# How to Identify Programming Skills from Source Code?

**Johnatan Oliveira**

[1]Department of Computer Science (DCC)
Federal University of Minas Gerais (UFMG)
Software Engineering Lab (LabSoft)
Belo Horizonte, Brazil
johnatan.si@dcc.ufmg.br

***Abstract.*** *Both open-source and proprietary software systems have become increasingly complex. Despite their growing complexity and increasing size, software systems must satisfy strict release requirements that impose quality, putting significant pressure on developers. Therefore, the success of software projects is dependent on the identification and hiring of qualified developers to build a solid and cohesive team with different programming skills. Our main goal is to develop and evaluate a method able to compute programming skills from source code analysis. Our method uses software metrics such as Changed Files and Changed Lines of Code, to compute the skills. Our results showed that our method is able of identifying programming skills of the developers about mainly libraries used, programming languages, and profile concerning back-end & front-end and unit test.*

**Keywords:** Skills, Program Skills, Mining Software Repositories

## 1. Introduction

Both open-source and proprietary software systems have become increasingly complex. Despite their growing complexity and increasing size, software systems must satisfy strict release requirements that impose quality, putting significant pressure on developers. Therefore, software projects' success depends on identifying and hiring qualified developers to build a solid and cohesive team with different programming skills. The current practice in industry and research for assessing programming skills is mainly based on proxy variables of skills, such as education, years of experience, and multiple-choice knowledge tests [Baltes and Diehl 2018]. However, poor hiring decisions may hurt the success of a software system [Marlow and Dabbish 2013]. In particular, software projects have many open positions and not as many qualified candidates [Baltes and Diehl 2018]. In this context, identifying qualified candidates with the right combination of skills is crucial to the success of a software project because these candidates may not actively apply themselves for open job positions.

Several approaches aim at addressing this problem by proposing models [Montandon et al. 2019, Bizer et al. 2011, Constantinou and Kapitsaki 2016, da Silva et al. 2015] and tools [Greene and Fischer 2016, Marlow and Dabbish 2013] to automatically identify the programming skills of developers via source code analysis. These models and tools rely, for instance, on information about who has changed each file to identify developers who might know technology and business logic in these particular parts of a system. With the emergence of social coding platforms, such as

GitHub, a lot of information about developers and projects is available. Such social and programming data about developers have a vast potential to support the effective identification of programming skills and expertise. Greene and Fischer (2016) proposed a tool named CVExplorer to identify developers with programming skills, such as expertise in programming languages.

Although these approaches may potentially identify the programming skills, we still lack approaches able to compute programming skills in 5 perspectives: i) which programming languages each developer knows, ii) main libraries, iii) back-end & front-end profile, iv) if the developer creates unit tests, and v) the developer's experience in mobile devices. Therefore, this project presents a proposal for a Ph.D. thesis in Computer Science to fill this gap. Our contribution targets both researchers and practitioners. Researchers can use our results to design studies related to skills and reflect on the complicated relationship between source code activities and skills. Practitioners can use it to refine the practical evaluation of professional skills for several purposes, from hiring procedures to the evaluation of team formation. Understanding the effectiveness of the different approaches is the first and necessary step for building tools that will genuinely indicate programming skills by analyzing activities in the software repository [Baltes and Diehl 2018].

## 2. Work Plan

To develop the proposed method, we established a set of tasks to be performed during the Ph.D. course. Figure 1 shows the steps. The tasks may have three possible status: Done, In Progress, and To-Do. Our study is divided in ten steps, as follows: i) As we can observe, the first step of the research was the literature review, from which we could understand the state-of-art of metrics, tools, and models to compute programming skills from the source code (done). We noticed that a few works investigated programming skills from source code in perspectives analyzed in this study. ii) From the previous step, we developed the metrics to compute programming skills (done).

We focus mainly on static metrics, for example, changed files and changed lines of code. iii) From the metrics, we develop a method able to compute programming skills (done). iv) In order to analyze our method, we conducted a clone of repositories from GitHub (done). In steps v and vi, our method made an analysis of changed files and changed lines of code in desktop and mobile app (in progress). We need to improve this method to achieve more accuracy (in progress). vii) From the method developed, we need to conduct an empirical study to evaluate them. For this, we select developers, repositories, programming languages, libraries, and target developers to survey (in progress). viii) From the previous step, we conducted a survey with developers from GitHub in order to identify opportunities to improve the method (in progress). ix) After all previous steps, our aim is to evaluate a tool that supports the method developed (to-do). x) In this last step, we compile all findings to contribute to science and conclude the Ph.D. (to-do).

## 3. Method Overview

This section presents our method developed to compute programming skills in six perspectives: i) which programming languages are used by developer, ii) main libraries are known by them, iii) profile as develop back-end & front-end, iv) unit test, v) frequency of commits, and vi) experience of the developer in mobile devices (Android).

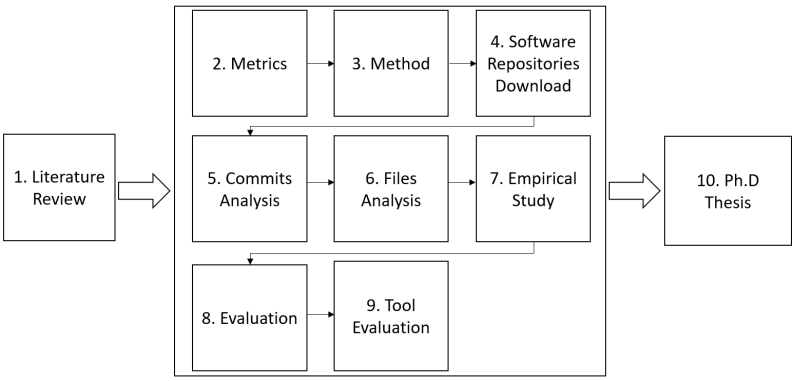


**Figure 1. Work plan for quantitative and qualitative studies**

### 3.1. Programming Skills of Library Experts

A library is a collection of objects, functions, and methods [Reif et al. 2016]. Its main benefits are supporting the work performed by developers in such a way that they can implement new features or maintain software systems with more quality [Reif et al. 2016]. Consequently, developers with more knowledge in a library, in particular, can work more actively in open source and proprietary projects [Bailey and Stefaniak 2001, Bailey and Mitchell 2006]. We rely on 3 metrics to identify the library experts. Table 1 shows these metrics and explains them in more details. Each metric is computed in relation to the amount of commits to a specific library. That is, when a commit to a library is identified, the metrics were calculated.

**Table 1. Proposed Metrics**

| Metric | Description |
|---|---|
| Number of Commits | This metric calculates the activity of each developer through the number of commits using a particular library. Through this metric, it is possible to measure the amount of use of the library in a project that a specific developer works. |
| Number of Imports | This metric presents the intensity of use of a particular library. For this metric, we count all imports to the library written by a developer. Repeated imports are included. |
| Lines of Code | To compute this metric, we developed a heuristic to count the amount of LOC related to a specific library, as follows. First, we obtain the ratio of changed LOC by the number of all imports in the file. Then, we multiply the ratio by the number of imports related to the library. |

We present the skills of each developer in 5 dimensions: Knowledge Amplitude, Knowledge Intensity, Code Proficiency, Collaboration, and Experience. To rank developers on each of the 5 skills, we created indexes. The indexes range from 1 (beginner) to 5 (senior). Table 2 describes the thresholds used to calculate scores for each skill. This table was designed by empirically analyzing the distribution of data in our study. Thresholds in this table can be adapted to specific contexts.

**Knowledge Amplitude–** This skill indicates the spread of a developer's knowledge of different libraries. To rank developers in this attribute, we count the number of unique imports written by the developer and related to a specific library.

**Knowledge Intensity–** This skill shows the intensity of use of a library. We count

**Table 2. Thresholds for skill scores**

| Rank | Knowledge Amplitude | Knowledge Intensity | Code Proficiency | Collaboration | Experience |
|---|---|---|---|---|---|
| 1 | #unique imports between (1 and <=10) | #tota limports between (1 and <=10) | #total LOC between (1 and <=500) | #different file between (0 and <= 5) | #different projects= 0 or 1 |
| 2 | #unique imports between (11 and <=50) | #tota limports between (11 and <=100) | #total LOC between (501 and <=5,000) | #different files between (6 and <= 10) | #different projects = 2 |
| 3 | #unique imports between (51 and <=100) | #tota limports between (101 and <=1,000) | #total LOC between (5,001 and <=50,000) | #different files between (11 and <= 20) | #different projects = 3 |
| 4 | #unique imports between (101 and <=500) | #tota limports between (1,001 and <=10,000) | #total LOC between (50,001 and <=500,000) | #different files between (21 and <= 30) | #different projects = 4 |
| 5 | #unique imports between (>=501) | #tota limports between (>=10,001) | #total LOC (>=500,001) | #different files >= 31 | #different projects >= 5 |

all imports to the library written by a developer. Repeated imports are included.

**Code Proficiency–** With this skill, we want to measure how much the developer is productive in a given library. Therefore, we count the total lines of code (LOC) added when this code is related to a library. That is, if the developer adds an import for the library, we assume that this developer is writing code related to the library. We count only non-commented LOC; blank lines and comments are not accounted for this criterion. Table 2 describes the thresholds considered for calculating code proficiency. To compute this metric, we developed a heuristic to count the amount of LOC related to a specific library, as follows. First, it obtains the ratio of changed LOC by the number of all imports in the file. Then, it multiplies the ratio by the number of imports related to the library. The heuristic takes into consideration 3 attributes, the number of library imports, the number of imports in general, and the number of LOC altered by a commit related to the library. The heuristic is then computed as follows:

$$LOC = \frac{\text{\# of LOC Altered by Commit}}{\text{\# of All Imports}} X \text{ \# of Library Imports}$$

**Collaboration–** This skill measures the developer's ability to collaborate with other peers in the same file. Therefore, we count the number of different source code files that the developer has modified in partnership with other developers.

**Experience–** This skill denotes the number of additional projects where the developer uses a specific library. Multitasking can produce the developer good dividends if done the right way.

### 3.2. Programming Skills Profile

Besides compute library experts, we developed 2 other metrics to compute programming skills in 3 dimensions: programming language, back-end & front-end profiles, and unit test. These 2 metrics are, (i) Changed Files and (ii) Changed Lines of Code. Changed Files compute the number of files change made by a specific developer, for example, Mary modified 3 Python, 2 Java, 2 CSS, and 1 HTML files. Therefore, according to the Changed Files, her skill in programming languages is distributed as follows: 37.5% Python, 25% Java, 25% CSS, and 12.5% HTML. In a similar fashion, the metric Changed Lines of Code

considers the number of LOC added or removed by a specific developer. For example, to compute the programming skills to Johns in programming languages, we need to count the total number of changed LOC for all files that share the same file extension. John, for instance, has a skill distribution in programming languages as follows: 7 Python, 2 CSS and 1 HTML. Therefore, in percentage we obtain, 70% Python, 20% CSS, and 10% HTML. This data are about programming languages.

Besides compute programming languages, we also analyzed the programmer (a) alignment between back-end front-end profiles and (b) skills in test development. For the former, we classified a set of programming languages as front-end (e.g., CSS), and another set as back-end technologies (e.g., Java). Therefore, we also used the models previously described to calculate back-end front-end profiles. For the latter, to calculate the test skills, we need to identify if a particular folder contains test files. That is, the source code is parsed and we search for directory structures mentioning tests. We manually validated a sample of our script output and verified a precision above 80%. We then consider it good enough for our purpose.

## 4. Preliminary Results

In this section, we present the dataset used to evaluate our method (section 4.1). Besides, we show the preliminary results in two perspectives: i) library experts (section 4.2) and ii) programming skills profiles (section 4.3 ).

### 4.1. Dataset

Our method computed 1,137 curricula from our dataset. To create our dataset, we conduct a process of filtering to select the developers. First, we select 2,000 developers randomly. Second, we made a filter by developers with the top-10 programming languages[1]: JavaScript, Python, Java, Go, C++, Ruby, PHP, TypeScript, C#, and C. Besides, we selected the style sheet language CSS and HTML. Third, from the last filter, we delete developers whose repositories have less than ten projects. Fourth, we select from the previous step developers with at least 100 commits in projects. In the fifth and last step, we select developers with at least a thousand LOC committed and obtained the 1,137 developers able to participate in our analysis.

### 4.2. Library Experts

In this section, we present a overview of some relevant findings achieved in perspective library experts. Our study shows that a significant amount of expert candidates makes commits, when writing code related to a specific library, performs many imports of particular libraries, and writes LOC in sequence when making an import of the library. The developers were invited (survey) to rank their knowledge using a scale from 1 (one) to 5 (five), where (1) means no knowledge about the library; and (5) means extensive knowledge about the library. If we analyze the data about the precision of the strategy from the sum of levels 3, 4 and 5 of Likert-type scale, we obtain on average 88.49% of accuracy in relation the knowledge of the developers, i.e., identification is correct in more than 88% of the cases. On the other hand, although a score 3 may represent an acceptable knowledge, if we followed a more conservative criterion, only classifying as library experts the developers that informed a higher ($\geq 4$) knowledge on the libraries, we obtain,

[1] https://madnight.github.io/githut/#/pull_requests/2020/2

on average, 63.31% of precision. This way, we conclude that most of the identified expert candidates identified by the strategy contain high knowledge about the evaluated libraries. In contrast, to a level of knowledge $< 3$, we achieved only 11.51% of the developers, i.e., possibly the strategy fails by selecting these developers.

### 4.3. Programming Skills Profiles

This section describes the survey results applied with developers from GitHub. Our dataset had the same setup used in the section 4.1. Figure 2 shows the opinion of the developers about our method. In this figure, we have 3 blocks, one for each evaluated programming skill. The first block of charts represents programming language skills. The second block depicts the back-end & front-end profiles. Finally, the last block shows the test skills. Concerning the Changed Files agreement, we observe in the programming language perspective that 27% of the participants answered "strongly agree" and "agree". On the other hand, we observe that to the Changed LOC, 14% and 22% of the participants answered respectively "strongly agree and "agree". Concerning disagreements, we have 12% and 18% to "strongly disagree" and "disagree" to the Changed Files. While for Changed LOC, we observe 18% and 19%, respectively, to "strongly disagree" and "disagree". This way, we note that the Changed Files models, in general, are better evaluated by developers to capture their programming language skills.

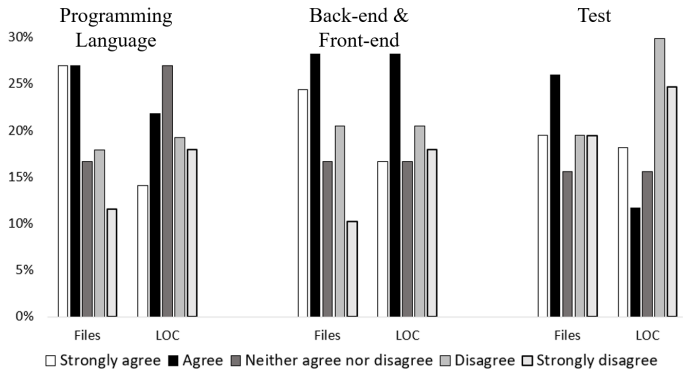


**Figure 2. Overview**

## 5. Related Work

The use of data present in GitHub to understand how software developers work and collaborate has become recurrent in software engineering studies [Destefanis et al. 2016]. Some studies seek to understand the behavior of developers concerning interaction with their peers [Ortu et al. 2015]. For example, studies that try to understand who are the developers with peaceful behavior and those with aggressive behavior and if these developers coexist productively in software development projects [Ortu et al. 2016]. Similar studies also try to understand if there is a relationship between bug resolution time and behavior of developers [Ortu et al. 2015]. Also, some studies investigate developers manners [Destefanis et al. 2016] and seek to understand the emotional behavior of software developers [Ortu et al. 2016].

Greene and Fischer (2016) have developed an approach to extracting technical information from GitHub developers. The work of researchers also does not differentiate

developers from their level of knowledge of technical skills, since a recruiter has several candidates for the same job position [Greene and Fischer 2016]. In addition, the work only shows the profile of the users in GitHub, and it does not extract other characteristics of their knowledge and skills. Besides, they do not provide actual data about the developer's knowledge production. Singer et al. (2013) investigate the use of profile aggregates in the evaluation of developer skills by developers and recruiters [Singer et al. 2013]. However, these aggregates only gather skills for individual developers, and it is not clear how they support the identification of relevant developers from a large dataset of candidates.

We believe that our method is complementary to the described related work, providing a different approach focusing on the identification of possible experts in specific technologies. To the best of our effort, we did not find a similar strategy. Hence, we cannot compare our proposal to other studies.

## 6. Conclusion and Future Work

This project presents a proposal for a Ph.D. thesis in Computer Science that focuses on the method to compute programming skills from source code. We believe that mining software repositories to understand how developers use specific technologies provide objective insight into their real expertise. Therefore, our method may be useful to support the assessment of developers' skills, which are often analyzed using curriculum information, interviews, and admission tests. Our method uses data from GitHub to compute programming skills from source code. We rely on a set of metrics and thresholds. Our preliminary results showed that our method is able compute programming skills.

As future work, we intend to conclude the last study steps, publish the study results as papers for the community, complete the study (to-do) and finish this Ph.D.'s is December 2022 or January 2023 with the thesis defense.

## References


Bailey, J. and Mitchell, R. B. (2006). Industry perceptions of the competencies needed by computer programmers: technical, business, and soft skills. *Journal of Computer Information Systems*, 47(2):28–33.

Bailey, J. L. and Stefaniak, G. (2001). Industry perceptions of the knowledge, skills, and abilities needed by computer programmers. In *Proceedings of the 2001 ACM SIGCPR conference on Computer personnel research*, pages 93–99. ACM.

Baltes, S. and Diehl, S. (2018). Towards a theory of software development expertise. ESEC/FSE 2018, page 187–200, New York, NY, USA. Association for Computing Machinery.

Bizer, C., Heath, T., and Berners-Lee, T. (2011). Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227.

Constantinou, E. and Kapitsaki, G. M. (2016). Identifying developers' expertise in social coding platforms. In *42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*.

da Silva, J. R., Clua, E., Murta, L., and Sarma, A. (2015). Niche vs. breadth: Calculating expertise over time through a fine-grained analysis. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*.

Destefanis, G., Ortu, M., Counsell, S., Swift, S., Marchesi, M., and Tonelli, R. (2016). Software development: do good manners matter? *PeerJ Computer Science*, 2:e73.

Greene, G. J. and Fischer, B. (2016). Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions. In *31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 804–809.

Marlow, J. and Dabbish, L. (2013). Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 145–156.

Montandon, J. E., Silva, L. L., and Valente, M. T. (2019). Identifying experts in software libraries and frameworks among GitHub users. In *16th International Conference on Mining Software Repositories (MSR)*, pages 276–287.

Ortu, M., Adams, B., Destefanis, G., Tourani, P., Marchesi, M., and Tonelli, R. (2015). Are bullies more productive?: empirical study of affectiveness vs. issue fixing time. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 303–313. IEEE Press.

Ortu, M., Destefanis, G., Counsell, S., Swift, S., Tonelli, R., and Marchesi, M. (2016). Arsonists or firefighters? affectiveness in agile software development. In *International Conference on Agile Software Development*, pages 144–155. Springer.

Reif, M., Eichberg, M., Hermann, B., Lerch, J., and Mezini, M. (2016). Call graph construction for Java libraries. In *24th Proc. of the ACM Int. Symposium on Foundations of Software Engineering (SIGSOFT)*, pages 474–486.

Singer, L., Figueira Filho, F., Cleary, B., Treude, C., Storey, M.-A., and Schneider, K. (2013). Mutual assessment in the social programmer ecosystem: an empirical investigation of developer profile aggregators. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 103–116. ACM.