Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.069707

ARTICLE

Individual Software Expertise Formalization and Assessment from Project Management Tool Databases

Traian-Radu Plosca[1,*], Alexandru-Mihai Pescaru[2], Bianca-Valeria Rus[1] and Daniel-Ioan Curiac[1,*]

[1]Department of Automation and Applied Informatics, Politehnica University of Timisoara, V. Parvan 2, Timisoara, 300223, Romania
[2]Department of Computer and Information Technology, Politehnica University of Timisoara, V. Parvan 2, Timisoara, 300223, Romania

*Corresponding Authors: Traian-Radu Ploscă. Email: traian.plosca@aut.upt.ro;
Daniel-Ioan Curiac. Email: daniel.curiac@aut.upt.ro

Received: 29 June 2025; Accepted: 13 October 2025; Published: 10 November 2025

ABSTRACT: Objective expertise evaluation of individuals, as a prerequisite stage for team formation, has been a long-term desideratum in large software development companies. With the rapid advancements in machine learning methods, based on reliable existing data stored in project management tools' datasets, automating this evaluation process becomes a natural step forward. In this context, our approach focuses on quantifying software developer expertise by using metadata from the task-tracking systems. For this, we mathematically formalize two categories of expertise: technology-specific expertise, which denotes the skills required for a particular technology, and general expertise, which encapsulates overall knowledge in the software industry. Afterward, we automatically classify the zones of expertise associated with each task a developer has worked on using Bidirectional Encoder Representations from Transformers (BERT)-like transformers to handle the unique characteristics of project tool datasets effectively. Finally, our method evaluates the proficiency of each software specialist across already completed projects from both technology-specific and general perspectives. The method was experimentally validated, yielding promising results.

KEYWORDS: Expertise formalization; transformer-based models; natural language processing; augmented data; project management tool; skill classification

1 Introduction

In the context of today's rapidly evolving technological landscape, the precise identification of software experts has become a critical factor when assembling effective development teams, improving productivity, and optimizing development timelines. However, it remains a core issue, especially in large companies where individual skills and contributions are not always visible.

Well-structured teams contribute to creating a collaborative and innovative work environment [1]. The traditional approach of finding experts through the human resources department is a time-consuming task that requires substantial resources from organizations [2]. Automatic software approaches can improve the detection of patterns and relationships that are often difficult to observe otherwise. This enables a more objective evaluation of the skills and experiences of team members. In addition, it can be continuously refined and adjusted as new data become available. This capacity for ongoing learning ensures that teams are consistently optimized to meet future needs and goals. Notable research works in this field include collaboration graphs from email lists [3], locating expertise within organizations through recommendation

systems [4], and semi-automated methods for assessing expertise using Machine Learning (ML) techniques to analyze the types of bugs developers have resolved [5].

In this paper, we propose a method for evaluating the software specialists' expertise by leveraging metadata from project management tools like Jira, Trello, Asana, and Azure DevOps. Our approach begins by formulating two mathematical models to evaluate specific technology expertise and general expertise. We then develop a technique that automatically identifies the relevant skills for each developer, using an ML model trained on labels extracted from the StackOverflow dataset and transferring the knowledge to the task-tracking tool dataset. This allowed us to profile developers across different technology stacks without the need for manual analysis of their tasks. We also estimate the task complexity based on metadata recorded by task-tracking tools, such as issue type, connections to other tasks, priority, number of comments, and story points, which enabled us to accurately assess the prior work of each software specialist. Finally, we calculate the expertise-related individual scores for each possible candidate for a team position.

Our contributions are as follows:

- We formalize two mathematical models to calculate specific technology-related expertise and generalized expertise of a software developer based on task metadata recorded by project management tools.
- We propose an automated method to identify the skills and programming languages associated with already-completed tasks using BERT-like models. We confirm that models trained on software engineering data are more efficient in identifying technical details than models trained on general language.
- In the experimental section, we validate the proposed system.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the related work in the field. Section 3 describes the features that need to be extracted from project management tool databases to effectively formalize the expertise, as well as our mathematical models to evaluate technology-related and overall expertise. In Section 4, we present the proposed methodology in detail. Section 5 presents the experimental validation and discusses the results. In Section 6, we outline the main research implications of our work. Section 7 describes the threats to validity. Finally, we conclude with Section 8.

2 Related Work

Previous research on identifying and assessing individual expertise has been focused on two specific areas. The first focus was placed on automatically identifying the skills and technologies related to a software specialist, while the second was centered on evaluating the level of expertise from available data.

2.1 Automatic Assessment of Programming Skills

Software engineering teams rely on project management tools to organize their day-to-day tasks. Generally, the tasks do not include metadata about the programming language or technology used. Although it is possible to manually add labels or tags to indicate the technologies, they are often omitted.

In this particular context, predicting team roles from available data can increase our understanding of workflows. In their pioneering study, Martens and Franke [6] demonstrated that machine learning techniques could be used to extract individual roles from software repositories. They classified team positions such as developer, architect, scrum master, product owner, and team manager based on data collected from task-tracking systems. This research highlights the potential of using machine learning to analyze existing software engineering data for role identification, which may improve team dynamics and resource allocation.

In our research, we are using StackOverflow and Jira datasets to understand and identify the technical skills of developers, based on multiple metrics. The use of StackOverflow data has already been studied for classification tasks. Alreshedy et al. [7] developed a Multinomial Naive Bayes classifier, named SCC, trained

on StackOverflow posts to predict programming languages, which achieved an F1 accuracy score of 75%. In their newer study [8], they utilized an improved version called SCC++, built on top of XGBoost and the Random Forest algorithms, that achieved an accuracy of 88.9% across different programming languages.

A recent paper proposes the automatic classification of issues by their type, using large language models (LLMs), particularly GPT-like models [9]. Another study proposes CaPBug, a framework for automatically classifying bugs using machine learning techniques such as Multinomial Naive Bayes, Random Forest, Decision Trees, and Logistic Regression [10]. Though this framework outlines the task type, we aim to conduct a more in-depth analysis of the specific technology involved in the issue.

Another branch of research employs pre-trained models to improve the accuracy of source code comprehension. Karmakar and Robbes [11] compared several pre-trained models (i.e., BERT, CodeBERT, CodeBERTa, and GraphCodeBERT). Their findings indicate that GraphCodeBERT outperforms other models in coding tasks, while BERT performed well in some other instances. This suggests that models pre-trained solely on natural language can still capture some software engineering semantics. Moreover, one relevant study demonstrated that CodeBERT, when specifically trained on source code, outperforms natural language models like RoBERTa in detecting software errors [12]. The results show the importance of task-specific pre-training in improving code understanding. Our approach builds upon these findings by utilizing BERT-like models for tasks that involve both source code and natural language inputs, such as task descriptions. By combining specialized terminology with code examples, we aim to improve the accuracy of programming language classification.

Before the pre-trained models, classical natural language processing techniques predominated in code-related tasks. From this perspective, a relevant study utilized statistical models, including n-grams, skip-grams, and Multinomial Naive Bayes (MNB), to classify programming languages from code fragments within GitHub data [13]. Similarly, Odeh et al. [14] used MNB classifiers for automatic programming language detection, demonstrating their effectiveness in editors such as Visual Studio Code or Notepad+. The authors of [15] used Support Vector Machines (SVMs) to classify source code into programming languages, demonstrating that traditional machine learning techniques can successfully perform this kind of task.

Another relevant study uses Long Short-Term Memory (LSTM) networks to automatically classify programming languages, receiving increased accuracy compared to the MNB method [16]. However, as noted by the authors of [17], such methods still exhibit lower efficiency in the absence of a natural language description accompanying the code. Our work addresses this limitation by integrating textual context in the form of task descriptions and summaries alongside code fragments. By this, our approach enables more accurate classification compared to the methods that rely only on code.

This paper provides an improved model for identifying programming skills used in tasks from our previous study [18], increasing the F1 score and incorporating additional classification labels.

2.2 Data-Driven Expertise Level Assessment

Understanding the expertise of a developer depends on the type of task he has worked on. Wysocki et al. [19] introduced a new concept for classifying project tasks to optimize management support systems within the software engineering industry. Their work suggests the importance of task-related data and presents a classification method for categorizing tasks. Another similar study highlights the importance of metadata used to classify Jira tasks [20]. Here, the authors managed to automatically assign Jira issues to developers. They utilized Latent Dirichlet Allocation (LDA) to perform topic modeling on the Apache dataset. A mathematical method is proposed in [21], where the authors implement an expertise recommender based on task descriptions from Jira. They use the term frequency-inverse document frequency (tf-idf)

method to recommend experts. In our study, we also focus on classifying expertise based on task categories utilizing the available metadata recorded by project management tools.

Two relevant studies highlight the benefits of systematically analyzing project repositories to obtain insights into software development processes [22,23]. They emphasize that issue tracking platforms contain rich metadata that can be mined to characterize developer activities and collaboration patterns. By utilizing this information, project monitoring can be improved, and behavioral trends among developers can be identified. Such insights provide a valuable foundation for our study to identify relevant attributes for assessing expertise.

Beyond software tools, an interesting method for identifying experts within the multisensor data fusion framework was proposed by Moreira and Wichert [24]. They define three sensors, each operating independently and presenting different candidates. The results from all sensors are combined using Shannon's Entropy formula to produce a final list of experts.

In social networks, the authors of [25] propose a method for identifying social media application users who can offer professional answers to questions posed by other users. They create dynamic profiles for each user based on data obtained from social network applications, as well as other metrics such as user quality and trust, using mathematical methods. McLean et al. [26] expand expertise identification to organizations by using corporate data, more specifically documents, as evidence of expertise. They extract data from the web to create a profile for each user. This data is then used to calculate a score for each individual. ExpFinder is another expert recommender system that utilizes datasets from the academic field [27]. Here, the authors use both an N-gram vector space model and μCO-HITS, comparing them with other expert finder models and demonstrating the effectiveness of their approach.

Rather than concentrating solely on programming languages or bug categorization, we analyze a wider range of inputs, including communications, comments, and task metadata, to create a comprehensive algorithm for calculating the expertise-related scores of each developer.

3 Software Development Expertise Formalization

Individual expertise in the software development industry can be perceived from two different points of view:

(i) **expertise in a specific technology**, which indicates the level of in-depth knowledge and skills related to a specific area, such as Java programming, data science, database management, computer vision, cybersecurity, mobile development, WordPress, etc. This type of expertise is essential for solving specific technical tasks.

(ii) **generalized expertise**, which refers to a wide range of skills, knowledge, and experiences that are applicable across various fields. It encompasses a broad knowledge base and abilities such as analytical thinking, problem-solving, adaptability, creativity, flexibility, etc.

Both components are essential in building high-performing teams for new projects. Expertise in a specific area can enable the solving of complex problems, the application of best practices based on technicality, and the optimization of processes. Having solid experience in specific technologies is beneficial for working with the latest technologies and addressing industry requirements. On the other hand, generalized expertise can contribute to adaptability across various themes and integrate diverse knowledge to develop innovative solutions. It may allow professionals to work efficiently in interdisciplinary teams and understand the broader context of a project.

The two types of expertise can be evaluated from historical data stored in company databases. By analyzing tasks from past projects, organizations can evaluate the expertise level of their employees to make informed decisions when forming teams.

The first step in developing expertise models involves identifying the relevant fields within historical data that should be considered. Table 1 presents the relevant features (i.e., fields) we identified in this respect within four project management tools: Jira, Trello, Asana, and Azure DevOps. The first column presents the features, while each subsequent column indicates whether these fields are available in the respective project management tool. If the field name differs from the specified one, we provided the tool's annotation for clarity.

Table 1: Features in different project management tools

Feature	Jira	Trello	Asana	Azure DevOps
ID	✓	✓	✓	✓
Title	Summary	Name	Name	✓
Description	✓	✓	Notes	✓
Comments	✓	✓	Stories	✓
Attachments	✓	✓	✓	✓
Labels	✓	✓	Tags	Tags
Assignee	✓	Member	✓	✓
Reporter	✓	x	x	✓
Start date	✓	✓	✓	✓
Due date	✓	✓	✓	✓
Parent	✓	x	✓	✓
Subtask	✓	Checklist	✓	Child
Type	Issuetype	x	x	Workitemtype
Priority	✓	x	✓	✓
State	✓	✓	Section	✓
Connections	Issuelink	x	✓	Relations
History	Changelog	Actions	Stories	✓
Time spent	✓	x	✓	CompletedWork
Story points	✓	x	✓	✓
Automation	✓	✓	✓	✓

Note: Underlined features mark the components used in expertise formalization.

We then modeled the expertise of a developer based on a set of carefully designed mathematical formulas that incorporate the various features we identified inside the issue-tracking tools. We specifically targeted information extracted from Jira, but the obtained expertise model can be easily adapted to other tools, including Trello, Asana, or Azure DevOps. These formulas consider key metrics such as task connections, the total number of task reopens, the size of the changelog, the number of comments exchanged between developers, the count of subtasks, the time spent on the issue, the task's priority, and the technology or programming language utilized in the task. By analyzing these parameters, we aimed to classify the developers based on their expertise in a particular area as 'novice', 'intermediate', or 'master', and based on their general expertise as 'junior', 'mid', or 'senior'.

3.1 Features Relevant to the Assessment of Expertise

We have identified nine features that may be used when formalizing the expertise of a developer. These features are underlined in Table 1 and briefly presented in Section 3.1. To maintain consistency across different scales, each value corresponding to these attributes will be normalized using a normalization method. We selected specific normalization methods best suited for the characteristics of each feature, but the approach is flexible and can support alternative normalization strategies.

3.1.1 Comments

The field *Comments* contains an array of JavaScript Object Notation (JSON) objects, each object storing information related to a comment between the developers involved in the given task. A higher number of comments may indicate increased task-related activity, and by this, a higher complexity. To express this variable, we used Eq. (1):

$$\varphi_{1,i} = \underset{i=1,\ldots,m}{normalize}(cmn_i) \tag{1}$$

where cmn_i represents the total number of comments within task i obtained by counting the number of the objects included in the *Comments* array, m is the total number of tasks, while *normalize* is a normalization function that outputs a value between 0 and 1. A suitable selection of this function is the min-max normalization because the data follows a consistent scaling while preserving the relative differences.

3.1.2 Subtask & Parent

The *Subtask* field contains an array of JSON objects that encompasses information about the subtasks of the current task (e.g., ID, name, project ID, description, type), while *Parent* is a JSON object with information regarding the parent task. Analyzing the content of both the *Subtask* and *Parent* we devise the s_i complexity-related parameter belonging to task i, as presented in Table 2.

Table 2: Subtask-related value definition

Condition	s_i	Complexity of the task
Subtask and *Parent* are empty	0	Easy
Parent is populated	1	Medium
Subtask is populated and *Parent* is empty	2	High

Afterward, the s_i value will be normalized using a predefined normalization function (2):

$$\varphi_{2,i} = \underset{i=1,\ldots,m}{normalize}(s_i) \tag{2}$$

In this case, a proper selection is the min-max normalization function.

3.1.3 Type

The *Type* field contains a string that describes the category of the task that, in our view, can be associated with a precise value of a complexity-related parameter, denoted by it_i. For example, if the *Type* field is populated with one of the *subtask*, *task*, *story*, *bug*, or *epic* strings, the it_i parameter may take the values presented in Table 3.

Table 3: Task-related value definition

Type	it_i	Complexity of the task
Subtask	0	Very easy
Task	1	Easy
Story	2	Medium
Bug	3	High
Epic	4	Very high

The it_i value is normalized to the $[0,1]$ interval using Eq. (3):

$$\varphi_{3,i} = \underset{i=1,\ldots,m}{normalize}(it_i) \tag{3}$$

a suitable choice in this regard being the min-max normalization.

3.1.4 Priority

The *Priority* field describes the urgency of a task using predefinite strings. Most projects use the following priority levels: *none*, *low*, *medium*, and *high*. These correspond to 0, 1, 2, and 3 priority-related values p_i, respectively. Some projects have proposed other items to define priority, for example, the Apache project uses the following categories: *none*, *trivial*, *minor*, *major*, *critical*, and *blocker*. In this case, the associated p_i values may be set to 0, 1, 2, 3, 4, and 5, respectively.

In order to rescale the p_i values to the $[0,1]$ interval, we may use the Eq. (4):

$$\varphi_{4,i} = \underset{i=1,\ldots,m}{normalize}(p_i) \tag{4}$$

where a suitable selection for the normalization function is the hyperbolic tangent, because a nonlinear scale might better reflect the escalating impact of the priority levels.

3.1.5 Connections

The *Connections* field is an array of JSON objects containing information about connected tasks or issues, such as issue links (e.g., the task is blocked by a specified task), parent-child relationships with other tasks, connections with external tools. The number of connections obtained by counting the number of JSON objects indicates the level of dependency other tasks have on the completion of task i. This integer, denoted by cn_i, may be normalized using Eq. (5):

$$\varphi_{5,i} = \underset{i=1,\ldots,m}{normalize}(cn_i) \tag{5}$$

In this case, an appropriate choice for the normalization function is min-max normalization.

3.1.6 History

The *History* field is an array of objects that records all changes made during task completion. We define a rn_i metric, which represents the number of reopens within task i and is calculated by counting the number of *History* objects that include the *status* subfield with the value "Reopened". The number of reopens is an indicator of complexity, which means that a task i was not entirely solved from the first attempt. A value of

0 indicates that the task was never reopened, while higher values show that the task i was reopened multiple times. The rn_i parameter is normalized using Eq. (6):

$$\varphi_{6,i} = \underset{i=1,\ldots,m}{normalize}(rn_i) \tag{6}$$

where a suitable choice for the normalization function is the hyperbolic tangent.

From another perspective, a large number of updates or transitions recorded in the *History* field of a task indicates a higher complexity. In this context, we define a new complexity-related parameter, namely chn_i, obtained by counting the number of changes that occur within the task (i.e., the number of objects in *History*). The chn_i may be rescaled to $[0,1]$ interval using Eq. (7):

$$\varphi_{7,i} = \underset{i=1,\ldots,m}{normalize}(chn_i) \tag{7}$$

where a proper selection is the sigmoid normalization function because it has a diminishing effect as it increases.

3.1.7 Time Spent & Story Points

Some projects enable time-tracking by allowing the developers to use the *Time spent* field. This field contains an integer that represents the total working time in seconds, which the developer logs while working on a task. To enhance clarity regarding the range, we convert the time spent to hours and denote this time parameter as t_i. For example, an easy task may take between 0 and 4 h, while a more complex task could require up to 40 h. The t_i parameter is normalized using Eq. (8):

$$\varphi_{8,i} = \underset{i=1,\ldots,m}{normalize}(t_i) \tag{8}$$

where the normalized value can be obtained using a sigmoid function.

Other projects utilize a story point methodology by recording the associated information in the *Story point* field. This field is an integer where a higher value assigned to a task may indicate a more complex task. In this case, we denote the complexity-related parameter as sp_i, with its normalized value given by Eq. (9):

$$\varphi_{9,i} = \underset{i=1,\ldots,m}{normalize}(sp_i) \tag{9}$$

A suitable selection of the normalization function is the min-max normalization.

3.2 Expertise Formalization

We intend to devise formulas regarding the two facets of the developer's expertise:

- **Technology-specific expertise**–refers to in-depth knowledge, skills, and experience within a specified field related to software development. Individuals recognized for such expertise are highly specialized in effectively solving problems within their domain by leveraging a deep understanding of complexities and nuances.
- **Generalized expertise**–covers a wide range of skills and knowledge spanning multiple technologies, rather than being confined to a single area. Professionals backed by such expertise excel at bridging ideas across fields, adapting to various contexts, and responding to problems with creative and flexible thinking.

Both these expertise forms can be assessed based on the developer's historical performance within tasks previously recorded by project management tools.

3.2.1 Task Complexity

In order to define the expertise, we introduce task complexity (TC) as a helping variable to capture the overall complexity of a task i. This indicator is a linear combination of the variables extracted from the project management tool databases and presented in Section 3.1, namely $\varphi_{k,i}$, with $k = 1, \ldots, 9$:

$$TC_i = w_1 \cdot \varphi_{1,i} + w_2 \cdot \varphi_{2,i} + w_3 \cdot \varphi_{3,i} + w_4 \cdot \varphi_{4,i} + w_5 \cdot \varphi_{5,i} + w_6 \cdot \varphi_{6,i} + w_7 \cdot \varphi_{7,i} + w_8 \cdot \varphi_{8,i} + w9 \cdot \varphi_{9,i} \quad (10)$$

where w_k are the weights used to control the balance between the components. If the sum of all the w_k weights is equal to 1, each w_k may be interpreted as the percentage by which the associated variable is taken into consideration.

3.2.2 Developer's Technology-Specific Expertise

We aim to classify the developer's level of expertise, denoted by TSEL (Technology-Specific Expertise Level) in three categories (i.e., *novice*, *intermediate*, and *master*), by considering both the number and complexity of tasks already completed and recorded in project management tool databases. Our proposed methodology is implemented in two steps: (i) finding the number of tasks corresponding to each category of task complexity that the developer had worked on; and (ii) assigning the developer's level of expertise.

In the first step, we use a classic statistical approach to classify the tasks related to a given software technology in three categories, namely: *easy*, *moderate*, and *complex*. This approach groups the tasks based on how their complexity deviates from the mean [28], and starts by computing the mean μ_e and standard deviation σ_e using the Eqs. (11) and (12):

$$\mu_e = \frac{1}{m} \sum_{i=1}^{m} TC_{i,e} \quad (11)$$

$$\sigma_e = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (TC_{i,e} - \mu_e)^2} \quad (12)$$

where $TC_{i,e}$ is the task-complexity variable calculated for the task i that uses the specified technology e, while m represents the total number of tasks corresponding to technology e that were previously covered by the project management tool.

Then, each task i related to the specified area e is categorized to be *easy* if $TC_{i,e} < \mu_e - \sigma_e$; *moderate* if $\mu_e - \sigma_e < TC_{i,e} < \mu_e + \sigma_e$; or *complex* if $TC_{i,e} > \mu_e + \sigma_e$. By counting the number of tasks in each of these three categories we will obtain the following variables: E—the number of easy tasks, M-the number of moderate tasks, and C—the number of complex tasks the developer has worked on.

In the second step of the methodology we define the developer's technology-specific expertise score γ in Eq. (13):

$$\gamma = \frac{v_E \cdot E + v_M \cdot M + v_C \cdot C}{m} \quad (13)$$

where v_E, v_M, and v_C are chosen weights to reflect the importance of each type of tasks in the computed expertise score. A suitable selection of these parameters may be: $v_E = 0.1$, $v_M = 0.5$, and $v_C = 1$.

Finally, a developer can be classified as *novice*, *intermediate*, or *master* in the technology denoted by e using the thresholds γ_{ni} and γ_{im} as described by Eq. (14):

$$TSEL = \begin{cases} novice, & 0 \le \gamma < \gamma_{ni} \\ intermediate, & \gamma_{ni} \le \gamma < \gamma_{im} \\ master, & \gamma_{im} \le \gamma \le 1 \end{cases} \tag{14}$$

where the thresholds γ_{ni} and γ_{im} may be reasonably selected to be 0.4 and 0.7, respectively.

3.2.3 Developer's Generalized Expertise

Our objective is to classify the developer's overall expertise level, denoted by GEL (Generalized Expertise Level) in three categories (i.e., *junior*, *mid*, and *senior*), by considering two aspects, namely the average technology-related expertise score and the maximum technology-related expertise score.

In the first step, we calculate the developer's technology-specific expertise scores γ_k with $k = 1, \ldots, K$ for each of the K technologies identified in the project management tool database using (13). We organize these technology-specific expertise scores in an array Γ, where the γ_k values are stored in descending order. The Γ array can be mathematically formalized as follows: $\Gamma = \{\gamma_k^{\downarrow}\}_{k=1}^{K}$, where $\{\gamma_i^{\downarrow}\}_{k=1}^{K}$ is a permutation of $\{\gamma_k\}_{k=1}^{K}$ such that: $\gamma_1^{\downarrow} \ge \gamma_2^{\downarrow} \ge \cdots \ge \gamma_K^{\downarrow}$. This array will allow us to compute two components used in devising the developer's generalized expertise score δ, namely the developer's maximal technology-related expertise score θ and the average developer's technology-related expertise score v using the Eqs. (15) and (16):

$$\theta = max(\Gamma) = \gamma_1^{\downarrow} \tag{15}$$

$$v = \frac{1}{p}\sum_{j=1}^{p}\gamma_j^{\downarrow} \tag{16}$$

where p is the number of different technologies taken into consideration. It is noteworthy to mention that in order to have a clear perspective on the developer's expertise, the number of technologies p that need to be taken into account when computing the average v (we need to consider the first p technologies from the Γ array) have to be selected such as $3 < p < 10$, a suitable value being $p = 5$.

The developer's generalized expertise score is determined using Eq. (17)

$$\delta = \begin{cases} 0, & \gamma_p^{\downarrow} = 0 \\ \alpha \cdot \theta + (1-\alpha)\cdot v, & \gamma_p^{\downarrow} \ne 0 \end{cases} \tag{17}$$

where α is a parameter that controls the balance between the two components, a suitable value being $\alpha = 0.6$. As it can be observed, in the case that $\gamma_p^{\downarrow} = 0$ (i.e., the number of technologies where the developer has a non-zero expertise score is less than p), we threshold the overall expertise score δ to 0, to signalize that the respective developer has not a sufficiently diversified expertise.

Using the overall expertise score provided by (17), we classify a software specialist as *junior*, *mid*, or *senior* using two thresholds, namely δ_{jm} and δ_{ms}, as described by Eq. (18):

$$GEL = \begin{cases} junior, & 0 \le \delta < \delta_{jm} \\ mid, & \delta_{jm} \le \delta < \delta_{ms} \\ senior, & \delta_{ms} \le \delta \le 1 \end{cases} \tag{18}$$

where the thresholds δ_{jm} and δ_{ms} may be reasonably selected to be 0.4 and 0.7, respectively.

4 Proposed Methodology

The formation of competent software development teams or the replacement of existing members often depends on the ability to identify and utilize individual expertise. Within the teams, expertise is typically associated with different roles, such as junior, mid, and senior positions, each contributing at various stages of the software development process. However, role definitions are not always clearly stated in the project management tools, leading to challenges in team management and formation by the human resources team.

A significant challenge in this respect is identifying the skills and technologies associated with a software developer from recorded historical data. In order to achieve this, we have to classify the tasks a developer has worked on taking into consideration two different perspectives: (a) the technologies that were employed during task completion; and (b) the level of complexity the task has.

To determine the developer's level of expertise in a given software technology or the overall expertise in the field, we have developed a carefully crafted methodology containing three stages that are presented in detail in the following Sections 4.1–4.3.

4.1 Stage 1: Identifying the Domains of Expertise from Project Management Tool Databases

Our methodology starts with identifying all software technologies that were employed during the completion of the tasks recorded by the project management tool database. The procedure, described in Fig. 1, utilizes Natural Language Processing (NLP) techniques (e.g., record filtering, field concatenation, transformer-based keyword extraction, knowledge transfer) to associate a software technology with each task, thereby providing a list of all previously used technologies.

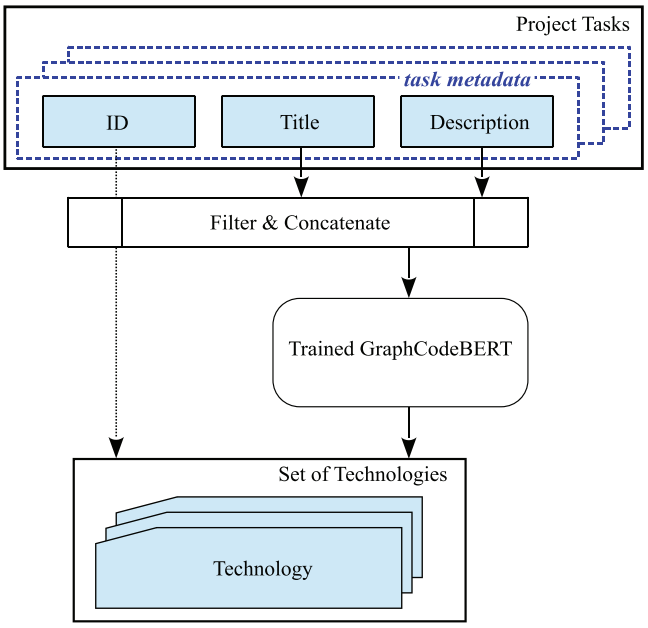

Figure 1: Obtaining the set of technologies

The procedure starts with acquiring three fields corresponding to each task recorded in the database, namely *ID*, *Title*, and *Description*. The records containing at least one empty field are filtered out and, for the remaining records, the *Title* and *Description* string-type fields are concatenated. By using a carefully trained GraphCodeBERT model, we associate a single most relevant technology to each task. Finally, a list of previously employed technologies is output.

Details regarding the selection of the model to be employed in extracting the software technology from the task *Title* and *Description* (i.e., GraphCodeBERT) and its training process are provided in the following paragraph.

Finding the Best Model to Identify the Technologies

We have trained several ML models to better understand the programming languages and technologies relevant to each task a software developer has worked on in order to assess the expertise. The utilized models are: BERT—a bidirectional encoder that has been trained on diverse data [29]; ALBERT—builds on the BERT architecture to improve training speed while maintaining prediction performance [30]; RoBERTa—an optimized version of the BERT model [31]; CodeBERT—a variant of BERT that has been specifically improved for programming understanding [32]; GraphCodeBERT—further upgrades CodeBERT by utilizing data flow graphs [33]; and, XLNet—uses a different architecture and was introduced to compare the performance of BERT-like models against a different architectural approach in terms of prediction accuracy [34]. By doing this, we not only focused on programming-specific models such as CodeBERT and GraphCodeBERT, but we also explored the effectiveness of other general language models, including BERT-like models like BERT, RoBERTa, and ALBERT, as well as different architectures such as XLNet.

One of the main challenges we had to solve was to determine the type of classification the model has to face. There are two main classification methods: single-label and multi-label. The single-label classification can be categorized into binary and multi-class classification. In our case, binary classification, which involves selecting one result from two options, is not suitable because we have multiple possible outcomes. Therefore, we have to choose to focus on either multi-class classification, where we choose a single label from a set of tags, or on multi-label classification, where we can select multiple tags that fit the input [35]. By examining the records provided by the project management tools, since these records generally lead to single labels and the datasets are too imbalanced to be used for a multi-label classification, we concluded that a multi-class classification is more appropriate in our case.

Another challenge that has arisen is the imbalance of the dataset. Relevant studies recommend using the class weights method to address this problem [36,37]. Even with balanced datasets, there can still be discrepancies between certain labels. In this situation, we used the class weights technique to improve the precision of the classes that appear less frequently. Furthermore, we have incorporated a label smoothing methodology. As suggested by one study, we applied label smoothing to our models to make the labels more distinguishable [38].

To implement our study, we loaded the mentioned ML models using the Hugging Face Transformers library [39] and selected the cross-entropy loss function due to its effectiveness in multiclass classification tasks and the SoftMax activation function to convert the raw output into probabilities for each class. The models were then applied to a Stack Overflow dataset, described in detail in the following paragraphs, that was previously split as follows: 70% for training, 20% for validation, and 10% for testing. The data was then applied to the method "AutoTokenizer", which automatically selects the appropriate tokenizer for each model. This process transforms the raw sequence from each text input into individual tokens. Since each of the loaded ML models has a maximum token length of 512 tokens and we have observed that more than half of the inputs exceed this token limit, we employed the sliding-window approach described in [40] to cope with higher numbers of tokens.

Since there are no available labeled datasets of project management tools with tasks linked to specific programming languages, we utilize a dataset of StackOverflow questions and answers, along with their associated tags, to fine-tune the ML models. The StackOverflow dataset contains 50.000 rows of questions obtained through the StackExchange API. For this, we used the endpoint https://api.stackexchange.com/2.

3/questions (accessed on 12 October 2025) with the following parameters: 'page', 'pagesize', 'order', 'sort', 'site', and 'filter'. A Python script was implemented to fetch 100 records per page in descending 'desc' order, sorted by 'creation', from the site 'stackoverflow', applying the filter 'withbody'. Each entry includes a unique question ID, the title of the question, a description, tags that classify the question into specific topics, and the creation time, indicating when the question was first published. An illustrative example is presented in Table 4.

Table 4: StackOverflow record example

Field	Data type	Example
Question ID	Long	34553034
Title	String	Why are Java Optionals immutable?
Body	String	I'd like to understand why Java 8 Optionals were. . .
Tags	String	<java><optional>
Creation date	Date	01/01/2016 02:03:20

To create a balanced dataset and optimize the learning process, from the original Stack Overflow dataset, we retained a minimum of 150 and a maximum of 300 records for each tag (i.e., label). In the end, we are left with a dataset consisting of 10,031 records across 44 different tags represented as word cloud in Fig. 2: java, javascript, android, c, jquery, c#, ios, c++, scala, kotlin, mysql, python, php, r, node.js, html, asp.net, typescript, django, css, postgresql, angular, spring, ruby, sql, reactjs, go, ruby-on-rails, react-native, vue.js, oracle,.net, dart, amazon-web-services, azure, tensorflow, docker, wordpress, bash, firebase, powershell, jenkins, matlab, and laravel.

Figure 2: Classification labels

After fine-tuning the six ML models on the mentioned Stack Overflow dataset, we implemented a transfer learning approach by applying them to Apache, Atlassian, and Moodle projects from the Jira dataset (available at the following links: https://issues.apache.org/jira/, https://moodle.atlassian.net/jira/, https://jira.atlassian.com/projects/, acceessed on 12 October 2025). Our goal was to predict the technologies utilized in the tasks a developer has worked on and, for this reason, we manually labeled 2561 Jira records, an example containing only the relevant fields being presented in Table 5.

The comparative results of using the six ML models are presented in Table 6. The training and testing process was conducted using Python 3.11 and PyTorch Lightning library trainer, with a batch size of 64 and Adam optimizer, on a Lenovo Legion equipped with a 6-core i7 2.60 GHz processor, 32 GB of RAM, and an NVIDIA GeForce GTX 1650 Ti GPU. Each model was fine-tuned over thirty epochs, applying early stopping techniques.

Table 5: Jira dataset

Parameter	Data type	Example
Task ID	String	13209399
Assignee ID	Int	4
Title	String	I get this error when I preview Options
Description	String	java.lang.NullPointerException at org.netbeans.modules . . .
Technology	String	java

Table 6: BERT, RoBERTa, ALBERT, XLNet, CodeBERT, and GraphCodeBERT results

Model	Stack overflow dataset	Jira dataset
BERT	Accuracy: 0.885 F1 score: 0.862	Accuracy: 0.702 F1 score: 0.689
RoBERTa	Accuracy: 0.946 F1 score: 0.941	Accuracy: 0.751 F1 score: 0.734
ALBERT	Accuracy: 0.935 F1 score: 0.929	Accuracy: 0.730 F1 score: 0.704
XLNet	Accuracy: 0.956 F1 score: 0.948	Accuracy: 0.793 F1 score: 0.793
CodeBERT	Accuracy: 0.978 F1 score: 0.972	Accuracy: 0.838 F1 score: 0.836
GraphCodeBERT	**Accuracy: 0.987** **F1 score: 0.986**	**Accuracy: 0.872** **F1 score: 0.867**

Note: The model in bold exhibits the best performance.

As shown in Table 6 and Fig. 3, all the models perform well on the training dataset. However, to determine the best-performing models, we had to test them on the Jira dataset. The table highlights notable differences in performance, with GraphCodeBERT achieving the highest accuracy, followed closely by CodeBERT. XLNet performs slightly worse than these two, but its results are still acceptable. The models with the lowest performance are BERT, ALBERT, and RoBERTa. On the other hand, they still achieve an accuracy of more than 70%, indicating that these base language models can successfully learn the technical content of software tasks.

4.2 Stage 2: Forming the De-Identified List of Developers

In order to obtain the list of developers that worked on already-completed tasks, a simple procedure, described in Fig. 4, can be used.

After acquiring the task's *ID* and *Assignee* fields for each recorder task, we filter out the records where the *Assignee* is unpopulated (i.e., 'unassigned'). Individuals who have a track record of completing tasks are included in the list of developers, which must later undergo a de-identification process, where names are replaced with unique identifiers to comply with privacy protection policies.

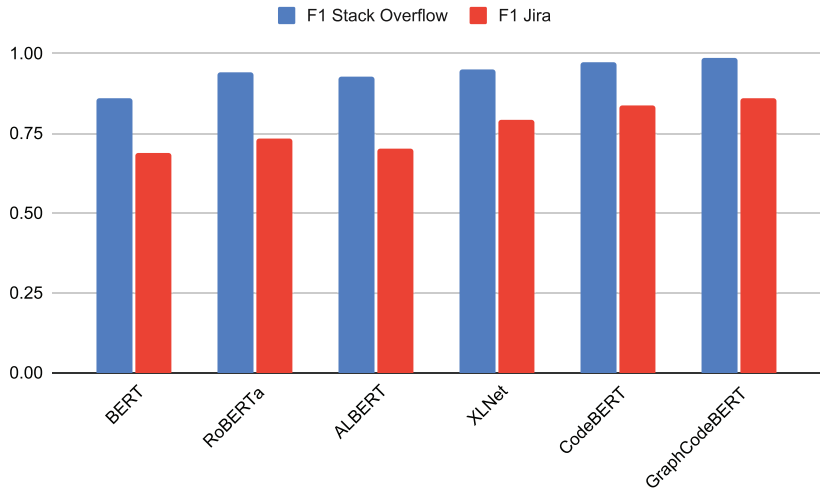

Figure 3: Model F1 scores across datasets

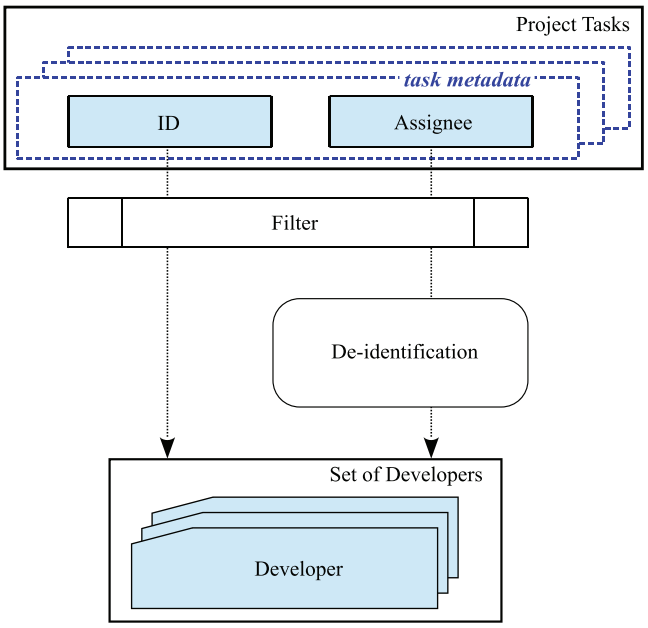

Figure 4: Obtaining the de-identified list of developers

4.3 Stage 3: Computing the Developers' Expertise

This stage is the kernel of our methodology, computing the technology-related and generalized forms of expertise for each developer identified in the second stage. The procedure is presented in Fig. 5.

For each task, a set of nine fields (i.e., *Comments*, *Parent*, *Subtask*, *Type*, *Priority*, *Connections*, *History*, *Time spent*, and *Story points*) is acquired. After filtering out the corrupted or inaccurate records, for each developer identified in the second stage, a set of expertise scores for each of the technologies identified in stage 1 and the generalized expertise score are computed using Eqs. (13) and (17). To this array of expertise scores we associate an array of expertise levels using Eqs. (14) and (18).

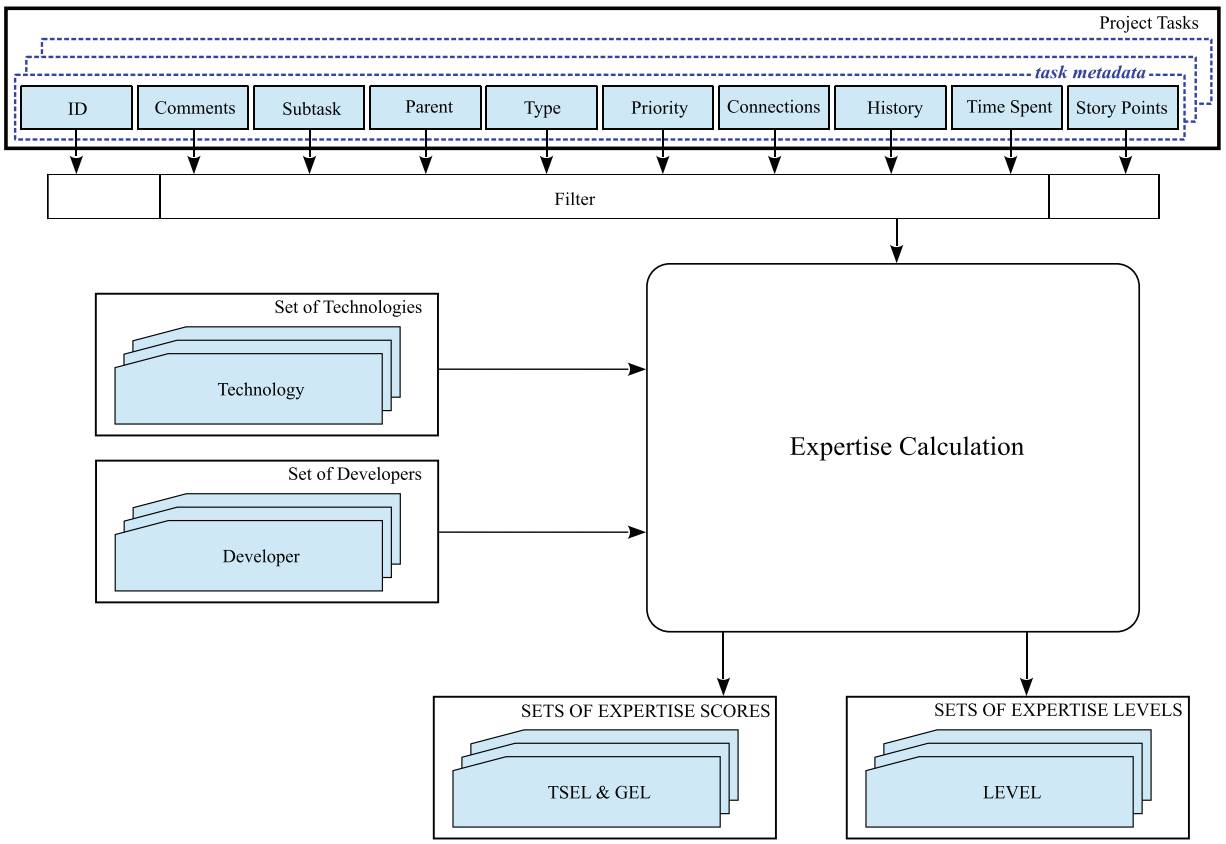

Figure 5: Developers' expertise computation

5 Experimental Case Study

To illustrate the effectiveness of our approach for evaluating software developers' skills, outlined in Section 4, we utilized a Jira dataset. For this, we collected all the 12,912 record data from the Apache project, specifically from the Apache Ignite subproject, which is a distributed database open-source project available at the following link: https://issues.apache.org/jira/browse/IGNITE-25788?jql=project%20%3D%20IGNITE (accessed on 12 October 2025). Additionally, we gathered 11,976 records from the Moodle project, focusing on the MDL subproject, which is an open-source learning platform available at this link: https://moodle.atlassian.net/jira/software/c/projects/MDL/issues (accessed on 12 October 2025).

In our practical example, we highlighted some features with higher weights because we believe they are more important due to their construction and logic. We excluded the Jira record fields (specified in Section 3.1) *Time spent* $\varphi_{8,i}$ and *Story points* $\varphi_{9,i}$ as the majority of tasks do not typically utilize these parameters, this being formalized by considering $w_8 = w_9 = 0$. We consider that the task reopens $\varphi_{6,i}$, extracted from the *History* field, is the most important feature, and therefore, we assign it a weight w_6 of 0.25. The primary reason is that an item marked complete but later reopened indicates unresolved work, additional costs, and potential quality issues. This makes reopens more critical than other fields that do not directly reflect rework or task incompleteness. The next important attributes in our view are *Comments* $\varphi_{1,i}$ and *Priority* $\varphi_{4,i}$, since the number of comments generally serves as an indicator of interaction and collaboration among team members, while the task's priority level shows the urgency and importance of a task within the project. For both of them, we have assigned weights w_1 and w_4 of 0.2. We assigned the weights w_2, w_3, w_5 of 0.1 to the complexity-related information extracted from *Subtask & Parent* $\varphi_{2,i}$, *Type* $\varphi_{3,i}$, and *Connections*

$\varphi_{5,i}$ fields, these features providing the context about the complexity and interconnection of the tasks. Lastly, the changelog-related parameter $\varphi_{7,i}$, extracted from the *History* field, received the lowest weight w_7 of 0.05. While the changelogs number does provide insights into the task's history, it does not always correlate with the task's importance (changes may indicate shifts between different statuses without necessarily implying that the task has been resolved or is of high priority). All the other parameters and normalization functions involved in our computations are those provided as default inside Section 3.

Following the first step of the methodology, described in Section 4.1, we obtained a list of 18 software technologies (e.g., Java, Python, Javascript, SQL) that were employed in the completion of the recorded 12,912 tasks. Furthermore, we identified 10 software technologies (e.g., PHP, MySQL, Android, iOS) associated with the 11,976 tasks in the Moodle project. Supplementary, the complete list of developers containing 257 anonymized professionals for the Apache project and 31 for the Moodle project was provided at the end of the second stage of the methodology as presented in Section 4.2.

For each identified developer, a set of expertise scores γ corresponding to all technologies identified in the first stage and the generalized expertise score δ are computed using Eqs. (13) and (17). Correspondingly, to every γ and δ values an expertise level is associated using Eqs. (14) and (18). We exemplify some expertise scores in Tables 7 and 8.

Table 7: Developers' expertise scores in Apache Ignite project

Candidate ID	γ_1 (Java)	γ_2 (C++)	γ_3 (C#)	γ_4 (Python)	γ_5 (Node.js)	γ_6 (Docker)	...	γ_{18} (SQL)	δ
1	0.337	–	–	0.705	–	–	...	–	0.0
2	0.237	–	–	–	–	–	...	–	0.0
3	–	–	–	–	–	–	...	0.841	0.0
4	–	0.582	0.565	0.087	–	–	...	0.275	0.556
...									
43	–	–	–	0.718	–	0.652	...	–	0.627
...									
67	0.346	0.977	0.525	–	–	–	...	0.344	0.862
...									
118	–	–	0.618	–	0.601	–	...	0.482	0.804
...									
257	0.427	–	–	–	–	–	...	–	0.0

For example, in Table 7, the user with ID 5 has the following non-zero results: $\gamma_8 = 0.756$ (Spring), $\gamma_1 = 0.446$ (Java), and $\gamma_{18} = 0.381$ (SQL), all the other technology-related expertise scores being 0. Since he only worked with 3 technologies, which is below the threshold $p = 5$, his generalized expertise score will be $\delta = 0$ according to Eq. (17). On the other hand, the developer with ID 67 achieved a generalized score expertise $\delta = 0.862$, since his technology-related expertise scores have non-zero values for 7 technologies: $\gamma_2 = 0.977$ (C++), $\gamma_7 = 0.832$ (.net), $\gamma_{10} = 0.776$ (HTML), $\gamma_3 = 0.525$ (C#), $\gamma_1 = 0.346$ (Java), $\gamma_{18} = 0.344$ (SQL), and $\gamma_{12} = 0.015$ (Javascript). The result does not consider γ_{18} and γ_{12} because we only calculate it based on the first $p = 5$ technologies. The same logic applies to Table 8, where the user with ID 1 has achieved a score of 0.706, with expertise scores across 5 technologies: $\gamma_1 = 0.835$ (PHP), $\gamma_5 = 0.760$ (CSS), $\gamma_3 = 0.563$ (MySQL), $\gamma_2 = 0.493$ (SQL), and $\gamma_2 = 0.471$ (HTML).

Table 8: Developers' expertise scores in Moodle project

Candidate ID	γ_1 (PHP)	γ_2 (SQL)	γ_3 (MySQL)	γ_4 (HTML)	γ_5 (CSS)	γ_6 (iOS)	...	γ_{10} (Android)	δ
1	0.835	0.493	0.563	0.471	0.760	–	...	–	0.706
2	0.29	–	–	–	–	–	...	–	0.0
3	0.476	0.199	–	–	–	–	...	–	0.0
4	0.144	–	–	–	–	–	...	–	0.0
...									
14	0.525	0.555	–	0.670	–	0.422	...	0.391	0.607
...									
31	0.641	0.145	–	0.648	–	–	...	–	0.0

By investigating the results presented in Tables 6–8, the following observations are worth mentioning:

(1) We successfully classified a task's technology based on task descriptions. The machine learning model used in our study can accurately identify the programming languages associated with a given task's title and description. In Table 6, we observe that the GraphCodeBERT model achieved an F1 score of 0.986 on the StackOverflow dataset and 0.867 on the Jira dataset. This demonstrates its high accuracy and its effectiveness in transferring knowledge across IT-related datasets.

(2) Models trained specifically on software engineering data demonstrate superior performance in classifying programming technologies. While BERT, ALBERT, RoBERTa, and XLNet can achieve a prediction confidence between 70% and 80%, they are outperformed by models that were specifically trained on software-related terminologies, such as CodeBERT and GraphCodeBERT. These models were pre-trained on six programming languages: Python, Java, JavaScript, PHP, Ruby, and Go [32,33]. As indicated by other studies, including [11], we have demonstrated that GraphCodeBert performs better on code-related assignments.

(3) We can distinguish between several types of developers. By analyzing both general knowledge and technology-specific expertise we can differentiate between various types of developers. A candidate with good general expertise but limited technology-specific knowledge is likely a versatile developer who can work across different projects but may need to specialize more in a particular technology. On the other hand, a candidate with high expertise in a specific technology but weaker general knowledge might struggle with wider software development projects. A balanced developer may excel in both areas, with a deep understanding of their primary technology while also being capable of solving complex, cross-domain problems efficiently. By applying this expertise evaluation methodology, developers are classified in a measurable and objective manner. Developers with the IDs 43, 67, and 118 from Table 7 and ID 1 from Table 8 have high general expertise, making them suitable for multiple roles. The same candidates also demonstrate a deep specialization in at least one particular technology, easily solving problems within the domain. Across the datasets, we found 16 developers having the *'senior'* level from the overall expertise point of view according to (18), the rest having a *'master'* or *'intermediate'* level in at least one domain according to (14).

We can also observe that the number of developers involved in creating distributed databases, such as Apache Ignite, is greater and spans multiple low-level technologies, including C++, Java, C#, and Docker. On the other hand, the Moodle platform has fewer developers, with a smaller pool of senior professionals. These developers focus on web programming languages like PHP and HTML, as well as mobile platforms such as

iOS and Android. While the tasks in both areas may seem similar, the complexity of low-level applications makes them more challenging to maintain and develop, requiring a higher level of expertise and knowledge.

(4) The developers that worked in the considered project fall mostly between *'senior'* and *'mid'* level on the general knowledge scale, and between *'intermediate'* and *'master'* on the specific domain knowledge scale. This pattern might be because the analyzed project is open source, which usually attracts developers with a higher level of expertise, as they volunteer their contributions. On the other hand, the results could vary regarding seniority levels in a closed-source environment.

(5) To the best of our knowledge, this is the first formalization of expertise from project management tools databases within the software engineering domain. While our work draws inspiration from prior research in expertise identification and task metadata [20,22,23], we aimed to advance these approaches by offering a more quantitative and scalable solution. For example, ExpFinder is a modern tool designed to identify experts in scientific fields based on their publications, while tools like CaPBug categorize and prioritize bugs using the metadata associated with tasks. Although both tools utilize informative metadata, they differ from our approach, as we evaluate developers' expertise based on metadata that reflects their actual interactions with the tasks. Also, instead of focusing solely on semantic similarity or expert rankings, we propose a methodology that combines programming technology identification, task complexity modeling, and mathematical formalization of expertise, enabling a more structured evaluation of software expertise.

6 Research Implications

The research implications provide the scientific relevance of this work and demonstrate how our methodology advances the formalization of developers expertise.

6.1 Implications for Researchers

Our study contributes to the research on data-driven expertise formalization by demonstrating how project management tools' metadata can be efficiently utilized.

From a research point of view, this work provides a foundation for future studies on metadata expertise modeling, the inclusion of additional features relevant to expertise assessment, validation across different domains, or the implementation of real-world services such as Application Programming Interfaces (APIs), dashboard interfaces, and Human Resource (HR) tools. As a direct consequence, the results of this study should be further developed by considering the following future research directions:

- A sensitivity analysis demonstrating the impact of different weight distributions.
- Implementing a multi-label approach to facilitate the assessment of multiple technologies.
- Exploring newer transformer models, such as Bloom, Mamba, or DeepSeek.
- Characterizing seniority levels in more detail.
- Analyzing closed-source projects to identify evolution and differences compared to the proposed open-source method.
- Exploring extensions or tools that connect to project management systems to find additional metadata.
- Developing automatic team formation based on the metadata related to developer expertise.

By expanding the study in these ways, future work can provide deeper insights and help ensure that the proposed methodology remains relevant while supporting developer expertise formalization.

6.2 Implications for Practitioners

While the current version is just a proof of concept, the results suggest several promising possibilities for practical applications. In enterprise settings, this approach could improve team formation, task allocation, or development planning.

In team formation, the proposed methodology can balance the mix of senior, mid, and junior developers while improving productivity. Rather than depending on intuitions, team formation becomes a data-driven process that helps prevent skill shortages and avoids an overconcentration of expertise within a few individuals.

From a task allocation perspective, our method enables project managers to assign work to developers whose expertise scores reflect either their readiness or potential for growth. This strategy boosts efficiency by reserving complex assignments for experienced engineers, but also enables junior developers to solve more challenging tasks with proper supervision. The approach benefits both project success and individual career growth by ensuring employees receive tasks suited to their skill level.

Beyond evaluation and training, this methodology can also support development planning. By tracking expertise trends, managers can plan how teams adopt new technologies, forecast future hiring needs, and identify supplemental training. This method may also reveal which employees are developing competencies aligned with organizational goals, helping to close gaps between current workforce capabilities and future project needs. The resulting expertise scores can serve as advisory results to make HR department practices more transparent and evidence-based, benefiting both the organization and its employees.

7 Threats to Validity

To ensure transparency and rigor, we discuss the main threats to the validity of our study: internal, external, construct, and conclusion validity.

Internal validity threats may arise from incomplete, inconsistent, or malicious entries in project management tools. Our methodology assumes that developers or project managers consistently complete the fields, as missing or incomplete data can decrease the accuracy of the results. Properly maintaining project data is crucial for reusing historical information to generate reliable expertise assessments.

External validity may be affected by differences between open source and closed source projects, such as task allocation, task conventions, or contribution practices. Another challenge could be the evolution of project management tools, which might result in future task metadata differing from that used in our experiment. Additionally, as frameworks and technologies advance in the software industry, models trained on current technologies might struggle with new programming languages, terminology, or context. Therefore, while our expertise formalization works in the examined setting, its applicability to other projects, tools, or time periods remains uncertain.

Construct validity might refer to compromised data, such as incomplete or ambiguous task descriptions, leading the machine learning model to misidentify the technologies related to the tasks. Moreover, when task complexity is inferred from potentially inaccurate or misleading metadata, there is a risk that the measured complexity does not actually align with the true difficulty experienced in practice. If these intermediate constructs are unreliable, the resulting expertise scores may not align with real world proficiency.

Conclusion validity could arise from the incorrect categorization of junior, mid, and senior expertise levels because such a mapping doesn't guarantee that those categories really correspond to actual professional seniority. For example, a developer can receive multiple complex tasks because of team allocation rules, rather than higher expertise, while another may appear as less expert simply due to the participation in activities that are not captured in project management tools.

8 Conclusions

In this paper, we introduced a new approach to quantify the expertise of software developers using task metadata from project management tools. We carefully selected a set of features from Jira databases to define mathematical models for both technology-specific and general expertise. By implementing a transfer learning procedure that utilizes a StackOverflow dataset to fine-tune BERT-like models and classify information from a Jira dataset, we achieved an accuracy of 87% and an F1 score of 0.867 in identifying the software technologies from the title and description of the tasks using the GraphCodeBERT model. We validated this method using a case study, where 288 software professionals worked across 24,888 tasks. Furthermore, we assessed the developers by calculating a domain-specific expertise score for each technology they were working with, and also their overall expertise. These insights can help project managers and HR departments better allocate resources, quickly identify experts, spot potential skill gaps, conduct a more objective evaluation of the employees, and form teams.

In the future, we want to improve our models to identify multiple technologies relevant to each task and to use advanced models such as Bloom, Mamba, or DeepSeek. Additionally, we plan to quantify the seniority levels of senior developers. Currently, developers are considered senior if they achieve a master's level in at least one technology. In the future, we want to explore what differentiates senior developers from other seniors and how they can be a better fit for specific roles by applying bonuses based on the number of tasks, technologies used, or time worked within each project. Furthermore, we intend to adapt this identification module to enable the automatic formation of teams based on different provided inputs.

Acknowledgement: The authors would like to thank the Politehnica University of Timisoara for providing computational support and access to resources required for this research.

Funding Statement: The work of the first two authors, namely Traian-Radu Ploscă and Alexandru-Mihai Pescaru, was supported by the project "Romanian Hub for Artificial Intelligence-HRIA", Smart Growth, Digitization and Financial Instruments Program, 2021–2027, MySMIS No. 334906.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Traian-Radu Ploscă and Daniel-Ioan Curiac; methodology, Traian-Radu Ploscă and Daniel-Ioan Curiac; formal analysis, Traian-Radu Ploscă and Bianca-Valeria Rus; implementation, Traian-Radu Ploscă, Alexandru-Mihai Pescaru, and Bianca-Valeria Rus; validation, Traian-Radu Ploscă, Alexandru-Mihai Pescaru, and Bianca-Valeria Rus; writing—original draft preparation, Traian-Radu Ploscă and Alexandru-Mihai Pescaru; writing—review and editing, Traian-Radu Ploscă, Alexandru-Mihai Pescaru, and Daniel-Ioan Curiac; supervision, Daniel-Ioan Curiac; critical review, Daniel-Ioan Curiac. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available on request from the first author (Email: traian.plosca@aut.upt.ro).

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Heerwagen JH, Kampschroer K, Powell KM, Loftness V. Collaborative knowledge work environments. Build Res Inform. 2004;32(6):510–28. doi:10.1080/09613210412331313025.
2. Wi H, Oh S, Mun J, Jung M. A team formation model based on knowledge and collaboration. Expert Syst Applicat. 2009;36(5):9121–34. doi:10.1016/j.eswa.2008.12.031.
3. Bird C, Gourley A, Devanbu P, Gertz M, Swaminathan A. Mining email social networks. In: Proceedings of the 2006 International Workshop on Mining Software Repositories; 2006 May 22–23; Shanghai, China. p. 137–43.

4. McDonald DW, Ackerman MS. Expertise recommender: a flexible recommendation system and architecture. In: Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work; 2000 Dec 2–6; Philadelphia, PA, USA. p. 231–40.
5. Anvik J, Hiew L, Murphy GC. Who should fix this bug? In: Proceedings of the 28th International Conference on Software Engineering; 2006 May 20–28; Shanghai, China. p. 361–70.
6. Martens B, Franke J. Identifying agile roles in software engineering projects using repository and work-tracking data. In: 2022 International Conference on Data and Software Engineering (ICoDSE); 2022 Nov 2–3; Denpasar, Indonesia. p. 83–8.
7. Alreshedy K, Dharmaretnam D, German DM, Srinivasan V, Gulliver TA. SCC: automatic classification of code snippets. arXiv:1809.07945. 2018.
8. Alrashedy K, Dharmaretnam D, German DM, Srinivasan V, Gulliver TA. Scc++: predicting the programming language of questions and snippets of stack overflow. J Syst Softw. 2020;162:110505. doi:10.1016/j.jss.2019.110505.
9. Colavito G, Lanubile F, Novielli N, Quaranta L. Leveraging GPT-like LLMs to automate issue labeling. In: 2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR); 2024 Apr 15–16; Lisbon, Portugal. p. 469–80.
10. Ahmed HA, Bawany NZ, Shamsi JA. Capbug—a framework for automatic bug categorization and prioritization using NLP and machine learning algorithms. IEEE Access. 2021;9:50496–512. doi:10.1109/access.2021.3069248.
11. Karmakar A, Robbes R. What do pre-trained code models know about code? In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE); 2021 Nov 15–19; Melbourne, VIC, Australia. p. 1332–6.
12. Briciu A, Czibula G, Lupea M. A study on the relevance of semantic features extracted using BERT-based language models for enhancing the performance of software defect classifiers. Procedia Comput Sci. 2023;225:1601–10. doi:10.1016/j.procs.2023.10.149.
13. Van Dam JK, Zaytsev V. Software language identification with natural language classifiers. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). Vol. 1. New York, NY, USA: IEEE; 2016. p. 624–8.
14. Odeh AH, Odeh M, Odeh N. Using multinomial naive bayes machine learning method to classify, detect, and recognize programming language source code. In: 2022 International Arab Conference on Information Technology (ACIT); 2022 Nov 22–24; Abu Dhabi, United Arab Emirates. p. 1–5.
15. Ugurel S, Krovetz R, Giles CL. What's the code? Automatic classification of source code archives. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2002 Jul 23–26; Edmonton, AB, Canada. p. 632–8.
16. Reyes J, Ramírez D, Paciello J. Automatic classification of source code archives by programming language: a deep learning approach. In: 2016 International Conference on Computational Science and Computational Intelligence (CSCI); 2016 Dec 15–17; Las Vegas, NV, USA. p. 514–9.
17. LeClair A, Eberhart Z, McMillan C. Adapting neural text classification for improved software categorization. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME); 2018 Sep 23–9; Madrid, Spain. p. 461–72.
18. Ploscă TR, Rus BV, Pescaru AM, Curiac DI. Identifying developer positions in AGILE projects using BERT architecture. In: 2024 International Symposium on Electronics and Telecommunications (ISETC); 2024 Nov 7–8; Timisoara, Romania. p. 1–4.
19. Wysocki W, Miciuła I, Mastalerz M. Classification of task types in software development projects. Electronics. 2022;11(22):3827. doi:10.3390/electronics11223827.
20. Diamantopoulos T, Saoulidis N, Symeonidis A. Automated issue assignment using topic modelling on Jira issue tracking data. IET Software. 2023;17(3):333–44. doi:10.1109/msr59073.2023.00039.
21. Heyn V, Paschke A. Semantic jira-semantic expert finder in the bug tracking tool jira. arXiv:1312.5150. 2013.
22. Reszka Ł, Sosnowski J, Dobrzyński B. Enhancing software project monitoring with multidimensional data repository mining. Electronics. 2023;12(18):3774. doi:10.3390/electronics12183774.

23. Dobrzyński B, Sosnowski J. Graph-driven exploration of issue handling schemes in software projects. Appl Sci. 2024;14(11):4723. doi:10.3390/app14114723.
24. Moreira C, Wichert A. Finding academic experts on a multisensor approach using Shannon's entropy. Expert Syst Applicat. 2013;40(14):5740–54. doi:10.1016/j.eswa.2013.04.001.
25. Bok K, Jeon I, Lim J, Yoo J. Expert finding considering dynamic profiles and trust in social networks. Electronics. 2019;8(10):1165. doi:10.3390/electronics8101165.
26. McLean A, Wu M, Vercoustre AM. Combining structured corporate data and document content to improve expertise finding. arXiv:cs/0509005. 2005.
27. Kang YB, Du H, Forkan ARM, Jayaraman PP, Aryani A, Sellis T. ExpFinder: an ensemble expert finding model integrating N-gram vector space model and μCO-HITS. arXiv:2101.06821. 2021.
28. William M, Sincich T. Statistics for engineering and the sciences. Saddle River, NJ, USA: Prentice-Hall International, Inc. USA; 1995.
29. Devlin J, Chang MW, Lee K, Toutanova K. Bert: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. New York, NY, USA: ACM; 2019. p. 4171–86.
30. Lan Z, Chen M, Goodman S, Gimpel K, Sharma P, Soricut R. Albert: a lite bert for self-supervised learning of language representations. arXiv:1909.11942. 2019.
31. Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, et al. Roberta: a robustly optimized bert pretraining approach. arXiv:1907.11692. 2019.
32. Feng Z, Guo D, Tang D, Duan N, Feng X, Gong M, et al. Codebert: a pre-trained model for programming and natural languages. arXiv:2002.08155. 2020.
33. Guo D, Ren S, Lu S, Feng Z, Tang D, Liu S, et al. Graphcodebert: pre-training code representations with data flow. arXiv:2009.08366. 2020.
34. Yang Z, Dai Z, Yang Y, Carbonell J, Salakhutdinov RR, Le QV. Xlnet: generalized autoregressive pretraining for language understanding. In: Advances in neural information processing systems. Vol. 32. New York, NY, USA: ACM; 2019. p. 5753–63.
35. Er MJ, Venkatesan R, Wang N. An online universal classifier for binary, multi-class and multi-label classification. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC); 2016 Oct 9–12; Budapest, Hungary. p. 003701–6.
36. Xu Z, Dan C, Khim J, Ravikumar P. Class-weighted classification: trade-offs and robust approaches. In: International Conference on Machine Learning. Westminster, UK: PMLR; 2020. p. 10544–54.
37. Madabushi HT, Kochkina E, Castelle M. Cost-sensitive BERT for generalisable sentence classification with imbalanced data. arXiv:2003.11563. 2020.
38. Gao Y, Si S. Label smoothing for enhanced text sentiment classification. arXiv:2312.06522. 2023.
39. Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, et al. Transformers: state-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. New York, NY, USA: ACM; 2020. p. 38–45.
40. Zhang Q, Chen Q, Li Y, Liu J, Wang W. Sequence model with self-adaptive sliding window for efficient spoken document segmentation. In: 2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU); 2021 Dec 13–17; Cartagena, Colombia. p. 411–8. doi:10.1109/asru51503.2021.9688078.