Notes on Turkish Software Industry: Developer Participation and Effort Estimation

Ekrem Kocaguneli
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26505, USA
ekocagun@mix.wvu.edu

Bora Caglayan, Ayse Tosun
Computer Engineering Department
Bogazici University
34342 Bebek, Istanbul, TURKEY
bora.caglayan@boun.edu.tr, ayse.tosun@boun.edu.tr

Ayse Bener
Ted Rogers School of Information Technology Management
Ryerson University
Toronto, CANADA
ayse.bener@ryerson.ca

ABSTRACT

Software effort estimation is critical for resource allocation and planning. Accurate estimates enable managers to distribute the workload among resources in a balanced manner. Organizations use different project management tools. However, the actual workload of developers may be different from the values observed in such tools. In this research, we provide an overview of effort estimation activities in Turkish software companies operating in various domains. We analyse the developer workload through churn data. As a result, we observe that in Turkish software industry, effort estimation is viewed as an important topic. However, there is a big room for research to transfer the ad-hoc methods employed to empirical ones. Furthermore, we observe that resource allocation after estimates does not conform to actual developer workloads. The common characteristic of developer contribution in different projects is: More than 80% of edits in code are performed by a small number of developers.

1. INTRODUCTION

Being able to estimate the development effort of a software project accurately is an important indicator of mature software development processes. Software effort estimation as well as resource planning and monitoring has been addressed in various contexts [3, 11, 13, 17]. A considerable number of the studies have used publicly available datasets in their experimental settings [11, 13, 17]. These results from publicly available datasets have provided a global picture of two issues: Software effort estimation and resource planning.

In our research, we tackle the two problems in a local context and provide a local picture of Turkish software industry. In terms of effort estimation we provide details about the estimation models that are used by companies in Turkey. As for the resource planning, we focus on programmer related activities and evaluate programmer contribution. Our research includes a wide range of companies from various domains: Banking, telecommunication as well as white good companies (embedded software development units). When evaluating our results, we compare them to findings coming from open source projects. Our purpose in that is to see whether there are any similarities between development activities in open source and closed company settings.

As a result of our research we have seen that the importance of software effort estimation as well as resource planning is well understood in Turkish software industry. Companies have their own ad-hoc models of dealing with effort estimation as well as resource planning. For effort estimation companies use linear regression models, a form of expert judgement or a combination of the two. The programmer activities and workload distribution is monitored and managed through third party or in-house developed project management tools.

Although both problems are tackled by companies, there is some discomfort in their implementation. The parameters of regression models used for software effort estimation are determined by humans. Estimation include features identified by experts and weights assigned to them also by experts. However, as the characteristics of the projects change over time and number of factors that are important in software effort increase, human judgement becomes less effective and the linear models based on human expert judgement become obsolete. One clarification we need to make is that, this way of model building is not a form of estimation through expert judgement. Estimation through expert judgement entails experts discussing and estimating the actual effort, whereas here experts only specify parameters of linear model [7].

Another problem regarding the monitoring of developer participation through project management tools is that, most of the time everyone completes similar work hours through the tool, whereas some developers are believed to produce more than the others. The fact that some developers producing more than the others is not a new concept. Koch has conducted an extensive case study in which he has questioned the manpower distribution on open source projects and has come up with the conclusion that commercial projects and open source projects share common characteristics [10]. In both commercial and open source projects, the manpower distribution and corresponding production per developer in terms of static code attributes like lines of code (LOC) show different characteristics. In Turkish software industry, we have seen similar results among the projects we have analysed. Although workload is evenly distributed to developers, more than 80% of all source code development is performed by less than 20% of the developers.

Our dataset is composed of six projects coming from multiple software companies from different domains in Turkey. For our analysis we make use of churn analysis as well as static code attributes. Since we collect our data from private companies, the proprietary rights is a big concern. We analyse the churn data of all the projects in our research and give the necessary information regarding our analysis. We also use these projects as the basis of software effort estimation related interviews. However, project names, company names as well as the developer names will be kept anonymous due to proprietary rights.

To guide us in the right direction we have formed three research questions. With these research questions our aim was to question how widely software effort estimation practices were employed in Turkish software industry and what was the implications of all the estimation and planning activities on the developer participation. The three research questions we formed are:

RQ1 How widely is effort estimation practices employed?

RQ2 Which effort estimation methods are used?

RQ3 How does estimation and planning activities reflect on developer participation?

The rest of the paper is organized as follows: In Section 2 we provide background information regarding software effort estimation modelling as well as case studies focusing on programmer participation, in Section 3 we give the details of our datasets. Then we continue with Section 4 in which we provide the methodology we adopted in this research. Section 5 provides the results we elicited from our analysis. In Section 6 we identify the potential threats to the validity of our research. Finally we conclude our research in Section 7.

2. BACKGROUND

In this research we present a snapshot of Turkish software industry in two parts: 1) Effort estimation methods currently adopted and 2) activity in terms of developer participation. The first part is be about our observations that we elicited during various projects with the industry. The second part is closer to a case study of developer participation in Turkish software industry and our conclusions will be based on churn analysis of various different projects coming from different software companies in Turkey. Therefore, in this section we provide some background information for both parts. In Section 2.1 we present some background information regarding software effort estimation and in Section 2.2 we present information regarding case studies.

2.1 Software Effort Estimation

We can categorize software effort estimation into two groups [16]: Expert judgement-based techniques and model-based techniques. Former approach entails experts discussing the effort of a new project and reaching a consensus on the estimated value, whereas the latter includes the use of a model that calculates the estimated value. Model-based methods may also use expert opinion for parameter specification etc. however, the estimated value is indicated by the model itself and not the expert.

A very widely used technique in software effort estimation domain is expert judgement [6]. The employment of expert judgment methods may be either through well defined methods like Delphi [2] or via ad-hoc methods like formal meetings between experts in an organization. Although expert judgement methods are highly regarded in various settings, they have their own pitfalls. To begin with, the application of expert judgement methods may create an atmosphere of competing interests. The competing interests may be between different departments of an organization or between experts with different levels of expertise. In the former case, the urgent needs of a particular department may result earlier deadlines for the project, which would in return affect the allocation of many resources and which eventually may challenge or even fail the project. In the latter case, estimations of a senior expert may be more dominant to those of a junior expert regardless of their accuracy. Secondly, human experts are evaluated as poor in terms of improving their estimation skills [7]. Lastly, an organization willing to completely rely on expert judgement methods for software effort estimation would keep in mind that lack of good experts will result in imprecise estimations. Furthermore bad estimations would affect critical decisions regarding whether or not to start new projects or when to cancel challenged projects.

Unlike expert judgement-based techniques, model-based methods rely on parametric or algorithmic models. Parametric methods adapt an expert-proposed model to local data. Therefore, for the use of parametric methods collection of local data is a must. One of the most widely known example to parametric methods is the COCOMO method proposed by Boehm [3]. Algorithmic methods are useful when local data does not conform to the specifications of a parametric method. Algorithmic methods employ various algorithms to evaluate local data, build a model and make an estimation. Some well known examples to algorithmic methods are linear regression, case based reasoning systems, neural nets and model trees [9, 11, 13, 17].

The latter approach is useful in the case where local data does not conform to the specifications of the expert's method. A few examples of induced prediction systems are linear regression, neural nets, model trees and analogies [13, 17]. These methods have been mostly aided with certain types of patches such as noise removal, instance selection, feature selection or by feature reduction. Depending on the fundamental assumptions of each algorithm, a different patch can be selected.

Certain organizations may choose to use an expert judgement and some other organizations may prefer to go with a model-based approach, or alternatively a combination of the two in different settings. Regardless of the adopted method, the goal of any estimation model is to attain high estimation accuracy values.

2.2 Type of Research

According to two-dimension classification scheme proposed by Basili et. al. [1], our research falls into the class of blocked subject-project studies, which examine objects across a set of teams and a set of projects. To increase the granularity of this classification we can say that our research is a case study of effort estimation practices and developer contribution in Turkish software industry. Any study can be a formal experiment, a case study or a survey depending how many teams and how many projects are considered in this study and what the scale of the study is [8]. Due to the nature of formal experiments, they are limited in size and need careful control. Case studies focus on what happens on a typical project and surveys on the other hand focus on what is happening over various types of projects. In our research we deal with different companies and different projects that were developed with different languages. Furthermore, we are more of an observer than an experimenter in our research. This is due to the fact that rather than being able to experiment the response of projects as well as project teams in response to a changing factor, we merely observed the effort estimation practices and developer contributions across multiple projects and multiple companies. From our reading of Kitchenham et. al. [8] our research conforms to the definition of a case study.

Different case studies regarding both effort estimation practices and developer participations have been conducted in different settings. For example Jorgensen et. al. addresses the effort estimation practices and budget overruns in Norwegian software industry [14]. Koch questions effort modelling as well as developer participation over open source software projects in his extensive survey [10]. Although similar case studies have been conducted by different researchers in different contexts previously, to the best of our knowledge there is no previous study evaluating the effort estimation practices and developer participation in the context of the Turkish software industry.

3. DATA

In our research we analyse 6 different projects coming from different companies. Both in terms of their development languages and in terms of their magnitude they are very different from one another. Ideally the suggested and practised strategy for selecting projects in case studies is random sampling [8, 14]. However, private organizations are not completely optimistic about sharing churn data and static code attribute information of all their projects. Therefore, in our research we have asked them to provide us with representative projects of their software development activities. This may have introduced a selection bias into our study and we will address this issue as threat to validity more broadly in Section 6.

The project names and their particular characteristics are provided in Figure 1. Due to proprietary rights, we will name the projects and developers with numbers. The properties of projects given Figure 1 are number of developers employed, total edited lines of code (*total edited LOC = added LOC + deleted LOC*), total number of commits that were performed during the lifecycle of the project and the language in which the project was developed. We see from Figure 1 that our sample set contains a wide variety of projects in terms the features given in the same Figure. The smallest project in terms of size (Project1) employs only 3 developers and a total of 5579 LOC were edited for this project. The biggest project in terms of size (Project3) on the other hand employs 98 developers and includes a total of 1324956 edited LOC. Therefore, although we were unable to apply random selection in our research, the organization selected representative projects cover projects with quite different characteristics.

4. METHODOLOGY

In this section we provide the details about the methodology we adopted and present our research questions that guided us in this research.

4.1 Estimation Models

Our method for observing the estimation models employed in organizations is to conduct various meetings with domain experts in these organizations. The aim of those meetings were two-fold: 1) Understanding the company practices and needs then 2) forming a questionnaire. The initial meetings consisted of half an hour to one hour individual interviews. Those meetings aimed at getting to know the companies, learning their problems as well as the aspects they would like to improve in their current estimation practices. We also wanted to interview a wide spectrum of positions in the company to get a grasp of different perspectives. For example in one of the reported companies, we have interviewed 2 senior executives, 5 analysts, 3 architects and 2 developers. The questionnaire that formed eventually pinpoints key aspects of software effort estimation in those companies. The domain experts to whom we have conducted this questionnaire are responsible for software effort estimation related activities or for the effort estimation model (if organization has any). Questionnaire consists of 25 questions. A small sample of the questionnaire is as follows:

- What kind of an output do you expect from an estimation model?
- What kind of an estimation do you perform in your projects?
- What kind of metrics do you extract from your project? And how do you evaluate them?

To be parsimonious in space, we do not provide all the questions and their possible answers here (for further information please feel free to contact any of the authors).

4.2 Metrics for Developer Participation

We analysed churn data to understand developer participation. From churn data analysis we first extracted the commit information of projects from source code control systems. Since different companies use different types of source control systems, this process takes some time. For each source

Project	Total # of developers	Total edited LOC	Total # of commits	Dev. Language
Project1	3	5,579	72	Java
Project2	110	623,173	12,384	Java+JSP
Project3	97	1,324,956	23,403	PL/SQL
Project4	7	7,034	212	Java
Project5	11	16,358	322	Java+JSP
Project6	19	68,550	2,387	C

Figure 1: 6 projects coming from different organizations. Projects have a wide diversity in terms of their size, developers they use and the languages they were developed in.

control system, we wrote scripts to extract the commit information.

In this research we extracted the commit information associated with each developer for each project. For each developer we extracted the information of how many lines of code (LOC) was added and deleted in each commit. Furthermore, we recorded the total number of commits performed by each developer. In this paper we will name the total of added and deleted lines of code as edited lines of code.

$$editedLOC = addedLOC + deletedLOC \quad (1)$$

After extracting the commit information related to each developer from the projects, we used two metrics to measure developer participation in projects. The first metric we used is the percentage edited LOC ($Perc.LOC$) for each developer in a project. Percentage edited LOC is the ratio of total edited LOC by a developer to the sum of edited LOC for all developers. The calculation of $Perc.LOC$ for $developer_i$ is given in Equation 2, where n corresponds to the total number of developers.

$$Perc.LOC = \frac{editedLOC_{developer_i} * 100}{\sum_{j=1}^{n} editedLOC_{developer_j}} \quad (2)$$

The second metric we used to evaluate the developer participation is the percentage of commits ($Perc.Commit$) performed by each developer during the development of a software project. Calculation of $Perc.Commit$ is similar to that of $Perc.LOC$. $Perc.Commit$ is the ratio of total number of commits performed by a single developer to the number of commits that was conducted by all the developers who worked in this project. The calculation of $Perc.Commit$ is given in Equation 3, where n is the total number of developers in that project.

$$Perc.Commit = \frac{totalCommits_{developer_i} * 100}{\sum_{j=1}^{n} totalCommits_{developer_j}} \quad (3)$$

5. RESULTS

In this section we present the results of our research. While presenting the results, we follow the research questions we have formed in this research.

RQ1 How widely is effort estimation practices employed?

Previously it was reported by Lederer et. al. in their survey that the software managers puts a high importance on the necessity of software effort estimation [12]. On a scale from 1 (min) to 5 (max) the average importance attributed to software effort estimation practice was 4.7 [12]. In our case study of Turkish industry we observed a similar tendency. However, rather than asking the importance attributed to software effort estimation, we asked whether the company conducts any estimation related activity or employs an estimation model. All the companies we questioned performs software estimation related activities. Of course the level of the activities change drastically, some companies have undergone into projects for developing their own algorithmic model, whereas some companies suffice with formal or informal meetings among domain experts. However, the fact that all companies perform estimation related activities at different levels show that all the companies are well aware of the software effort estimation concept. Therefore, we can say that effort estimation practices are widely employed, but the level of employment differs greatly from company to company.

RQ2 Which effort estimation methods are used?

Every company in our case implicitly or explicitly employs a software effort estimation model. 2 of the companies employs a linear regression based model, in which attributes and the weights related to these attributes are defined by domain experts (the explicit cases). One of these two companies have recently switched to a machine learning based estimation method. The other companies employs expert judgement in their estimation activities. However, they do not strictly follow a Delphi like expert judgement estimation method. Instead they follow their ad-hoc processes that entails a number of meetings between domain experts for discussing the cost of a project. This shows that mostly used estimation method is expert judgement. That is probably due to the fact that it is a method that requires the least effort. However, since the companies do not record the initial expert estimates of a project it is almost impossible to track the efficiency of this method. Although being the minority case, two companies use an algorithmic model and only of these two companies have undergone the effort of developing a complex estimation model that employs machine learning algorithms.

RQ3 How does estimation and planning activities reflect on developer participation?

After initial estimates of software effort, each company plans the allocation of their resources. For resource allocation and planning of effort, companies use various project management tools. Some of the companies prefer their in-house developed project management tools, whereas some prefer to use open source or commercial tools. From our interviews with domain experts in each company, it is our understanding that managers try to distribute the total effort evenly to employees working in particular projects. Keeping track of planned efforts is again performed on the project management tools. Each employee to whom a task is assigned and a certain hours of effort is allocated is responsible to record the actually performed effort into the management tool. However, the consensus among the domain experts from different companies is that the actual effort values is nothing but a confirmation of the planned values. Therefore, the virtually even allocation of resources on the project management tools may be misleading. In that case using churn data may be helpful to get a better view of actual effort spent by developers. On the other hand, none of the companies we have conducted this case study uses churn data for observing the participation of developers to the project.

For each project in these companies, we extracted churn data from their source control system and analysed the developer contribution. We have seen that a big portion of the whole development activity is performed by a very limited number of developers. So the answer to our third research question is that reflection of estimation and planning activities on the developer participation is relatively weak. In other words, the effort of managers to evenly distribute the development effort among developers does not effectively occur in actual development environment. In the next paragraphs, we will evaluate the results for each project separately.

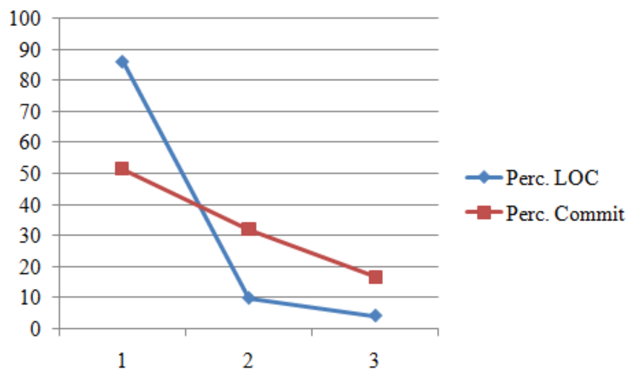

Figure 2: The *Perc.LOC* and *Perc.Commit* information for Project1. Developer 1 is responsible for more than 80% of the whole edited LOC.

In Figure 2 we see the *Perc.LOC* and *Perc.Commit* information regarding Project1. The x-axis corresponds to different developers whereas the y-axis is the percentage value associated with *Perc.LOC* and *Perc.Commit*. Project1 is the smallest project among 6 projects both in terms of size as well as in terms of employed developers. We can see that the majority of the edited LOC in context of this project was developed by a single developer. Developer 1 has edited more than 80% of the total edited LOC. The difference between developers in terms of *Perc.Commit* metric becomes less. However, this may be due to the fact that Developer 1 prefers to commit less often and the other two developers prefer to commit their changes more often.

Figure 3 provides the *Perc.LOC* and *Perc.Commit* values of Project2. Project2 is has the highest number of developers employed. A total of 110 developers have contributed to this project during its development lifecycle. The difference between developer contribution in Project2 is less than that of Project1. However, this may be due to the effect of distributing a value of 100% to 110 developers. We see in Figure 3 that after 8 developers, the contribution of individual developers goes below 2%. Furthermore, we can see from Figure 3 that the contribution of more than half of the developers is less than 1%.

In Figure 4 we see the *Perc.LOC* and *Perc.Commit* values of Project3. Project3 is another densely populated project with 97 developers in total. The behaviour of developer contribution of Project3 is similar to that of Project2, i.e. only a small group of developers have a contribution of more than 2% and a large number of developers (after developer 31) is responsible for less than 1% of the total edited LOC. We also observe from Figure 4 that the behaviour of *Perc.LOC* is very similar to that of *Perc.Commit*. Although there are some irregularities between the two lines, this may be due to different commit habits of individual developers.

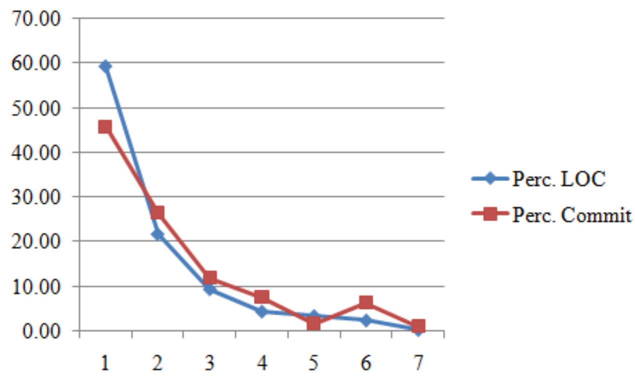

Figure 5: Project4 *Perc.LOC* and *Perc.Commit* information. Developer 1 and developer 2 has more than 80% of the *Perc.LOC*, whereas the rest is shared by other 5 developers.

Figure 5 summarizes the developer contribution of Project4. Project4 is a relatively small project when compared to Project2 and Project3. The contribution pattern among 7 developers employed in Project4 is similar to previous projects. Only two developers account for more than 80% of the whole edits and the remaining 5 developers account for less than 20% of the edits. Furthermore, like previous projects the *Perc.LOC* and *Perc.Commit* lines go hand in hand in Project4 as well.

Perc.LOC and *Perc.Commit* information associated with Project5 are given in Figure 6. Project5 has 11 developers employed in the development phase. Among 11 developers the first three developers edits more than 80% of all the edited LOC. With Developer 5 the contribution of develop-

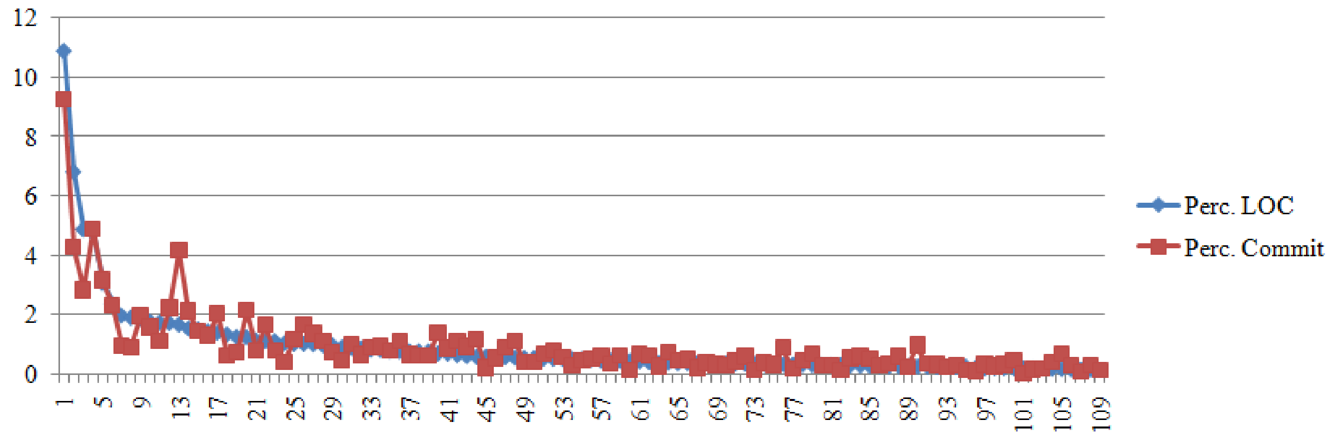

Figure 3: Project2 has the highest number of developers employed. Only 8 developers contribute more than 2% to the project. The participation of more than half of the developers is less than 1%.

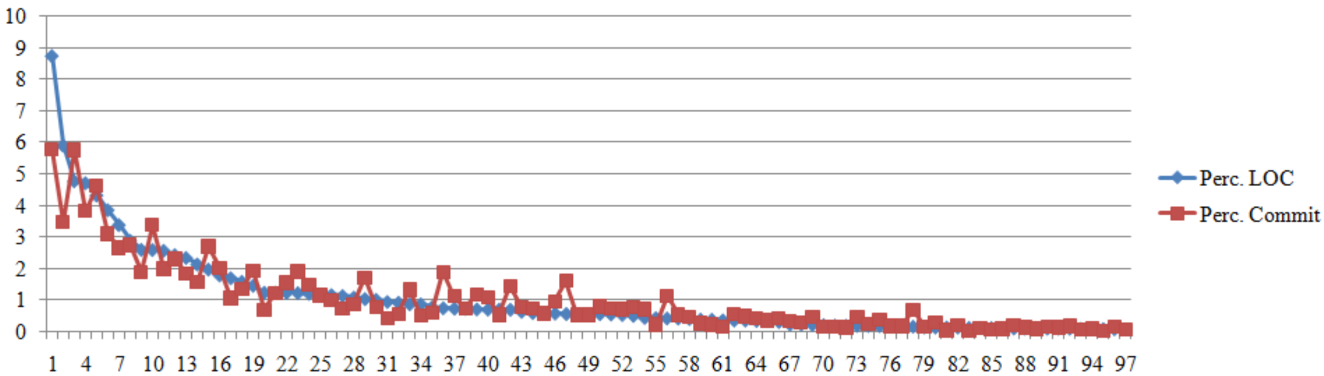

Figure 4: Project3 has 97 developers participating in the development activity. Majority of the developers have limited contribution in the development activity. After developer 31, the participation of developers is less than 1%.

ers in terms of $Perc.LOC$ goes below 5% per developer. One suspicious point in Figure 6 is that there is a considerable difference between $Perc.LOC$ and $Perc.Commit$ values of Developer 1, who is the top developer. The likely explanation to that case could be that Developer 1 was reviewing the code of other programmers and some of the developers were committing through Developer 1. Although this scenario is very likely in open source communities, this is not the case for this particular company and project. After our inquiry with the company, we have learned that such a review and commit scenario does not exist for Project5.

The last project in our research is Project6. Figure 7 provides the $Perc.LOC$ and $Perc.Commit$ information of Project6. In Project6 a total of 19 developers are employed. Among 19 developers the first two developers alone have the total of more than 50% of $Perc.LOC$. Starting with Developer 5, the individual participation of developers fall below 5% in terms of $Perc.LOC$. Furthermore, as in the case of all the previous projects the $Perc.Commit$ line in Figure 7 behaves very similar to the line of $Perc.LOC$.

6. THREATS TO VALIDITY

The most obvious threat to our research is the selection process of the projects. Unlike widely used method of random selection of projects, our projects are selected by the experts of the organizations. Expert selection is based on the fact that projects shall be representative of the company. When considering the wide range of projects in our research we can consider the dataset as representative of the organizations. However, it cannot be proven that there is no selection bias coming from domain experts.

The number of projects evaluated is also a very big threat to the validity of our results. We are aware of the fact that only six projects can be viewed insufficient to draw sound conclusions in an empirical study. In addition, those projects were selected by domain experts, which also limits the validity of the findings. It would have been much better, if we could have used many more projects and if we have selected those projects randomly without human intervention. However, the data shared by the companies (detailed project plans and individual employee performance) is viewed very sensi-

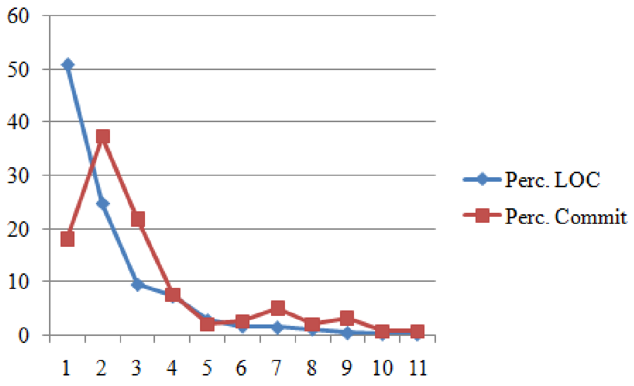

Figure 6: *Perc.LOC* and *Perc.Commit* values of Project5. Similar trend of previous projects continues here. The first three developers account for more than 80% of all edited LOC.

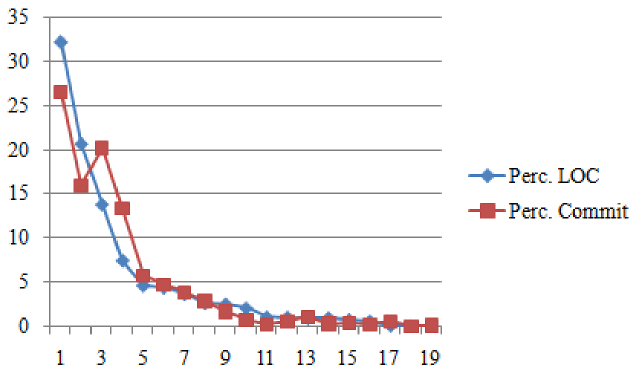

Figure 7: Project6 *Perc.LOC* and *Perc.Commit* values. Out of 19 developers, the first 2 developers account for more than 50% of the total edited LOC.

tive and confidential by managers. Therefore, although we forced our chances to get as many projects as we can and although we are aware of the limitations, we are bound to decisions made by company managements.

Another threat to validity is the selected metrics to evaluate the developer participation. Churn data stored in source control systems of open source projects and private organizations have some characteristic differences. Therefore, use of *Perc.LOC* and *Perc.Commit* may be questionable. However, edited LOC and commit count are successfully used in studies about open source projects [4, 10], so we also based our metrics on edited LOC and commit count as well.

Final threat to validity of our results is the complexity of the code that is committed by a developer. In other words, the differences between developers in terms of *Perc.LOC* can be due to the fact that some developers may be working on more complex parts of the software, hence being able to produce less LOC. Unfortunately we were able to get the source code of a single project to analyse the complexity of committed code. Depending on our analysis we did not see such an effect. The average complexity of the committed code by each developer was more or less the same. Of course we cannot generalize our finding to all 6 projects. However, this may be a hint that code complexity is not a strong validity threat for this research.

7. CONCLUSION

In this research we observed the effort estimation and its use in planning activities in the local perspective of Turkish software industry. We also addressed the implications of estimation and planning activities on developer contribution in software development. Our dataset in this research included 6 projects coming from different companies that specialize in different domains. Our analysis concerning these 6 projects were performed in two phases. For effort estimation activities, we conducted personal meetings with domain experts to gain an insight into their estimation and planning processes. For the developer contribution analysis on the other hand, we extracted churn data from source control systems of different companies and derived our metrics from the extracted churn data.

One implication of our study is that Turkish industry is well aware of the software effort estimation concept and appreciates its importance. All companies in our research implicitly or explicitly employ certain estimation methods. However, estimation practices are mostly far from having a scientific basis. Rather than following literature in effort estimation, organizations suffice with ad-hoc developed processes. Among the companies we worked with, only one company had a solid machine learning based effort estimation model that was developed and implemented by our research team.

Another implication of our research is about developer participation. The workload distribution that is being tracked by managers through project management tools does not reflect the actual participation of developers. In all 6 projects we analysed in our research, we see that majority of the edited LOC is committed by a very limited number of developers. The fact that most of the code is developed by a limited number of developers is previously reported in open source domain [4, 5, 10, 15]. Skewed effort distribution was studied in multiple projects and it was observed that %90 of the churn were done by the %20 of the developers [10] in an open source setting. Although software development in a private organization has very different characteristics in comparison to open source software development, they share common properties in terms of developer participation.

8. ACKNOWLEDGMENT

This research is supported in part by Turkish State Planning Organization (DPT) under project number 2007K120610.

9. REFERENCES

[1] V. R. Basili, R. W. Selby, and D. H. Hutchens. Experimentation in software engineering. *IEEE Trans. Softw. Eng.*, 12(7):733–743, 1986.

[2] B. Boehm, C. Abts, and S. Chulani. Software development cost estimation approaches: A survey. *Annals of Software Engineering*, 10:177–205, 2000.

[3] B. W. Boehm. *Software Engineering Economics.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[4] B. Caglayan, A. Bener, and S. Koch. Merits of using repository metrics in defect prediction for open source projects. In *FLOSS '09: Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 31–36, Washington, DC, USA, 2009. IEEE Computer Society.

[5] K. Crowston, K. Wei, Q. Li, and J. Howison. Core and periphery in free/libre and open source software team communications. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 118.1, Washington, DC, USA, 2006. IEEE Computer Society.

[6] M. Jø rgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70:37–60, February 2004.

[7] M. Jorgensen and T. Gruschke. The impact of lessons-learned sessions on effort estimation and uncertainty assessments. *IEEE Trans. Softw. Eng.*, 35(3):368–383, May-June 2009.

[8] B. Kitchenham, L. Pickard, and S. L. Pfleeger. Case studies for method and tool evaluation. *IEEE Softw.*, 12(4):52–62, 1995.

[9] E. Kocaguneli. Better methods for configuring case-based reasoning systems. Master's thesis, Bogazici University, 2010.

[10] S. Koch. Effort modeling and programmer participation in open source software projects. *Information Economics and Policy*, 20(4):345 – 355, 2008. Empirical Issues in Open Source Software.

[11] Y. Kultur, B. Turhan, and A. B. Bener. ENNA: software effort estimation using ensemble of neural networks with associative memory. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 330–338, New York, NY, USA, 2008.

[12] A. L. Lederer and J. Prasad. Information systems software cost estimating: a current assessment. *J Inf technol*, 8:22–33, 1993.

[13] T. Menzies, Z. Chen, J. Hihn, and K. Lum. Selecting Best Practices for Effort Estimation. *IEEE Trans. Softw. Eng.*, 32:883–895, 2006.

[14] K. Moløkken-Østvold, M. Jørgensen, S. S. Tanilkan, H. Gallis, A. C. Lien, and S. E. Hove. A survey on software estimation in the norwegian industry. *IEEE International Symposium on Software Metrics*, pages 208–219, 2004.

[15] G. Robles, S. Koch, J. M. Gonzlez-Barahona, and J. Carlos. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In *In Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS*, pages 51–55, 2004.

[16] M. Shepperd. Software project economics: a roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 304–315, 2007.

[17] M. Shepperd and G. Kadoda. Comparing software prediction models using simulation. *IEEE Trans. Softw. Eng.*, pages 1014–1022, 2001.