TOWARDS AN AUCTION-BASED REWARD MECHANISM FOR EFFECTIVE BUG RESOLUTION

ÇAĞDAŞ ÜSFEKES

JULY 2019

TOWARDS AN AUCTION-BASED REWARD MECHANISM FOR EFFECTIVE BUG RESOLUTION

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED

SCIENCES OF

ÇANKAYA UNIVERSITY

BY

ÇAĞDAŞ ÜSFEKES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF

COMPUTER ENGINEERING

JULY 2019

Title of the Thesis: **Towards an Auction-based Reward Mechanism for Effective Bug Resolution.**

Submitted by **Çağdaş ÜSFEKES**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.

Prof. Dr. Can ÇOĞUN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Sıtkı Kemal İDER
Head of Department

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

Dr. Eray TÜZÜN
Co-Supervisor

Assoc. Prof. Dr. Murat YILMAZ
Supervisor

Examination Date: 31.07.2019

Examining Committee Members

Prof. Dr. Mehmet Reşit TOLUN	(Başkent Univ.)	
Assoc. Prof. Dr. M. Alp ERTEM	(Çankaya Univ.)	
Assoc. Prof. Dr. Murat YILMAZ	(Çankaya Univ.)	

STATEMENT OF NON-PLAGIARISM PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Çağdaş ÜSFEKES

Signature :

Date : 31.07.2019

ABSTRACT

TOWARDS AN AUCTION-BASED REWARD MECHANISM FOR EFFECTIVE BUG RESOLUTION

ÜSFEKES, Çağdaş

M. Sc., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Murat YILMAZ

Co-Supervisor: Dr. Eray TÜZÜN

July 2019, 72 pages

Bug management involves all the processes from discovering to reporting in a project. A bug may occur at any stage of software development lifecycle, and managing software development processes better reduces the number of bugs that may occur. In particular, in large-scale software development projects, the approach called ALM (Application Life Cycle Management) has been developed to define software development processes well and to manage their relationship with each other. ALM is a set of process covering the development, management and maintenance of source code in software development projects.

One of today's software engineering problems is the inability to use the Application Lifecycle Management (ALM) tools efficiently in software development. Within the delivery process fewer bugs improves software quality and customer satisfaction, however it may not be enough to test the product well. It is also significant that the

relevant bug records are distributed to software practitioners as efficiently as possible and quickly resolved during business planning. At this point, using gamification and reward mechanisms can be more efficient in the distribution and solution of software bugs. Gamification provides methods that make learning easier and therefore eliminates barriers to work efficiency by combining methods developed with new technology for a business or process with traditional game methods. Rewarding mechanisms aim to motivate each player in the game in line with a goal and increase the efficiency of the players by rewarding them provided that they are successful.

In this study, the effect of gamification on software developers' bug solving was observed by using Monte Carlo simulation. The study was carried out on a project developed within HAVELSAN. Firstly, a pilot project was selected and the resolution times of the bug records in this project were examined. Later, another study in which gamification was used in the training of real users was examined and the effect of gamification on the test results was calculated mathematically. We calculated a metric named as “gamification ratio” by comparing the pre-test and post-test results in this study. Monte Carlo simulation was designed on the value obtained with this calculation and the impact of the bug records of the pilot project on the resolution times was examined. In the simulation, virtual auctions and virtual players were created and these auctions were bid by the players. Each auction item has been created through bug records. After all, by comparing the bug resolution times of the pilot project with the resolution times obtained at the end of Monte Carlo simulation, it was observed that gamification increased the efficiency obtained from the bug resolution.

Keywords: Gamification, Bug management, Software productivity, Monte Carlo simulation, Application lifecycle management.

ÖZ

ETKİLİ HATA ÇÖZÜMÜ İÇİN İHALE TABANLI ÖDÜLLENDİRME MEKANİZMASI

ÜSFEKES, Çağdaş

Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı

Tez Yöneticisi: Doç. Dr. Murat YILMAZ

Eş - Tez Yöneticisi: Dr. Eray TÜZÜN

Temmuz 2019, 72 sayfa

Hata yönetimi, bir projede hatanın keşfedilmesinden raporlanmasına kadar geçen süreci kapsar. Hata, yazılım geliştirme yaşam döngüsünün herhangi bir adımında çıkabilir ve yazılım geliştirme süreçlerini iyi yönetebilmek çıkabilecek hata sayısını azaltır. Özellikle geniş ölçekli yazılım projelerinde yazılım geliştirme süreçlerini daha iyi tanımlayabilmek, birbirleri ile olan ilişkilerini yönetebilmek için UYY (Uygulama Yaşam Döngüsü Yönetimi) adlı yaklaşım geliştirilmiştir. UYY, yazılım geliştirme projelerinde kodun geliştirilmesi, yönetilmesi ve bakımının yapılması gibi adımları kapsayan bir süreçler bütünüdür.

Bugün, yazılım mühendisliğinde karşılaşılan zorluklardan biri Uygulama Yaşam Döngüsü Yönetimi (UYY) araçlarının yazılım geliştirmede verimli şekilde kullanılamamasıdır. Teslimat sürecinde, geliştirilen ürünün olabildiğince az hata içermesi, yazılım kalitesinin ve müşteri memnuniyetinin artmasını sağlar fakat bunun için sadece

ürünün iyi test edilmesi yeterli olmayabilir. İş planlaması sırasında ilgili hata kayıtlarının yazılım geliştiricilere zaman açısından olabildiğince verimli dağıtılması ve hızlı çözülmesi de önemlidir. Bu noktada oyunlaştırma ve ödüllendirme mekanizmalarını kullanmak yazılım hatalarının dağıtımı ve çözümünde daha verimli olunmasını sağlar. Oyunlaştırma, bir iş veya süreç için yeni teknolojiyle geliştirilen araçlar ile geleneksel oyun metotlarını birleştirerek öğrenme işlemini zorlaştıran, iş verimini azaltan engelleri ortadan kaldıran yöntemler sunar. Ödüllendirme mekanizmaları ise oyun içindeki her oyuncuyu bir hedef doğrultusunda motive edip, başarılı olması koşuluyla ödüllendirerek oyuncudan alınan verimi artırmayı amaçlar.

Bu çalışmada, oyunlaştırmanın yazılım geliştiricilerin hata çözmelerine olan etkisi Monte Carlo simülasyonu kullanarak gözlemlenmiştir. Çalışma HAVELSAN bünyesinde geliştirilmiş bir proje üzerinde gerçekleştirilmiştir. İlk olarak bir pilot proje seçilmiş ve bu projedeki hata kayıtlarının çözüm süreleri incelenmiştir. Daha sonrasında, oyunlaştırmanın gerçek kullanıcıların eğitiminde kullanıldığı bir başka çalışma incelenmiş ve oyunlaştırma kullanımının eğitim sonucu yapılan sınav sonuçlarına etkisi matematiksel olarak hesaplanmıştır. Bu hesap ile elde edilen değer üzerinden Monte Carlo simülasyonu tasarlanmış ve pilot projeye ait hata kayıtlarının çözüm sürelerine olan etkisi incelenmiştir. Simülasyonda sanal açık artırmalar ve sanal oyuncular yaratılmış, açık artırmalara bu oyuncular tarafından teklif verilmiştir. Her açık artırma öğesi hata kayıtları üzerinden yaratılmıştır. Pilot projeye ait hata çözüm süreleri ile simülasyon sonunda elde edilen çözüm süreleri kıyaslanarak oyunlaştırmanın hata çözümünden alınan verimi artırdığı gözlemlenmiştir.

Anahtar Kelimeler: Oyunlaştırma, Hata yönetimi, Yazılımda verimlilik, Monte Carlo simülasyonu, Uygulama yaşam döngüsü yönetimi.

ACKNOWLEDGEMENTS

I would like to thank to my thesis advisor, Assoc. Prof. Dr. Murat YILMAZ and my thesis co-advisor Dr. Eray TÜZÜN. They are one of the best supervisors who I have ever met in my academic life. They supported and guided me in this thesis study patiently. I had lots of useful discussions with them while studying in this thesis and I got lots of valuable comments from them.

I would like to express my deep gratitude to Yagup MACİT. He supported me during my thesis study and helped me about creating auction-based bug management simulation model. He is one of the best software engineer and software architect who I have ever met.

Finally, I wish to thank the thesis committee for their kindness during the presentation of this thesis study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF FORMULAS

LIST OF ABBREVIATIONS

ALM	Application Lifecycle Management
TFS	Team Foundation Server
VS	Visual Studio
MTTR	Mean Time to Resolve
MCTS	Monte Carlo Tree Search
GVGP	General Video Game Playing
URL	Uniform Resource Locator
LINQ	Language Integrated Query
XML	Extensible Markup Language
HTML	Hyper Text Markup Language
IEEE	The Institute of Electrical and Electronics Engineers
CCB	Configuration Control Board
IDE	Integrated Development Environment
DEVOPS	Development and Operations

CHAPTER 1

1. INTRODUCTION

1.1 Introduction

Defects are instances that cause incorrect results in the analyzing, software developing, designing, software testing, or in the deployment stages of a software project [102]. Defects in source code created during the software development process are called ‘bugs’. There may be several amendments to the source code throughout this process, either to improve its quality and performance or in resolving or attempting to resolve a different problem. Nevertheless, largescale source code amendments may inadvertently create critical bugs. It is therefore crucial to effectively employ bug management systems in order to minimize the occurrence of potential bugs, and also to preserve the quality of the source code within its evolved status during the software development process.

A well-designed bug management system facilitates the deployment of software development projects within planned budgets and according to scheduled delivery targets. A critical bug diagnosed within the advanced phases of the software development process may negatively impact the overall timeline of a project. There are several software development management systems available for the successful execution of the software development project timeline such as Application Lifecycle Management (ALM). Application Lifecycle Management encompasses the processes of version control system, requirement, build and test management with deployment, as well as monitoring and feedback.

The current study concerns the use of gamification along with ALM to improve the software developers’ motivation in the bug solution process within a software development project. Gamification provides users with a motivating and engaging environment for information exchange by distributing game resources as rewards such as recognition, badges or credits. Gamification in daily life creates an enjoyable working environment for the participants. In addition to daily life, gamification may also be applied to simulation software.

Monte Carlo [71] is a type of simulation based on the generation of values randomly. In a Monte Carlo simulation, a transaction is repeated multiple times in order to achieve a real-like value, and the more the transaction is repeated, the more real-like the resulting value.

The aim of this study is to observe the impact of gamification on the bug solution process through the use of a Monte Carlo simulation. To this aim, a completed software development project of HAVELSAN, a Turkish defense industry corporation, was selected as the pilot project, and named as "Project X" throughout the study. Bug solution timing in Project X was analyzed as the first step. Then, a different study examined the impact of gamification on the training of football referees. The referees were separated into two different groups, with the first group educated by using conventional methods, while the second group was trained using gamification. A pre and post tests were applied to all groups in order to see the differences and changes in their respective results.

In this study, a Monte Carlo simulation was designed using bug solution timing in Project X as well as mathematical data obtained from a reference study. An auction-based bug management model was developed for resolving bugs in the simulation, with participants (software developers) and auctions (bugs) created virtually. During the simulation, the participants were made to bid for auctions. Data obtained at the end of the study were compared to the data obtained through Project X in order to evaluate the impact of game systems on the bug solution process.

1.2 Objective of This Study

We aim to investigate the impact of using gamification in software development processes by using Monte Carlo simulation. Bug management is one of the processes of software development projects and it directly affects product quality. Gamification improves the efficiency of personal learning, working skills. In this study, we aim to show improvement of productivity on bug resolution in a simulation using gamification.

1.3 Research Questions

In this study, we aim to improve the software bug management process. Solving bugs in a shorter time is one of the most important part of this goal. In line with this goal, the research questions are listed as follows;

- RQ1: How can the auction-based reward mechanism help the bug resolution process?
    - RQ1.1: How can we minimize the bug resolution time?
    - RQ1.2: How can we demonstrate the effectiveness of auction-based reward mechanism in the bug resolution process?

In this study, the overall structure has five chapters with including this introductory chapter. The other chapters as follows:

Chapter 2 begins by the definition of software development, application lifecycle management (ALM) and literature view of game theory in software engineering with reward mechanisms. Then it continues with the definition and literature view of simulation and modeling.

Chapter 3 starts with the brief description of Mean Time to Resolve (MTTR) metric, followed by the description of the metric defined in this study: "Gamification Ratio," and gives detailed information about how we used it in the simulation. After that we explain our simulation model that depends on Monte Carlo simulation.

Chapter 4 starts with the introduction of our simulation implementation. Then it continues with the system description and tools that we used. After that, we explained the system functions and encountered problems. Details about the simulation results are then given and compared with the project bug data source sand basic statistical information about the winner users is presented.

Chapter 5 explains the conclusion of the study detailed and summarizes the study by giving a discussion about our method. The validation of threats is then classified in four categories and list the threats.

CHAPTER 2

2. BACKGROUND AND RELATED WORK

2.1 Introduction

This chapter starts with a brief review of the software development, with the emphasis of software development processes, agile software development, and ALM. Next, we provide details related to the definition of games and gamification with a review of the theory of games in software engineering's literature and reward mechanisms. The next part continues with the literature review of applying gamification techniques in a bug tracking context. The chapter ends with the definition of Monte Carlo simulations and sample usage of it in various studies.

2.2 Software Development

Software development is a process of understanding, describing, designing, coding, testing, bug fixing and documenting software applications, software frameworks or any other software components [6]. Software development can include research about new development techniques, methodologies, prototyping, and modifications. It should be reusable and support re-engineering [64]. Software engineering is a type of engineering which provides developing software in a systematic method [65].

The software development process describes the procedures for creating software products and services [33]. These processes can be used by software organizations or individuals. Software development process divides all development work into small parts to improve the design, project management, and product management. The software development process includes some related activities. These activities help to develop products and provide a road map for software development with a budget and plan. Each activity includes a task which is the smallest work unit [66]. Software development companies can be considered as social organizations built on employee's skills. Based on

software organizations' skills, resources and goals, they directly use a process or modify a process regarding their needs [33] [67].

The software process improvement methods should include different actions to improve the software project's quality [2, 26]. These actions should be reevaluated by considering factors affecting software development activities (e.g. human characteristic in coding, social relationship problems, e.g.). In the software development process, software practitioners are seldom working by themselves; in almost all cases they are working in various teams so all of the working actions can be considered as an activity of social relationships [27]. The software developers who is working in various development teams are affected by different social factors. These factors are not limited with their interdependence, rationality, characters and working conditions [28]. Improving the quality of software projects and finish the project in planned budget is an important aim of improvement of software processes [3]. To achieve this point, a coordination mechanism from development to maintenance and management are needed. As an example, while the project is getting larger, source code and technical documentation readability is decreasing. Therefore, the members of a software development team has to work in coordination while the team is getting larger. The level of coordination about this teams affect the software product's quality. At this point, the software development problems can be assigned to the correct team members [84] [103].

2.2.1 Agile Software Development

Agile software development is an approach which is organized solutions and requirements by cross-functional teams in software development [5]. With agile software development, some type of lightweight software development methods is described. At first "Rapid Application Development" was announced [54]. Then "Dynamic System Development Method", "Scrum", "Crystal" and "Extreme Programming" and "Feature-driven Development" announced. After that, Manifesto for Agile Software Development was published [57]. The agile manifesto focuses on customer collaboration and interaction while working on software.

Agile Software Development methods divide the main work into small pieces to minimize the planning and design phases. The name of these parts is iteration or sprint in Scrum [56]. Iterations can be one week to four weeks [56]. Each iteration includes a cross-functional team that is working on planning, analyzing, designing, developing, and testing. In cross functional teams people who has different expertise work for a common

goal [56]. At the end of each iteration, the output is shown to customers. Working with iterations decreases the overall risk and allows change product easily.

In Scrum teams, an assigned member is held responsible to represent stakeholders. This member is called “Product Owner” of the team. Product owner provides communication between stakeholders and the team. When each iteration ends, this member and stakeholders review progress and order product priorities [58]. In each iteration, product owner assigns the bugs or tasks to the other team members (developers).

2.2.2 Application Lifecycle Management (ALM)

Application lifecycle management (ALM) is a ruleset for managing and integrating the processes which are related to the development, governance and maintaining of a software project [98]. ALM can be divided into three areas. These are, operations, development and governance. In the governance part, we have to be sure that the software product always provides what the customer needs [1] [47]. Governance step includes all of the steps of Application lifecycle management. Development step is a common part of all lifecycle of software product [1] [47]. After the development step finishes operation step starts [1]. When the deployment step is finished, every software project needs to monitoring and managing. In operation step, project builds and deployments are managed.

ALM 1.0 proposes to use different tools for every discipline and every tool works properly between each other [1]. These tools include development, test, build, design and management modules [50]. In ALM 1.0 various software companies developed different tools and also there were orchestration problems between these tools. Therefore, “ALM 2.0” is announced [50, 32]. In ALM 2.0 all of the software development processes can work in one tool [49, 51]. Tools that appropriate to the solution of ALM 2.0 follow a role-based and authority-based approach throughout this process. In ALM 2.0, data modeling is suitable for communication between different tools.

With the advantages of the ALM 2.0, large companies started to use ALM process in their projects [104]. In ALM 2.0 all steps of a project can be followed in one tool. In a project from start point to endpoint software development engineers, test engineers, configuration managers, project managers, etc. are using ALM tools. ALM tools supports software engineers work with a single framework. This framework includes everything for the many modules that the software development process requires [49]. These modules are as

follows (i) requirement, (ii) test, (iii) build, (iv) project, (v) source code and (vi) task management. A software project development process depends on these modules and each of them has chain-like structure. These various processes are integrated between each other successfully in ALM and this ability is important about software products' delivery process. Employees who are from different roles use relevant ALM tools. So all of these employees know their toolset. The whole spectrum of the ALM process is addressed with various ALM tools. In particular, it is challenging for software practitioners to fully engage with the tasks that are assigned to them in these ALM tools. In this study, we aim to create a reward mechanism to use ALM tools more efficiently.

2.3 Definition of Games and Gamification

The notion of games is relevant to studies of social aspects of software development, which have gained increasing attention among researchers [59]. Recently, several researchers have conducted research to explore the potential usage of games in software development activities in terms of collective behavior: altruism and selfishness that ultimately affect the software product health. Games are a special kind of social activities, which can easily highlight the social interactions or engagements that could offer a variety of measurable societal outcomes. Over the last decade, games have reshaped the ways of communication by the help of social media to promote cooperation and competition. Serious games are used for game-based social skill training that helps individuals to gain social responsibility through the creation of fun and engaging environments. Emerging trends improve the popularity of researchers and practitioners who have redefined the notion of games in non-gaming contexts. Consequently, the term gamification (i.e. the use of game elements in non-gaming practices) becomes an emerging subject for improving the software development processes. It not only has a great potential to align individuals' motivations with software development task but also helps to address a variety of information technology related issues [84] [103].

2.3.1 Theory of Games in Software Engineering Literature

Research into games has a long history. The theory of games first appeared in the literature at the 1930s. A game highlights strategic interaction among individuals, teams, units, or infrastructures. Historically, research investigating the individuals' interactions

associated with games has focused on analytical methods and tools to aid the decision-making process [2]. Around the early 1960s, small-scale research and case studies began to emerge linking theory of games with social science successfully. Especially, in the last 15 years games become popular and companies are using game-based techniques to analyze the characteristic features of their employees.

Game Theory is a set of analytical tools, which can be used to model the interactions between participants (e.g. individuals, companies, nations, etc.) in a game form [60]. In addition, it can be used to explore the actual or essential decisions and behaviors, and ultimately their consequences that may include tradeoffs or conflicts among individuals. The most important fact about game theory is that it assumes all players are rational. In other words, all players follow the rules of a game and hence their goal is to win. In the last decade, use of game theory has become widespread, not only in economics but used in the fields of psychology, biology, and computer science [3]. Game theory has both cooperative and non-cooperative forms. However, it is mostly known for its "non-cooperative" form [3]. In this approach, the goal is to design a controlled competition where selections of participants are likely to affect every single player's benefits. These players are considered as successful when they mind their benefits based on choice architecture. This architecture is the designing various types of ways about which choices could be shown to consumers and the effect of the selected way on consumer's decision [99]. Nash [4] coined "Nash Equilibrium", which describes the optimal outcomes of a game by predicting the outcome of strategic interactions.

There are many examples of using the theory and practices of games and the use of game elements to address a set of problems in software development. For example, Cockburn [8] accepted that software development is a kind of game based on limited project resources, communication, and coordination skills. Baskerville [9] analyzed high-speed internet development from a balancing game viewpoint that depends on high usage of resources. Lagesse [7] created a game theoretic model for assigning tasks to software practitioners. Sullivan [10] worked to evaluate software design decisions by economic approach. Sazawal and Sudan [11] combined the theory of games and decision modeling structure to improve software design. In this work, they designed a game called "software design evaluation". This game aims to address problems between developers and customers. Moreover, they suggested a lightweight game theoretical analysis technique to assess software development teams.

Gao [12] designed a game theoretic model to configure software products and decision errors. Gao-hui [13] worked about depending corporate software developments to game

theory. Soska et al. [14] worked about students in academic life. In this work, they designed a card game to teach students about software testing. In addition, Pedreira et al. [15] created a systematic map about the usage of gamification in software engineering. By this work, they aimed to find opportunities for next works. In recent years, gamification becomes more popular in software engineering research. Sweedyk [16] worked about the popularity of gamification in academic programs and conferences. In 2016, Kitagawa and others created a game theory on code review. Code review has a big effect on software quality in the development process and it aims to decrease the number of bugs [17]. Szabo [18] applied "Game Dev Tycoon" game to students for teaching software engineering. This game is about business simulation. Amir [19] worked about getting systems more gamified and effective with using gamification. Ranganathan [20] used gamification in hardware engineering. He supported a low power timer on the circuit by a game theory which depends on "Nash equilibrium". Nash equilibrium is a solution concept for an individual game. In this theory there are at least two or more players. Each player is assumed to know strategies of equilibrium for the other players [61].

A game is a useful tool to reveal interpersonal conflicts. This situation is known as a "social dilemma". A social dilemma is a situation where people takes advantage of selfishness unless everyone selects an egoistic option. In this case, the whole group loses [62]. "Prisoner's dilemma" is a basic framework that often used by researchers to observe such issues. Hazzan and Dubinsky [21] suggest that "prisoner's dilemma" is useful to highlight the problems in software development. Fejis [22] designed a game theoretic model for software developers and testers. He worked about the results of this game and said that these results may cause "prisoner's dilemma". Costa [23] combined the "Prisoner's dilemma" with gamification and he designed war and peace game by using this combination. In another work, Mortensen [24] used "prisoner's dilemma" in security and privacy of web technologies. In this work, he defined seven strategies and created a strategy to exceed "prisoner's dilemma" of web technologies by using a set of strategies.

Several lines of evidence suggest that building a mechanism for automating software development activities by designing game-like activities is essential [29, 30, 31]. Yılmaz [32] designed a game-like approach to explore the effects of team personality characteristics in software development activities. Yılmaz et al. [31] created a gamification approach to improve the software development process. The idea of creating an economic mechanism for software development is introduced by [30], which was one of the first serious discussions about the subject matter. One study by Yilmaz et al. [33] proposed an economic mechanism for improving the software development process. Yılmaz and O'Connor [34] suggested a complementary approach to ScrumBan to

improve the software development process using gamification. In another work, Yilmaz and O'Connor [35] considered software development as an economic activity and created a market-based approach to investigate task assignment problems. Collectively, these studies confirm that using game-like approaches in the software development activities have a significant impact on software productivity improvements. One study by Jurado et al. [36] proposed a system to design game elements. This system includes three components. These components are; game environment, game environment process, and component to measure and evaluate. This study analyzes knowledge management, and game elements to determine the relationship of motivation for contribution, collaboration and participation in the definition of knowledge management steps, especially in the scenarios of software development projects [36].

2.3.2 Reward Mechanisms

A reward mechanism can be considered a feedback device, which is an important aspect of game design. A considerable amount of literature has been published on computational features of these mechanisms. Houk et al. [37] investigated the models of intelligent behaviors and their relation to reward mechanisms. In another work, Singh [38] proposed a reward mechanism for online learning systems. Singh studied to categorize web pages to predefined subjects which is based on an available text in URL. Lua [39] worked on a reward mechanism which is designed for P2P systems. They designed a reward mechanism for reducing the costs of P2P systems. Wang and Sun [40] researched reward mechanisms which was designed for computer games. In this work, they discussed how reward mechanisms can be used to motivate or change behaviors in the physical world.

Reward mechanisms have a crucial impact on associative learning and cognition [63]. In addition, they are likely to game elements. If a reward system is constructed properly, it is likely to improve the motivation of the participants. Game elements potentially assist people to solve problems in an enjoyable way, e.g. while they are working on routine tasks. Walz [41] defined a game as a closed system that depends on social and cultural fundamentals of cultural values. Gonzales [42] described the advantages of games for teaching a process in computer engineering. Qu [43] worked on teaching software engineering. Largo [44] collected lots of feedback from students and he examined game elements in the learning process.

Large corporations are using more complex systems. These can be engineering management tools, financial automation tools, etc. To use these systems efficiently,

employees must be experienced. At this point, employees make more effort to use these systems efficiently. In this process using gamification accelerates the employees learning process. For example, in software engineering, Pariza [45] designed a game about traceability in software tests and while conducting source code inspections. He designed a game about traceability in software tests and code artifacts [46].

2.4 Application of Gamification in Bug Management

Bug management is one of the process of software development lifecycle. Solving bugs in late periods of software development cause high cost. Fixing bugs in development period increases the quality of the product. Finishing software development period with zero bugs is impossible. Test engineers can detect bugs with running test scenarios before deploying products to the customer side.

Gamification can be used in bug tracking because game elements and game scenarios can motivate the developers to solve more bugs. Lotufo [68] used Stack Overflow bug source in a work. Stack Overflow is a website to search answers about software development failures. They use game elements to address these problems by motivating the contributors. Sasso [69] used gamification to gamify software engineering and also bug reporting. Sasso [69] used building blocks to define a game-based system. In other work, Fraser [70] try to set a new view to testing and detecting bugs using gamification. Zheng [92] at al. created a bug management framework to support development of product. In this work, they focused on hardware products and they developed this bug framework which is depend on activities that evaluate and define the design failures. Aqlan [93] combines data analyze methods with the simulation modelling for developing an approach to use in bug management. In another work, Rahman [94] designed an approach for life cycle of bug management to improve software quality. The main goal of this study is defining a bug management process and its details. Taba [95] defines an extensive approach to inspecting of software project. This model offers unique equipment for collecting prevalent barriers to inspection. Weerd [96] defines a new conceptual approach to integrate software product management (SPM). In other work, Nair [97] defines an effective bug management process for project managers. This research enables to increase the quality of software projects and helps to the project managers about resource allocation on project management [97].

2.5 Simulation and Modeling

In simple terms, simulation and modeling is a substitute for every experiment in physical the world. Computers can be used to calculate the result of these experiments. As such simulation and modeling can simplify understanding the behavior of a system without testing in the real world. Using simulation and modeling within engineering is well recognized. This helps to increase the quality of software projects and decrease the costs of project [100]. In this study, we studied Monte Carlo simulation to validate our results.

The simulation which based on Monte Carlo method is a type of stochastic simulation system which depends on choices by selecting randomly for modeling aspects of the real-life system [71]. In this simulation technique, a condition is repeated multiple times to calculate nearest results. This simulation is used in mathematics and physic and it can be used in a wide variety of settings, from medicine to the software industry. We can use Monte Carlo simulation in three areas. These areas are sampling, estimation, and improvement [72] [73]. Monte Carlo is concerned with “Sampling”. “Sampling” is a process that simulates the non-virtual system behaviors like telecommunication network systems [72]. In "Estimation", the focus is guessing the numerical values about a simulation model. An example usage of Monte Carlo method can be given as expecting the productivity of a product line. Another usage of Monte Carlo method can be given as using random variables to evaluate the multi-dimensional integrals [72]. Monte Carlo method is used to refine noisy functions by using random variables [72].

Raychaudhuri [74] describe the Monte Carlo method is creating multiple samples as randomly to calculate and analyze the simulation results. Burgin [75] used the Monte Carlo methods in super-recursive software development algorithms. Kalantari [76] used Monte Carlo methods in a neural network to clear the noises from data. In another work, Neese [77] used Monte Carlo method to solve complex bounded integrals within seconds using Java programming language. Soemers [78] studied on a research that using Monte Carlo Tree Search (MCTS) in General Multimedia (Video) Game Playing. General Video Game Playing (GVGP) is a child of Artificial Intelligence (AI) and Monte Carlo Tree Search (MCTS) is a type of search method for playing of game and it is not a domain-based approach. In another work Maia [79] used MCTS and Google Maps in location-based games. They presented a study that shows the improvements in game balancing using the two most popular location-based games "Ingress" and "Pokemon Go". And Lorentz [80] studied on MCTS by using evaluation methods. They designed an algorithm named as MCTS-EPT and this algorithm depends on three various games. These games

are; Havannah, Breakthrough, and Amazons. Malefaki [81] designed a rejuvenation in a computer system to protect from software aging. At the end of this study, they calculated and compared performance results to show the effect of software rejuvenation. In another work, Pacagnella [82] used Monte Carlo simulation to calculate cost estimations in software development projects. Madani [83] suggested to solve multi-criteria decision making problems by using non-cooperative gamification approaches.

CHAPTER 3

3. METHODOLOGY

3.1 Introduction

This section describes the methodology of the current thesis. It begins with software metrics and a brief description of Mean Time to Resolve (MTTR). Mean Time to Resolve is a measuring type of the repairable items' maintenance. MTTR calculates the average time that takes to resolve a non-working component like software bugs. Next, the chapter continues with the definition of "Gamification Ratio". We reference another study [88] to calculate "Gamification Ratio". The reference study includes two training examinations. The first exam is performed before training and the other is performed after training. We calculate MTTR values on these exam results to calculate gamification ratio value. Next, we designed a Monte Carlo simulation to simulating the use of auction-based bug management. Gamification ratio is used as an input in this Monte Carlo simulation.

In this study, we aim to show that using reward mechanisms in software development projects decreases the bug resolution time. Reward mechanisms help to improve the motivation of employees about their work. In this research, auction-based reward mechanism is designed to enable software developers to select bugs by themselves. After all, software developers can solve bugs in less time with higher motivation and we can minimize the bug resolution time using reward mechanisms.

A Monte Carlo simulation is designed to simulate auction-based reward mechanism. In the simulation algorithm "Gamification Ratio" and bug resolution time of Project X are used as inputs. Then, the algorithm applies the "Gamification Ratio" to the real bug resolution time and creates virtual bug items with possible virtual resolution time. Mean Time to Resolve (MTTR) method is used to calculate the average resolution hours in simulation. By this way, we can demonstrate the effectiveness of auction-based bug management and we can calculate the impact of reward mechanism on bug resolution process.

3.2 Research Design

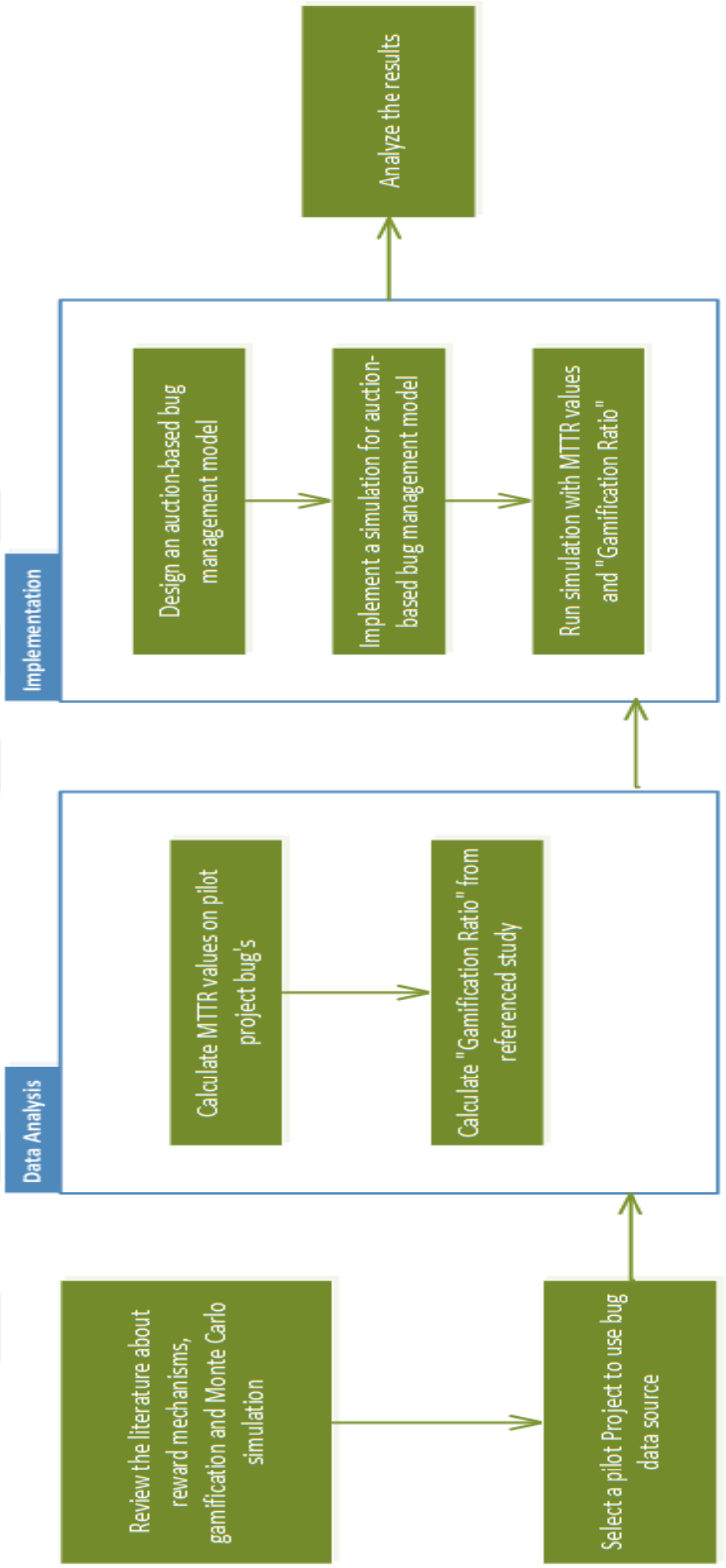

Figure 1 The Research Design Process

In this study, we categorized our research process into two parts. The first part focuses on data analysis and the second part focuses on implementation (see Figure 1). In the first part, we reviewed the literature about Application Lifecycle Management (ALM), gamification, reward mechanisms and Monte Carlo simulation. Then we selected a pilot project to analyze bug resolution time values. In the second part, we calculated Mean Time to Resolve (MTTR) of bugs. Then we studied on another referenced study to calculate "Gamification Ratio". After all, we designed a Monte Carlo simulation algorithm to simulate an auction-based bug management system. The simulation uses MTTR values and "Gamification Ratio" as input and creates new MTTR values as output.

The purpose of this simulation (see Figure 1) is to shorten bug resolution time. In this model, every single bug is equal to a single auction and this model can provide software developers to select any bug by themselves. The team leader or product owner does not need to assign any bug to any software developer. Using an auction-based system increases the motivation of employees about their work. By this way, software developers can solve bugs with higher motivation in less time and bug resolution time decreases using gamification.

3.3 Auction-Based Bug Management Model

Application lifecycle management (ALM) systems do not necessarily suggest the most efficient methods to software developers while they are assigning bugs. The goal of this model is using individual choices to improve software productivity while developers are assigned bugs. Users can join multiple auctions which are defined in this software model. Auctions can be related to requirement analysis, software testing or etc. Therefore, users can choose the bugs that motive them the most from a pool with resource distribution method. This model is proposed as a resource management framework to define the task choices based on the priority of the software developer's selection. This system aims to make the task assignment and time planning in an efficient way. The workflow of the system is seen in Figure 2.

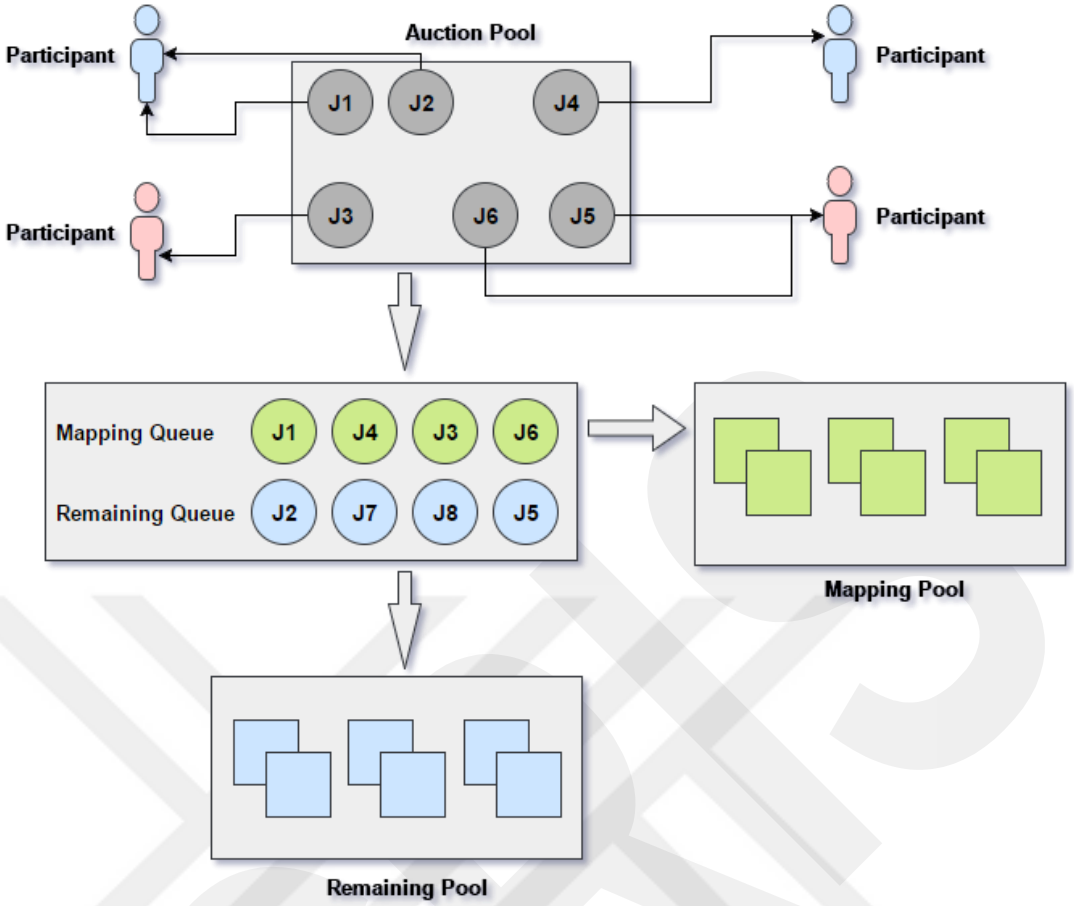

Figure 2 Auction-based Bug Management

The main aim of this mechanism (see Figure 2) is to reform the software development activities in a resource economy model where software practitioners have initial credits, which enable them to select these bugs regarding their preferences. Based on the proposed model, we announce the bugs to the software developers in an auction like structure. Similar to story cards, these bugs are based on their effort and complexity points. A practitioner requests a set of specific bugs depending on the amount of credit they might be able to pay by using the auction mechanism. From these requests, the proposed mechanism selects the practitioner who desires to do this job the most. In this way, a gamification based value mapping occurs between bugs and resources. The system ensures that a user has to bid on their own budget and allows the price to stay constant over time. The system uses a set of game elements to motivate its users such as giving reputation, badges, and leaderboards (i.e. to create community leader with more privileges). Consequently, participants who finish their bugs in time are rewarded by the system based on the importance of their achievements. All this information is announced to system participant to foster their motivation (see Figure 2) [84] [103].

Using this technique, individuals in a team can bid for the work they would like to perform and in the context of their available credits. We believe that this could have interesting ramifications for productivity and knowledge diversity among individuals in software development teams. Here, we suggest that this is a useful vehicle for risk reduction in software companies since everyone has the right to bid for work in the context of their credit position. Let's look to a metaphor – a golfing handicap. In amateur golfing competitions, individuals participate in competitions but their score is modified on the basis of their handicap /ability with the result that the winner is not the player who shot the absolute score for the round of golf but rather the winner who shot the lowest score taking into account their own ability. This means that everyone competes with the ability to win the competition and everyone is trying to improve his or her own personal performance.

3.4 Variables and Measures

In this study, we used two variables to create a simulation model for auction-based reward mechanism. These variables are;

- MTTR values of bug items
- Gamification Ratio

In this part, we described the definition of MTTR (Mean Time to Resolve) and how we calculate the MTTR values of bug items. Then, we gave a reference study [88] to calculate the "Gamification Ratio" value.

3.4.1 Mean Time to Resolve (MTTR)

A software metric is a way of measuring software characteristics as countable and quantifiable. In the software development process, there is various type of metrics that are related to each other. In this study, we used "Mean Time to Resolve (MTTR)" as a software metric to calculate the time intervals on bug resolution.

In software development projects, one of the critical customer satisfaction criteria is to be able to fix bugs in short periods of time. Time to fixing a selected bug is the time elapsed between when a bug is reported (i.e. entered into the "Proposed" state in the bug management tool) until a resolution to the bug is verified by the test engineer (i.e.

entering a “Closed” state in the bug management tool) [89]. We selected Team Foundation Server (TFS) as a bug management tool because bug data source was located in TFS. Team Foundation Server is developed by Microsoft and it is useful on application lifecycle management processes. Team Foundation Server has some type of work items and users can create custom work item types. These are examples of some work item types:

- Backlog
- Task
- Bug
- Requirement
- Nonconformity
- Test Case
- Test Suite
- Test Plan

In this study, we focused on “Bug” work items. The workflow of a bug item is shown in Figure 3.

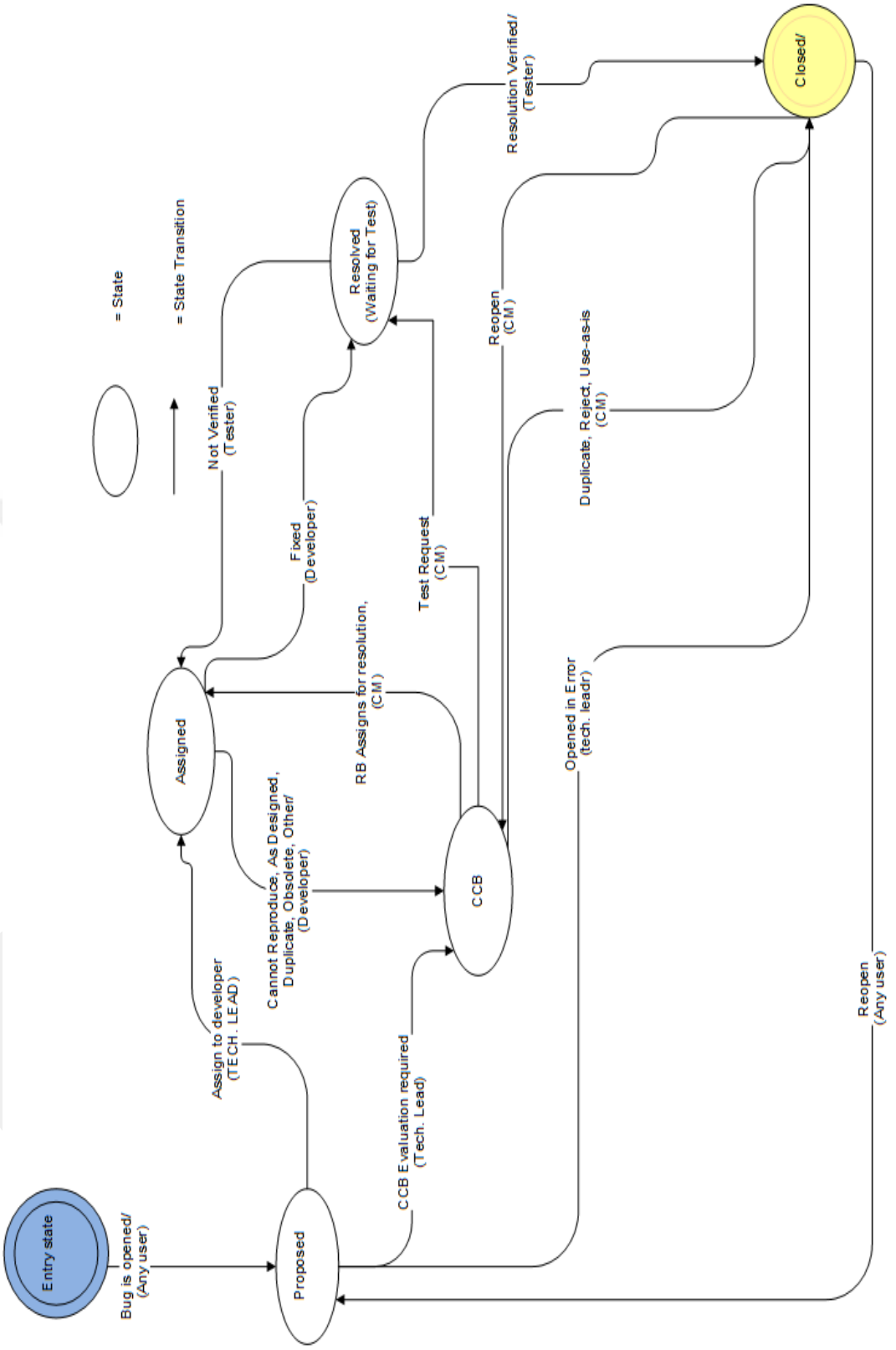

Figure 3 Bug workflow schema

Throughout its life cycle, a bug item always begins with “Proposed” state and ends with “Closed” state. It is necessary to calculate the duration using timestamps from “Proposed” and “Closed” state. This metric is usually measured in days or hours. Then we can calculate and use “Mean Time to Resolve” (MTTR) as a metric to examine this perspective. Mean Time to Resolve is a measuring type of the repairable items’ maintenance [89]. It calculates the time required to fix a non-working component or machine. It is the average repair time for failed items divided by the number of maintenance operations for failures during in a time range [90]. Fousch [91] has previously focused on software solutions for MTTR predictions. The formula for MTTR is given as follows;

$$MTTR = \frac{\sum_1^n Elapsed\ time\ to\ fix\ a\ bug(i)}{n}$$

Formula 1 MTTR formula

Calculating elapsed time to fix a bug is shown in Formula 2.

$$Elapsed\ time\ to\ fix\ a\ bug = Timestamp\ [Closed] - Timestamp\ [Proposed]$$

Formula 2 Elapsed time to fix a bug

If we further expand the formula, we will have the following Formula 3, where n is the total bug count in project.

$$MTTR = \frac{\sum_1^n Timestamp\ [Closed](i) - Timestamp\ [Proposed](i)}{n}$$

Formula 3 MTTR detailed formula

This is an important metric to calculate the bug resolution time of the team. Although it is useful to show which failure took long a time to repair, MTTR gives an overall indicator of the performance of the team. Since in general, your team can resolve bugs for the customers, the happier customers will be, this metric is directly related to customer satisfaction.

The metric also would provide an indicator of the team's efficiency. By analyzing this metric, we can explore various types of bottlenecks in the overall bug resolution process.

3.4.2 Gamification Ratio

In this chapter, the gamification ratio based on a previous related work which was published in 2016. Gulec and Yılmaz [88] examined decision-making skills on 54 Turkish football referees. Firstly, we gave a brief description of this study and then we described how we calculate the gamification ratio.

Gulec and Yılmaz [88] created two groups as an experimental and control group from 54 referees. The football referees were divided into two groups randomly. The first group named as "control group" (included 27 referees) and the second named as "experimental group" (included 27 referees). The control group members were not allowed to use the game system. The control group members could train by using only the LOG book (Law of the game). The football referees who are located in the experimental part could access the game system. However, the LOG book was forbidden to experimental group members [88]. Experimental group members are trained with a serious game and control group members are trained by classical referee training system from a book. All of these groups are tested before and after training. They called the test before training as "Pre Test" and called the test after training as "Post Test".

A pre-test was applied to the all of the group of referees before the beginning of the education period. By this test, the referees' knowledge level has been determined. This test included 50 questions and the exam questions were divided into two parts thus [88]:

- Test Questions
- Multimedia Questions

By these question types, 15 multimedia (video) questions and 35 test questions were selected.

After the training finished, a post test was applied to a part of football referees. The post test was the same as pre-test. Exam content and question count were same because questions' the difficulty degree should be equal in pre-test and post-test because we should provide validity on these tests. The pre-test scores and answers were not exposed in the training period.

The pre and post-test results for experimental group members are seen in Figure 4.

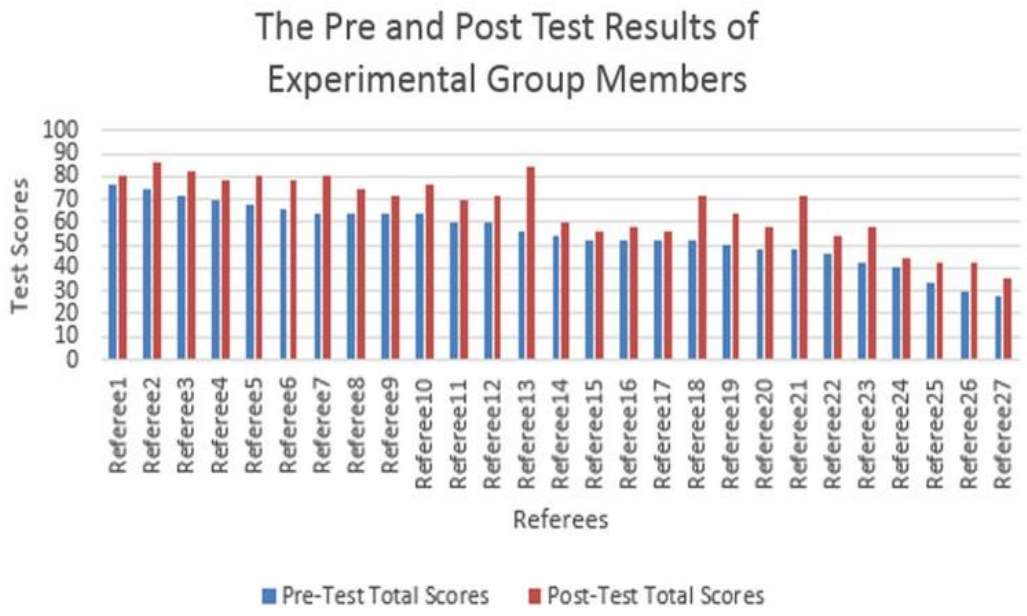

Figure 4 Experimental group test scores [88]

The post and pre-test scores of control group members are shown in Figure 5.

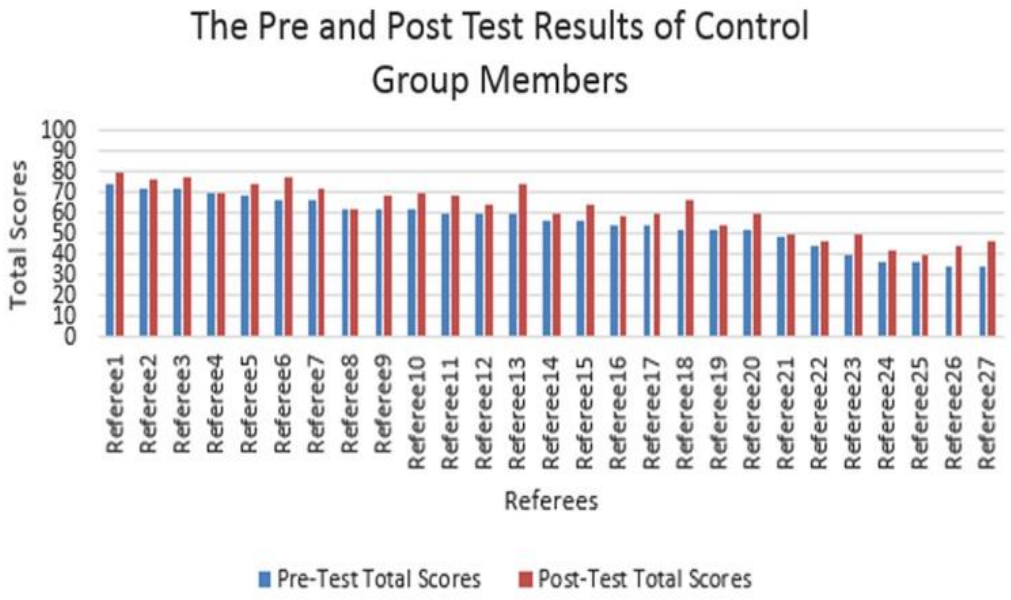

Figure 5 Control group test scores [88]

Average test results for all groups and all question types in pre-test are shown in Table 1.

Table 1 Pre-test results [88]

	Test (70 points)	**Multimedia (30 points)**	**Totally (100 points)**
Experimental Referees	38.3	16.8	55.1
Control Referees	38.4	17.3	55.7

Average test results for all groups and for all question types in post-test are shown in Table 2.

Table 2 Post-test results [88]

	Test (70 points)	**Multimedia (30 points)**	**Totally (100 points)**
Experimental Referees	43.6	22.6	66.1
Control Referees	41.2	20.9	62.0

Now we can crosscheck the post-test scores with the pre-test scores. Firstly, we calculated the increase of success (IoS) on the experimental group. This group are trained by gamification methods and group members' average score point is 55.1 in pre-test and 66.1 in post-test. We set the increase of success for the experimental group as a variable IoS(E). IoS(E) calculation is shown in Formula 4.

$$IoS(E) = 66.1 - 55.1 = 11$$

Formula 4 Calculation of IoS for experimental group

We converted the IoS(E) value to percentage in Formula 5.

$$IoS(E)\ \% = \frac{11 * 100}{55.1} = 19.96\ \%$$

Formula 5 Percentage of IoS(E)

Secondly, we calculated the increase of success on a control group. Control group are trained by classical referee training system and group members' average score point is 55.7 point in the pre-test and 6.20 point in the post-test. We set the increase of success for the control group as a variable IoS(C). IoS(C) calculation is shown in Formula 6.

$$IoS(C) = 62.0 - 55.7 = 6.3$$

Formula 6 Calculation of IoS for control group

We converted the IoS(C) value to percentage in Formula 7.

$$IoS(C)\ \% = \frac{6.3 * 100}{55.7} = 11.31\ \%$$

Formula 7 Percentage of IoS(C)

If we look at these results, we can analyze that the experimental group is 19.96 % more successful in using gamification in the training period. However, control group is 11.31 % more successful in using classical referee training system.

As a result, we can calculate the effect of using gamification in this training system by taking difference of two values in Formula 8 and Formula 9.

$$Gamification\ Ratio = IoS(E) - IoS(C)$$

Formula 8 Formula of gamification ratio

$$Gamification\ Ratio = 19.96\ \% - 11.31\ \% = 8.65\ \%$$

Formula 9 Calculation of gamification ratio

In this study, we set the gamification ratio to 8.65 % and used it in Monte Carlo simulation application. The detailed information about how we used this ratio will be given in the next chapter.

3.5 The Auction-Based Bug Management Simulation

In this study, we designed a simulation that uses Monte Carlo method. The simulation includes five parts. These parts are:

- Loading bug data
- Creating virtual bugs (auctions)
- Creating virtual users
- Running simulation
- Calculating winner users and MTTR values

In the Monte Carlo simulation, all parts execute in an order like seen in Figure 6. In first part, user loads the bug data as a file input. This file contains fixing the time and completed work values of bug items. In second part simulation creates virtual bug items. We named as “Auction” to virtual bug items in the simulation model. In third part, the simulation creates virtual users. Each user has the same credit value at the start of the simulation. In fourth part, the simulation starts to run and random users bid random bugs. In each bidding action, users spend credit and if a user wins an auction, user earns the credit which is equal to the point of auction. In fifth part, the simulation calculates the MTTR values of virtual bug items and lists the winner users with their credit value.

In the last part, we can compare the simulation results with the previous MTTR values which are calculated from project bug data. However, we can show the effect of gamification on bug resolution using Monte Carlo method. The activity diagram of Monte Carlo simulation is seen in Figure 6.

The simulation can export the simulation results to XML or text file. We can run the simulation by changing parameters in every run and we can save results to separate files. By this functionality, we can see which variable change the results.

Monte Carlo simulation uses two parameters as input. These are:

- Project bug data
- Gamification Ratio

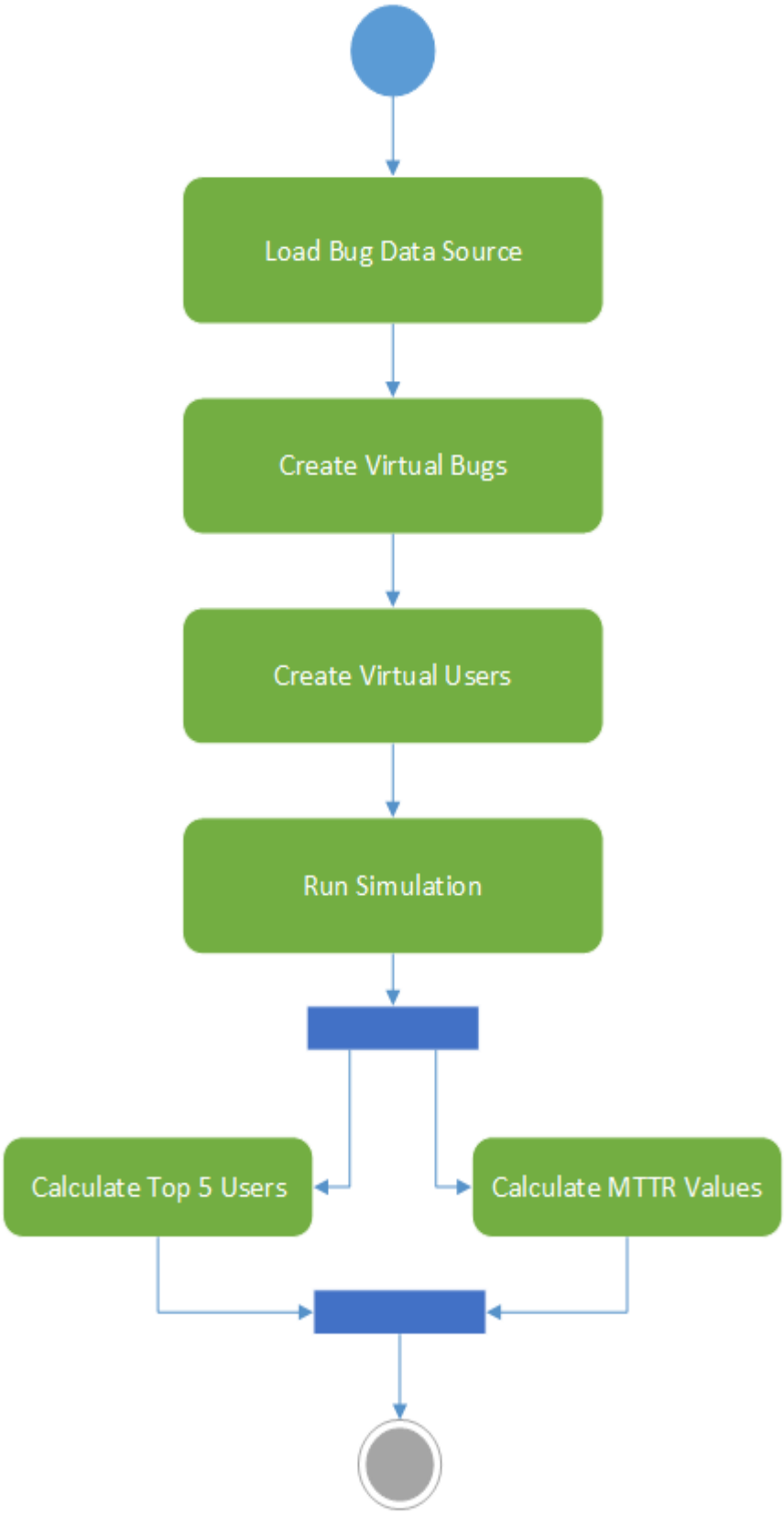

Figure 6 Activity diagram of Monte Carlo simulation

3.5.1 Analyzing Bug Data

In this step, we worked on a pilot project's (Project X) bug items. These bugs were generated in a real project's lifetime and categorized by milestones. Bug source has four milestones. These milestones are seen in Figure 7.

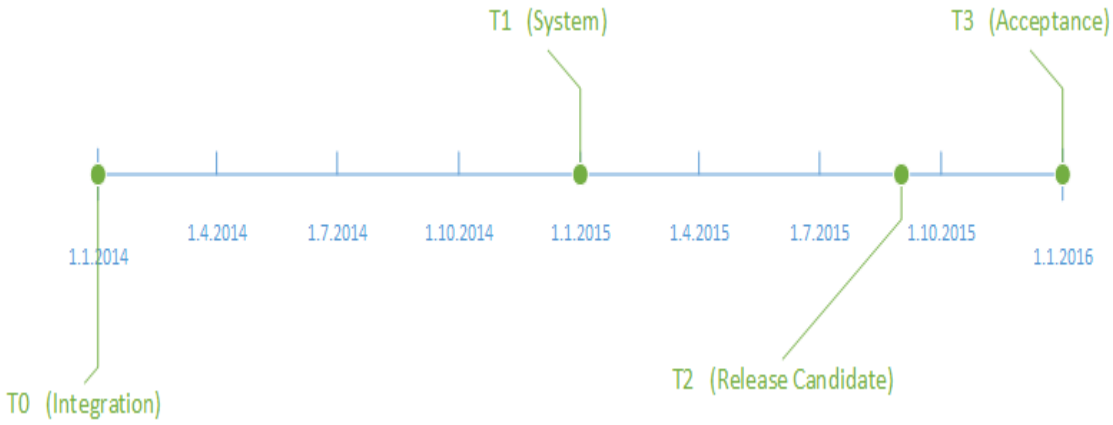

Figure 7 Project X milestones

We calculated bug counts, percentages of bug counts and MTTR values for all milestones to compare them with the simulation results which are calculated after running Monte Carlo simulation.

3.5.2 Calculating Gamification Ratio

As previously mentioned, we calculated the gamification ratio value that depends on another study. In the referenced study, the pre-test was applied to 27 football referees before education and a post-test was administered after education. This education was performed by using gamification. We compared post-test and pre-test scores and calculated 8.65 % difference. This value named as "gamification ratio" in this study and used in Monte Carlo simulation. Gamification ratio is described clearly in Chapter 3.4.2.

3.5.3 Creating Monte Carlo Simulation

At the final step, we designed a Monte Carlo simulation algorithm. This algorithm uses project bug data source and gamification ratio as input and creates a virtual auction-based

simulation. At the end of this simulation, we can compare the simulation MTTR values with the project MTTR values to see the effect of gamification. The simulation has five parts.

- **Loading bug data:** In this part, the simulation loads the fields of pilot project bug items. The simulation reads fixing time values and completed work values from bug items.

- **Creating virtual bugs (auctions):** The simulation creates a virtual bug item for every real bug item. Virtual bug items named as auction items. Each auction has four fields: These are:

    1. **Point:** Point is a numerical value that is created randomly. It has minimum and maximum limit. These limit values are determined by user. Point can be called as a price of the auction. This means if the user wins this auction, he/she earns the point as credit**.**

    2. **Fix Time:** Fix time is a numerical value that is read from bug items. Fix time is bug resolution day count. This value is calculated by subtracting the proposed date from closed date. Proposed and closed date information is located in the selected ALM tool.

    3. **Team:** The bug source includes five different development teams. These teams are loaded in simulation and each auction receives a team value randomly.

    4. **Completed Work:** Completed work is a numerical value that is read from bug items. Completed work is required field and each bug has completed work value. This value indicates how much time it was spent to resolve a bug. The simulation sets completed work values to every auction one by one.

- **Create virtual users:** The simulation creates virtual users to bid auctions. User count and credit per user can be defined at the start of the simulation.

- **Run simulation:** The simulation needs some parameters to run. These parameters are;

    1. **Auction Count:** The value of how many virtual bugs (auctions) will be created in simulation.

    2. **Min. Auction Point:** The value of minimum point for a single auction.

    3. **Max. Auction Point:** The value of maximum point for single auction.

    4. **User Count:** The value of how many users will be created in simulation.

    5. **Credit per User:** The value of how much credit each user will have at the beginning of the simulation.

    6. **Fixing Hours:** The file which includes the fixing hour for every bug item.

    7. **Completed Work Hours:** The file which includes the completed work hour for every bug item.

After all parameters have entered, the simulation loads the auctions and users, then runs every auction in a different thread. When every auction enters a thread, the simulation needs to check users. Simulation checks;

1. There is at least one user can bid at least one auction

2. There is at least one user have enough credit to bid an auction

If at least one user is found who provides these conditions, the simulation starts to bid current auction.

The algorithm searches the available users to bid the current auction. Then gets every available user and creates a bidding hour using current auction's fix time and gamification ratio. Bidding hour is selected randomly from a range. This range has minimum and maximum values. Randomly selection is shown in formula 10.

$$Bidding\ Hour = RANDOM(Min\ [i], Max\ [i])$$

Formula 10 Calculation of bidding hour

Calculation of Minimum and maximum values are shown in formula 11.

$$Min\ [i] = FixTime\ [i]$$

$$Max\ [i] = FixTime\ [i] \times (1 - "GamificationRatio")$$

Formula 11 Calculation of minimum and maximum bidding hour range

After calculation of bidding hour, the algorithm bids the current auction and subtracts the auction point from current user's credit. This bidding process continue in a loop until the simulation is finished. The simulation pseudocode is seen Figure 8.

```
FUNCTION Main()
BEGIN
    CREATE user list
    CREATE auction list

    FOR get each auction from auction list
    BEGIN
        CHECK user can bid at least one auction
        CHECK user who has enough points

        IF at last one user available THEN
            CALL SimulateSingleAuctionBidding() with auction
    END
    CALCULATE winner users
END


FUNCTION SimulateSingleAuctionBidding (Auction auction)
BEGIN
    GET available bidding users for current auction

    FOR get each user from user list
    BEGIN
        CALCULATE bidding hour
        BID auction
        DECREASE user credit
    END
END
```

Figure 8 Simulation pseudocode

- **Calculating winner users:** At the end of the simulation, we can analyze statistical information about the simulation. We can order the users by their credit (winner users), by bidding count or their teams. We can compare the MTTR values of simulation and pilot project. Ultimately, we can show the effect of gamification on bug management as a numeric value.

CHAPTER 4

4. DESIGN AND IMPLEMENTATION

4.1 Introduction

The goal of this chapter is to give the detailed description of Monte Carlo simulation algorithm. This chapter includes system description, the information about the tools which have been used to develop Monte Carlo simulation application. We describe the back-end and front-end of application and the flow mechanism. We give detailed information about the system functions and we describe the implementation of simulation step by step. Then we describe the execution model of Monte Carlo simulation.

4.2 System Description

This auction-based bug management application is a simulation tool, which is designed to simulate bug resolving in a software development project. This simulation tool aims to improve software development quality in the projects of HAVELSAN. HAVELSAN is a Turkish Systems and Software corporation having a business in various domains. This company works in three business areas. The main area is simulation and training systems and the others are command, control and e-government projects. HAVELSAN has various software development project portfolio of around 50 projects in different sizes at any given time.

In this study, we explored one of the projects in the defense industry with around 60 personnel. Project X (we could not introduce the name of project because of security) started in 2014 and finished in 2016. In the project, the team used Microsoft Team Foundation Server for integrated ALM.

Project X had four milestones T0 (Integration), T1 (System), T2 (Release Candidate), and T3 (Acceptance) with a total of 1065 bugs. We calculated the sum of bugs in these periods and calculated the percentages of them. The bug counts and percentages in Project X are shown in Table 3.

Table 3 Bug counts in milestones (Day)

Time	**Bug Count**	**Percentage**
T0	488	45.8 %
T1	441	41.5 %
T2	115	10.8 %
T3	21	1,9 %
Total	1065	100 %

According to IEEE [45], a bug is a failure part or data in a software project. In Figure 3, we have provided the workflow of a bug. The lifecycle of a bug starts with a user (mostly test engineers) report a bug in the system. This bug report is reviewed by the development tech lead for initial triage, following which there are mainly two alternatives. Either the tech lead would assign the bug to a developer to get it fixed, or if a bug is affecting more than one system, the tech lead would escalate to the Configuration Control Board (CCB). Later on, after evaluation in CCB, the bug would be assigned to a developer, or might be closed by the CCB. In the Resolved state, a test engineer would test the proposed fix. If the fix is verified, the bug would be closed, otherwise the test engineer would return the bug to the developer in the Assigned State.

We can classify software anomalies in two groups. First group is “Defect Classification” and the other one is “Failure Classification” [87]. In this study, we concentrated on “Defect Classification” items.

4.3 Tools

In this part, we give brief information about the tools which we used in the study. We selected a pilot project named as “Project X” and analyzed this project’s bug source to calculate MTTR values. Project X keeps the bug work items in Microsoft Team Foundation Server (TFS). We develop our Monte Carlo simulation using object-oriented C# programming language and Visual Studio 2017. Visual Studio (VS) is an integrated

development environment (IDE). C# programming language, Team Foundation Server and Visual Studio are designed by Microsoft.

4.3.1 Team Foundation Server (TFS)

Microsoft Team Foundation Server (TFS) is a tool of Application Lifecycle Management (ALM) to manage a software development project. The new version of TFS named as "Azure DevOps". TFS includes requirement, task, test, build and source code management. Users can create or modify own work item types or use custom work item types in TFS. The most used work item types listed as follows:

- Task
- Bug
- Backlog Item
- Requirement
- Change Request
- Test Case
- Test Plan
- Test Suite
- Nonconformity

In this study, we used around 1200 bugs of Project X from TFS 2017. TFS has a powerful API to read, create, update or delete any of work item like Bug, Task or Test Case etc. We used TFS API to export bug items to Excel file. Excel has available mathematical functions and we used these functions to calculate MTTR values easily. Each work item includes around 30 fields. We filtered these fields and exported some of them to the Excel file. Most commonly used fields and types are shown in Table 4.

Table 4 Bug Work Item Fields

Field Name	**Type**	**Description**
Original Estimate	Decimal	Working hour which is assigned while creating bug (approximate value)
Completed Work	Decimal	Working hour which is calculated after resolving bug (exact value)
Activated Date	Date Time	Start date on the bug
Resolved Date	Date Time	Resolve date of bug
Area Path	String	The project area of bug
Assigned To	String	The assignee person of bug
Description	String	The description of bug
State	String	The state of bug
Reason	String	The reason about bug

The screenshot of a bug item from TFS is seen in Figure 9. (Not from Project X)

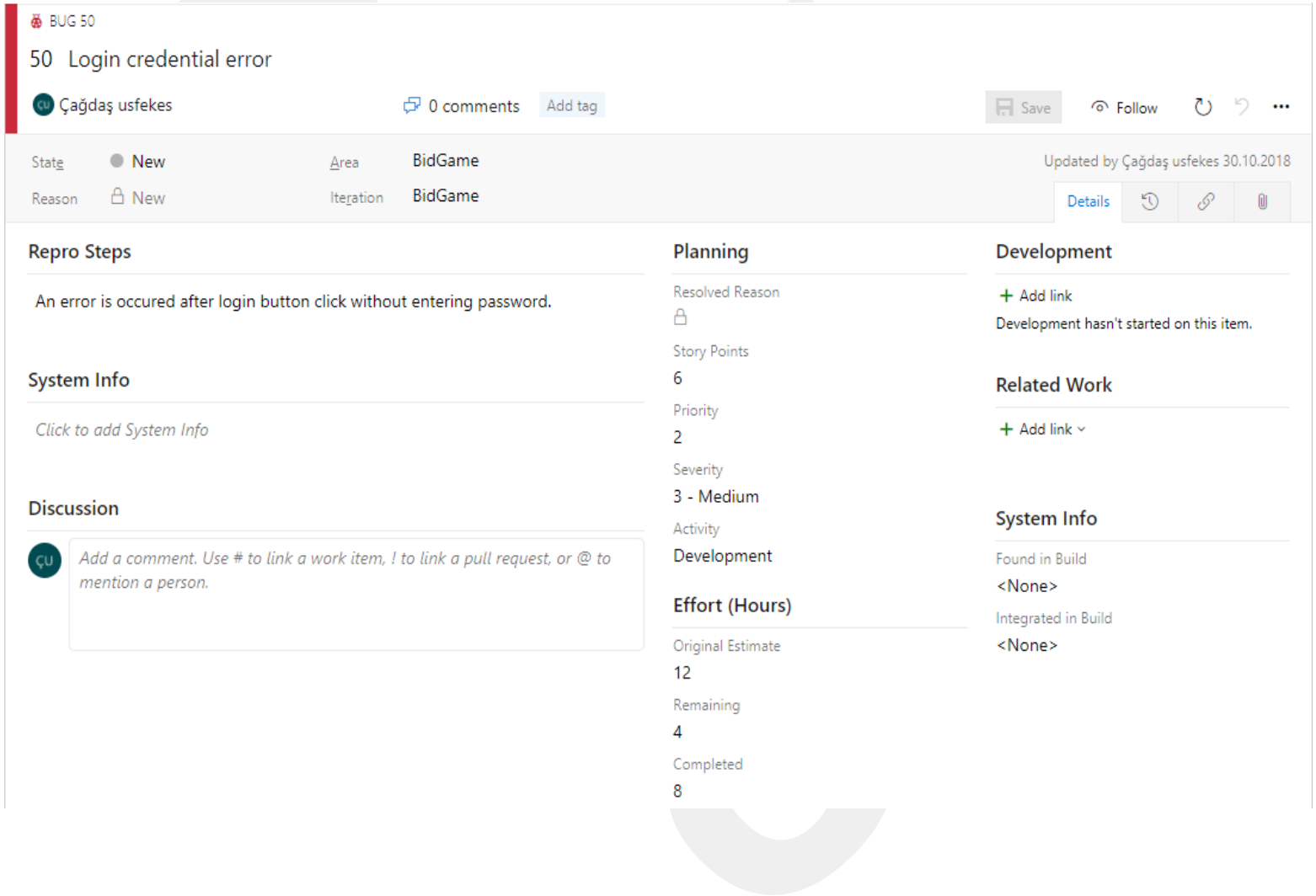

Figure 9 Bug work item

4.3.2 Visual Studio (VS)

Visual Studio (VS) is one of the popular IDE (Integrated Development Environment) that is developed by Microsoft. Software developers can develop applications with various programming languages by using Visual Studio. The most popular programming languages which is supported by Visual Studio is seen in Table 5.

Table 5 Supported Programming Languages by VS

C#	Go	Visual Basic
C	Ruby	PHP
C++	JavaScript	PowerShell
Python	HTML	Perl
Java	TypeScript	Coffee Script

Visual Studio has extended features that provides coding more efficiently. Visual Studio Code, Azure DevOps (Team Foundation Server) and Visual Studio App Center can be given as example to these features.

Visual Studio includes various software development environments like web development (ASP.NET, ASP.NET MVC), mobile development (Xamarin, Windows Mobile), Office Add-ins (Word, Excel, PowerPoint), Windows Forms Applications. WPF (Windows Presentation Framework). In this study, we selected Windows Forms Application to develop Monte Carlo Simulation. Windows Forms Applications allows to develop classic windows style applications with rich interfaces.

In Visual Studio, each software development project stored under a solution file. One solution includes one or more projects and each project includes one or multiple code files, interface files, images etc. Windows Form Applications create two different files for every windows form interface. One of them includes interface items (textbox, button, label, progress bar, panel, checkbox) and the other one includes all codes about the current windows interface. The solution of Monte Carlo simulation is seen in Figure 10.

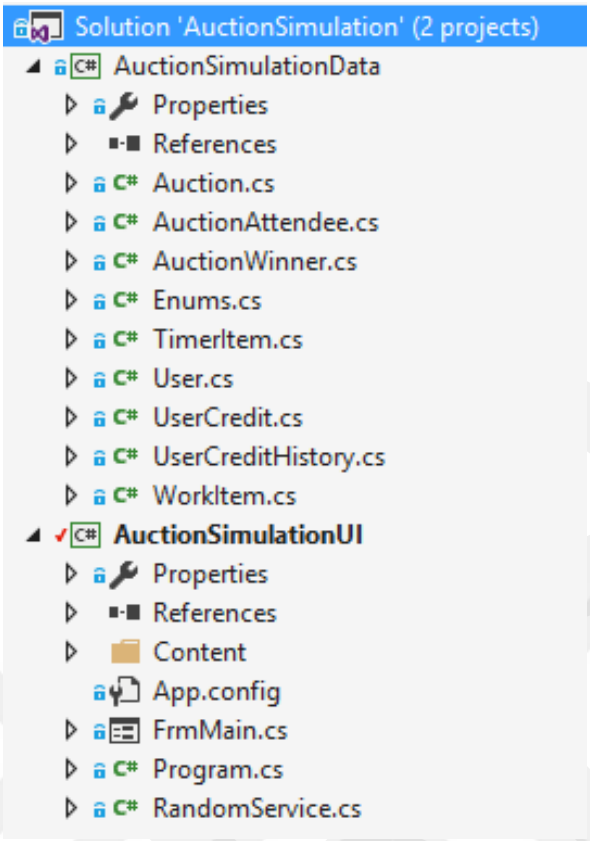

Figure 10 Monte Carlo Simulation Solution

4.4 System Functions and Implementation

Based on system description which is explained in 4.2 section, this system is an auction-based bug management simulation application. The system's functions are divided into two groups according to user items and auction items (work items). In the simulation, random users can bid random auctions. While bidding action is running, the system checks various requirements about the selected user and auction.

The class diagram of Monte Carlo simulation can be seen in Figure 11. All class and enumeration items are listed in this diagram.

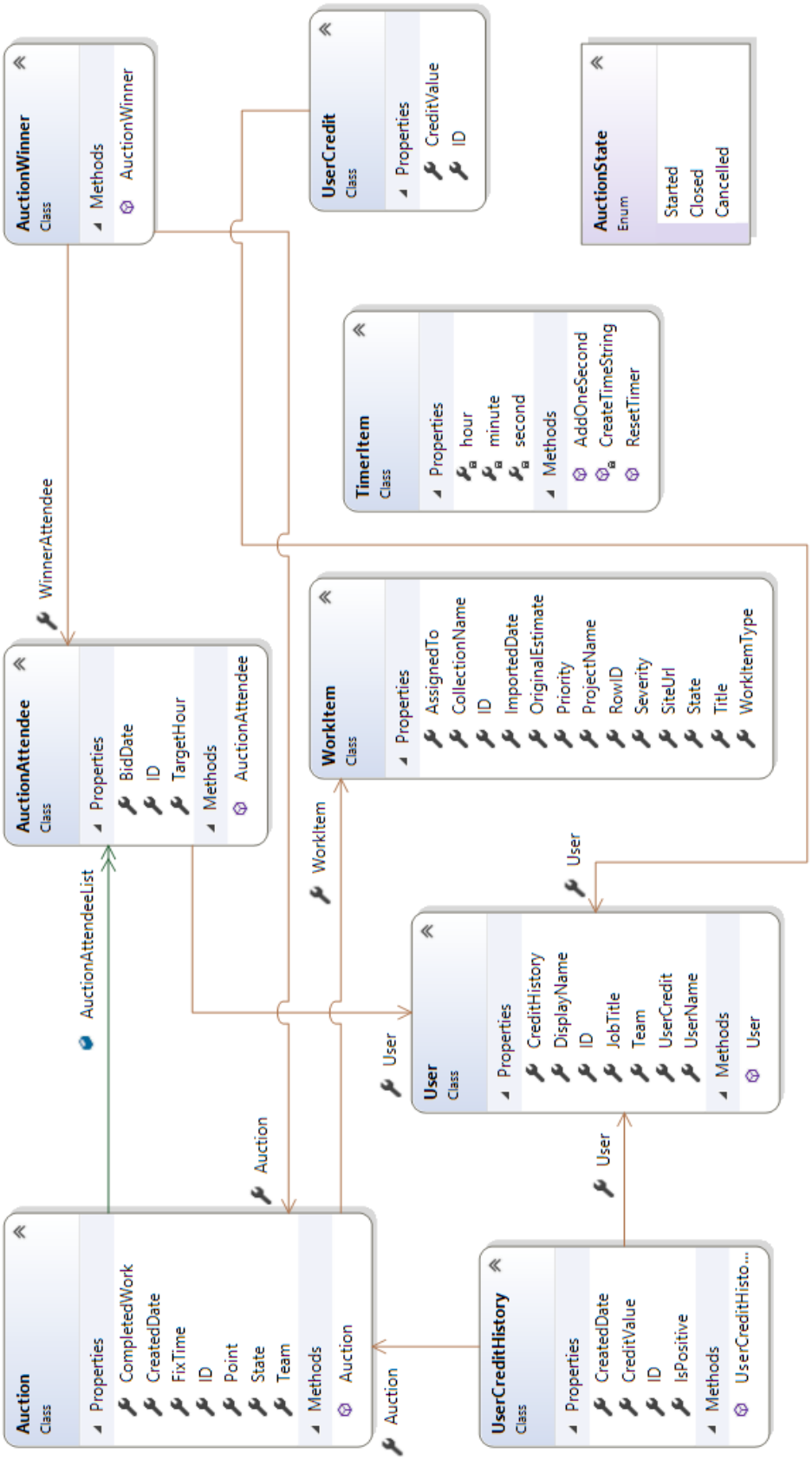

Figure 11 Class Diagram of Monte Carlo Simulation

“WorkItem” and “User” are the base classes of this system. Work items are exported from Project X and users are generated dynamically during the simulation. “UserCreditHistory” and “UserCredit” are the sub classes of the “User” class and they hold the credit information about a user. “Auction” class is inherited from “WorkItem” class. “AuctionAttendee” and “AuctionWinner” are the sub classes of “Auction” class. “AuctionAttendee” class describes which users bid an auction and “AuctionWinner” class shows the winner user of an auction. “TimerItem” class shows how long the simulation is run.

4.4.1 Monte Carlo Simulation Back-End Model

The Monte Carlo simulation includes 8 classes and 1 enumeration which are developed by using C# programming language. The main classes of the simulation model are “Auction”, “User” and “UserCredit”. The others are helper classes or inherited from main classes. All classes are listed below with properties and descriptions.

Table 6 “Auction” Class Definition

Property Name	**Type**	**Description**
ID	Integer	The unique id of auction
WorkItem	WorkItem	The reference work item of auction
CreatedDate	Date Time	The creation date of auction
Point	Integer	The point value of auction
State	AuctionState	The state enumeration value of auction
FixTime	Decimal	The fixing hour value of work item
CompletedWork	Decimal	The completed work hour of work item
Team	String	The team value of work item
AuctionAttendeeList	AuctionAttendee[]	The users who bid the auction

Table 6 shows an auction item and its properties. Auctions are inherited from work items.

Table 7 "User" Class Definition

Property Name	**Type**	**Description**
ID	Integer	The unique id of user
UserName	String	The username of user
DisplayName	String	The display name of user
JobTitle	String	The job description of user
Team	String	The team value of user
UserCredit	UserCredit	The credit value of user
CreditHistory	UserCreditHistory[]	The all credit history of user

Table 7 shows a user item and its properties. All users have credit (money) to bid auctions.

Table 8 "UserCredit" Class Definition

Property Name	**Type**	**Description**
ID	Integer	The unique id of user credit
User	User	The owner user of credit
CreditValue	Integer	The value of credit

Table 8 shows a credit value item and its properties. Credit value item shows how much credit does a user have?

Table 9 "AuctionAttendee" Class Definition

Property Name	**Type**	**Description**
ID	Integer	The unique id of attendee user
BidDate	Date Time	The date of bidding auction
TargetHour	Decimal	The time that user declares to solve bug
User	User	The user who bids auction

Table 9 shows an auction attendee item and its properties.

Table 10 "AuctionWinner" Class Definition

Property Name	**Type**	**Description**
Auction	Auction	The auction items
WinnerAttendee	AuctionAttendee	The winner user of current auction

Table 10 shows an auction winner item and its properties. Auction winner shows the winner user of an auction.

Table 11 "UserCreditHistory" Class Definition

Property Name	**Type**	**Description**
ID	Integer	The unique id of credit history
Auction	Auction	The related auction of credit history
User	User	The owner user of credit history
CreditValue	Integer	The value of credit
IsPositive	Boolean	The flag which shows credit earned or spent
CreatedDate	Date Time	The creation date of credit history

Table 11 shows a user credit history item and its properties. User credit history shows a single credit operation about a user. If user earns credit, “IsPositive” property will be true and if the user spends credit this property will be false.

Table 12 “WorkItem” Class Definition

Property Name	**Type**	**Description**
ID	Integer	The unique id of work item
SiteUrl	String	The ALM site url of work item (TFS)
CollectionName	String	The collection name of work item in TFS
ProjectName	String	The project name of work item in TFS
Title	String	The title of work item
State	String	The state of work item
WorkItemType	String	The work item type of work item
AssignedTo	String	The owner of work item
Priority	Integer	The priority of work item
Severity	String	The severity of work item
OriginalEstimate	Decimal	The original estimation value of work item
ImportedDate	Date Time	The imported date of work item from TFS to simulation

Table 12 shows a work item and its properties. Work items are exported from Project X.

Table 13 “AuctionState” Enumaration Definition

Property Name	**Type**	**Description**
Started	Integer	The auction is started to bidding
Closed	Integer	The auction is closed to bidding
Cancelled	Integer	The auction is cancelled

Table 13 shows auction state enumeration and its values.

Table 14 "TimerItem" Class Definition

Property Name	**Type**	**Description**
Hour	Integer	How many hours of simulation is running
Minute	Integer	How many minutes of simulation is running
Second	Integer	How many seconds of simulation is running

Table 14 shows timer item and its properties. Timer item indicates how long the simulation is run.

Table 15 "RandomService" Class Definition

Method Name	**Input Type**	**Output Type**
GetTeamList()		String[]
CreateRandomUsers()	Integer, Integer	User[]
CreateUserCredit()	Integer, Integer	UserCredit
CreateRandomAuctions()	Integer, Integer, Integer, Decimal[], Decimal[]	Auction[]
GetRandomBiddingHour()	Decimal, Decimal	Decimal

Table 15 shows random service and its functions.

4.4.2 Monte Carlo Simulation Front-End Model

The Monte Carlo simulation interface is designed in Windows Forms Applications by using C# programming language. Visual Studio provides rich controls to design classic windows forms easily. Software developers can use various controls like Textbox, Button, Panel, RadioButton, CheckBox, ProgressBar or ListBox while coding a windows forms application.

In this study, we designed a simple interface to manage auction-based bug management simulation. We separated the screen into two parts. In the first part, (left of screen) we located a control set that allows entering simulation parameters. Monte Carlo simulation needs 7 parameters except “Gamification Ratio”. The screenshot of the parameters panel is seen in Figure 12.

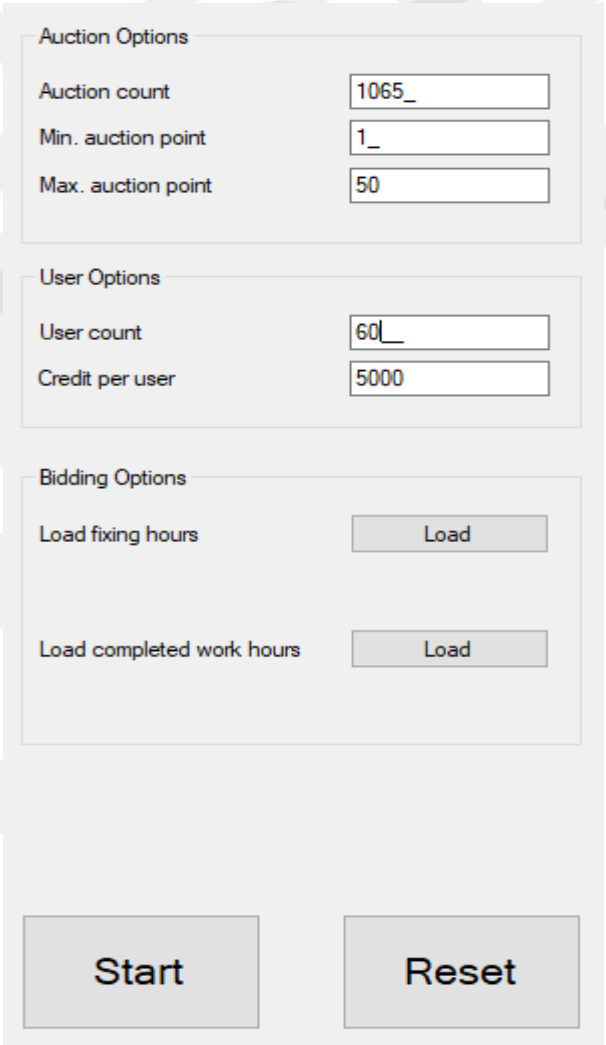

Figure 12 Monte Carlo Simulation Parameter Settings

All numerical parameters have default values. Default parameters are calculated according to the Project X. The detailed information about parameters are like that:

- **Auction Count:** This value shows how many auctions will be created in simulation. In project, X we have 1065 bugs and we set this value as default.

- **Minimum Auction Point:** The minimum limit for an auction point. Auction point cannot be a negative number and we set the minimum positive number to this parameter. "1" means the easiest auction in simulation.

- **Maximum Auction Point:** The maximum limit for an auction point. Auction point calculation depends on "Priority" and "Severity" fields of current auction (work item). Highest "Priority" value can be 5 and highest "Severity" value can be 10 in Project X. We calculated the maximum auction point by multiplying highest "Priority" and "Severity" values.

- **User Count:** This value shows how many user will be created in simulation. In Project X, there are 6 teams and every team includes around 10 people. We calculated the default value as the total population of Project X.

- **Credit Value per User:** The initial credit value for every user. Every user spends credit while bidding an auction. We set the default value 5000 credit. This is enough to run simulation around 1000 auction.

- **Fixing Hour File:** This file includes the all fixing hours of Project X bug items. Fixing hours are used while creating auctions.

- **Completed Work Hour File:** This file includes the all fixing hours of Project X bug items. Completed work hours are used while creating auctions.

When user clicks "Start" button, the simulation starts to run with selected parameters. "Reset" button sets the default values for all parameters. While simulation is running, a progress bar appears and it shows how many auctions are bid, total auctions count, percentage of bid auctions and how long has the simulation been running. The screenshot of information bar is seen in Figure 13.

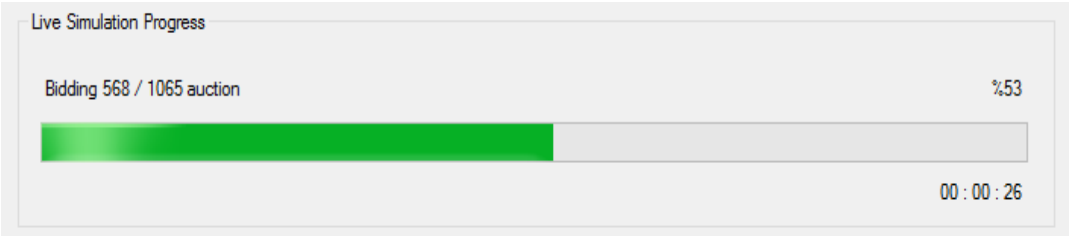

Figure 13 Monte Carlo Simulation Progress Bar

At the end of the simulation run, statistical information about Monte Carlo simulation results are presented below the progress bar. The other detail information is saved to a text file automatically. Presented information can be used to answer the following questions:

- How many user(s) bid at least one auction?
- How many user(s) win an auction?
- How many auction(s) were bid by at least one user?
- What is the average point of all auctions?

The information bar is shown in Figure 14.

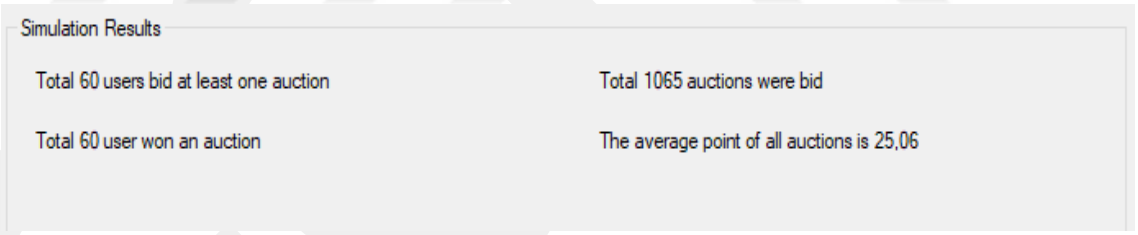

Figure 14 Monte Carlo Simulation Information Bar

4.4.3 Monte Carlo Simulation Execution Model

Monte Carlo simulation needs a set of required parameters to execute. These parameters are explained in previous part. After all required parameters are set, the simulation creates virtual users and virtual auctions. Next, the simulation is ready to run in a loop. The loop count is equal to the user count.

In every step of the loop, the simulation bids a different auction and checks a set of conditions. If one of these condition is not provided, simulation finishes otherwise the simulation continues to run in current loop. The conditions are like that:

- **Checking users' credit:** The simulation checks is there at least one user can bid at least one auction. If there is at least one user, the simulation continues, otherwise simulation ends.

- **Checking minimum auction:** The simulation calculates the minimum auction point of all auctions. Then it checks is there at least one user whose point is bigger than minimum auction point. If there is at least one user, the simulation continues, otherwise simulation ends.

- **Checking user-auction map:** In the simulation, one user can bid the same auction only one time. If all users bid all the auctions one time, the simulation finishes.

If all of these conditions are provided, the simulation starts to bid current auction. The function "SimulateSingleAuctionBidding()" is called to bid every auction. The function creates a new thread for every auction and disposes the thread after execution finishes. This function takes the current auction item as input parameter.

When the simulation starts to bid an auction firstly available users are determined to bid the auction. These users are determined by their credit value. If a user has not enough credit to bid an auction, he/she cannot bid the auction. While bidding process is continue, the system calculates a bidding hour. The bidding hour value is calculated by a function and this function uses two parameters as input. These are:

- **Fix time (decimal):** Every auction item inherited from a real bug item from Project X. Fix time value is a numeric field that is read from bug item.

- **Gamification ratio (decimal):** This numeric ratio is calculated by a reference study. Gamification ratio is explained in the methodology section.

The function calculates and return a new bidding hour by using fix time and gamification ratio. Function code is shown in Figure 15.

```
public static decimal GetRandomBiddingHour(decimal fixingTime,
    decimal gamificationRatio)
{
    decimal subtractValue = fixingTime * gamificationRatio;

    decimal minHourRange = fixingTime - subtractValue;
    decimal maxHourRange = fixingTime;

    int rndHour = random.Next(Convert.ToInt32(minHourRange * 100),
        Convert.ToInt32(maxHourRange * 100));

    decimal newBiddingHour = (decimal)rndHour / 100;
    return newBiddingHour;
}
```

Figure 15 Calculation of Bidding Hour

The simulation creates a range for bidding hour by using fix time and gamification ratio. Then the function selects a random value from this range and return it to the simulation.

After the simulation is finished, the winner users of auctions are listed by their bidding hour values. We can calculate the new MTTR values from the simulation results and compare them with the pilot project's MTTR values to show the effect of using gamification in software development projects.

4.5 Analysis and Test Results

We ran the auction simulation using 1065 bugs and 60 users. Then we calculated and compared the MTTR values for two scenarios. The first scenario was depending on real project data from Project X. The second scenario ran the Monte Carlo simulation with parameters in Table 3 and using the gamification ratio which is drawn from previously published study by the authors [88]. The main difference between the two scenarios is the use of gamification ratio. By this ratio, we can see the effect of using gamification in bug management.

We calculated MTTR values for two scenarios by the formula (1). We included 1065 bugs into this formula. MTTR results for the Monte Carlo simulation is shown in Table 16.

Table 16 Monte Carlo simulation MTTR values (Days)

Time	MTTR	Min. Time	Max. Time
T0	50.30	0.06	633.66
T1	47.12	0.02	307.12
T2	73.11	1.41	182.31
T3	28.76	5.01	68.02

The MTTR values for the T0, T1, T2 and T3 milestones of Monte Carlo simulation is seen in Figure 16.

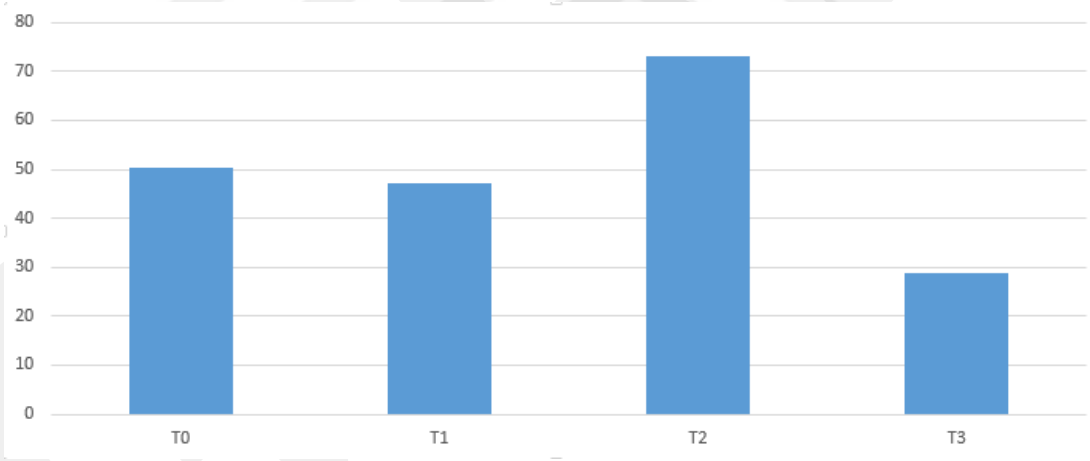

Figure 16 MTTR values for Monte Carlo simulation

MTTR values for the Project X is shown in Table 17.

Table 17 Project X MTTR values (Days)

Time	MTTR	Min. Time	Max. Time
T0	54.61	0.04	686.76
T1	51.87	0.02	310.76
T2	78.10	2.03	195.83
T3	33.75	5.79	71.82

The MTTR values for the T0, T1, T2 and T3 milestones of Project X is seen in Figure 17.

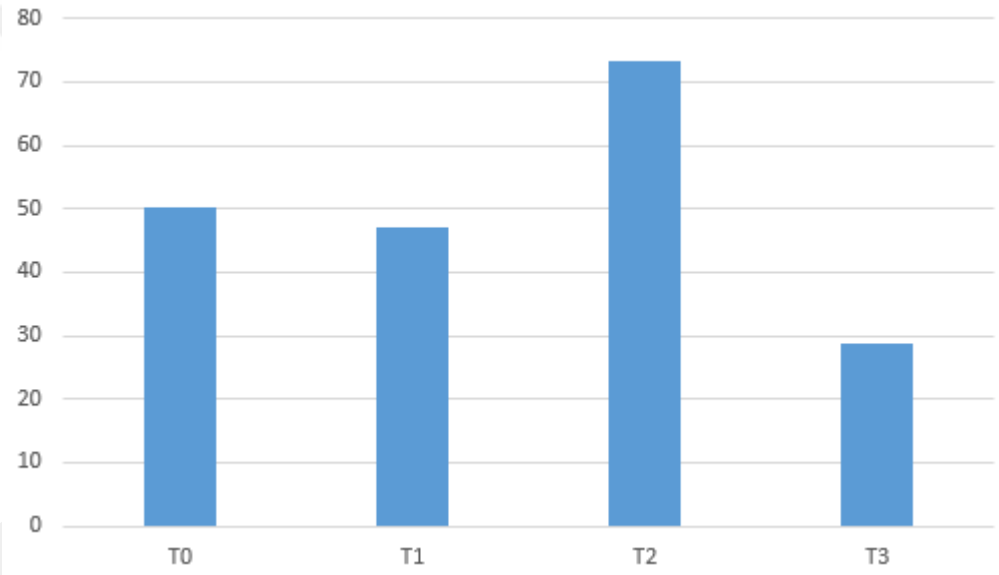

Figure 17 MTTR values for Project X

Then we compared actual MTTR values for Project X with the Monte Carlo simulation, as shown in Table 18.

Table 18 Project X and Monte Carlo simulation MTTR Comparison

	Project X	Monte Carlo Simulation
Number of bugs that used	1065	1065
MTTR values (day)	54.58	49.82

The comparison of MTTR values in Project X and auction-based Monte Carlo simulation is shown in Figure 18. In this graph, we can see the effect of gamification on MTTR values.

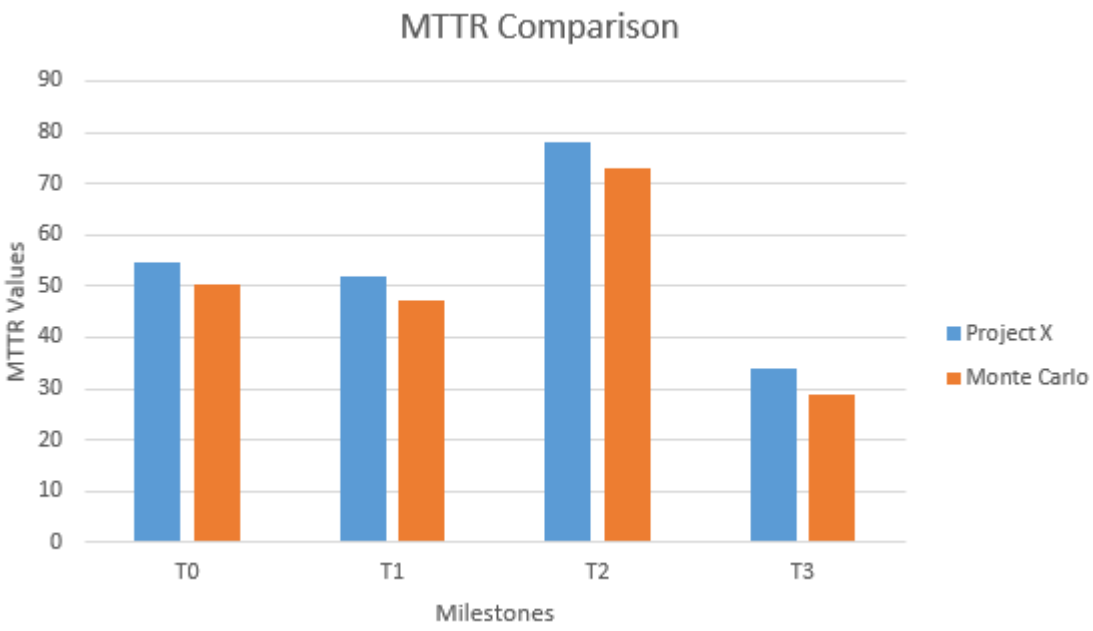

Figure 18 MTTR comparison of Project X and Monte Carlo simulation

We listed the top 5 users who has maximum points, won auction counts and their teams. The list is shown in Table 19.

Table 19 Top 5 users

Username	**Point**	**Won Auction Count**	**User Team**
User 3	2456	58	Maintenance
User 7	2256	48	Planning
User 32	1748	32	Infrastructure
User 16	1290	18	Maintenance
User 57	967	10	Infrastructure

We conducted experiments with a set of parameters (see Figure 12) and the average results are shown in Table 16. We repeated the simulation for five times and we got close results. The average of MTTR values were between 49.05 days and 52.52 days for every repetition.

The MTTR values for the first repetition is seen in Table 20.

Table 20 Monte Carlo simulation MTTR values (Repetition 1)

Time	**MTTR**	**Min. Time**	**Max. Time**
T0	48.25	0.05	632.21
T1	49.10	0.06	302.18
T2	72.10	1.49	176.56
T3	29.66	7.02	69.01

The MTTR results for second repetition is seen in Table 21.

Table 21 Monte Carlo simulation MTTR values (Repetition 2)

Time	**MTTR**	**Min. Time**	**Max. Time**
T0	49.33	0.09	628.11
T1	43.18	1.00	306.00
T2	76.13	1.34	171.52
T3	29.36	7.01	65.02

The MTTR values for the third repetition is seen in Table 22.

Table 22 Monte Carlo simulation MTTR values (Repetition 3)

Time	MTTR	Min. Time	Max. Time
T0	49.11	0.06	624.11
T1	43.99	1.04	311.40
T2	73.54	1.32	169.53
T3	32.31	7.09	64.03

The MTTR values for the fourth repetition is seen in Table 23.

Table 23 Monte Carlo simulation MTTR values (Repetition 4)

Time	MTTR	Min. Time	Max. Time
T0	49.45	0.02	629.12
T1	45.78	0.24	316.42
T2	78.54	1.42	172.43
T3	36.31	7.78	67.09

The MTTR values for the fifth repetition is seen in Table 24.

Table 24 Monte Carlo simulation MTTR values (Repetition 5)

Time	MTTR	Min. Time	Max. Time
T0	45.45	0.08	628.42
T1	44.88	0.14	313.79
T2	77.11	1.89	171.72
T3	35.01	7.34	64.19

4.6 Revisiting the Research Questions

In this section, our three research questions from Chapter 1 are discussed and detailed answers are given for all questions. We defined three objectives. These are;

- Analyze Project X bug data (MTTR calculation).
- Create an auction-based bug management simulation.
- Compare the MTTR values of Project X and the auction-based bug management simulation.

Depending on three objectives, we defined three research questions. The questions and answers are explained below this paragraph.

RQ1: How can the auction-based reward mechanism help the bug resolution process?

To address the first question, we studied on a referenced paper. Gulec and Yilmaz [88] designed a game system to train the football referees. The first group of referees (control group) was trained by using handbook and the second group (experimental group) was educated by using gamification. The same exam was applied to all groups at the end of the training. Exam results show that the second group (trained by using the game system) is 8,65 % more successful than the first group. According to these results, we decided to use the reward mechanism to help software developers to solve bugs in less time. The reward mechanism can encourage participants (software engineers) to solve problems in more enjoyable ways while they are trying to solve bugs about their jobs. By this way, software engineers can resolve more bugs during development process.

RQ1.1: How can we minimize the bug resolution time?

To address the second question, we should study on MTTR calculation. In Project X, there are 1065 bug items and MTTR value is 54.58 days. In auction-based bug management simulation, there are 1065 auction items (bugs) and the MTTR value is between 49.05 days and 52.52 days in all repetitions. We can see that using gamification helps software developers to solve bugs. Software developers (participants) were allowed to choose bugs to resolve by themselves during the simulation and they had higher motivation while working on bugs. If a software developer has high motivation, he/she

can solve bugs in less time and we can minimize the average bug resolution time in a software development project.

RQ1.2: How can we demonstrate the effectiveness of auction-based reward mechanism in the bug resolution process?

To address the third question, we designed a Monte Carlo simulation for an auction-based bug management. Monte Carlo is a type of simulation that depends on randomly choices. In auction-based bug management system, every single bug item is equal to an auction and software developers can bid an auction which they want to solve. The team leader does not assign a bug to a developer. The team members (developers) can solve bugs with high motivation in auction-based working model. The improvement of motivation is a result of using auction-based reward mechanism in Monte Carlo simulation.

CHAPTER 5

5. CONCLUSION AND FUTURE WORK

In the field of software development, MTTR is a notable metric with lower MTTR figures correlating closely with greater satisfaction for the customer. In order to reduce MTTR, an innovative gamification approach is presented in this study. We initiated this project to devise a structure that would incentivize, using Monte Carlo simulation methods, software developers to improve their bug tracking and investigation skills. Following five experiments, the results seemed to indicate that the gamified variant (i.e., incentive mechanism-based simulation) yielded outcomes that were superior to the normal run. It could be observed in the data distribution of the study that a sequence of dichotomous event outcomes occurred in a selected period including a number of bugs being resolved over 51.45 days [25].

Our research endeavored to present and advance a model that would expound on and analyze auction-based incentive systems for bug tracking in software development environments. The findings present a potential procedure for designers of mechanism (software managers) to evaluate any possibilities that increase the probability of managers making better decisions. As attested in previous related research showing the great complexity of software development process decisions [85] and the dependence on the manner in which many individuals perform in software development [86], measures taken to deal with this complexity via gamification may facilitate handling it through the enlistment of software developers at a higher level through the software development social setting of gamification, which would probably result in more timely higher quality work [25].

Our identified approach provides individual developers with the benefit of selecting those bugs that are most likely to be resolved. Individuals will of course occasionally fail to be accurate in their assessment of their abilities. Nevertheless, providing them with the means to identify issues that they believe can be resolved is considered by the authors to be a way for matching particular developers with particular individual bugs. Moreover, when developers quote an estimated period time leading to resolution, they become bound to the time(s) declared. In the event of failing to meet their declared target(s), there

is the potential risk of appearing incompetent in the company of their peers should this failure continue. Nevertheless, this can assist individual developers to concentrate on determining and pinpointing more accurate bug resolution durations. Moreover, at a later time, a combination of known developer predictive resolution duration accuracy and bids placed across various auctionable bugs to identify the stronger economic distributions of bugs to bug resolvers may be used by a development team, which would be an improvement in the effective removal of bugs by applying gamification techniques [25].

Our current study, for the first time, delves into applying an auction mechanism to software development. In software development, for a better sense of bug trend dynamics, the classification of MTTR has increased in importance. Our research presents interesting possibilities to expand on what we know about software metrics. This makes it possible for us to specify better software product dependability.

Preliminary prototype and simulation results had been disclosed to the company, from whom we received immediate commendations. Nevertheless, there is still more work required for a better and more complete understanding of the ramifications of an auction-based incentive mechanism. For further study, we intend to test the system on an intermediate sized software company.

5.1 Threats to Validity

Yılmaz [29] describes the "threats to validity" as a set of possible factors which can change the correctness, usefulness of study and trust-ability in a negative way. However, Fayter et al. [101] define threats as the factors which cause to get worse quality results. We can classify the validation of threats as four categories. These are;

- **Construct Validity:** Constructs and valid operational measures should represent the subject clearly [29].

- **Interval Validity:** Any invisible factors which affect the validity should be predicted and conceptual definitions should match with the operational results [29].

- **External Validity:** The research should provide equipment to extrapolate on research results [29].

- **Reliability:** The research should be stable about measuring instrument and the researchers can repeat the study with the current results [29].

There are however, various limitations in our study which should be discussed. Firstly, similar to other methods based on the theory of probability Monte Carlo approaches are data-intensive. Therefore, they cannot produce significant results unless a considerable set of data has been generated - which has the effect of introducing a computational burden. Apparently, more experiments need to be conducted under various data scenarios. An auction-based bug management is a socio-technical process where all on different trials needs to be run to determine parameters which should have to be set by the researcher. This may impose time constraints while modeling the system. A further limitation can be seen in the assumption that the gamification ratio from earlier research will retain validity in the context of this gamification experiment. Clearly, further study should be conducted to examine this assumption. It should, however, be noted that a new gamification ratio could be established for individual teams.

Table 25 summarizes the potential threats to validity for this study.

Table 25 Threats to validity items on Monte Carlo simulation

Threat Category	**Threat Description**
Internal Validity	Monte Carlo simulation is data-intensive. Data size is important about the simulation result is consistent or not.
Construct Validity	Auction-based operations are socio-technical processes and all system parameters should have to be set by the researcher otherwise different trials can give different results.
External Validity	The data which depends on the calculation of 'Gamification Ratio' has a limitation on Monte Carlo simulation.
Reliability	The measurement methods should be accurate and stable where the methodologies and measuring methods could be reused by other researchers.

5.2 Future Work

In this study, we designed an auction-based bug management simulation using Monte Carlo method. In the future, we are planning to develop an auction-based bug management tool. This tool will be a web-based application and has the ability to create auctions with work items from Team Foundation Server (TFS). We will determine one or more administrators on this tool and they have ability to create auctions. All auctions will start and finish in scheduled time zone. All of the software engineers (gamers) can login to this system and they can see the open auctions. Software engineers can bid any auction if they have enough credit. One gamer will win the auction at the end of auction. If winner software engineer can solve the bug in promised time range, he/she will earn more credit, otherwise will not earn any credit.

When the bug management tool development will be finished, we will select another pilot project in HAVELSAN. Firstly, we will create two different groups as an experimental and control group. The software engineers will be divided into these two groups randomly. Control group will work with the classical model. In classical model, team leaders will assign bugs to developers. Then we will collect some bug data like resolving time, priority and severity value etc. to calculate MTTR value of control group. The experimental group will work using the auction-based bug management tool and we will collect the same bug data for experimental group during the development process. At the end of the project lifecycle, we will compare the MTTR values of control and experimental group to show the effectiveness of auction-based game systems in bug resolution process.

REFERENCES

1. **Chappell, D.** (2008), "*What is Application Lifecycle Management?*", Chappell & Associates.

2. **Yilmaz, M., O'Connor, R.** (2011), *"Oyun Kuramı Kullanarak Yazılım Takımlarının Üretkenliğini Artırmak İçin Geliştirilen Bir Yazılım Süreç Mühendisliği Yaklaşımı"*, UYMS.

3. **Zahran S.** (1998), *"Software Process Improvement: Practical Guidelines for Business Success",* Addison Wesley.

4. **Maskin E.** (2008), *"Nash equilibrium and mechanism design, Institute for Advanced Study, Princeton University",* United States.

5. **Dingsøyr T., Dybå T., Moe N. B.** (2010), *"Agile Software Development: Current Research and Future Directions"*, 1st ed. Springer.

6. **Deek F. P., McHugh J. A., Eljabiri O. M.** (2005), *"Strategic software engineering: an interdisciplinary approach*", CRC Press.

7. **Lagesse B.** (2006), *"A Game-Theoretical model for task assignment in project management"* in 2006 IEEE International Conference on Management of Innovation and Technology, Singapore, pp. 678-680.

8. **Cockburn A.** (2006), *"Agile software development: the cooperative game."* Addison-Wesley, "A Game-Theoretical model for task assignment in project management," in 2006 IEEE International Conference on Management of Innovation and Technology, Singapore, pp. 678-680.

9. **Baskerville R. L., Levine L., Ramesh B., Pries-Heje J.** (2004) *"The high speed balancing game: How software companies cope with internet speed"* Scandinavian Journal of Information Systems, vol. 16, no. 1, pp. 11–54.

10. **Sullivan K., Chalasani P., Jha S.** (1997), *"Software design decisions as real options"* University of Virginia, Tech. Rep.

11. **Sazawal V., Sudan N.** (2009) *"Modeling software evolution with game theory"* Trustworthy Software Development Processes, vol. 5543, pp. 354–365.

12. **Xing G., Weijun Z., Shue M.** (2013), *"A game-theory approach to configuration of detection software with decision errors"*.

13. **Gao-hui N.** (2006), *"Analysis on Enterprise's Software Project Management Based on Game Theory, Management Science and Engineering"*.

14. **Soska A., Mottok J., Wolff C**. (2016), *"An experimental card game for software testing: Development, design and evaluation of a physical card game to deepen the knowledge of students in academic software testing education"*, Global Engineering Education Conference (EDUCON), IEEE.

15. **Pedreira O., García F., Brisaboa N., Piattini M.** (2015), *"Gamification in software engineering – A systematic mapping, Information and Software Technology"*, v. 57.

16. **Sweedyk E., Keller R. M.** (2005), *"Fun and games: a new software engineering course"*, ITiCSE '05 Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, Pages 138-142.

17. **Kitagawa N., Hata H., Ihara A., Kogiso K., Matsumoto K.** (2016), *"Code review participation: game theoretical modeling of reviewers in gerrit datasets"*, CHASE '16 Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering, Pages 64-67.

18. **Szabo C.** (2014), *"Evaluating GameDevTycoon for teaching software engineering"*, Proceeding SIGCSE '14 Proceedings of the 45th ACM technical symposium on Computer science education, Pages 403-408.

19. **Amir B., Ralph P.** (2014), *"Proposing a theory of gamification effectiveness"*, Proceeding ICSE Companion 2014 Companion Proceedings of the 36th International Conference on Software Engineering, Pages 626-627.

20. **Ranganathan N., Murugavel A.** (2003), *"A low power scheduler using game theory"*, CODES+ISSS '03 Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, Pages 126-131.

21. **Hazzan O., Dubinsky Y.** (2005), *"Social perspective of software development methods: The case of the prisoner dilemma and extreme programming"* in Extreme Programming and Agile Processes in Software Engineering. Springer, pp. 74–81.

22. **Feijs L.** (2001), *"Prisoner dilemma in software testing"* Computer Science Reports, vol. 1, pp. 65–80.

23. **Costa C. J., Costa P. J.** (2011), *"A peace war game application"*, OSDOC '11 Proceedings of the 2011 Workshop on Open Source and Design of Communication, pp. 71-74.

24. **Mortensen P., Wai C.** (2007), "*Avoiding the prisoner's dilemma of the web*", DUX '07 Proceedings of the 2007 conference on Designing for User eXperiences.

25. **Usfekes C. et. al.** (2019), *"An Auction-Based Serious Game for Bug Tracking",* IET Software, DOI: 10.1049/iet-sen.2018.5144.

26. **Conradi R., Fuggetta A.** (2002), *"Improving Software Process Improvement"* IEEE Software, vol. 19, no. 4, pp. 92–99.

27. **Dittrich Y., Floyd C., Klischewski R.** (2002), *"Social thinking software practice."* The MIT Press.

28. **Grechanik M., Perry D. E.** (2004), *"Analyzing Software Development as a Noncooperative Game*" in IEE Seminar Digests, vol. 29.

29. **Yilmaz M.** (2013), *"A software process engineering approach to understanding software productivity and team personality characteristics: an empirical investigation."* PhD thesis, Dublin City University.

30. **Yilmaz M, O'Connor R.** (2010), *"Maximizing the value of the software development process by game theoretic analysis",* In: 11th International Conference on Product Focused Software, Limerick, Ireland. ISBN 978-1-4503-0281-4.

31. **Yilmaz M., Yilmaz M., O'Connor R., and Clarke P.** (2016) *"A gamification approach to improve the software development process by exploring the personality of software practitioners"*, In: Clarke, Paul and O'Connor, Rory and Rout, Terry and Dorling, Alec, (eds.) Software Process Improvement and Capability Determination. Communications in Computer and Information Science, 609 . Springer, pp. 71-83. ISBN 978-3-319-38980-6

32. **Schwaber C., et al.** (2006), "*The Changing Face of Application Lifecycle Management*", Forrester Research, August 18.

33. **Yilmaz M., O'Connor R., Collins J.** (2010), *"Improving software development process through economic mechanism design",* In: 17th European Software Process Improvement Conference, Grenoble, France. ISBN 978-3-642-15666-3.

34. **Yilmaz M., O'Connor R.** (2016) *"A Scrumban integrated gamification approach to guide software process improvement: a Turkish case study*", TehnickiVjesnik (Technical Gazette), 23 (1). pp. 237-245. ISSN 1330-3651.

35. **Yilmaz M., O'Connor R.** (2012) *"A market based approach for resolving resource constrained task allocation problems in a software development process*", In: 19th European Conference on Systems, Software and Services Process Improvement (EuroSPI 2012), Vienna, Austria.

36. **Jose L. J., César A. C., Francisco L. G., Luis M.** (2016), *"Designing Game Strategies: An Analysis from Knowledge Management in Software Development Contexts, Serious Games"*, Interaction and Simulation, pp.64-73

37. **James C. H., Joel L. D., David G. B.** (1994), *"Models of Information Processing in the Basal Ganglia*", MIT Press, pp. 185 – 185.

38. **Neetu S., Narendra S. C.** (2014), *"Differential Reward Mechanism Based Online Learning Algorithm for URL-based Topic Classification*", IEEE, Computational Intelligence and Communication Networks (CICN).

39. **Lua K., Wanga S., Xiea L., Wanga Z., Li M.** (2016), "*A dynamic reward-based incentive mechanism: Reducing the cost of P2P systems*", vol. 112, pp. 105 – 113.

40. **Wang H., Tsai C.** (2011), "*Game Reward Systems: Gaming Experiences and Social Meanings*".

41. **Walz S. P., Deterding S.** (2014), "*Gamification and Learning*", MIT Press, pp. 688.

42. **González C. S., Carreño A. M.** (2014), *"Methodological proposal for gamification in the computer engineering teaching*", IEEE, Computers in Education (SIIE).

43. **Qu W., Zhao Y., Wang M., Liu B.** (2014), *"Research on teaching gamification of software engineering*", IEEE, Computer Science & Education (ICCSE).

44. **Largo F. et. al.** (2016), "*Gamification of the learning process: lessons learned*", IEEE, IEEE Revista Iberoamericana de Tecnologias del Aprendizaje, pp. 1 – 1.

45. **Parizi R. M.** (2016), "*On the gamification of human-centric traceability tasks in software testing and coding*", IEEE, Software Engineering Research, Management and Applications (SERA).

46. **Parizi R. M., Kasem A., Abdullah A.** (2015), *"Towards gamification in software traceability: Between test and code artifacts"*, Software Technologies (ICSOFT), 2015 10th International Joint Conference on.

47. **Lacheiner H., Ramler R.** (2011), "*Application Lifecycle Management as Infrastructure for Software Process Improvement and Evolution: Experience and Insights from Industry*", 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, doi: 10.1109/SEAA.2011.51, pp. 286-293.

48. **Aytekin A. İ. et. al.** (2015), "*Uygulama Yaşam Döngüsü Yönetimi – Sistematik Eşleme Çalışması*", UYMS.

49. **Shaw K.** (2007), "*Application Lifecycle Management for the Enterprise*", Serena Software, White Paper, http://www.serena.com/Docs/Repository/company/Serena_ALM_2.0_For_t.pdf.

50. **Kääriäinen J., Välimäki A.** (2008), "*Impact of Application Lifecycle Management – A Case Study*", In: International Conference on Interoperability of Enterprise, Software and Applications (I-ESA), Berlin, Germany, pp. 55–67.

51. **J. K, Välimäki A.** (2009), "*Applying Application Lifecycle Management for the Development of Complex Systems : Experiences from the Automation Industry*" in EuroSPI, pp. 149–160.

52. **Kitchenham B. A., Dyba T., Jorgensen M.** (2004), *"Evidence-Based Software Engineering*", Proc. Of the 26th International Conference on Software Engineering (ICSE '04), Scotland, UK, pp. 273-281.

53. **Macit Y. et. al.** (2014), "*Büyük Ölçekli Bir Organizasyonda Uygulama Yaşam Döngüsü Yönetimi Uygulama Deneyimi*", Proceedings of the 8th Turkish National Software Engineering Symposium.

54. **Martin J.** (1991), "*Rapid Application Development.",* Macmillan. ISBN 0-02-376775-8.

55. **Beck K. et. al.** (2010), *"Principles behind the Agile Manifesto"*, Agile Alliance. Archived from the original on 14 June 2010.

56. **Krajewski L. J., Ritzman L. P.** (2005), "*Operations Management: Processes and Value Chains*", Pearson Education, Upper Saddle River.

57. **Beck K. et. al.** (2001), "*Manifesto for Agile Software Development*", https://agilemanifesto.org/

58. **Gangji A., Hartman, B.** (2015), "*Agile SCRUM for Denver Web Development*", Neon Rain Interactive. Retrieved September 25.

59. **Mangalindan J. P.** (2012), "*Play to win: The game-based economy*", Fortune, Archived from the original on 2012-11-12.

60. **Yilmaz M., V. O'Connor R., Collins J.** (2010), "*Improving Software Development Process through Economic Mechanism Design",* EuroSPI 2010, pp 177-188.

61. **Osborne M. J., Rubinstein A.** (1994), "*A Course in Game Theory",* Cambridge, MA: MIT, 1994. Print.:14

62. **Allison S. T., Beggan J. K., Midgley E. H.** (1996), *"The quest for "similar instances and simultaneous possibilities*": Metaphors in social dilemma research".

Journal of Personality and Social Psychology. 71: 479–497. doi:10.1037/0022-3514.71.3.479.

63. **Schultz W.** (2015), "*Neuronal reward and decision signals: from theories to data*", Physiological Reviews, pp 853–951.

64. **DRM Associates** (2002), "*New Product Development Glossary*", Retrieved 2006-10-29.

65. **Laplante P.** (2007), "What Every Engineer Should Know about Software", ISBN: 978-0849372285

66. **ISO/IEC** (2008), "*Systems and software engineering Software life cycle processes*", Amendment to ISO/IEC 12207-2008.

67. **Persse J.R.** (2006), "*Process Improvement Essentials*", O'Reilly Media, Inc.

68. **Lotufo R., Passos L., Czarnecki K.** (2012), *"Towards improving bug tracking systems with game mechanisms"*, Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, pp.2-11.

69. **Sasso T. D., Mocci A., Lanza M., Mastrodicasa E.** (2017), "*How to Gamify Software Engineering*", Software Analysis, Evolution and Reengineering (SANER).

70. **Fraser G.** (2017), "*Gamification of software testing*", Proceedings of the 12th International Workshop on Automation of Software Testing, pp.2-7.

71. **Metropolis N., Ulam S.** (1949), *"The Monte Carlo method"*, Journal of the American Statistical Association Vol. 44, No. 247, pp. 335-341.

72. **Kroese D. P., Brereton T., Taimre T., Botev Z. I.** (2014), "*Why the Monte Carlo method is so important today*". WIREs Comput Stat. 6: 386–392. doi:10.1002/wics.1314.

73. **Pham H.** (1999), "*Software Reliability"*, John Wiley & Sons Inc., p:567, ISBN 9813083840, 1999, "Software Validation. The process of ensuring that the software is performing the right process. Software Verification. The process of ensuring that the software is performing the process right.".

74. **Raychaudhuri S.** (2008), "*Introduction to Monte Carlo simulation"*, Proceedings of the 40th Conference on Winter Simulation, pp. 91-100.

75. **Burgin M. S., Maurice J. A.** (2009), "*Monte Carlo methods and super-recursive algorithms*", SpringSim '09 Proceedings of the 2009 Spring Simulation Multiconference Article No. 140.

76. **Kalantari N., Bako S., Sen P.** (2015), "*A machine learning approach for filtering Monte Carlo noise*", ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2015, Volume 34 Issue 4, 2015, Article No. 122.

77. **Joshua A. N., Gregory K. H., Jin W.** (2014), "*Bounded Monte Carlo integration using Java*", Proceedings of the 2014 ACM Southeast Regional Conference Article No. 29.

78. **Dennis J. N., Soemers J., Chiara F. S., Torsten S**. (2016), "*Enhancements for real-time Monte-Carlo Tree Search in General Video Game Playing*", Computational Intelligence and Games (CIG).

79. **Maia L. F., Viana W., Trinta F.** (2017), "*Using Monte Carlo tree search and google maps to improve game balancing in location-based games*", Computational Intelligence and Games (CIG).

80. **Lorentz R.** (2016), "*Using evaluation functions in Monte-Carlo Tree Search*", Theoretical Computer Science, Volume 644, pp. 106-113.

81. **Malefaki S., Koutras V. P., Platis A. N.** (2012), *"Modeling Software Rejuvenation on a Redundant System Using Monte Carlo Simulation*", Software Reliability Engineering Workshops (ISSREW).

82. **Pacagnella A. C., Pacifico O., Terra L. A.** (2015), *"Cost estimation in software development projects with Monte Carlo simulation"*, IEEE Latin America Transactions, pp. 3051 – 3058.

83. **Madani K.** (2011), "*A Monte-Carlo game theoretic approach for Multi-Criteria Decision Making under uncertainty*", Advances in Water Resources, Volume 34, Issue 5, pp. 607-616.

84. **Usfekes C. et. al.** (2017), *"Examining Reward Mechanisms for Effective Usage of Application Lifecycle Management Tools."*, 24th European Conference, EuroSPI 2017

85. **Clarke P., O'Connor R.V., Leavy B.** (2016), "*A Complexity Theory viewpoint on the Software Development Process and Situational Context"*, In: proceedings of the International Conference on Software and Systems Process (ICSSP), Co-Located with the International Conference on Software Engineering (ICSE), pp. 86-90, DOI:10.1145/2904354.2904369.

86. **Clarke P. and O'Connor R.V.** (2015), *"Changing situational contexts present a constant challenge to software developers*", 22nd European Conference on Systems, Software and Services Process Improvement (EuroSPI 2015), Springer-Verlag.

87. **IEEE** (2010), "*1044-2009 - IEEE Standard Classification for Software Anomalies*", ISBN: 0-7381-0406-X.

88. **Gulec U., Yilmaz M.** (2016), "*A serious game for improving the decision making skills and knowledge levels of Turkish football referees according to the laws of the game"*.

89. **Lapp S. A.** (1986), *"Derivation of an Exact Expression for Mean Time to Repair"*, IEEE Transactions on Reliability, pp. 336 – 337.

90. **Institute for Telecommunications Sciences** (2008), "*Mean Time To Repair definition",* Archived 2008-09-25 at the Wayback Machine.

91. **Fousch R.J.** (1989), "*PC software solutions for MTTR predictions*", Reliability and Maintainability Symposium.

92. **Zheng H., Liu W., Xiao C.** (2018), "*An activity-based defect management framework for product development"*, Computers & Industrial Engineering.

93. **Aqlan F., Ramakrishnan S., Shamsan A.** (2017), "*Integrating data analytics and simulation for defect management in manufacturing environments*", Simulation Conference (WSC).

94. **Rahman A., Hasim N.** (2015), "*Defect Management Life Cycle Process for Software Quality Improvement*", Artificial Intelligence, Modelling and Simulation (AIMS).

95. **Taba N. H., Ow S. H.** (2012), "*Improving Software Quality Using a Defect Management-Oriented (DEMAO) Software Inspection Model*", Modelling Symposium (AMS).

96. **Weerd I. V., Katchow R.** (2009), "*On the integration of software product management with software defect management in distributed environments*", Software Engineering Conference in Russia (CEE-SECR).

97. **Gopalakrishnan T. R., Suma V., Shashi K. N. R.** (2011), *"An analytical approach for project managers in effective defect management in software process*", Software Engineering (MySEC).

98. **Tuzun E. et. al.** (2019), "*Adopting integrated application lifecycle management within a large-scale software company: An action research approach"*, Journal of Systems and Software, pp. 63-82.

99. **Scheibehenne B., Greifeneder R., Todd P.** (2010), *"Can there ever be too many options? A meta-analytic review of choice overload*". Journal of Consumer Research. 37 (3): 409–25.

100. **National Science Foundation (NSF)** (2006), "*Report on Simulation-Based Engineering Science*", Blue Ribbon Panel.

101. **Fayter D., McDaid C., Eastwood A.** (2007), "*A Systematic Review Highlights Threats to Validity in Studies of Barriers to Cancer Trial Participation*", Journal of Clinical Epidemiology, vol. 60, no. 10, pp. 990- 991.

102. **Shen T. -. Yu, V. Y., Dunsmore H. E.** (1988), "*An analysis of several software defect models*" in IEEE Transactions on Software Engineering, vol. 14, no. 9, pp. 1261-1270.

103. **Usfekes C. et. al.** (2017), "*Systems, Software and Services Process Improvement*", EuroSPI 2017, vol. 748, pp. 259-268.

104. **Tuzun E. et. al.** (2019), "*Adopting integrated application lifecycle management within a large-scale software company: An action research approach*", Journal of Systems and Software, vol. 149, pp. 63-82.