International Journal of Software Engineering and Knowledge Engineering

RANKING DEVELOPERS' IMPORTANCE FACTORS BASED ON TEAM LEADER PERSPECTIVE

GUILHERME COSTANTIN TANGARI
MARCELO DE ALMEIDA MAIA
Faculty of Computing, Federal University of Uberlândia
Uberlândia, Minas Gerais, Brazil
guilhermecostantin@mestrado.ufu.br
marcelo.maia@ufu.br

Several companies use amount of deliveries as a metric for performance evaluation of developers. However, the productivity of a developer and his importance for the company is not only related to the amount of produced lines of code. There are a variety of factors that can contribute to the relevance of developers for their teams. This paper aims at mapping some of these factors, measuring those that are more important for companies and propose an evaluation model of developer importance that considers more than just deliveries. We have found that some factors are more important than others and that there are minor differences for different companies. We have also developed a high accuracy classifier that can indicate the importance of the developer based on a set of attributes.

Keywords: productivity; developers importance; pattern recognition; human factors.

1. Introduction

All kinds of companies have been investing in techniques to increase productivity in order to increase competitiveness, and this is no different in the software industry, which still continues investing in new methods, tools and best practices that could lead organizations to productivity improvement [3]. Traditional productivity metrics for software development are based either on lines of code (LOC) or function points (FP) [5], for example, the amount of LOC or FP developers deliver per hour. A slightly more abstract definition for productivity is the ratio of delivered outputs to consumed inputs, where outputs may be LOC, FP, or other relevant delivery, and inputs are the resources used to produce that output, e.g., time, people [2], [6]. Nonetheless, the use of only these traditional metrics can mislead the management of software teams. LOC does not take into account the effort and knowledge required to write the code. Complex problems often require experienced developers to solve them, and often, they do not require lots of lines of code. In that case, experienced developers would be penalized. Several companies are beginning to gain awareness of these issues and are committed to improve the way they evaluate developer performance. This work aims at investigating how team leaders understand the

notion of importance, indicating which factors are most relevant in their overall assessment of developers. We are interested in the investigation of these questions:

1. What are the most important criteria used by leaders while assessing developers?
2. Is it possible to build a classifier for developers' importance with high accuracy, using the proposed criteria? This question can be refined in other two:
   (a) Is it possible to have a generic classifier, i.e., company-independent? or
   (b) Is it more appropriate to build customized classifiers for each company?

In this paper, we used 16 factors, elicited on previous studies [4], which can have an influence in the leaders evaluation. We conducted a survey with team leaders representing software companies, where they evaluated their developers based on those factors. The result was analyzed in attempt to recognize a pattern in their evaluation. We identified those factors that mostly influence the leaders evaluation about their developers, and also built a high accuracy classifier for developers importance. These contributions may even help human-resource managers to select candidates based on a target profile.

2. Methodology and Results

To investigate the current practice of developers evaluation, we decided to perform a survey with human subjects in real software companies to extract the desired information and analyze it. In this survey, we ask respondents firstly to classify the importance of a subject developer and then fill the rest of the survey with the respective developers characteristics. To analyze the obtained data, we use automatic classification methods to understand how those characteristics affect leaders' classifications, and as a product we still may have a classifier that can be used to help leaders and human-resource managers to gain more insight about their teams productivity.

The remainder of this section is aimed at explaining how the survey was designed, how we conducted our data analysis, how the classifier was constructed, and the respective results.

2.1. *Survey*

We used an approach called Goal Question Metric (GQM) [1] that helped us define our survey. GQM is a top-down approach, that is based on the assumption that first, to measure something, you need to specify goals, from which it is possible to derive questions that define those goals, and then specify the metrics that need to be collected to answer those questions.

The survey was applied to software companies that have a software development environment with a minimum hierarchical structure where exists the role of leaders, or managers, or chief engineers, etc. (for future references, we call that person, the leader). Eleven respondents (leaders) provided 61 answers (unique developers evaluated). There were eight companies involved in the collected data. All participant

companies work on their own products (i.e., they are **not only** software factories selling manpower), they vary in size, considering amount of employees (developers), they vary in the sector of operation (ERP, Telecom) and they have some variation in the used technologies.

Overall Survey Results. We asked the leaders to classify each developer in five degrees of importance. The classes (and their respective relative frequency from leaders' classification) are *Very important* (41%), *Important* (31%), *Average importance* (20%), *Less important* (3%) and *Very less important* (5%).

Analyzing the survey results, we came to the conclusion that the leaders were conservative in some degree to classify their developers in the lowest classification of importance. From this analysis, we decided to group the developers in only two classes based on the original five classes, *High importance*, that aggregate the result of the *Very important* and *Important* classes, and *Low importance*, that aggregate the results of the rest of the classes.

2.2. *Feature Selection*

In order to conduct the analysis to determine which factors are the most relevant and have major influence in the leader evaluation, we use the algorithm called GainRatioAttributeEval (a feature selection algorithm) to select a subset of features in order to improve the quality and achieve the best possible accuracy of our classifier. To apply the algorithm and build the classifier, we use WEKA, an open-source software for data mining and machine learning.

Table 1 shows the feature rank ordered by the Average merit (the rate that the attribute influences the classification). The higher the Average merit, the more important is that feature for the classification algorithm, i.e., the more discriminative is that feature to identify the developer's importance.

Table 1. Attribute Ranking

Features	**Average merit**
Proactivity	0.168
Subjective evaluation of the productivity	0.156
Capacity of solving complex problems	0.126
Focus on the results	0.112
Past experiences	0.107
Creativity	0.095
Planning and organization	0.09
Generalization (diversity of skills)	0.08
Specialization (expert in some technology or tool)	0.071
Focus on the customer	0.063
Time of work in the organization	0.063
Willingness to help a colleague	0.057
Leadership	0.052
Communication with the team members	0.039
Entrepreneurship	0.015
Main behavior of the developer	0.016

2.3. *Classification*

After producing the feature ranking, we use that rank to construct a classifier using the most important features. To determine how many features need to be selected to get higher performance, we conducted an exhaustive test (we ran the classifiers with a crescent numbers of features selected, from 2 to 16) and chose the configuration with better performance (8 attributes).

To evaluate the performance of a classifier, we used 10-fold cross-validation that divided the dataset in 10 equal parts (called folds), take 9 pieces to use for training and use the last piece for testing, and then do it 9 more times, always alternating the piece used for testing, that way, a single fold will be used 9 times for training and 1 for testing. The final result is the average of the 10 runs.

Different classifiers were constructed and evaluated considering the data gathered from all companies and also from the individual company with the largest number of observations.

The used machine-learning algorithms are J48, a tree classifier, and Naïve Bayes. There is no strong reason to choose them from other options, yet they tend to produce high quality classifiers in general, whenever possible.

Classification Results. Using the reduced set of classes (*High importance*, *Low importance*), we achieved better classifier accuracy, as expected. The algorithm Naïve Bayes had better performance, achieving a relevant accuracy of 85% against the 75.8% of correctness of J48, considering the data of all companies.

When analyzing the individual company, we noticed some major changes in some features position in the feature ranking: behavioral characteristics (creativity) and attributes related to the developers commitment with the company (focus on results) in some cases were more important than the classic metric of productivity. We credit those differences to the culture and values of that particular company that have a pressure for innovative products and management efficiency. Although different companies may assess the importance factors with some variation, and thus, specific classifiers for different companies produce more tuned results, using the general framework assessment is still fairly accurate for all of them.

2.4. *Threats to Validity*

We recognize some threats to this work. The first one is related to the used sample of companies. Although there is some reasonable representativeness on that sample, results may vary if replicated in other different companies. Other threat is related to the leader's beliefs on the notion of productivity. Different leaders in the same company could had answered differently the survey. We tried to mitigate this threat considering a reasonable number of different leaders, even in the same company, in some cases. There could also be other factors to define productivity that were not addressed in this study, however we tried to mitigate this threat considering previous factors already studied in the literature.

3. Discussion and Conclusion

The first point considered in this discussion is the creation of the reduced new set classes. The classification provided by leaders tended to be more positive, maybe because they would not like to say that they maintain developers with low importance in their teams. So, a neutral classification could be interpreted as negative, because in that case, the leader do not consider that developer as an important one. We grouped the original set of 5 classes in only 2, creating a new set of classes that proved, as expected, to improve the performance of all the classification algorithms applied. This new reduced set of classes preserves the meaning of the original classification performed by the leaders.

We provided a set of criteria used by the leaders of IT companies to evaluate their developers, and also ranked those criteria, finding that capacity of solving complex problems, quantitative evaluation of productivity and proactivity were generally the most important factors.

The classification results evaluating all companies together with Naïve Bayes provided a classifier with 85.2% accuracy that can be considered a successful and useful result. The use of this classifier can help leaders conducting more coherent analysis of the team profile and the human resource managers to look for candidates that have the needed characteristics and more potential to become an important part of the team.

The limited number of developers and companies involved in this study may limit the generalization for other contexts. Nonetheless, we have observed several intersections in different companies that can explain why the general classifier had reasonably high accuracy.

A qualitative analysis, considering the culture of the company and their values, and the application of that classifier in the collaborators of open-source software repositories, to validate the results or spot the differences, could be suggestions of future work.

Acknowledgements. We would like to acknowledge CAPES, CNPq and FAPEMIG for partially funding this research.

References

[1] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, 2:528–532, 1994.

[2] B. W. Boehm. Improving Software Productivity. *Computer*, 20(9):43–57, 1987.

[3] de Barros Sampaio et al. A review of productivity factors and strategies on software development. In *Software Engineering Advances (ICSEA)*, pages 196–204. IEEE, 2010.

[4] G. C. Tângari and M. A. Maia. Developers' importance from the leader perspective. In *Proc. of SEKE'2015*, pages 1–6.

[5] S. Wagner and M. Ruhe. A Structured Review of Productivity Factors in Software Development Technical . Technical report, 2008.

[6] C. E. Walston and C. P. Felix. A method of programming measurement and estimation, 1977.