GALATASARAY UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

AWARD DETERMINATION FOR CROWDSOURCED SOFTWARE DEVELOPMENT

Aslı SARI

July 2017

AWARD DETERMINATION FOR CROWDSOURCED SOFTWARE DEVELOPMENT

(KİTLE KAYNAK ESASLI YAZILIM GELİŞTİRMEDE ÖDÜL BELİRLEME)

by

Aslı SARI, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

in the

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

July 2017

This is to certify that the thesis entitled

AWARD DETERMINATION FOR CROWDSOURCED SOFTWARE DEVELOPMENT

prepared by **Aslı SARI** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering** at the **Galatasaray University** is approved by the

Examining Committee:

Assoc. Prof. Dr. Gülfem IŞIKLAR ALPTEKİN (Supervisor)
Department of Computer Engineering
Galatasaray University ----------

Assist. Prof. Dr. Ayşe TOSUN
Department of Computer Engineering
Istanbul Technical University ----------

Assist. Prof. Dr. Keziban ORMAN
Department of Computer Engineering
Galatasaray University ----------

Date: ----------

ACKNOWLEDGEMENTS

I would like to thank my advisor Assoc. Prof .Dr. Gülfem IŞIKLAR ALPTEKİN and Assist. Prof. Dr. Ayşe TOSUN for their guidance and help in the course of my research. Also, I would like to thank to my family who continuously supported me, especially while I was writing this thesis.

July 2017
Aslı SARI

TABLE OF CONTENTS

LIST OF SYMBOLS

AMK : Amazon's Mechanical Turk
COCOMO : Constructive Cost Model
CSE : Crowdsourcing in Software Engineering
CSMs : Crowdsourcing Software Development Markets
DEA : Data Envelopment Analysis
DI : The Degree of Influence
FC : Function Count
FP : Fixed Price
FP : Function Point
FPA : Function Point Analysis
IPR : Intellectual Properties Right
LOC : Lines of Code
MY/YR : Manyear/Year
PC : Processing Complexity
PCA : Processing Complexity Adjustment
PM : Person-months
SLR : Systematic Literature Review
SLIM : Software Lifecycle Management
SLOC : Source Lines of Code
UFP : Unadjusted Function Point
3GL : Third-Generation Programming Language

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Crowdsourcing is an emerging paradigm that outsources the software tasks to the large group of people via open call format. The effect of crowdsourcing in software engineering has increased dramatically in recent years. This thesis study first provides a systematic survey on emerging issues of crowdsourcing in software engineering. It involves a comprehensive survey on business models, technological platforms and frameworks, practices in software engineering, software economics, task award mechanisms, crowd selection, task decomposition strategies, and assisting tools. Then, an award determination model is proposed to be useful in crowdsourced software projects. The applicability of the model is shown on sample projects of TopCoder. The introduced award determination model is a way to analyze and discuss features of crowdsourcing in software economics aspects. In the thesis, the Putnam's SLIM model, which is proposed for effort estimation of software development, is applied to award determination in competition-based crowdsourced software development.

Keywords: Crowdsourcing, crowdsourcing in software engineering, effort estimation, award determination, software cost.

ÖZET

Kitle kaynak, dışarıdan temin edilen geniş bir insan grubuna, açık çağrı biçimiyle yazılım işlerininin yaptırıldığı, yeni nesil bir yazılım geliştirme yaklaşımdır. Son yıllarda, yazılım mühendisliğinde kitle kaynak yaklaşımının kullanımının çarpıcı biçimde artması, akademik yazında ilgili çalışmaların sayısını da etkilemiştir. Bu tezde, yazılım mühendisliğinde kitle kaynak yaklaşımının, seçilen araştırma soruları ışığında bir sistematik akademik yazın araştırması yer almaktadır. Sistematik akademik yazın araştırmasında, cevap aranan sorular aşağıda verilmiştir:

1. İş modelleri
2. Teknolojik platformlar ve sistemler
3. Kitle kaynak esaslı yazılım mühendisliği uygulamaları
4. Kitle kaynak esaslı yazılım ekonomisi
5. Görev-ödül mekanizmaları
6. İnsan topluluğu (kitle) seçimi
7. Görev parçalama stratejileri
8. Yardımcı araçlar

Akademik yazın araştırmaları sonucu elde edilen bilgiler ışığında, kitle kaynak esaslı yazılım geliştirmede kullanılacak bir ödül tahmin modeli önerilmiştir. Bu amaç doğrultusunda, bu tezde Putnam'ın SLIM efor tahmin modeli çalışılmıştır. Putnam'ın SLIM efor tahmin modeli öncelikle bir projede çalışan insan sayısını, zamanın bir fonksiyonu olarak tanımlar (Pillai & Nair, 1997). Önerilen model, TopCoder'daki örnek projeler üzerinde uygulanmıştır. Sonuç olarak, önerilen modelin kitle kaynak yaklaşımında ödül miktarını belirlerken kullanılabileceği hakkında sonuçlar elde edilmiştir.

Anahtar Kelimeler: Kitle kaynak, yazılım mühendisliğinde kitle kaynak, efor tahmini, ödül belirleme, yazılım maliyeti.

1. INTRODUCTION

Crowdsourcing is an emerging phenomenon based on outsourcing the work to undefined large network of individuals by means of open call requesting for participation. Howe (2006) used the crowdsourcing term firstly in a *Wired* magazine article. He defined the crowdsourcing in his blog as "outsourcing the act of a company or institution to an undefined generally large network of people in the form of open call". A crowdsourced work can be performed collaboratively or individually (Archak, 2010). According to Howe's definition, the main motivations behind the crowdsourcing are the open call format and the large network of potential laborers i.e. the crowd. Several crowdsourcing approaches have introduced both in academy and in industry. Therefore, crowdsourcing has various application domains: Recruiting participants for opinion collection tasks, recruiting participants for a basic task, recruiting participants for tasks that require expertise, recruiting participants for competitive tasks or recruiting participants for collaborative donation tasks (Hosseini et al., 2015). In crowdsourcing, participants may answer different *opinion-based problems*, such as online survey. Hence, the correctness of an answer depends on the view of the participants. *The basic problems* in crowdsourcing are defined as easy and simple ones that do not necessitate special knowledge or expertise (e.g. counting the number of stones in different pictures). On the other hand, *complex problems* are difficult and require expertise. *Competitive tasks* can be easy or difficult depending on the presence of tangible awards, and awards can be given to some competitors rather than all participants. In *collaborative donation tasks*, also known as *crowdfunding projects*, everyone donate and support participants via their non-compulsory money.

These problem types constitute the taxonomy of crowdsourcing (Figure 1.1).

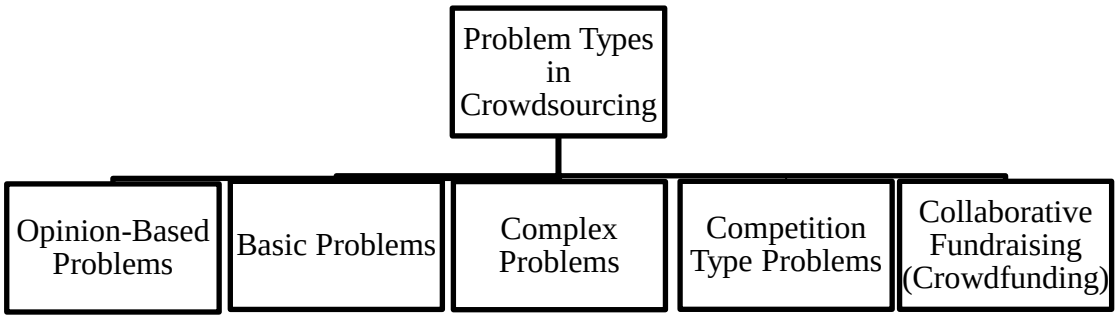

Figure 1.1: Problem Types in Crowdsourcing (Hosseini et al., 2015)

Boudreau & Lakhani (2013) proposes another classification approach of crowdsourcing according to the way of working with the crowd: contest, collaborative community, complementor and labor market. In crowd contests, the organization proposes a specific problem such as technical, analytical, scientific or design problems with tangible prizes. It then broadcasts an invitation with deadline to submit solutions in order to assess a good solution through many independent solutions. Crowd contest is a good opportunity for complex or novel problems to assess high-value solutions among multiple independent experimentation and diverse solutions. However, there are some concerns about management in running contest crowdsourcing. That is to say, the problem must be generalized to be easily understandable for people, be abstracted from company specific details and be structured for the implementation. Collaborative communities aim to accumulate ideas of multiple contributors in such as wikis, open-collaboration projects or frequently asked questions and aggregate them into coherent and value creating combination. Protection of intellectual property, controlling of the crowd and cohesiveness among them are strengths of collaborative communities. The third model is crowd complementors that are market for goods or services such as open operational, product, or marketing data initiatives, content mashup, applications to be built on your core product or technology, effectively transforming that product into a platform that generates complementary innovations while provide solutions to many different problems. For instance, Apple's iTunes, which compromises of large number of geographical distributed developers. On the other hand, protection of the functions and information in the core product is a challenge for this model due to using technological interfaces or hooks. The last model is the crowd labor market, which acts as an intermediary between buyers and sellers to match workers to human computation

and repeated tasks such as third-party intermediaries such as Elenca, oDesk and Guru. Identification of tasks for appropriate workers is the main difficulty of this model. Figure 1.2 summarizes these models.

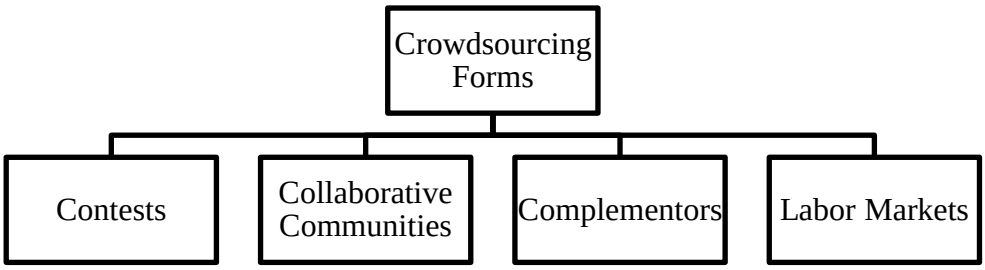

Figure 1.2: Crowdsourcing Forms (Boudreau & Lakhani, 2013)

A crowdsourcing system has three categories of components and their interactions between them (Vukovic, 2009). Crowdsourcing requestor initiates crowdsourcing process by submitting a task request, pays and awards the successful completion of the task. Crowdsourcing requestor has several appointments in terms of describing and management of crowdsourcing requests. Members of the crowd generate the crowdsourced tasks by submitting their solutions. Via a crowdsourcing platform, providers (crowd) complete tasks and requestors pay awards. Crowdsourcing platform also authenticates information of providers and requestors, executes crowdsourcing requests in different modes and forms in terms of advertising them on the marketplace, bidding for them via requestor or competition. Moreover, there are six connections among these components (Zhao & Zhu, 2014). Submitting a task and its related request, validating as evaluation of the feedback and selecting the appropriate ones and awarding for some crowdsourcing contests are three connections between the assigner as requestor and the platform. In addition, there are three actions between the providers as the crowd and the platform. Push and pull indicate functionalities such as personalized recommendation and customization provided by the platform to attract, intent and sustain the crowd. Participation considers people who join in some of the projects and take some actions to respond to the tasks. Bidding for some types of crowdsourcing systems is defined as a submission of produced outcomes of participations to the competitions. Besides, there is a direct link between the requestor and the crowd without the intermediary platform. These links consist of inquiry about

some details of the task to support their works, negotiation with the requestor for the requirements and awards or request a reply for their concerns. All these interactions between the requestor and the crowd are achieved by email, telephone or face-to-face communications.

Crowdsourcing in software engineering (CSE) has emerged from this concept and various software engineering tasks in terms of requirements extraction, design, coding and testing is crowdsourced to the developers in the form of open call format (Mao et al., 2015). There are several crowdsourcing models for software development in terms of peer production, competitions and microtasking (Latoza, 2016). Peer production is an example of open source in which large group of people contribute to software projects such as Linux, Apache and Firefox without monetary award. The second crowdsourcing model is the competitions, which have some similar aspects with outsourcing. TopCoder.com, a software development portal, is the commercial pioneer of this model. In competition-based crowdsourcing, client requests a work and pays for its completion. Unlike outsourcing, workers are considered as contestants rather than collaborators. The last crowdsourcing model is microtasking, in which complex tasks are decomposed into several standalone microtasks to be completed within a short time. This approach is pioneered by Amazon's Mechanical Turk (AMK). The choice of the type of crowdsourcing approach depends on companies' business goals (Zhao & Zu, 2014 ; Naik, 2016). In *insourced software development*, a software project or task is achieved by in-house expertise and resources rather than subcontracting to an external provider. When comparing crowdsourcing and insourcing as software development approaches, insourcing enables companies to keep complete control over the whole software development process at a higher cost to meet requirements of dependable and trustworthy software projects such as reliability, safety and security. A company subcontracts all or part of a software project to external agents i.e. third-party service provider in *outsourced software development*. Some people argue that crowdsourcing is based on Web 2.0 form of outsourcing, and plays significant role on advancement of Internet platform and its interactive technologies. However, there is a contract between client and supplier in outsourcing to define needs and goods or services are provided according to it in a cost-effective way. On the other hand, the client proposes tasks via open call and individuals within the crowd participate voluntarily. In o*pen sourced*

software development, an existing software project is improved collaboratively by allowing essential elements of a product, such as source code of software to public, without financial award or ownership. However, crowdsourcing is more private than open sourcing with respect to investment of organization for the solutions or ownership or intellectual properties right (IPR) of feedbacks. In addition, contributors satisfy with finding a better solution to the problem in open source whereas contributors in crowdsourcing expect monetary awards. The last difference is dependability of contribution among participants. In other words, members of the crowd contribute independently, such as idea competition or design contest, or collaboratively, such as Wikipedia or citizen science in crowdsourcing. On the other hand, participants work together, and there are dependencies between their contributions in open source.

Current practice of crowdsourcing in software engineering involves developing an online marketplace. C*rowdsourcing software development markets (CSMs)* concept is a growing interest for companies rather than traditional software outsourcing markets. CSM is based on outsourcing short-term projects that last only several days on a *fixed price (FP)*. The fixed price is a contracting method in which the project price is recognized before work begins and is paid when predefined milestones are reached (Gefen et al., 2016).

1.1 Motivations and Challenges of Crowdsourcing

Crowdsourcing has become popular by capturing considerable attention from the world. It provides increased development speed by means of many contributions of workers, which lead to generate alternative solutions for the same task. In addition, crowdsourcing facilitates flexibility in the use of specialist freelancers as democratization and liberating and learning new technologies (Latoza, 2016). Iterative and collaborative software development are opportunities for rapid feedback from the customer. Collaboratively defining the requirements and the scope of the software, splitting up the software into components and services that need to be developed, breaking down the work into smaller pieces or tasks and collective intelligence are other benefits of CSE (Satzger et.al., 2014). Obtaining quality of software, reducing the time to acquire the software product, cost reduction, diversity of solutions, many ideas

creation, recruitment as many contestants as possible, teaching contestants new knowledge and skillsets by competitions, funding as sponsors for projects, raising the publicity of organization among other participants as marketing are goals for CSE (Wu et al., 2013).

On the other hand, crowdsourcing has challenges in terms of allocating people for special tasks, collaboration for self-contained tasks and knowledge management (Machado et al., 2014). In addition, Dwarakanath et al. (2015) state issues in crowdsourced software development in terms of task management (decomposition a high level problem into a number of atomic tasks), security, management of the responses, provisioning of resources for the crowd, collaboration between individuals in the crowd, crowd selection strategies, and program management for monitoring crowd activities. In order to solve these challenges, software crowdsourcing models require new workflows (Latoza, 2016). These workflows encounter quality issues, crowd selection, coordination of contributions and share knowledge across the crowd. Naik (2016) emphasizes that large number of the crowd leads to several difficulties in terms of quality, liability, intellectual property rights, information security, privacy and security. Furthermore, every type of software projects may not be appropriate for crowdsourcing (Naik, 2016). It is said that less complex and standalone software development tasks without interdependencies are more suitable for software crowdsourcing. Therefore, types of tasks play significant role on success of CSE.

1.2 Thesis Objectives

This thesis provides provision of a detailed insight of emerging research areas of CSE. The thesis involves a systematic literature review. Systematic literature review examines emerging issues and literature in order to construct proposed model by means of searching keywords in the databases or libraries and answering research questions. It focuses on the following research areas: Analyzing the business models used for CSE, investigating technical infrastructure on which crowdsourcing process is implemented, identifying crowdsourced software development methodologies, identifying software process area(s) that crowdsourcing is utilized, identifying effort estimation approaches in crowdsourced software development, identifying the factors that affect effort

estimation, investigating task award strategies in CSE, analyzing strategies for crowd selection or formation in software engineering, investigating micro-tasking process performed in CSE, and identifying assisting tools for CSE. Comparison of the literature review results with previous literature reviews enables determining the emerging and abandoned research topics in CSE.

The findings of the literature review directed us to better examine the economic aspects of CSE. An award determination model that is based on the Putnam's SLIM model is introduced and adapted for CSE. The aim of using the Putnam's SLIM model in this thesis is to estimate the award of software projects and accordingly the required effort. This award determination approach will direct companies when deciding whether to outsource tasks the large group of people via open call format or not (i.e. making the decision of crowdsourcing or insourcing).

2. BASIC DEFINITIONS AND USED APPROACHES

2.1 Software Cost/Effort Estimation Methodologies

Predicting the required effort to develop software is an essential topic for researchers and practitioners. In academic literature, various studies are conducted to propose appropriate cost estimation methods for predicting the required effort. Software sizing is an important step in the process of cost estimation, for which several methods for software sizing are introduced (Boehm et.al., 2000 ; Leung & Fan, 2002; Aljahdali & Sheta, 2010). A commonly used software sizing method is using the *line of code*, which is the number of lines of the delivered source code of software (Leung & Fan, 2002). Another method is the *software science* that consists of the code length of source code and the volume of the amount of required storage space (Leung & Fan, 2002). The *function points* is another approach for software sizing with respect to functionality of the program (Leung & Fan, 2002). The *feature points* is an extension of function points for measurement of highly algorithmic complex systems with few input or output (Leung & Fan, 2002). The last commonly used method is the *object point* that is based on the number and the complexity of the screens, reports and 3GL components (Leung & Fan, 2002).

On the other hand, the software cost estimation models may be as algorithmic and non-algorithmic approaches (Leung & Fan, 2002). In non-algorithmic methods, the *analogy costing* requires one or completed projects, which are similar to the new project, and it performs estimation through reasoning by analogy using the actual costs of previous projects. One or more experts estimate, each with respect to their own methods and experiences in *expert judgment* approach (e.g. Delphi technique). In *Parkinson method*, the cost is determined by the available resources. The best price wins the project in *price-to-win* method, which is based on customer's budget rather than software

functionality. The cost of each component in the software system is estimated individually and the results are summed up to produce an estimate for the overall system in *bottom-up* approach; whereas the total cost is apportioned into the various components of the software in *top-down* method.

Algorithmic approaches are based on mathematical models to estimate the cost as a function of a number of variables (Leung & Fan, 2002). In algorithmic models, there are cost factors besides the software size to distinguish among the existing algorithmic methods by means of selection of cost factors (Leung & Fan, 2002). The taxonomy of cost factors involves four types (Leung & Fan, 2002): *Product factors*, such as required reliability, product complexity, *computer factors* e.g. execution time constraint, main storage constraint, *personnel factors* in terms of analyst capability, application experience, and *project factors*, such as use of software tool and required development schedule. In the effort function of algorithmic models, $x_1, x_2, \dots, x_n$ are the cost factors, and it can be seen in Eq. (2.1.1).

$$Effort = f(x_1, x_2, \dots, x_n) \tag{2.1.1}$$

We can also classify the algorithmic methods in respect to the form of the function of f (Leung & Fan, 2002). In *linear models*, linear effort function can be seen as in Eq. 2.1.2. It includes $a_1, \dots, a_n$ coefficients, which are chosen in order to best fit the completed project data (Leung & Fan, 2002):

$$Effort = a_0 + \sum_{i=1}^{n} a_i x_i \tag{2.1.2}$$

In *multiplicative models*, the effort function also includes $a_1, \dots, a_n$ which are chosen as coefficients to best fit the completed project data. The effort function is given as:

$$Effort = a_0 \prod_{i=1}^{n} a_i^{x_i} \tag{2.1.3}$$

In *power function models*, effort is expressed as in Eq. (2.1.4), where *S* is the code size, and *a* and *b* are functions of other cost factors.

$$Effort = a \,.\, S^b \tag{2.1.4}$$

The common models of power function methodology are the *Constructive Cost Model (COCOMO)* and *Putnam's model*. Boehm (1981) has firstly proposed COCOMO. The model enables the identification of the developed time, the effort and the maintenance effort, as mathematical equations (Aljahdali & Sheta, 2010).

Another algorithmic model is based on linear regression. The ultimate goal of the *regression model* is to find the function $f(x)$ which best models the training data (Oliveira, 2006). To predict the total effort in man-months of future software projects, *linear regression* finds the line that minimizes the sum of squares errors on the training set (Oliveira, 2006). That is to say, linear regression method is another utilization approach for cost estimation as model calibration (Leung & Fan, 2002). *Discrete models*, such as *Aron model* and *Boeing model*, are other effort, duration, difficult and other cost factor-related models (Leung& Fan, 2002). In *other models*, *Price-S* computes project cost and schedule by estimating project size, type and difficulty. *SoftCost* is related to size, effort and duration and this model uses Rayleigh probability distribution to address risk (Leung & Fan, 2002).

2.2 The Putnam Model

The Putnam model plays significant role on predicting the costs and delivery schedules of software projects. The Putnam model performs software life-cycle in terms of the Rayleigh distribution of project personnel level versus time (Han et al., 2005). The Putnam model has concerns about the number of people working on a project as a function of time, which is characterized by Rayleigh distribution (Warburton, 1983).

That is to say, Rayleigh curve indicates the rate at which resources are consumed by software engineering projects (Parr, 1980). Rayleigh equation indicates manpower which is measured in people per unit time as a function of time usually expressed in manyear/year (MY/YR) (Pillai & Nair, 1997). Approximately ± 25% of the expected manpower value during the manpower buildup phase of the profile is showed as a tolerance by Putnam (Pillai & Nair, 1997). Manpower as a function of time is expressed as (Pillai & Nair, 1997):

$$\dot{p} = 2 \,.\, K \,.\, a \,.\, t \,.\, e^{-at^2} \tag{2.2.1}$$

According to Eq. (2.2.1), $\dot{p}$ represents the manpower in MY/YR, K represents the total area under the curve and a is a constant, i.e. $a = 1/(2t_d^2)$ in which t_d is the time for manpower to peak. Cumulative number of people used by the system at any time t is formulated as (Putnam, 1978):

$$y = K(1 - e^{-at^2}) \text{ MY} \tag{2.2.2}$$

Putnam utilizes the Rayleigh curve together with a number of empirically derived assumptions in order to obtain following equation (Kitchenham & Taylor, 1984):

$$S_S = C_K \,.\, K^{1/3} \,.\, t_d \tag{2.2.3}$$

In Eq. (2.2.3), S_S indicates the number of source statements in the final product, t_d is the time at which the manpower curve reaches a maximum, and it is identified with the development time, i.e. $t_d = t^{4/3}$, and C_K is the technology factor as a constant for development environment (Kitchenham & Taylor, 1984). Therefore, Putnam model assumes a relationship between product size, development time and total effort for a particular project (Kitchenham & Taylor, 1984). Besides, Putnam model defines productivity, as in Eq. (2.2.4) (Kitchenham & Taylor, 1984). Eq. (2.2.4) formulates productivity considering the code of the end product, and the effort which is required to

produce it. According to the formula, total effort to produce the code includes overhead and also test and validation effort (Kitchenham & Taylor, 1984).

$$Productivity = \frac{\begin{matrix} Total \\ Size\ of\ End\ Product\ Code \end{matrix}}{Total\ Effort\ to\ Product\ Code} \quad (2.2.4)$$

2.3 Productivity

Productivity achieves product with quality at low cost. Therefore, several methodologies have been proposed to achieve high productivity in software projects. Two major approaches have been used for estimating software productivity (Woodfield et. al., 1983). The first approach is based on lines of code per programmer/ month, on work unit and the second approach is based on the cost per line of code, on cost unit. Woodfield et al. (1983) propose productivity model as function of problem size, resources consumed in production and the quality of the end product. Moreover, *Data Envelopment Analysis (DEA)* is a performance evolution method in which input parameters are used as constraints. DEA maximizes efficiency, which is generally measured as output per input, as a function of output parameters (Saikia et. al., 2016).

$$Productivity = \frac{y}{x} \quad (2.3.1)$$

In Eq.(2.3.1), y indicates the output, which is measured by source lines of code (SLOC), function points (FP), or object points in software projects, and x is effort as the number of person-months (PM), and productivity refers to the number of FP developed per PM (Stensrud & Myrtveit, 2003). Stensrud & Myrtveit (2003) state that the more FP per PM leads to higher productivity. Besides, Moser (1996) points out incremental productivity in terms of incremental function developed per person-days of effort.

The authors state that there is strong relationship between software size and both productivity and defect rate (Maccormack et al., 2003). In other words, larger projects

lead to high rate of productivity and lower levels of customer-reported defects per LOC. Early prototyping implies lower defect rate, but higher productivity.

Anselmo & Ledgard (2003) discuss issues of productivity in terms of software complexity, scalability and reusability in software development environments. When considering measuring of end product, Anselmo & Ledgard (2003) state functionality as determination of size and complexity of the function space specified for a software product, complexity i.e. difficulty in developing a piece of software and quality such that availability of its specified functions, time and cost to support that software to maintain an acceptable level of availability, which must be determined by the users of that software. Affecting factors of productivity such as independence, understandability, flexibility, visibility and abstraction are also determined as affecting for the man-hours, time to develop and support a software product (Anselmo & Ledgard, 2003).

In addition, software and programming productivity can be determined as measure of the time and/or cost required to deliver and maintain software systems (Duncan, 1988). For software productivity, there are two major dimensions: The first one is the change in the quantity of software produced during the development at a given cost. The second dimension is the quality of the final software system. Duncan (1988) emphasizes that engineering productivity metric enables determining the quantity of the produced code at each development-month as an indicator of improvement of programmer productivity. When considering software development life-cycle, requirements analysis phase is suitable for productivity measurement (Moses et al., 2006). In addition, language generation type, application domain, development type e.g. enhancement, new development and system size influence on productivity, and language type, development type and project team size affect to effort (Moses et al., 2006). Furthermore, Briand et al. (1998) state a productivity model which is strongly related to the cost overhead.

There are several project characteristics that are proposed for productivity, such as the size and complexity of the project, project duration, newness of the project and team size (Blackburn et al., 1996). For instance, productivity decreases with project duration,

and team size is inversely proportional with productivity. Basili et al. (1996) emphasize the difficulty of measurement of size of the project due to object-oriented mechanisms, such as inheritance and aggregation of classes in their proposed productivity formulation. Therefore, there are some difficulties about measurement of size of the software projects owing to using the programming language. In a research, the identification of the organization perspectives for productivity measurement in software projects achieves different evaluation of the inputs and outputs of a production process (Júnior, 2009).

There are several productivity approaches that are pointed out the combination of size and resources (Card, 2006). *Physical productivity* is the ratio between the amount of product and the resources consumed as usually effort. *Functional productivity* is the ratio of the amount of the functionality delivered to the resources consumed as usually effort. *Economic productivity* is the ratio of the value of the produced product and the cost of the resources used to produce it. Kitchenham & Mendes (2004b) propose *size-based effort estimation model* to emphasize the relationship between different size measures in different aspects of software product and effort for measuring the productivity.

Another study is related to measurement of software productivity which is related to the measurement of the output and input to the software development process (Yu et.al., 1991). According to this study, productivity factors improve the quality of the software, when accurately measuring software product attributes during the development process.

2.4 Function Point Analysis

In the award determination model, the size of the project is calculated using the Function Point Analysis (FPA). Alberth (1979) introduced Function Point (FP) methodology in order to measure functionality delivered by software. FPA counts the used functions, which are meaningful to user in the software application. FP count for a software product starts with classifying and counting the five user functions: *External Input Types, External Output Types, Logical Internal File Types, External Interface File Types and External Inquiry Types* (Low & Jeffery, 1990).

For each function type, the weights used for each function types in respect to their complexities are shown Table (2.4.1).

Table 2.4.1: The Raw Function Point Worksheet (Low & Jeffery, 1990)

Function Type / Complexity	**Simple**	**Complexity Average**	**Complex**
External Input	x3	x4	x6
External Output	x4	x5	x7
Logical Internal File	x7	x10	x15
External Interface File	x5	x7	x10
External Inquiry	x3	x4	x6

The number of each function is multiplied by corresponding weight in the given table. The total number of Function Count (FC) leads to *Unadjusted Function Point (UFP)* (Albrecht & Gaffney, 1983). Eq. (2.4.1) defines FC known as UFP in which z_{ij} is the count for component i at level j e.g. outputs at high complexity and w_{ij} is the fixed weight assigned by the Albretch procedure, seen as Table 2.4.1 (Matson et al., 1994).

$$Function\ Count = \sum_{i=1}^{5}\sum_{j=1}^{3} w_{ij}z_{ij} \tag{2.4.1}$$

The degree of influence (DI) is determined by answering 14 *General System Features*, which take values between 0 and 5, to signify none to essential for adjustment of application and environment complexity. In order to define *Processing Complexity (PC)*, 14 general system features are:

1. Data Communication
2. Distributed Functions
3. Performance
4. Heavily Used Configuration
5. Transaction Rate
6. Online Data Entry
7. End User Efficiency

8. Online Update
9. Complex Processing
10. Reusability
11. Installation Ease
12. Operational Ease
13. Multiple Sites
14. Facilitate Change

After this step, *Processing Complexity Adjustment (PCA)* is calculated by Eq. (2.4.2).

$$PCA = 0.65 + (0.01\ x\ PC) \tag{2.4.2}$$

The Function Points (FPs) delivered by an application program is measured by Eq. (2.4.3).

$$FP = FC\ x\ PCA \tag{2.4.3}$$

Finally, we have calculated estimated LOC of the given software project by means of FP and using the corresponding programming language coefficient, as seen in Eq. (2.4.4) (Borandag et al., 2013).

$$LOC = FP\ x\ Programming\ Language\ LOC\ Coefficient \tag{2.4.4}$$

In addition, we have interpreted results between project size in LOC and cost per LOC of high level languages. For instance, the cost per LOC of high level language such as Java is $6.25 (Jones, 2008). Moreover, the project cost per line is determined as $3.98 for JavaScript programming language[1].

[1] http://www.yegor256.com/2014/04/11/cost-of-loc.html

3. LITERATURE REVIEW

Systematic literature review examines emerging issues and literature in order to construct proposed model by means of searching keywords in the databases or libraries and answering research questions. In this section, the guideline steps is used after giving a brief introduction to the crowdsourcing concept (Kitchenham, 2004a).

The effect of crowdsourcing has increased dramatically both in academic research and industry, recently. In our systematic literature survey study, at the end of the quality assessment process, 46 primary studies are selected and analyzed. These primary studies are grouped in respect to our research questions in order to extract useful information. Our research questions can be grouped into four main classes as: framework used for crowdsourcing, software economic aspects of crowdsourcing, crowd building, and related tools.

3.1 Research Questions

This subchapter provides state of the art of crowdsourcing in software engineering research area, together with the practitioner's view. Kitchenham & Charters (2007) have proposed a *systematic literature review (SLR)* procedure, which is also taken as a guide for the SLR in this thesis. The research questions that are examined are:

- RQ1.1: What are the business models used for crowdsourcing in software engineering (CSE)?
- RQ1.2: What are the technological platforms used for management of CSE?
- RQ2: How are crowdsourced software development processes modeled?
- RQ3: For which software process area(s) crowdsourcing is utilized?
- RQ4.1: What kind of effort estimation approaches are employed in crowdsourced software development?

- RQ4.2: What are the cost drivers used in effort estimation?
- RQ5: How are task awards determined in CSE?
- RQ6: What kind of strategies exist for crowd selection or formation in software engineering?
- RQ7: How tasks are decomposed into micro-tasking process performed in CSE?
- RQ8: Which tools are used to assist CSE?

3.2 Searching Keywords

In order to select the most relevant studies with respect to our research questions, we have identified five keywords and formulated their combinations by means of logical operators. The chosen keywords are:

- crowdsourcing OR crowdsourced OR crowd
- crowdsourcing OR crowdsourced OR crowd AND “software engineering”
- crowdsourcing OR crowdsourced OR crowd AND “software development”
- “competitive programming”
- “crowdsourced development”

As databases, we have chosen IEEEXplore, ACM Digital Library, Web of Science, SCOPUS, SpringerLink and ScienceDirect. These libraries are popular and well-known databases for searching computer science related issues. Therefore, we have conducted our SLR by means of these content-richest libraries. The searching process is conducted without any date limit and only academic publications are included. We have retrieved a total of 5295 publications.

Table 3.2.1 gives detailed information about the number of retrieved papers in these databases.

Table 3.2.1: Number of Studies Retrieved in Databases

Database	Number of publications	First Filtering			Second Filtering	
		Excluded	Duplicated	Included	Excluded	Included
IEEE Xplore	291	234	16	34	1	12
ACM	190	151	10	27	0	14
Science Direct	1827	1812	84	9	0	3
Web of Science	51	45	8	2	1	2
Scopus	517	406	36	24	1	4
Springer Link	2419	2383	110	24	1	11
TOTAL	**5295**	**5031**	**264**	**124**	**120**	**46**

3.3 Screening of Relevant Papers

The screening of relevant papers is the process in which the inclusion and exclusion criteria and quality assessment process for further selection of primary studies are determined. By means of these conducted steps, we have determined papers, which are related to our research questions and constituted our final list of primary studies.

3.3.1 Inclusion and Exclusion Criteria

We have determined inclusion criteria as English written publications, crowdsourcing and software engineering related publications, scientific publications published in conferences, journals and chapters from books. On the other hand, we have excluded non-scientific publications, presentations, newspaper and magazine articles, blog posts, presentations, abstracts only and publications which are out of our research topics.

3.3.2 Study Selection and Data Extraction

First, the five search strings are searched in all the databases. As a result, 5295 metadata information of the paper are found. The second step is achieved by a first pilot study, which is based on randomly selected 100 papers. Each selected papers'

abstract, keyword and title information have been reviewed by each pair of researchers in order to distinguish relevant papers to our research questions. During this pilot study, approximately 65 papers have been reviewed by each researcher. Disagreements among the researchers are discussed and solved after this step. We have conducted first filtering on 5295 papers' abstract, title and keywords, such that we have removed duplicate and irrelevant studies from the list. As the result of the first filtering, 124 papers are remained.

3.3.3 Quality Assessment

Second pilot study is conducted on 124 relevant papers. In second pilot study, we have randomly selected 15 full texts of papers, which have been read by each individual researcher. Each paper is analyzed with respect to research questions. They are evaluated by giving a *YES/NO* answer. Before consensus, we have discussed our disagreements, which are generally on the research questions related to business models, technological platforms and assisting tools. We have resolved these disagreements by reviewing papers together in our meetings.

In quality checklist step, 124 papers are evaluated with respect to quality assessment checklist, which is based on *YES/NO* questions. In order to determine *quality checklist score*, *YES* answers are counted for each paper. We have kept the papers with a score of three or more YES. 46 studies have passed our quality assessment criteria as the primary study of our SLR after this step. The list of primary studies are available[2]. 46 primary studies are read and evaluated by each researcher with their different perspectives to extract answers for our research questions.

3.3.4 Data Synthesis

In order to synthesize the information from 46 primary studies, we have conducted thematic analysis. Thematic analysis is a qualitative analytic method to identify, analyze and report patterns or themes within data (Braun & Clarke, 2006). There are two ways to identify patterns or themes within data: *Inductive* or *deductive*. In an

[2] http://tinyurl.com/kzp6spl

inductive approach, the themes are strongly related to the data themselves, while a deductive approach provides a less rich description of the data overall, and more a detailed analysis of some aspect of the data. In this SLR study, we have performed inductive approach. That is to say, we have read primary studies to extract keywords used in the primary studies and group them to define final themes, which are used as information for our research questions.

3.4 Results

3.4.1 Business Models Used for CSE

Business models are related to participation of crowd workers, interaction among them, submission activity and final selection for the best solution. The most popular model is the competition-based, where participants compete with each other and the winning solution is chosen by the client. In this model, the task is broadcasted via platform to the crowd and there is no interaction among the crowd workers. TopCoder is a commercial pioneer of this model. In addition, competition-based model compromises of two models. The first one is based on game theory, where the crowd workers participate and submit the solution with respect to other registered workers' activities (Wu et al., 2013; Wu et al., 2015). On the other hand, the crowd workers meet with each other, discuss and provide feedback to each other in collaborative models (Ramakrishnan, 2014).

The crowd workers determine their bid depending on their effort and cost for achieving the task. The submission for the task is determined by the winning bid in auction-based model (Satzger et. al., 2014). Furthermore, client can select crowd workers based on their reputations, skills, qualification or trustworthiness measures and may invite them to the task in invitation-based model. After invitation, workers who accept the invitation can submit their solution.

In collaborative model, workers can see their solutions and revise their solutions to provide a better solution for the task.

This business models are team-based as workers' formation or the subtasks are assigned to workers (Tung & Tseng, 2013; Vukovic & Das, 2013; Alvertis et al., 2016).

The last proposed model is performed by an intermediary, such as a broker, by which all crowdsourcing activities can be achieved (Edgeman et al., 2013). According to these types of business models, crowdsourcing activities and interactions among the crowd effect the selection of business models in software engineering.

All these business models are summarized in Table 3.4.1.1

Table 3.4.1.1: Taxonomy of Business Models

Business Models	**Primary Studies**
Competitive	(Archak, 2010; Nag et al., 2012; Wu et al., 2013; Tsai et al., 2014; Xu & Wang, 2014; Yakushin & Lee, 2014; Dwarakanath et al., 2015; Hasteer et al., 2015 ; Li et al., 2015; Xie et al., 2015; Baba et al., 2016; Dwarakanath et al., 2016; Weidema et al., 2016)
• Game theory-based • Collaborative	(Wu et al., 2013; Wu et al., 2015) (Wu et al., 2013; Hu & Wu, 2014; Ramakrishnan & Srinivasaraghavan, 2014)
Auction-based	(Satzger et al., 2014)
Invitation-based	(Vuković, 2009; Xiao & Paik, 2014; Zogaj et al., 2014; Dwarakanath et al., 2015; Luz et al., 2015; Dwarakanath et al., 2016)
Collaborative	(Vukovic & Das, 2013; Wu et al., 2013; Groen, 2015; Latoza et al., 2015; Li et al., 2015; Zhao & Hoek, 2015; Aletdinova et al., 2016; Hu & Jiau, 2016)
• Team-based • Assignment of workers to subtasks	(Alvertis et al., 2016) (Tung & Tseng, 2013; Vukovic & Das, 2013)
Via a broker/intermediary	(Edgeman et al., 2015)

3.4.2 Technological Platforms Used for Management of CSE

Technological platforms that are used for management of CSE perform the whole process of crowdsourcing activities. The most popular examples of commercial platforms are TopCoder as a competitive business model, and several collaborative business models such as AppStori (Wu et al., 2013; Li et al., 2015).

Moreover, enterprise crowdsourcing utilizes also technological platforms. In this technological platforms, design, testing and integration phase play significant role on the evaluation, testing and integration into the existing applications of the crowd's solutions (Dwarakanath et al., 2015). In addition, hierarchical components of the task such that implementation independently in predefined time, which are based on iterative task decomposition process are achieved by the technical architects in the enterprises. All related artifacts in terms of user interfaces, test cases and component description are uploaded to the platform. Crowd workers' solutions are automatically integrated with the existing system after test cases generation by the platform. In addition, cloud-based crowdsourcing platform is the other proposed technical platforms for crowdsourcing in our primary studies (Xu et al., 2015; Wu et al., 2015). Table 3.4.2.1 illustrates these technological platforms.

Table 3.4.2.1: Technological Platforms

Technological Platforms	**Primary Studies**
Commercial platforms	(Yan & Wang, 2013; Ramakrishnan & Srinivasaraghavan, 2014; Zogaj et al., 2014; Luz et al., 2015; Aletdinova et al., 2016)
TopCoder	(Mao et al., 2013; Wu et al., 2013; Stol & Fitzgerald, 2014; Ågerfalk et al., 2015; Hasteer et al., 2015; Li et al., 2015; Yang et al., 2016)
AppStori	(Wu et al., 2013; Li et al., 2015)
Enterprise crowdsourcing mechanisms	(Vukovic & Das, 2013; Scupola & Nicolajsen, 2014; Dwarakanath et al., 2015; Edgeman et al., 2015; Dwarakanath et al., 2016)
Cloud-based crowdsourcing	(Vuković, 2009; Tsai et al., 2014; Wu et al., 2015; Xu et al., 2015)

3.4.3 Crowdsourced Software Development Processes Models

Testing process is frequently discussed with the aspect of crowdsourcing (Machado et al., 2014; Zogaj et al., 2014). In addition, TopCoder's process model has been studied in several primary studies (Mao et al., 2013; Dwarakanath et al., 2015; Hasteer et al., 2015). According to these studies, TopCoder generates specification, implementation and testing. In addition, parallel process development activities, such as design and coding are proposed for crowdsourced software development (Dwarakanath et al., 2015). Several existing software development methodologies (*waterfall* and *Scrum*) have been proposed in several papers (Stol & Fitzgerald, 2014; Dwarakanath et al., 2015).

3.4.4 Crowdsourced Software Process Area(s)

Coding is the most popular software development activity of implementation of crowdsourcing. In addition, requirements engineering, design, development and testing are another software development activities in which crowdsourcing approach is applied

(Wu et al., 2013; Vukovic & Das, 2013; Jiang & Matsubara, 2014; Latoza et al., 2014; Satzger et al., 2014; Wu et.al., 2015) (Table 3.4.4.1).

Table 3.4.4.1: Crowdsourcing Process Areas

Process Area(s)	**Primary Studies**
Requirements Engineering	(Jiang & Matsubara, 2014; Satzger et al., 2014; Groen, 2015; Wu et al., 2015; Alvertis et al., 2016; Hu & Jiau, 2016)
Design (UI, Architecture)	(Archak, 2010; Stol & Fitzgerald, 2014; Saremi & Yang, 2015; Xu et al., 2015; Wu et al., 2015; Zhao & Hoek, 2015; Weidema et al., 2016)
Coding/Development	(Archak, 2010; Nag et al., 2012; Mao et al., 2013; Hu & Wu, 2014; Latoza et al., 2014; Ramakrishnan & Srinivasaraghavan, 2014; Stol & Fitzgerald, 2014; Yakushin & Lee, 2014; Ågerfalk et al., 2015; Dwarakanath et al., 2015; Hasteer et al., 2015; Mao et al., 2015; Saremi & Yang, 2015; Wu et al., 2015; Xie et al., 2015; Xu et al., 2015; Dwarakanath et al., 2016; Weidema et al., 2016; Yang et al., 2016)
Testing	(Mäntylä & Itkonen, 2013; Tung & Tseng, 2013; Latoza et al., 2014; Machado et al., 2014; Yan et al., 2014; Zogaj et al., 2014; Ågerfalk et al., 2015; Guo et al., 2016)
• Usability Testing	(Scheneider & Cheung, 2013; Bruun & Stage, 2015)

3.4.5 Effort Estimation Approaches in Crowdsourced Software Development

COCOMO and linear regression models with price drivers have been studied for effort estimation in crowdsourced software development (Mao et al., 2013; Dwarakanath et al., 2015). Dwarakanath et al. (2015) propose effort estimation methodology for the special web application by means of COCOMO II estimation technique. Before they build use-cases and design wireframes, functional requirements are gathered in the analyse phase following by utilization of COCOMO II for effort estimation of this web application. In this paper, the unadjusted function points are used as the sizing metric of the software. The function points and their complexity levels in a software project are great opportunities for quantification the amount of information processing

functionality. According to web application experiment, 50 unadjusted function points lead to 3.5 person-months or 532 hours of effort. In addition, the required effort is divided into different phases of the software development process. Their results show that the overall effort and cost spent in building application including design is well within COCOMO II.

Another effort estimation technique is achieved by price predictor models and a multiple linear regression model with 16 price drivers on the TopCoder platform (Mao et al., 2013). In their study, the result of a total of 12 different prediction models is presented. In addition, Naïve, Random and COCOMO'81 approaches are used as comparison models to emphasize that COCOMO'81 model, the Naïve model and the Random method are outperformed by all 9 predictive models. Their results show that between size and effort for crowdsourced projects on TopCoder, there is not an obvious correlation. The best performing model is C4.5, and it is a motivation for them to value in predictive modeling for crowdsourced software development and other well performing models, as well.

3.4.6 Cost Drivers Used in Effort Estimation

In crowdsourcing aspect, cost drivers are categorized as monetary and/or non-monetary awards. Expected duration, complexity and quality of input, effort and reputation mechanisms have influence on determining the awards. Table 3.4.6.1 summarizes cost driver-related primary studies.

Table 3.4.6.1: Cost Drivers

Cost Drivers	**Primary Studies**
Expected duration	(Stol & Fitzgerald, 2014; Ågerfalk et al., 2015; Bruun & Stage, 2015)
Input complexity	(Archak, 2010; Mao et al., 2013; Stol & Fitzgerald, 2014; Ågerfalk et al., 2015)
Quality of input	(Mao et al., 2013; Stol & Fitzgerald, 2014; Zogaj et al., 2014)
Effort	(Jiang & Matsubara, 2014; Weidema et al., 2016)
Online reputation	(Archak, 2010; Wu et al., 2013)

3.4.7 Determination of Task Awards in CSE

In order to determine task awards for crowdsourcing in TopCoder, cash prizes and long-term incentives (e.g. online reputation awards) are offered in contests (Scupola & Nicolajsen, 2014; Stol & Fitzgerald, 2014; Ågerfalk et al., 2015; Saremi & Yang, 2015). Awards are determined in respect to TopCoder awarding mechanism, performance-based, game theory-based, contest theory-based, deadline driven, and revenue-sharing model (Table 3.4.7.1). In CSE, the award mechanisms are strongly related with pricing mechanisms. Each of these task awarding mechanisms has special features. For instance, TopCoder awarding mechanism provides cash prices to winners, whereas performance-based offers a monthly fee, bonus rates or commission rates.

Consequently, Table 3.4.7.1 gives information about task awarding mechanisms in our primary studies.

Table 3.4.7.1: Task Awarding Mechanism

Task-Awarding Mechanism	**Primary Studies**
TopCoder awarding mechanism	(Archak, 2010; Nag et al., 2012; Scupola & Nicolajsen, 2014; Stol & Fitzgerald, 2014; Ågerfalk et al., 2015; Saremi & Yang, 2015)
Performance-based scheme	(Scupola & Nicolajsen, 2014; Stol & Fitzgerald, 2014; Zogaj et al., 2014; Ågerfalk et al., 2015; Xu et al., 2015; Weidema et al., 2016)
Game theory-based model	(Wu et al., 2013; Hu & Wu, 2014; Wu et al., 2015)
Contest theory-based model	(Wu et al., 2015; Xu et al., 2015)
Deadline-driven reward optimization	(Satzger et al., 2014)
Revenue sharing-based model	(Jiang & Matsubara, 2014)

3.4.8 Strategies for Crowd Selection or Formation in CSE

Crowd selection is another emerging issue for crowdsourcing in software engineering. Due to the fact that many workers with different perspectives, backgrounds and experiences complete the same task and this leads to generate alternative solutions. By selecting the best alternative solutions or requesting more work, which is combining aspects of several alternatives, crowdsourcing can perform higher quality solutions (Latoza & Hoek, 2016). Several studies use student groups as crowds (Nag et al., 2012; Mäntylä & Itkonen, 2013; Ramakrishnan & Srinivasaraghavan, 2014; Brunn & Stage, 2015; Latoza et al., 2015). In addition, the members of the crowd may be trained or untrained. Developer recommendation for crowdsourcing considers the research area for matching the best workers to the tasks in the crowdsourcing activities (Satzger et al., 2014; Xiao & Paik, 2014; Yan et al., 2014; Zogaj et al., 2014; Mao et al., 2015; Guo et al., 2016; Weidema et al., 2016).

Table 3.4.8.1: Crowd Types

Crowd Type	**Primary Studies**
Public-private partnership	(Ramakrishnan & Srinivasaraghavan, 2014)
Entrepreneurs	(Scupola & Nicolajsen, 2014)
The people	(Scheneider & Cheung, 2013; Yan & Wang, 2013; Hu & Wu, 2014; Yakushin & Lee, 2014; Zogaj et al., 2014; Weidema et al., 2016)
Experts	(Mäntylä & Itkonen, 2013; Tung & Tseng, 2013; Hu & Wu, 2014; Satzger et al., 2014; Tsai et al., 2014; Xiao & Paik, 2014; Yakushin & Lee, 2014; Bruun & Stage, 2015; Mao et al., 2015; Latoza et al., 2015; Luz et al., 2015; Dwarakanath et al., 2016; Guo et al., 2016; Weidema et al., 2016)
Group identified per interests or work activities	(Nag et al., 2012; Tung & Tseng, 2013; Ramakrishnan & Srinivasaraghavan, 2014; Satzger et al., 2014; Scupola & Nicolajsen, 2014; Bruun & Stage, 2015; Latoza et al., 2015)
Mixed group	(Ramakrishnan & Srinivasaraghavan, 2014; Edgeman et al., 2015)

3.4.9 Micro-Tasking Process in CSE

The task decomposition approaches are performed with respect to dependability of tasks, characteristics of tasks in terms of size, dimension and aspects features and process types (i.e. dynamic, iterative, aggregated and sequential). They are shown in Table 3.4.9.1.

Table 3.4.9.1: Task Decomposition Methodologies

Task Decomposition Methodology	**Category**	**Primary Studies**
Dependability of tasks	• Dependent • Independent	(Jiang & Matsubara, 2014) (Jiang & Matsubara, 2014; Dwarakanath et al., 2015; Weidema et al., 2016)
Characteristics of tasks	• Size • Dimension • Aspects	(Machado et al., 2014) (Mäntylä & Itkonen, 2013) (Zogaj et al., 2014)
Process types	• Dynamic • Iterative • • Aggregated • • Sequential	(Latoza et al., 2014; Zhao & Hoek, 2015) (Luz et al., 2015) (Luz et al., 2015) (Guo et al., 2016)

3.4.10 Assisting Tools for CSE

Assisting tools for testing activities in terms of software testing, collaborative testing and usability testing processes achieved by crowdsourced testing have been increased dramatically in recent years. Moreover, mobile and web-based environments are notices as more appropriate environments for assisting tools for crowdsourced software development, which are shown in Table 3.4.10.1.

Table 3.4.10.1: Tools in Crowdsourced Software Development

Tool Name	**Type**	**Primary Studies**
CrowdTest.me	Testing service	(Machado et al., 2014)
CrowdDesign	Framework	(Weidema et al., 2016)
UCFrame	Web framework	(Hu & Jiau, 2016)
iTest	Framework	(Yan et al., 2014)
CrowdCode	Web application	(Latoza et al., 2014)
Collaborative Testing System	Web application	(Tung & Tseng, 2013)
User Story Mapping	Web-based tool	(Satzger et al., 2014)
MyERP‟s SaaS-ERP System	System	(Groen, 2015)
The CloudTeams Persona Builder	Application	(Alvertis et al., 2016)
Office Robot	Cloud–based tool	(Tsai et al., 2014)
Code Hunt	Web platform	(Xie et al., 2015)
CrowdRex	Framework	(Mao et al., 2015)
Online Store	Web site	(Scheneider & Cheung, 2013)

3.5 Interpretations

Based on our findings, we conclude that various business models have been adapted to CSE. The business model selection is dependent on the clients' preferences in terms of how the crowd is formed, which workers would be assigned to the task, and the maturity of task description. In a competitive model, the task specifications should be clearly defined and related artifacts need to be provided to the crowd. Any worker would like to participate is welcome to join, but it is more difficult to assess the suitability of tasks to workers and quality of the final solution with this model. In a collaborative model, on the other hand, better solutions through continuous collaboration between crowd actors could be achieved. If the clients would like to find the right developers for their tasks, a collaborative model should be preferred over the others. In all these business models, a technical platform to manage and coordinate all the tasks is vital. Selection of these platforms is also important, because they incorporate internally used systems of companies with the development environments, testing tools and bug repositories that are used by the crowd workers. There are popular platforms like TopCoder and AppStori that could be used to run the crowdsourcing

activity, while many studies also offered their own platforms like cloud-based, or enterprise crowdsourcing as they fit to the domain, architecture and development methodology. Studies sharing their experiences on these platforms should provide more insights to the interested parties in the future. In terms of software development methodology, it is clear that CSE requires a unique solution with new process areas that are not available in traditional and agile development. More focus on project planning, requirements engineering and testing processes are needed to manage crowd recruitment, award and schedule estimation, task decomposition, and integration of solutions to have a stable, working product. Companies who develop in an agile fashion with continuous delivery and frequent relationship with the customer require more structured policies and procedures to handle change requests and assess the overall quality of their final product.

It is obvious that an effort estimation approach is mandatory for each project in CSE, similar to typical software development projects. Before determining the offered awards, even before making the decision of crowdsourcing, the client needs to know the approximate cost of its project. There were not many studies reporting effort estimation models in the literature. Those, which examined it concluded that the projects in CSE may be relatively predictable with the existing effort estimation models. We believe that the effort related models are among the research gaps in CSE. We believe strongly that embedding the specific criteria of CSE into these models will increase prediction rates. The complexity of the task and its expected duration are the most frequently employed factors among cost drivers. The quality of the input both involves the completeness of the task and achievement of several quality thresholds. The only distinctive cost driver of CSE is the online reputation. Developers of typical software projects usually does not motivate for getting prestige from their own designs or codes. We believe that the behaviors of developers towards having online reputation may be an independent topic for future research.

As TopCoder has been the commercial pioneer of CSE, and its award mechanism was appeared in public, many studies made use of its awarding approach. CSE, quintessentially, is highly convenient for competitive models, where game theory would be the fundamental technique. In related literature, several game theory-based models

are proposed, however we believe that the number and the scope of them should be extended. We need more case studies to demonstrate the applicability and success of these kinds of economic models. However, it is nearly impossible to access real life data in CSE. Formerly, several academic studies have used TopCoder data; but nowadays we cannot have it. We consider that this is the main challenge behind the evolution of the pricing mechanisms proposed for CSE. Additionally, the effect of task decomposition and/or education level of the crowd may be examined in detail to further integrate to proposed awarding mechanism.

The composition of the crowd is directly related to the quality level of its outputs. Several studies worked with students, since they are the most attainable crowd. Besides the crowd source, the second important factor is its level of training. The quality and the cost of a task are directly influenced from the crowd's education level. Many studies utilized already existing communities via AMK or TopCoder, whereas some studies are conducted among a group of employees. The AMK community and the companies' employees are usually considered among the expert crowds. We believed that the education/training level of crowd should be considered as another cost driver. The effect of the crowd's education level on the software project effort may be an independent research area.

4. PROPOSED AWARD DETERMINATION MODEL

The main motivation behind this thesis study is to introduce an effort estimation approach for crowdsourced projects for competition-based crowdsourced software projects (e.g. Topcoder). Various factors affect effort estimation in competition-based crowdsourcing. For instance, effort may depend on the productivity of the developer/ developer organization, which can be interpreted as the effectively spent time of each contributor in competitions. However, measuring the productivity of each contributor is a challenging task, since we do not know whether he/she spends his/her time effectively in competitions. In other words, time spent by highly concentrated participators is not the same as the less concentrated participators in the competition. In addition, time spent is strongly based on participators' background and qualification. Moreover, the effort may consider the overhead of losing participants in the competitions. The publicly available data is another emerging issue for effort estimation. In this thesis, the research on effort estimation models have been limited, because we could not reach to real life crowdsourcing data. Instead, we have studied another significant process in competition-based crowdsourcing: Award determination. Doing so, we have adapted Putnam model for award determination in contest-based crowdsourcing projects (Liebowitz, 1999). Determining award used in crowdsourced software projects requires analyzing appropriate factors and understanding the metrics related to the development process. We believed that the award of an offered task should be inversely proportional to the duration and platform's productivity, but proportional to the project size and its complexity. As a result, the main contribution of this thesis study is applying recently studied effort estimation method i.e. the Putnam Model to the award determination process for crowdsourcing from a new perspective. We chose the cost drivers used for effort estimation among the ones obtained from the systematic literature review, as shown in Table 3.4.6.1. The expected duration of a task refers to the time in days in our proposed award determination model (Stol & Fitzgerald, 2014; Ågerfalk et al., 2015;

Bruun & Stage, 2015). The input complexity parameter among cost drivers is represented as the scaling factor **(B)** in our proposed model (Archak, 2010; Mao et al., 2013; Stol & Fitzgerald, 2014; Ågerfalk et al., 2015). In addition, online reputation can be expressed as a productivity metric of members of the crowd for future award determination studies (Archak, 2010; Wu et al., 2013). On the other hand, unmeasured quantities of effort and quality of input, which are among the cost drivers in related literature are not considered in our proposed model (Mao et al., 2013: Jiang & Matsubara, 2014; Stol & Fitzgerald, 2014; Zogaj et al., 2014; Weidema et al., 2016).

Another contribution of this thesis study is the software size evaluation via *Function Point Analysis (FPA)*. The size in terms of SLOC is considered as the size parameter in Putnam Model (Khuttan et al., 2014).

Putnam model is introduced for cost estimation, where Eq. (4.1) gives Putnam's equation for software estimation (Liebowitz, 1999):

$$Effort = \left[\frac{Size\ x\ B^{\frac{1}{3}}}{Productivity}\right]^3 x\ \frac{1}{Time^4} \tag{4.1}$$

According to this equation, *effort* is the total effort expanded to complete the project, *size* is the finished size of the project in SLOC, **B** is a scaling factor in terms of project size, that increases with the system size, *productivity* parameter represents the efficiency of the overall development environment for the project and *time* is the length of the time the project takes to complete. We have thought that award can also be estimated using this approach, hence we adapted Eq. (4.1) for award determination to achieve our ultimate goal.

$$Award = \left[\frac{Size}{Productivity\ x\ Time^{\frac{4}{3}}}\right]^3 x\ B \tag{4.2}$$

We have studied award determination by means of Eq. (4.2). *Size* is the software project size, and determined by *Function Point Analysis* in *Line of Code (LOC)*, *productivity* refers to the *process productivity* that is the ability of a particular software organization to produce software of a given size at a particular defect rate. In demonstrative examples, it is taken as a percentage value between 0 and 100. *Time* is the total schedule of the project in days, *B* refers to scaling factor in terms of the project size and award is the total award offered for the particular task/project.

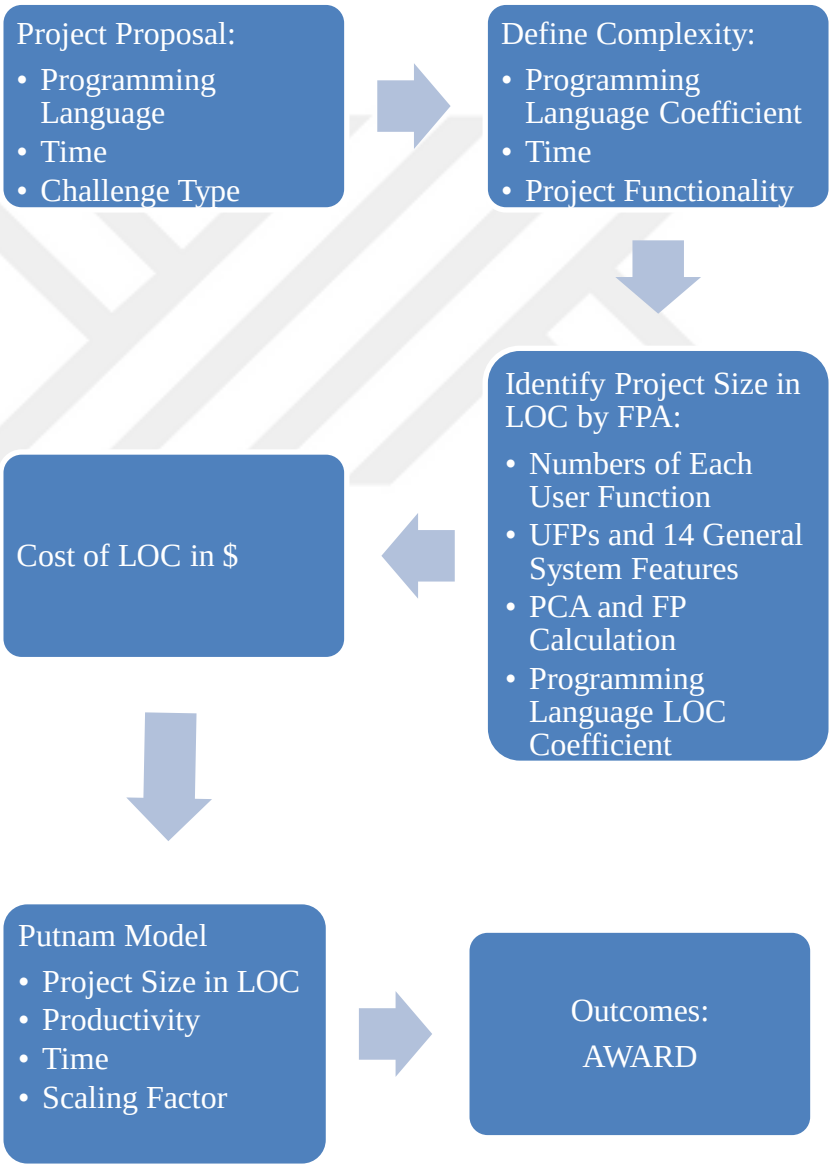

Figure 4.1: Flow Chart of Award Determination Process

According to Figure 4.1, the approach is initialized by defining the project characteristics, which are the programming language, time and project challenge type in order to calculate project complexity for each selected example projects in TopCoder challenges. We have examined programming language coefficient, time and project functionality for determination of project complexity. Then, as the next step, the identification of the project size in LOC is generated by FPA methodology, which is given in detail in Section 2.4. Following the size calculation, the cost of the sample projects are calculated in $[3] (Jones, 2008). In addition, we have studied Putnam model with the project size in LOC, productivity, time and scaling factor for projects with different complexities. Finally, we have deduced the relationships between the parameters of the approach and the defined award.

4.1 Demonstrative Examples

We used recently offered software projects in TopCoder, as our numerical examples. TopCoder has four main types of competitions: *Design Challenges*, *Development Challenges*, *Data Science Challenges* and *Competitive Programming*. Competitive programming is a special competition, where all contestants compete online and solve the same problems with the same deadline. In addition, there is no programming language used in design challenges, which is necessary for us to calculate project size in LOC. Therefore, we have excluded design challenge in this thesis, and examined only three project types in TopCoder for demonstration of the applicability of our proposed model. Each project type has its own parameter values for each challenge. Table 4.1.1 summarizes the parameters, their selected ranges and values for future studies.

[3] http://www.yegor256.com/2014/04/11/cost-of-loc.html

Table 4.1.1: Challenge Types and Used Parameters with Their Selected Values

Competition Type / Parameters	**1st Development Challenge**	**2nd Development Challenge**	**Data Science Challenge**
Size in LOC	3226.08	8371.35	1921.25
FPA	68.64	157.95	36.25
Productivity	[0.4 - 1]	[0.4 - 1]	[0.4 - 1]
Time (days)	[5-50]	[5-50]	[20-50]
B	1.054, 1.120 1.197	1.054, 1.120 1.197	1.054, 1.120 1.197

The first development challenge is a demo development application for understanding effect of weather on sales for different business units and materials selected and the second development challenge is the development of IoT devices management platform. The data science challenge is the identification of propeller objects in Saturn's rings. We have studied our proposed award determination model on these three challenges. In order to analyze applicability of the award determination approach and deduce several outcomes, we have realized sensitivity analysis by incrementing each parameter in respect to the others. The sizes in LOC are determined for two development challenges as 3226.08 and 8371.35, respectively, and for data science challenge as 1921.25. Function points measures are determined for data science project as 36.25, and as 68.64 and 157.95 for two development challenges. All those computations are explained in details in following three subchapters. Productivity parameter is selected as a percentile value, between 0 and 1. In addition, time is chosen in the range of 20 to 50 days, due to short deadlines of proposed projects in TopCoder. The time parameter is increased one by one between 20 and 30, and got the following values: 32, 35, 40, 42, 45, 50. B is determined as 1.054, 1.120 1.197 (Jensen et al., 2006). All those parameters are examined in respect to the purpose of analysis methodologies in our award determination model; therefore, the selected parameters and their values are illustrated in tables before analysis and implementation of the model for each selected project challenges.

4.1.1. Demonstrative Example 1: Development Challenge Project

The first project that is examined in Topcoder is called as *IBM Cognitive-Weather Sales Performance Analytics-Service Proof of Concept*. Developing a demo application for understanding relationship between the effect of weather on sales for different business units and materials via cognitive thinking and technology is the main motivation of this challenge.

According to FPA calculation methodology in subchapter 2.4 and proposed project details in TopCoder, the raw function point worksheet is illustrated as below, Table 4.1.1.1.

The Table 4.1.1.1 shows the raw function point for each user functions and their complexities in order to calculate UFP.

Table 4.1.1.1: The Raw Function Point Worksheet for Development Challenge Project (Low & Jeffery, 1990)

Function Type / Complexity	Simple	Average	Complex	Total
External Input	3x3	x4	x6	9
External Output	x4	x5	1x7=7	7
Logical Internal File	x7	x10	1x15=15	15
External Interface File	2x5=10	1x7=7	x10	17
External Inquiry	x3	1x4=4	6	4

According to Table 4.1.1.1, UFP of this project is 52, and the PC value is determined by values in Table 4.1.1.2.

Table 4.1.1.2: Processing Complexity (PC) for Development Challenge Project (Albrecht & Gaffney, 1983)

General System Features	Degree Of Influence (DI)
1. Data Communication	5
2. Distributed Functions	4
3. Performance	5
4. Heavily Used Configuration	5
5. Transaction Rate	5
6. Online Data Entry	5
7. End User Efficiency	5
8. Online Update	5
9. Complex Processing	5
10. Reuseability	5
11. Installation Ease	4
12. Operational Ease	4
13. Multiple Sites	5
14. Facilitate Change	5

The total DI is 67. Besides, PCA is found 1.32 and FPs are calculated as 68.64. In order to define LOC of this project, the average value of JavaScript programming language is selected (47) for using in Eq. 4.2.4 as programming language LOC coefficient[4]. The overall LOC of this development project is found as 3226.08, which is defined as average complex project for this challenge.

In addition, the proposed award for this challenged project is $1,200 for first place and $600 for second place in TopCoder, respectively. However, the expected cost of 3226.08 LOC is found as $12839.79 by means of multiplication with pure coding cost i.e. $3.98 cost per LOC for JavaScript[5]. Besides, TopCoder's offered award per JavaScript LOC i.e. unit cost per JavaScript LOC offered by TopCoder is determined as 0.371 such that $1,200 is divided by 3226.08 LOC.

[4] http://www.qsm.com/resources/function-point-languages-table
[5] http://www.yegor256.com/2014/04/11/cost-of-loc.html

4.1.2 Demonstrative Example 2: Data Science Challenge Project

The second FPA is examined for the project, which defines propeller objects in Saturn as much as possible by images in *NASAView* program. Considering implementation of this challenge in TopCoder, the raw function point worksheet is illustrated in Table 4.1.2.1.

Table 4.1.2.1: The Raw Function Point Worksheet for Data Science Challenge Project (Low & Jeffery, 1990)

Function Type / Complexity	**Simple**	**Average**	**Complex**	**Total**
External Input	x3	x4	x6	
External Output	x4	x5	1x7=7	7
Logical Internal File	x7	x10	x15	
External Interface File	2x5=10	x7	x10	10
External Inquiry	x3	x4	2x6=12	12

According to Table 4.1.2.1, the UFP is 29. The PC value of this challenge is calculated as 60 by means of total DI values in Table 4.1.2.2. Moreover, PCA value is 1.25, and FPs is measured as 36.25. Java is using as programming language in this project, and its average LOC coefficient value is 53[6].

Lastly, the LOC of this challenge is assigned as 1921.25, which defines this challenge as a project of simple complexity.

[6] http://www.qsm.com/resources/function-point-languages-table

Table 4.1.2.2: Processing Complexity (PC) for Data Science Challenge Project (Albrecht & Gaffney, 1983)

General System Features	**Degree Of Influence (DI)**
1. Data Communication	5
2. Distributed Functions	5
3. Performance	5
4. Heavily Used Configuration	5
5. Transaction Rate	5
6. Online Data Entry	0
7. End User Efficiency	5
8. Online Update	0
9. Complex Processing	5
10. Reusability	5
11. Installation Ease	5
12. Operational Ease	5
13. Multiple Sites	5
14. Facilitate Change	5

There was not a determined award for this challenge in TopCoder. On the other hand, the expected cost of 1921.25 LOC is calculated as $12007.81 by means of multiplication with pure coding cost i.e. $6.25 cost per LOC for Java. Therefore, it is expected that TopCoder offers a lower value than this for award to participants to utilize crowdsourcing methodology. If this project was achieved by JavaScript, the expected award for 1921.25 LOC by multiplication with $3.98 would be $7646.57.

According to the values of expected award, programming language plays significant role on the expected awards for crowdsourcing projects. In this example, we can interpret that, the pure coding code cost for Java would be higher than the one of JavaScript, hence the proposed award would be higher than 0.371 * 1921.25 $.

4.1.3 Demonstrative Example 3: 2nd Development Challenge Project

The last studied project in TopCoder is called: *IoT Hub Consumer*. The main purpose of this challenge is the development of an IoT device management platform. The raw function point worksheet is illustrated in Table 4.1.3.1.

Table 4.1.3.1: The Raw Function Point Worksheet for Second Development Challenge Project (Low & Jeffery, 1990)

Function Type / Complexity	**Simple**	**Average**	**Complex**	**Total**
External Input	x3	x4	3x6=18	18
External Output	x4	x5	3x7=21	21
Logical Internal File	x7	x10	2x15=30	30
External Interface File	x5	x7	3x10=30	30
External Inquiry	x3	x4	3x6=18	18

According to the values in Table 4.1.3.1, UFP of the 2nd development challenge project is 117, and the PC value is defined by accumulating of each 14 general system features, as illustrated in 4.1.3.2, which leads to the total degree of influences.

Table 4.1.3.2: Processing Complexity (PC) for Second Development Challenge Project (Albrecht & Gaffney, 1983)

General System Features	**Degree Of Influence (DI)**
1. Data Communication	5
2. Distributed Functions	5
3. Performance	5
4. Heavily Used Configuration	5
5. Transaction Rate	5
6. Online Data Entry	5
7. End User Efficiency	5
8. Online Update	5
9. Complex Processing	5
10. Reusability	5
11. Installation Ease	5
12. Operational Ease	5
13. Multiple Sites	5
14. Facilitate Change	5

The total DI is calculated as 70, and PCA is defined as 1.35. Moreover, FP is 157.95. The project size in LOC is calculated by the average LOC value of Java programming language (53), and it is 8371.35, which is defined as the most complex project among these challenges.

The proposed award for this challenge is $1,000 for 1st place and $500 for 2nd place. On the other hand, the cost of 8371.35 LOC is calculated as $52320.93 by means of multiplication with pure coding cost i.e. $6.25 cost per LOC for Java. When comparing the effect of programming language on award determination for crowdsourcing projects, the expected award is $33317.97 by multiplication of 8371.35 LOC and JavaScript pure coding cost i.e. $3.98[7]. Those award values state that the award is proposed higher for projects developed by Java programming language than projects developed by JavaScript. In addition, the Java unit award per LOC offered by TopCoder is 0.119 which is calculated by division of 1000 over 8371.35. On the other hand, Java unit award per LOC offered by TopCoder is normally expected higher than the JavaScript unit award offered by TopCoder. However, the popularity of the programming language can affect this inference. Besides, the TopCoder's unit award per Java LOC is very small than pure coding cost of Java i.e. $6.25.

4.2 Sensitivity Analysis

In this subsection, we have examined how the variables vary with each other and the effect of different project complexities on award determination in CSE by using MATLAB functions to visualize these changes. All these outcomes are deduced by using Eq. 4.2.

Table 4.2.1 illustrates selected values for size in respect to measured function point and other selected values for parameters for each selected challenges according to project descriptions on TopCoder. For illustration, there are 26 and 31 registrants for 1st and 2nd development challenges in TopCoder, whereas 338 registrants for data science challenge in which the high number of registrants is assumed to lead to high productivity. Therefore, the productivity for data science challenge is chosen higher than the productivity for development challenges. In addition, the number of registrants and the size of the projects can determine the project complexity, therefore; the scaling factor of 2nd development challenges is defined as 1.197 (scaling factor of a complex project), while the scaling factor for data science challenge is defined as 1.054 (scaling factor of a simple project), and 1st development challenge is defined as 1.120 (scaling

[7] http://www.yegor256.com/2014/04/11/cost-of-loc.html

factor of an average project). The deadlines for two development challenges are the same (5 days), however there was not a predefined deadline for data science challenge in TopCoder, and it is selected randomly as 30 days. In addition, measured function points and size in LOC of all selected demonstrative examples are calculated using the methodology given in previous chapter. Figure 4.2.1 shows relationship between FPs and award in terms of selected values of each parameter according to Table 4.2.1.

Table 4.2.1: Size in Function Points and Other Parameters for Each Selected Challenge

	1st Development Challenge	2nd Development Challenge	Data Science Challenge
Measured FPs	68.64	157.95	36.25
Time	5	5	30
Productivity	0.4	0.6	0.8
B	1.120	1. 197	1.0.54

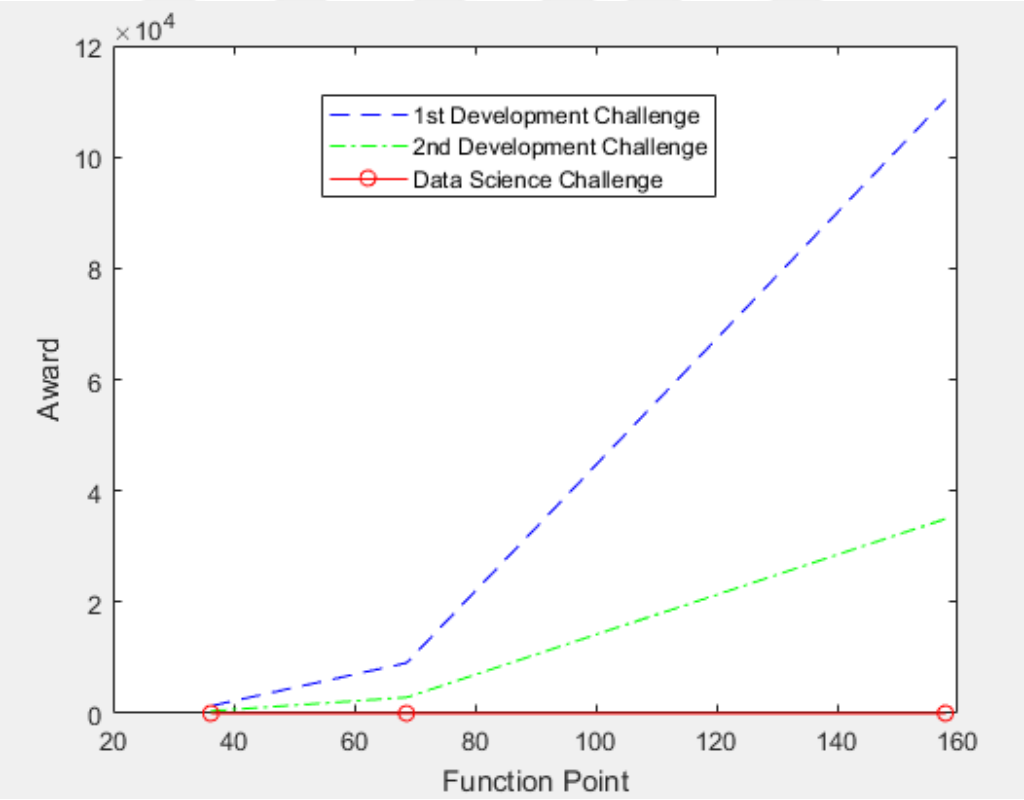

Figure 4.2.1: Function Point vs. Award

Figure 4.2.1 illustrates the variations of award in respect to function points. When the function point is increased, award is also increased. The selected values for each parameter play significant role on award calculation. For instance, the award for data science challenge is the lowest one due to the fact that it has the lowest measured function points. We can observe that, even if the function point increases, time and productivity values are inversely proportional to the award. According to Figure 4.2.1, the highest award value of 1st development challenge is 110335.92 and it is highest than the award value which is calculated in subsection 4.1.1 as $12839.79 and proposed award by TopCoder for this challenge. The highest value of 2nd development challenge is 34939.70, which is lower than our calculated award (i.e. $52320.93 in subsection 4.1.3), but is highest than TopCoder's proposed award. Finally, the highest award value for data science challenge (10.01) is also lower than our calculated award in subsection 4.1.2, which is $12007.81, and it is expected to be lower than the TopCoder's offered award.

According to the proposed award determination model, project size is directly proportional to the award. The highest measured function point leads to the highest award value irrespective of project properties such as productivity, time and scaling factor. In other words, other parameters' values in our award determination model can increase the value of award regardless of complexities of crowdsourcing projects.

Table 4.2.2: Size in LOC and Other Parameters for Selected Development and Data Science Challenges

	1st Development Challenge	**2nd Development Challenge**	**Data Science Challenge**
Measured Size in LOC	3226.08	8371.35	1921.25
Time	5	5	30
Productivity	0.4	0.6	0.8
B	1.120	1.197	1.054

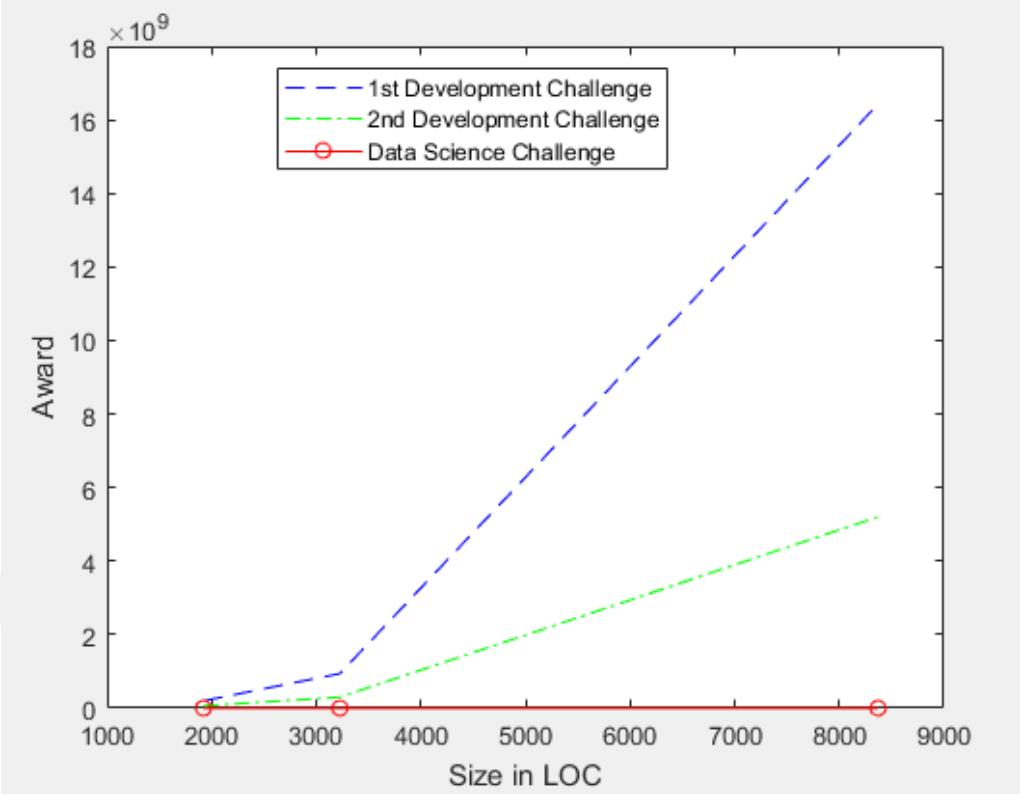

Figure 4.2.2: Size in LOC vs. Award

Figure 4.2.2 shows the relationship between size in LOC and award according to the selected values of each parameter, as seen in Table 4.2.2. For instance, the highest award belongs to the 1st development challenge with a project of 3226.08 LOC. In addition, 1st development challenge has the highest award in all project size in LOC that shows the important effect of selected parameter values. Moreover, the Figure 4.2.2 shows that the highest values for each selected challenges which are $1.64x10^{10}$ 5201718926 and 853740 for respectively 1st development, 2nd development and data science challenge. All those proposed award values are higher than the TopCoder's offered awards for 1st and 2nd development challenges. The award values in Figure 4.2.2 are also higher than the calculated award i.e. $12839.79 for 1st development challenge, $52320.93 for 2nd development challenge according to the pure coding cost of programming languages in demonstrative example subchapters. Besides, the Figure 4.2.2 states that the highest award value for data science challenge is higher than the estimated award by means of pure coding cost i.e. $12007.81 in data science demonstrative example chapter.

Although the project is complex, selected parameter values in our proposed award model can determine low award values, as in seen in Figure 4.2.2. That is to say, 1st development challenge has lower project complexity in terms of scaling factor and productivity; but higher award value than 2nd development challenge.

We have also studied the impact of productivity on award as seen in Table 4.2.3 and Figure 4.2.3.

Table 4.2.3: Productivity and Other Parameters for Selected Development Challenges

	1st Development Challenge	**2nd Development Challenge**	**Data Science Challenge**
Productivity	[0.4, 0.42, 0.44, …., 0.98 1]	[0.4, 0.42, 0.44, …., 0.98 1]	[0.4, 0.42, 0.44, …., 0.98 1]
Measured Size in LOC	3226.08	8371.35	1921.25
Time	5	5	30
B	1.120	1.197	1.054

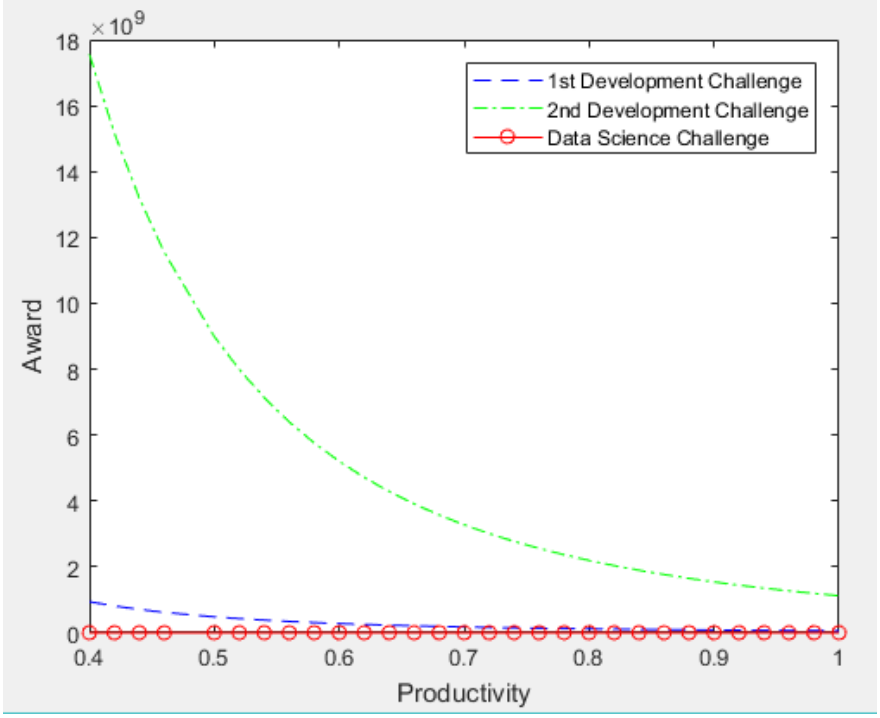

Figure 4.2.3: Productivity vs. Award

The 2nd development challenge with the highest project size in LOC has the highest award in our defined productivity range. In addition, 1st development challenge and data science challenge are observed to have almost the same award values as productivity increases. According to Figure 4.2.3, when productivity is selected as 0.4, the differences between award values are too large for 1st development challenge and 2nd development challenge. This large award gap can be the result of difference of project sizes in LOC of selected projects. Figure 4.2.3 illustrates that the highest award of the 1st development challenge is 940120298.5, the award value of 2nd development challenge 1.75 x 10^{10} and 144187.38 for data science challenge. These maximum award values for each challenge are higher than TopCoder's proposed award values for 1st and 2nd development challenges and our estimated award in demonstrative examples were $12839.79 for the 1st development challenge, $52320.93 for the 2nd development challenge and $12007.81 for the data science challenge. Consequently, higher value for productivity leads to lower award values in CSE. In other words, the award payment can be gradually decreased, when the productivity value is increased steadily for crowdsourcing projects irrespective of their size.

The time also plays significant role on award determination, as illustrated in Table 4.2.4 and Figure 4.2.4.

Table 4.2.4: Time and Other Parameters for Selected Development Challenge

	1st Development Challenge	**2nd Development Challenge**	**Data Science Challenge**
Time	[20, 21, …., 49, 50]	[20, 21, …., 49, 50]	[20, 21, …., 49, 50]
Measured Size in LOC	3226.08	8371.35	1921.25
Productivity	0.4	0.6	0.8
B	1.120	1.197	1.054

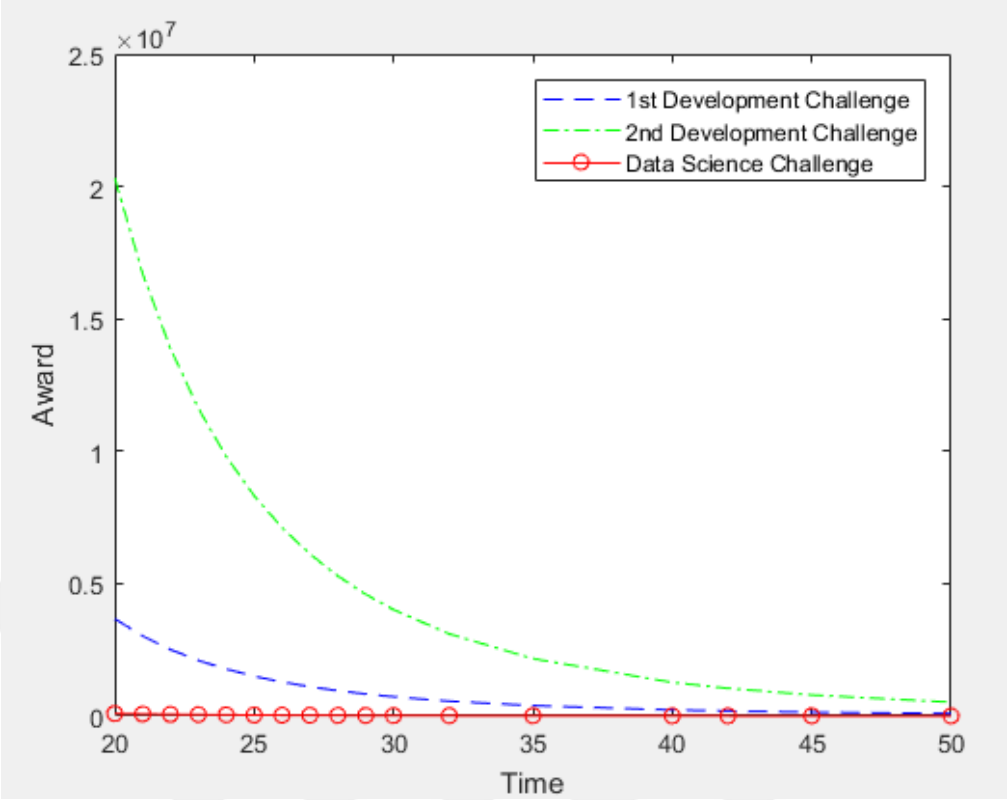

Figure 4.2.4: Time vs. Award

According to Figure 4.2.4, when the time increases, there is a dramatic decrease on the award for 2nd development challenge. Moreover, there is not much difference between award values of 1st development and data science challenges as time increases.

The data science challenge also has the lowest award values, (initial value of 20) compared with 1st and 2nd development challenges. Figure 4.2.4 gives also information about award value for 1st development challenge as 3672344.91, for 2nd development challenge i.e. 20319214.55 and 91243.57 for data science challenge. All these values are higher than our calculated values in demonstrative examples and TopCoder's proposed award values for 1st and 2nd development challenges.

Accordingly, selected higher time interval values for challenges play significant roles on minimum award payment irrespective of complex projects.

The impact of the scaling factor (B) in terms of project size is illustrated Figure 4.2.5. The corresponding parameters are represented in Table 4.2.5.

Table 4.2.5: Scaling Factor and Other Parameters for Selected Development Challenge

	1st Development Challenge	2nd Development Challenge	Data Science Challenge
B	[1.054, 1.120, 1.197]	[1.054, 1.120, 1.197]	[1.054, 1.120, 1.197]
Measured Size in LOC	3226.08	8371.35	1921.25
Productivity	0.4	0.6	0.8
Time	5	5	30

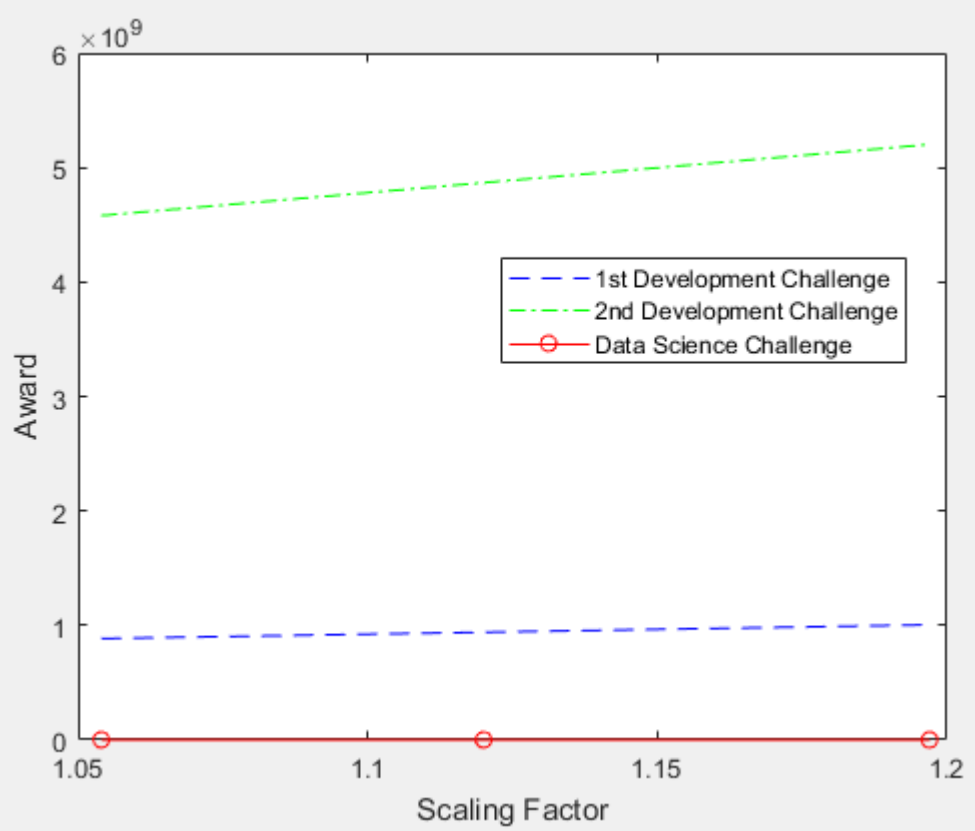

Figure 4.2.5: Scaling Factor vs. Award

There is a linear relationship between scaling factor and award. Figure 4.2.5 emphasizes that there is not significant increase on award values, even if scaling factor increases. 2nd development challenge has the highest award value, whereas data science's award is the lowest in all selected scaling factor values. According to Figure 4.2.5, the maximum award values are 1004753569 for 1st development challenge, 5201718926 for 2nd development challenge and 11720.45 for data science challenge.

However, 1st and 2nd development challenges' award values are higher than the award determined by TopCoder and our estimated award values for these challenges in demonstrative examples were $12839.79 for 1st development challenge and $52320.93 for 2nd development challenge; however data science award value in Figure 4.2.5 is lower than the estimated award value in demonstrative example which was $12007.81 for data science challenge.

As a result, both scaling factor and project complexity can be selected as minimum according to the project properties in order to determine the award with the lowest value for crowdsourcing projects.

We have also studied the relationship between projects size in LOC, productivity, and award. Table 4.2.6 gives the selected parameter values, and Figure 4.2.6 illustrates the variations.

Table 4.2.6: Productivity and Other Parameters' Values

	1st Development Challenge	**2nd Development Challenge**	**Data Science Challenge**
Productivity	[0.4, 0.42, 0.44,…,0.98, 1]	[0.4, 0.42, 0.44,…,0.98, 1]	[0.4, 0.42, 0.44,…,0.98, 1]
Measured Size in LOC	3226.08	8371.35	1921.25
Time	5	5	30
B	1.120	1.197	1.054

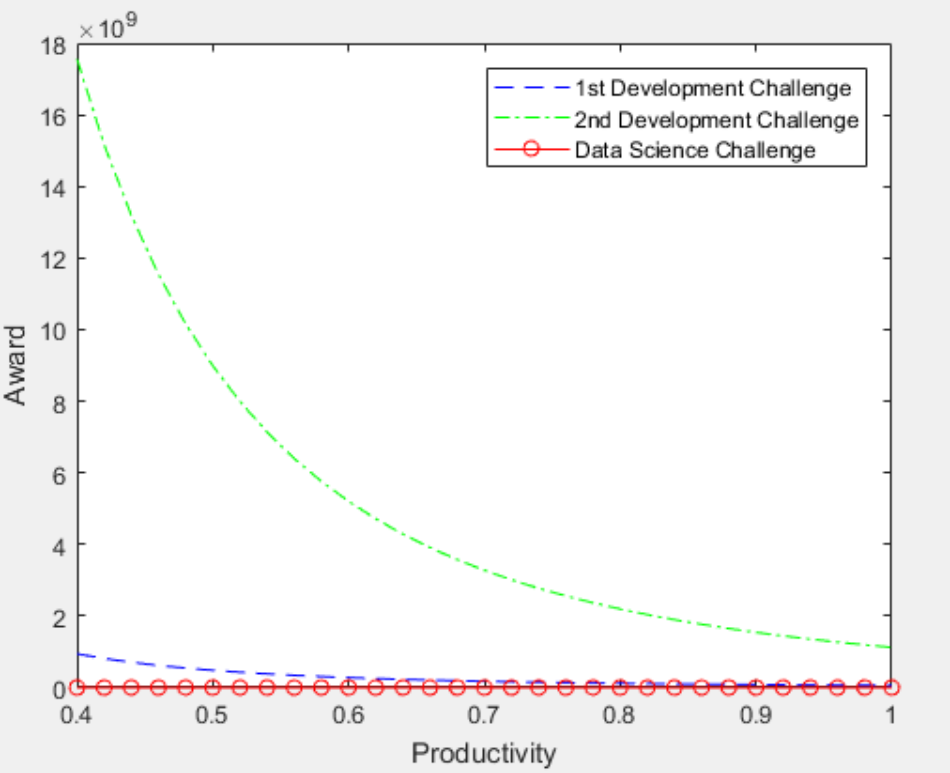

Figure 4.2.6: The Effect of Size and Productivity on Award

Figure 4.2.6 points out that there is the largest difference on award between 2nd development challenge, which has the highest project size in LOC and 1st development and data science challenges, when the productivity is fixed to 0.4. However, this largest difference has decreased dramatically as the productivity value reaches 1. We have finally examined the effect of time and size on award, as shown in Table 4.2.7 and Figure 4.2.7, respectively.

Table 4.2.7: Time and Other Parameters' Values

	1st Development Challenge	**2nd Development Challenge**	**Data Science Challenge**
Time	[20, 21, …., 45, 50]	[20, 21, …., 45, 50]	[20, 21, …., 45, 50]
Measured Size in LOC	3226.08	8371.35	1921.25
Productivity	0.4	0.6	0.8
B	1.120	1.197	1.054

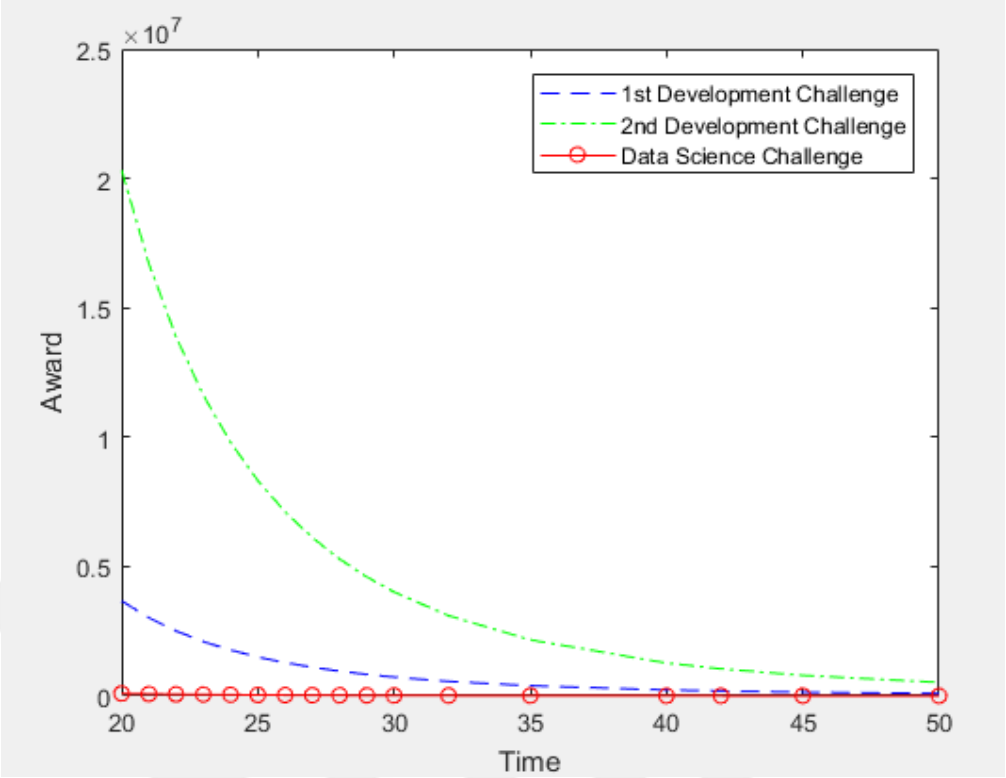

Figure 4.2.7: The Effect of Time and Size on Award

Figure 4.2.7 compares awards of three projects with different sizes in LOC in respect to time. We observe that project size in LOC does not have a significant effect on the award; because of the fact that, the award of 1st, 2nd development challenges and data science challenge are 0.36, 2.03 and 0.09 award/LOC on 20 days, respectively. According to these estimated award values, the award values of highest and lowest project size are almost the same.

5. DISCUSSION AND THREATS TO VALIDITY

We have revealed that there is a linear relationship between measured function points and award. That is to say, as function points are increased, the award payment need also be increased. The effect of size on award shows a similar trend. When the project size increases, the award payment need also be increased. The increase of the scaling factor has a direct influence on the award. However, productivity is inversely proportional with award payment. In addition, award payment is decreased when the time is increased. Moreover, when productivity and size are increased, award is decreased. On the other hand, when size in LOC and time are increased, award payment is also increased. Therefore, we can conclude that the selected values for each parameter play significant role on award determination.

We have encountered some drawbacks about Putnam's SLIM model during our research. Putnam model assumes the size of the software delivered is known or can be estimated accurately (Khuttan et al., 2014). However, uncertainty of the software size estimation leads to inaccurate effort/ award determination. Besides, Putnam's SLIM model is said to be useful for the projects that exceed 70.000 LOC (Keyes, 2002). When considering award determination model, Putnam Model has various equations with various variables, therefore, selecting the appropriate approach with respect to research goals is the most important issue. In addition, there is not a commonly accepted approach on the values of scaling factors. The performance of the proposed award determination equation is strongly related to the selection of values of these parameters. Moreover, determination of the project size using FPA can be considered as subjective. Hence, approaches that are more objective may increase the accuracy of the estimations and the efficiency of the determinations.

6. CONCLUSION

This study first provides comprehensive survey on crowdsourcing in software engineering in terms of its business models, its technological platforms, its practices and applications in software engineering, economic issues as effort estimation and cost drivers, crowd selection strategies, task decomposition methods and assisting tools for crowdsourcing in software engineering. This thesis study proposes a novel evaluation framework to explore future researches of crowdsourcing that require further insights. We suggest that future researches should explore ethical issues and crowd motivation. In addition, the outcomes of our proposed award determination model emphasize that future studies should be done to study efficient award mechanisms.

In addition, the thesis introduces an approach on award determination for competition-based crowdsourced software projects. This approach utilizes the Putnam Model, which is introduced in software engineering literature in order to determine effort of software development. Using Putnam's effort estimation model for crowdsourcing in our study, we have deduced several outcomes. Selected parameters and their values play significant role on task award determination of crowdsourcing projects. That is to say, when the project size in terms of function point or LOC increases, the offered award for this challenge can be low because of high value of productivity and time values. On the other hand, the award values decrease dramatically, when the productivity and time values increase. In addition, award is directly affected by project complexity, when considering constant productivity and time factors for crowdsourcing projects. There is a linear relationship between scaling factor and the award, which is deduced from Putnam's effort equation. In other words, the higher project size and scaling factor values lead to higher offered award. These perspectives lead to the question of how to determine award in a most appropriate way for crowdsourcing projects. We can conclude that Putnam model is applicable to CSE as an award determination approach.

However, it is an introductive approach, which does not consider online reputation or project size popularity. The proposed award model will be expanded to become a more comprehensive pricing model for using in CSE.

REFERENCES

Ågerfalk, P. J., Fitzgerald, B., & Stol, K. J. (2015). Software sourcing in the age of open: Leveraging the unknown workforce, Springer.

A lbrecht, A. J. (1979). Measuring application development productivity, *IBM Applications Development Symposium*, pp. 83.

Albrecht, A. J., Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: a software science validation., *IEEE Transactions on Software Engineering* : (6) : 639-648.

Aletdinova, A., Kravchenko, M., & Bakaev, M. (2016). Crowdsourcing and the effectiveness of C2G interaction in Russia, *In Proceedings of the International Conference on Electronic Governance and Open Society: Challenges in Eurasia*, ACM, pp. 202-211.

Aljahdali, S., & Sheta, A. F. (2010). Software effort estimation by tuning COOCMO model parameters using differential evolution, *2010 IEEE/ACS International Conference on Computer Systems and Applications*, AICCSA, IEEE, pp. 1-6.

Alvertis, I., Papaspyros, D., Koussouris, S., Mouzakitis, S., & Askounis, D. (2016). Using crowdsourced and anonymized personas in the requirements elicitation and software development phases of software engineering, *2016 11th International Conference on Availability, Reliability and Security*, ARES, IEEE, pp. 851-856.

Anselmo, D., & Ledgard, H. (2003). Measuring productivity in the software industry., *Communications of the ACM* : 46(11) : 121-125.

Archak, N. (2010). Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on TopCoder.com, *In Proceedings of the 19th International Conference on World Wide Web*, WWW'10, ACM, pp. 21-30.

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). How reuse influences productivity in object-oriented systems., *Communications of the ACM* : 39(10) : 104-116.

Blackburn, J. D., Scudder, G. D., & Van Wassenhove, L. N. (1996). Improving speed and productivity of software development: a global survey of software developers., *IEEE Transactions on Software Engineering* : 22(12) : 875-885.

Boehm, .B. (1981). Software engineering economics, Englewood Cliffs, NJ, Prentice-Hall.

Borandag, E., Yucalar, F., Sahinaslan, Ö. Yazılım projelerinde büyüklük tahmini, *Akdeniz Bilisim Konferansı*, pp. 185-189

Boudreau, K. J., & Lakhani, K. R. (2013). Using the crowd as an innovation partner., *Harvard Business Review* : 91(4) : 60-69.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology., *Qualitative Research in Psychology* : 3(2) : 77-101.

Briand, L. C., El Emam, K., & Bomarius, F. (1998). COBRA: a hybrid method for software cost estimation, benchmarking, and risk assessment, *Proceedings of the 1998 International Conference on Software Engineering*, IEEE, pp. 390-399.

Bruun, A., & Stage, J. (2015). New approaches to usability evaluation in software development: Barefoot and crowdsourcing., *Journal of Systems and Software* : 105: 40-53.

Card, D. N. (2006). The challenge of productivity measurement, *In Pacific Northwest Software Quality Conference*, pp. 1-10.

Collofello, J. S., Woodfield, S. N., & Gibbs, N. E. (1983). Software productivity measurement, *In Proceedings of the National Computer Conference*, ACM, pp. 757-762.

de Aquino Júnior, G. S., & de Lemos Meira, S. R. (2009). Towards effective productivity measurement in software projects, *Fourth International Conference on Software Engineering Advances*, ICSEA'09 , IEEE, pp. 241-249.

Duncan, A. S. (1988). Software development productivity tools and metrics, *In Proceedings of the 10th International Conference on Software Engineering*, IEEE Computer Society Press, pp. 41-48.

Dwarakanath, A., Chintala, U., Virdi, G., Kass, A., Chandran, A., Sengupta, S., & Paul, S. (2015). CrowdBuild: a methodology for enterprise software development using crowdsourcing, *In Proceedings of the Second International Workshop on CrowdSourcing in Software Engineering*, IEEE Press, pp. 8-14.

Dwarakanath, A., Shrikanth, N. C., Abhinav, K., & Kass, A. (2016). Trustworthiness in enterprise crowdsourcing: a taxonomy & evidence from data, *IEEE/ACM International Conference on Software Engineering Companion*, ICSE-C, pp. 41-50.

Edgeman, R., Engell, T., Jensen, N. G., Vrtik, M., Eskildsen, J., & Tambo, T. (2015). Structured crowdsourcing: A B2B innovation roadmap, *10th European Conference on Innovation and Entrepreneurship*, ECIE, pp. 165-174.

Gefen, D., Gefen, G., & Carmel, E. (2016). How project description length and expected duration affect bidding and project success in crowdsourcing software development., *Journal of Systems and Software* : 116 : 75-84.

Groen, E. C. (2015, August). Crowd out the competition, *1st International Workshop on Crowd-Based Requirements Engineering*, CrowdRE, IEEE, pp. 13-18.

Guo, S., Chen, R., & Li, H. (2016). A real-time collaborative testing approach for web application: Via multi-tasks matching, *2016 IEEE International Conference on Software Quality, Reliability and Security Companion*, QRS-C, IEEE, pp. 61-68.

Han, Y., Wu, X., & Yue, C. (2005). Optimizing financial budget for software implementation based on the development effort and cost function., *Advances in Engineering Software* : 36(10) : 699-706.

Hasteer, N., Bansal, A., & Murthy, B. K. (2015). Crowdsourced software development process: Investigation and modeling through Markov decision theory., *International Journal of Software Engineering and Its Applications* : 9(9) : 41-54.

Hosseini, M., Shahri, A., Phalp, K., & Ali, R. (2015). Recommendations on adapting crowdsourcing to problem types, *9th International Conference on Research Challenges in Information Science*, RCIS'15, IEEE, pp. 423-433.

Howe, J. (2006). The rise of crowdsourcing., *Wired Magazine* : 14(6) : 1-4.

Hu, W. C., & Jiau, H. C. (2016). UCFrame: A use case framework for crowd-centric requirement acquisition., *ACM SIGSOFT Software Engineering Notes* : 41(2) : 1-13.

Hu, Z., & Wu, W. (2014). A game theoretic model of software crowdsourcing, *8th International Symposium on Service Oriented System Engineering*, SOSE, IEEE, pp. 446-453.

Jiang, H., & Matsubara, S. (2014). Efficient task decomposition in crowdsourcing, *In International Conference on Principles and Practice of Multi-Agent Systems*, Springer, pp. 65-73.

Jones, C. (2008). A short history of lines of code (LOC) metrics., *Capers Jones & Associates LLC : Narragansett* : 1-12.

Keele, S. (2007). Guidelines for performing systematic literature reviews in software engineering., *In Technical Report*, Ver. 2.3, EBSE Technical Report.

Keyes, J. (2002). Software engineering handbook, CRC Press.

Khuttan, A., Kumar, A., Singh, A. (2014). A survey of effort estimation techniques for the software development., *International Journal Of Scientific & Technology* Research : 3(7).

Kitchenham, B., & Taylor, N. R. (1984). Software cost models., *ICL Technical Journal* : 4(1) : 73-102.

Kitchenham, B. (2004a). Procedures for performing systematic reviews., *Keele : UK : Keele University :* 33 : 1-26.

Kitchenham, B., & Mendes, E. (2004b). Software productivity measurement using multiple size measures., *IEEE Transactions on Software Engineering* : 30(12) : 1023-1035.

LaToza, T. D., Towne, W. B., Adriano, C. M., & Van Der Hoek, A. (2014). Microtask programming: Building software with a crowd, *In Proceedings of the 27th annual ACM symposium on User interface software and technology*, ACM, pp. 43-54.

LaToza, T. D., Chen, M., Jiang, L., Zhao, M., & Van Der Hoek, A. (2015). Borrowing from the crowd: A study of recombination in software design competitions, *In Proceedings of the 37th International Conference on Software Engineering*, Volume 1, IEEE Press, pp. 551-562.

LaToza, T. D., & Hoek, A. van der (2016). Crowdsourcing in software engineering: Models, motivations, and challenges., *IEEE software* : 33(1) : 74-80.

Leung, H., & Fan, Z. (2002). Software cost estimation., *Handbook of Software Engineering* : *Hong Kong Polytechnic University* : 1-14.

Li, W., Tsai, W. T., & Wu, W. (2015). Crowdsourcing for large-scale software development, *In Crowdsourcing*, Springer Berlin Heidelberg, pp. 3-23.

Liebowitz, J. (1999). Knowledge management handbook. CRC press.

Low, G. C., Jeffery, D. R. (1990). Function points in the estimation and evaluation of the software process., *IEEE Transactions on Software Engineering* : 16(1) : 64-71.

Luz, N., Silva, N., & Novais, P. (2015). A survey of task-oriented crowdsourcing., *The Artificial Intelligence Review* : 44(2) : 187.

MacCormack, A., Kemerer, C. F., Cusumano, M., & Crandall, B. (2003). Trade-offs between productivity and quality in selecting software development practices., *IEEE Software* : 20(5) : 78-85.

Machado, L., Pereira, G., Prikladnicki, R., Carmel, E., & de Souza, C. R. (2014). Crowdsourcing in the Brazilian IT industry: what we know and what we don't know, *In Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, ACM, pp. 7-12.

Mäntylä, M. V., & Itkonen, J. (2013). More testers–the effect of crowd size and time restriction in software testing., *Information and Software Technology* : 55(6) : 986-1003.

Mao, K., Yang, Y., Li, M., & Harman, M. (2013). Pricing crowdsourcing-based software development tasks, *In Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, pp. 1205-1208.

Mao, K., Yang, Y., Wang, Q., Jia, Y., & Harman, M. (2015). Developer recommendation for crowdsourced software development tasks, *Symposium on Service-Oriented System Engineering*, SOSE, IEEE, pp. 347-356.

Mao, K., Capra, L., Harman, M., & Jia, Y. (2017). A survey of the use of crowdsourcing in software engineering., *Journal of Systems and Software* : 126 : 57-84.

Matson, J. E., Barrett, B. E., & Mellichamp, J. M. (1994). Software development cost estimation using function points., *IEEE Transactions on Software Engineering* : 20(4) : 275-287.

Model, F. O. (2006). Software estimating models: Three viewpoints.

Moser, S., & Nierstrasz, O. (1996). The effect of object-oriented frameworks on developer productivity., *Computer* : 29(9) : 45-51.

Moses, J., Farrow, M., Parrington, N., & Smith, P. (2006). A productivity benchmarking case study using Bayesian credible intervals., *Software Quality Journal* : 14(1) : 37-52.

Nag, S., Heffan, I., Saenz-Otero, A., & Lydon, M. (2012). SPHERES zero robotics software development: Lessons on crowdsourcing and collaborative competition, *In Aerospace Conference*, IEEE, pp. 1-17.

Naik, N. (2016). Crowdsourcing, open-sourcing, outsourcing and insourcing software development: A comparative analysis, *2016 IEEE Symposium on Service-Oriented System Engineering*, SOSE, IEEE, pp. 380-385.

Parr, F. N. (1980). An alternative to the Rayleigh curve model for software development effort., *IEEE Transactions on Software Engineering* : (3) : 291-296.

Pillai, K., & Nair, V. S. (1997). A model for software development effort and cost estimation., *IEEE Transactions on Software Engineering* : 23(8) : 485-497.

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

Ramakrishnan, S., & Srinivasaraghavan, V. (2014). Delivering software projects using captive university crowd, *In Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, ACM, pp. 115-118.

Saikia, G., Shivagunde, S., Saradhi, V. V., Kannao, R. D., & Guha, P. (2016). Multiple kernel learning using data envelopment analysis and feature vector selection and projection, *23rd International Conference on Pattern Recognition*, ICPR , IEEE, pp. 520-524.

Saremi, R. L., & Yang, Y. (2015). Dynamic simulation of software workers and task completion, *In Proceedings of the Second International Workshop on CrowdSourcing in Software Engineering*, IEEE, pp. 17-23.

Satzger, B., Zabolotnyi, R., Dustdar, S., Wild, S., Gaedke, M., Göbel, S., & Nestler, T. (2014). Toward collaborative software engineering leveraging the crowd, *Economics-Driven Software Architecture*, Elsevier, pp. 159-182.

Schneider, C., & Cheung, T. (2013). The power of the crowd: Performing usability testing using an on-demand workforce, *In Information Systems Development*, Springer, pp. 551-560.

Scupola, A., & Nicolajsen, H. W. (2014). The impact of enterprise crowdsourcing on company innovation culture: The case of an engineering consultancy, *In Scandinavian Conference on Information Systems*, Springer International Publishing, pp. 105-120.

Stensrud, E., & Myrtveit, I. (2003). Identifying high performance ERP projects., *IEEE Transactions on Software Engineering* : 29(5) : 398-416.

Stol, K. J., & Fitzgerald, B. (2014). Two's company, three's a crowd: a case study of crowdsourcing software development, *In Proceedings of the 36th International Conference on Software Engineering*, ACM, pp. 187-198.

Tsai, W. T., Wu, W., & Huhns, M. N. (2014). Cloud-based software crowdsourcing., *IEEE Internet Computing* : 18(3) : 78-83.

Tung, Y. H., & Tseng, S. S. (2013). A novel approach to collaborative testing in a crowdsourcing environment., *Journal of Systems and Software* : 86(8) : 2143-2153.

Vukovic, M. (2009). Crowdsourcing for enterprises, *2009 World Conference on Services-I*, IEEE, pp. 686-692.

Vukovic, M., & Das, R. (2013). Decision making in enterprise crowdsourcing services, *In International Conference on Service-Oriented Computing*, Springer, pp. 624-638.

Warburton, R. D. H. (1983). Managing and predicting the costs of real-time software., *IEEE Transactions on Software Engineering* : (5) :562-569.

Weidema, E. R., López, C., Nayebaziz, S., Spanghero, F., & van der Hoek, A. (2016). Toward microtask crowdsourcing software design work, *IEEE/ACM 3rd International Workshop on CrowdSourcing in Software Engineering*, CSI-SE, IEEE, pp. 41-44.

Wu, W., Tsai, W. T., & Li, W. (2013). An evaluation framework for software crowdsourcing., *Frontiers of Computer Science* : 7(5) : 694-709.

Wu, W., Tsai, W. T., Hu, Z., & Wu, Y. (2015). Towards a game theoretical model for software crowdsourcing processes, *In Crowdsourcing*, Springer Berlin Heidelberg, pp. 143-161.

Xiao, L., & Paik, H. Y. (2014). Supporting complex work in crowdsourcing platforms: A view from service-oriented computing, *23rd Australian Software Engineering Conference*, ASWEC, IEEE, pp. 11-14.

Xie, T., Bishop, J., Horspool, R. N., Tillmann, N., & de Halleux, J. (2015). Crowdsourcing code and process via code hunt, *IEEE/ACM 2nd International Workshop on CrowdSourcing in Software Engineering*, CSI-SE, pp. 15-16.

Xu, X. L., & Wang, Y. (2014). Crowdsourcing software development process study on ultra-large-scale system, *In Advanced Materials Research*, Vol. 989 of *Trans Tech Publications*, pp. 4441-4446.

Xu, X., Wu, W., Wang, Y., & Wu, Y. (2015). Software crowdsourcing for developing Software-as-a-Service., *Frontiers of Computer Science* : 9(4) : 554-565.

Yakushin, D., & Lee, J. H. (2014). Cooperative robot software development through the internet, *2014 IEEE/SICE International Symposium on System Integration*, SICE, IEEE, pp. 577-582.

Yan, J., & Wang, X. (2013). From open source to commercial software development-the community based software development model.

Yan, M., Sun, H., & Liu, X. (2014). iTest: testing software with mobile crowdsourcing, *In Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, ACM, pp. 19-24.

Yu, W. D., Smith, D. P., & Huang, S. T. (1991). Software productivity measurements, *Proceedings of the Fifteenth Annual International In Computer Software and Applications Conference*, COMPSAC'91, IEEE, pp. 558-564.

Zhao, M., & van der Hoek, A. (2015). A brief perspective on microtask crowdsourcing workflows for interface design, *In Proceedings of the Second International Workshop on CrowdSourcing in Software Engineering*, IEEE Press, pp. 45-46.

Zogaj, S., Bretschneider, U., & Leimeister, J. M. (2014). Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary.

BIOGRAPHICAL SKETCH

ASLI SARI
Istanbul, 1990.
GALATASARAY UNIVERSITY, Istanbul, Turkey
M.Sc., Computer Engineering, July 2017 (Expected). Advisor: Gülfem Işıklar Alptekin
BAHÇEŞEHİR UNIVERSITY, Istanbul, Turkey
B.Sc., (Summa Cum Laude) Computer Engineering, June 2014.

AWARDS

2014-Highest 2nd Ranked Student at Bahçeşehir University
2014-Highest Ranked Student Engineering Department at Bahçeşehir University
2014-Highest Ranked Student Computer Engineering Department at Bahçeşehir University

PUBLICATIONS

- A. Sarı, A.Tosun and G. Işıklar Alptekin, 2017, "A Systematic Literature Review on Crowdsourcing in Software Engineering", ACM IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2017). Submitted full text article.
- A. Sarı and G. Işıklar Alptekin, 2017, "A Survey on Crowdsourcing Approach", 21st Conference of the International Federation of Operational Research Societies (IFORS 2017). Accepted abstract.
- A. Sarı and G. Işıklar Alptekin, 2017, "An Overview of Crowdsourcing Concepts in Software Engineering", 8th International Conference on Applied Informatics and Computing Theory (AICT'17). Accepted full text article.