IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received November 3, 2020, accepted November 17, 2020, date of publication December 3, 2020,
date of current version December 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3042429

Knowledge-Oriented Models Based on Developer-Artifact and Developer-Developer Interactions

EDSON M. LUCAS[1], TOACY C. OLIVEIRA[2], DANIEL SCHNEIDER[3,4], (Member, IEEE), AND PAULO S. C. ALENCAR[5], (Member, IEEE)
[1]IPRJ/UERJ, Polytechnic Institute, Rio de Janeiro State University, Nova Friburgo 28625-570, Brazil
[2]PESC/COPPE, Federal University of Rio de Janeiro, Avenida Horácio Macedo 2030, Centro de Tecnologia, Bloco H, sala 319, Cidade Universitária, Rio de Janeiro 21941-914, Brazil
[3]Tércio Pacitti Institute of Computer Applications and Research, Federal University of Rio de Janeiro, Rio de Janeiro 21941-901, Brazil
[4]Postgraduate Program in Informatics, Federal University of Rio de Janeiro, Rio de Janeiro 21941-901, Brazil
[5]David Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada

Corresponding author: Edson M. Lucas (emlucas@iprj.uerj.br)

This work was supported by the Advanced High Performance Computing Center (NACAD/COPPE) at Federal University of Rio de Janeiro (UFRJ), and by the Natural Sciences and Engineering Research Council of Canada (NSERC).

ABSTRACT INTRODUCTION: Software development is organized around developers working collaboratively promoting two types of interactions for knowledge sharing. Developer-Artifact interactions indicate developers define or access pieces of information within artifacts. Developer-Developer interactions indicate the exchange of information among developers using a collaboration platform to clarify an issue, promote an idea, or expose any thoughtful comment. PROBLEM: The number of such interactions grows over time and makes it difficult to capture and assess the evolution of the developers' knowledge about specific software project artifacts and tasks. Further, this knowledge decreases over time due to the natural limitations of human cognition that restrict our capabilities to cope with information overload. Besides, who has more knowledge about specific project elements are important to promote collaboration. AIMS: The K_a, K_s, K_c, and K_p models capture the evolution of the developers' knowledge about software project elements such as artifacts, tasks, similar tasks, and the whole software project. These models represent not only the knowledge developers have about these elements but also capture how this knowledge decreases over time based on forgetting and relearning functions. EVALUATION: An experimental study analyzed some developers' interactions on artifacts for the purpose of predicting the evolution of developers' knowledge in six software projects. The results show that the developers' rankings by performed tasks and by our models have 72% or more of similarity. CONCLUSION: Our models can capture and assess the evolution of the developers' knowledge and help to identify which developers have more knowledge about specific elements of software projects.

INDEX TERMS Knowledge model, interactions in software development, expert recommendation, recommendation system.

I. INTRODUCTION

The modern software development workflow is people-centered and most of the time conducted collaboratively [1]. Typically, this workflow suggests developers follow daily routines involving executing a set of tasks, interacting with other developers and manipulating artifacts such as source code, documentation, and configuration files. As a result, developers continuously and routinely interact with each other (Developer-Developer interactions) and with artifacts (Developers-Artifact interactions). Developer-Developer interactions occur face-to-face or through communication tools, when developers need to clarify some questions about performing a task. Developer-Artifact interactions happen when a developer reads or writes an artifact. Reading is used to gain knowledge about the artifact, typically to identify the parts of the artifact that may be changed to complete a task's goal. Writing occurs by adding or removing information from

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

artifacts that need to evolve, sometimes to accommodate a new feature, fix a bug or improve code.

The development workflow is typically orchestrated by a project plan (or sprint plan) describing the tasks developers agree on executing [2]. When a developer is assigned to a task, he or she usually tries to leverage on his/her past experience to remember similar performed tasks and manipulated artifacts. He/she may also try to figure out who is the most knowledgeable developer in the task's subject for a future call for help [3]. Remembering being in a similar development context, with similar developers and artifacts, is useful as it may help developers to avoid recurring problems and adopt best practices. In order to leverage on experience, developers intuitively browse their memory and the project's history in a cycle of discovery and remembrance of a task context. When a project history has a small number of tasks and artifacts, remembering task contexts is feasible. However, a typical software projects may last several months or years, accumulating a vast amount of tasks, artifacts and developers. As a result, intuitively browsing such vast amount of tasks, artifacts, developers and their relationships might be difficult, if not impossible. Moreover, the developers' knowledge about the history of tasks is limited as they tend to forget the past because of the natural limitations of human cognition that restricts our capabilities to cope with information overload [4], [5]. In this scenario, a model and the associated tool support, to measure the developers' knowledge of a given project element (task or artifact) can help other developers when dealing with a similar context along the project.

Our literature review has showed that Fritz *et al.* [6] proposed a model to infer the degree of developers' knowledge about software artifacts. The proposed measure assigns a degree of knowledge to a developer about an artifact over time, a combination of interest and disinterest of the developer over the artifact considering the authoring information. Rigby *et al.* [7] defined "knowledge loss of a project" as the number of files that are abandoned in turnover at quarterly intervals. Line-of-Code ownership means that the last developer changing a line of code has the ownership of a file and a file is abandoned when 90% or more of the lines are abandoned, possibly by a developer who has left the project. Other proposals use different words related to Knowledge such as Expertise, Expert, Affinity, Experience, Compatibility, and Skill to measure the relationship between developers and artifacts [8]–[13]. The result of this review shows that no work has so far been done concerning the fact that humans tend to forget bits of information [14]–[19], although there is evidence that developers tend to forget information about artifacts they have not manipulated for a while [20]–[22].

In this article, we present four knowledge-oriented models to represent the developer's knowledge about the elements of a software project. The K_a, K_s, K_c, and K_p models capture the evolution of the developers' knowledge about artifacts, tasks, similar tasks, and the whole software project, respectively. The proposed models combine information from Developer-Developer and Developer-Artifact interactions with a state-of-the-art way to capture the developers' forgetting and relearning. Given that the term *measurement is defined as a quantitatively expressed reduction of uncertainty based on one or more observations* [23], we claim that the proposed models measure developer's knowledge based on Developer-Artifact and Developer-Developer interactions taking into account knowledge depreciation over time, i.e., interactions from a distant past contribute less to the knowledge measure than the most recent interactions. According to our literature review, there is no research addressing explicitly the human limitations related to forgetting and relearning, and inferring knowledge about artifacts, tasks, similar tasks, and the whole software project.

An evaluation of the proposed models was performed using data from six open-source projects. The official ranking of the project based on the number of performed tasks was compared with the ranking produced with our K_p model for measuring developers' knowledge about the whole project. The results presented an average correlation greater than or equal to 72% between the two rankings. The model for the whole project depends on the models of similar tasks, tasks, and artifacts. There is also evidence that these models can capture developers' knowledge about artifacts, tasks, and similar tasks and use this knowledge to identify experts, i.e., the persons who have the most knowledge about specific project elements. These models, which measure knowledge about similar tasks and the whole software project, use a new meta-heuristic for clustering based on software modules [24].

The article is organized as follows. Section II presents an overview of our approach. Section III describes the knowledge-oriented models. Section IV presents an evaluation of the proposed model. A discussion about the results of the evaluation is presented in Section V and Section VI describes related work. Finally, Section VII wraps up this article with our conclusion and future work.

II. APPROACH OVERVIEW

An essential procedure in software development is the manipulation of software artifacts, such as models, source code and configuration files, to implement new software functionalities or amend existing ones [25]. Such manipulation is typically supported by an Integrated Development Environment (IDE) such as Eclipse or VSCode, through which developers can easily browse or create project-related artifacts. As illustrated in Figure 1, a developer $d1$ can use the IDE to read, write or update artifacts, establishing a series of Developer-Artifact interactions that can be used as a proxy to the developers' knowledge on a single artifact, i.e., if the developer $d1$ manipulates the artifact $a1$ several times during the artifact lifetime, $d1$ must be knowledgeable of $a1$.

Software development is also collaborative in the sense that developers exchange information regarding topics of interest, sometimes to clarify a reasoning path or to simply ask for a specific information. Note that in Figure 1 the developer $d1$ interacts with the developer $d2$ (and with other developers) establishing a series of Developer-Developer interactions.

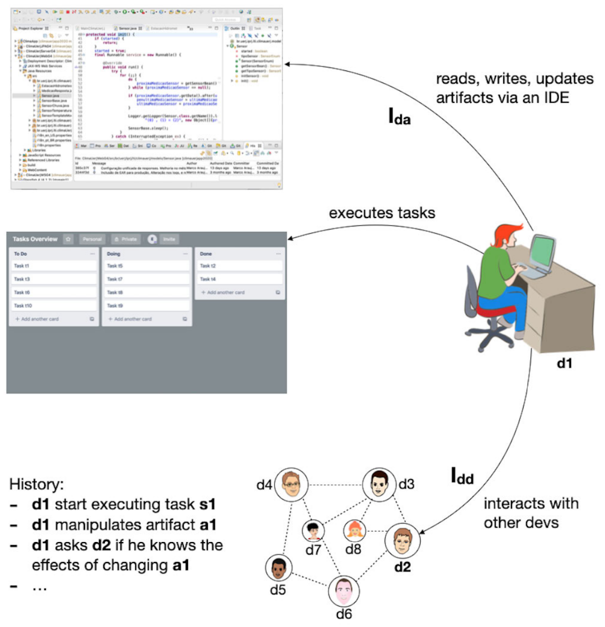

reads, writes, updates
artifacts via an IDE
I_{da}
executes tasks
d1
I_{dd}
interacts with
other devs
History:
- **d1** start executing task **s1**
- **d1** manipulates artifact **a1**
- **d1** asks **d2** if he knows the effects of changing **a1**
- ...
d4
d3
d7
d8
d2
d5
d6

FIGURE 1. Software project context.

These interactions can also be used as a proxy to the developers' knowledge about a specific topic, if developer-developer communications are properly harnessed. Developer-Artifact and Developer-Developer interactions typically occur in the context of a task, where a task describes the piece of work to be performed in the context of a software project. Examples of a task are ''fixing an issue'' or ''adding a new feature''. When a task is handled by a developer, he or she often manipulates artifacts and communicates with other developers.

Over the lifetime of a software project, the combination of Developer-Artifact interactions, Developer-Developer interactions and Developer-Task executions, enables creating a project history that can be analyzed to help discovering the most knowledgeable developers, similar tasks and specific task contexts. This article introduces Developer-Artifact (I_{da}) and Developer-Developer (I_{dd}) interaction degrees to define knowledge-oriented models that can be used to measure the amount of knowledge developers have about artifacts and tasks. We claim that using our models to find the most knowledgeable developers for a given task or artifact may help developers when new similar tasks need to be performed. The rationale supporting our claim is that when a new task $s1$ is under responsibility of developer $d1$, he or she may need to define $s1's$ context including: a) the tasks that are similar to $s1$ that contain information he or she may leverage on; b) the developers that may help him on clarifying issues regarding $s1$, i.e. developers that are knowledgeable on tasks similar to $s1$; c) the artifacts that were manipulated by developers when executing tasks similar to $s1$, i.e. developers that are knowledgeable on artifacts that might be required to handle $s1$. Building this context for a new task is infeasible for real-world projects as $d1$ would need to skim through very large artifacts and developers. In addition, developers tend to forget previous work and their knowledge about specific artifacts fade over time.

Figure 2 illustrates our approach to discovering knowledgeable developers. First we need to record the Developer-Artifact and Developer-Developer interactions that happen when developers are handling their tasks into a Project Interaction History (Step 1 in Figure 2). After recording these interactions, we apply our K_a and K_s models to measure the developer's knowledge on artifacts and tasks (Step 2 in Figure 2), and then apply a clustering technique to gather tasks that are similar based on the artifacts such tasks interact with (Step 3 in Figure 2). Finally, we measure the developer's knowledge about similar tasks and the whole project history using our K_c and K_p models (Step 4 in Figure 2).

More specifically, the K_a model measures a developer's knowledge about an artifact over time based on all edit interactions that the developer had with an artifact (see Section III-A). The K_s model measures the developer's knowledge about a task over time considering the result of the developer's knowledge about every artifact associated with a task, i.e. K_a applied to every artifact associated with the task based on edit interactions (see Section III-B). The K_c model measures the developer's knowledge about a cluster of similar tasks considering the developer's knowledge about every task belonging to a cluster (see Section III-C). Finally, the K_p model measures the developer's knowledge about the entire history of project tasks based on the developer's knowledge about similar task clusters (see Section III-D).

III. KNOWLEDGE-ORIENTED MODELS TO MEASURE DEVELOPER's KNOWLEDGE IN SOFTWARE PROJECTS

Our models use degrees of interaction between developers and artifacts (I_{da}), and among developers (I_{dd}) to measure developers' knowledge about a software development project. Figure 3 shows the representation of I_{da} and I_{dd} interactions based on the W3C provenance model [26]. We define *Artifact*, *Task*, and *Developer* types. *Artifact* is an *Entity* specialization for representing digital files such as source code and documentation. The *Developer* type represents an *Agent* type, a person who has knowledge and authorization to perform specific tasks. The *Task* is an *Activity* specialization and may represent an error or the addition of new functionality to the software. Depending on the tool used, the task may be called by different names. In Bugzilla, a task is called *Bug* and in GitHub it is named an *Issue*.

The I_{da} interaction is depicted by the *WasChangedBy* relationship between *Artifact* and *Task* in Figure 3. This relationship ensures that an artifact can be changed by one or more tasks. For each change in an artifact, the degree and final time of the change are recorded. The degree in this case is a measure of the size of a change (e.g., in WasChangedBy). This information depends on the specific development environment. For example, the degree may be expressed by the number of rows changed, the number of characters changed or the result of an interaction model (e.g., the one supported by

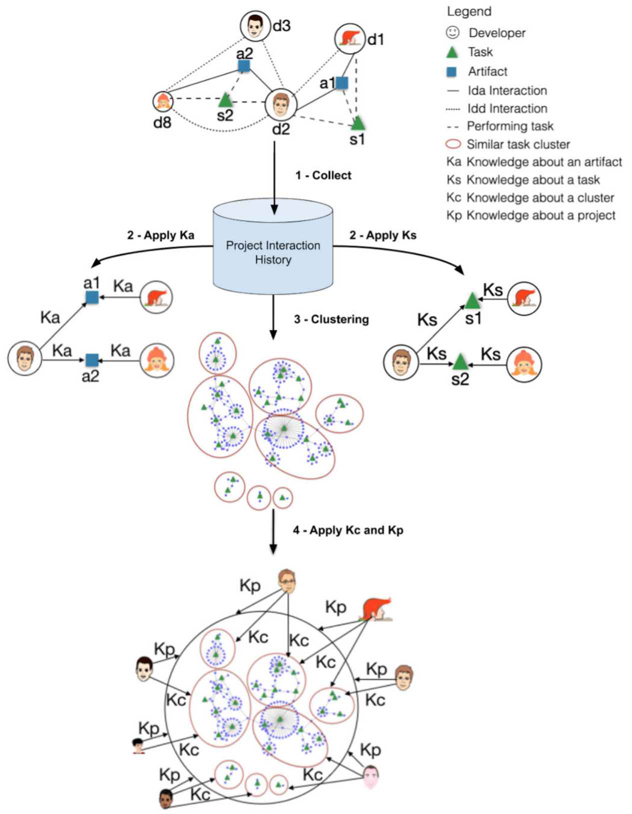


FIGURE 2. Knowledge-oriented models overview.

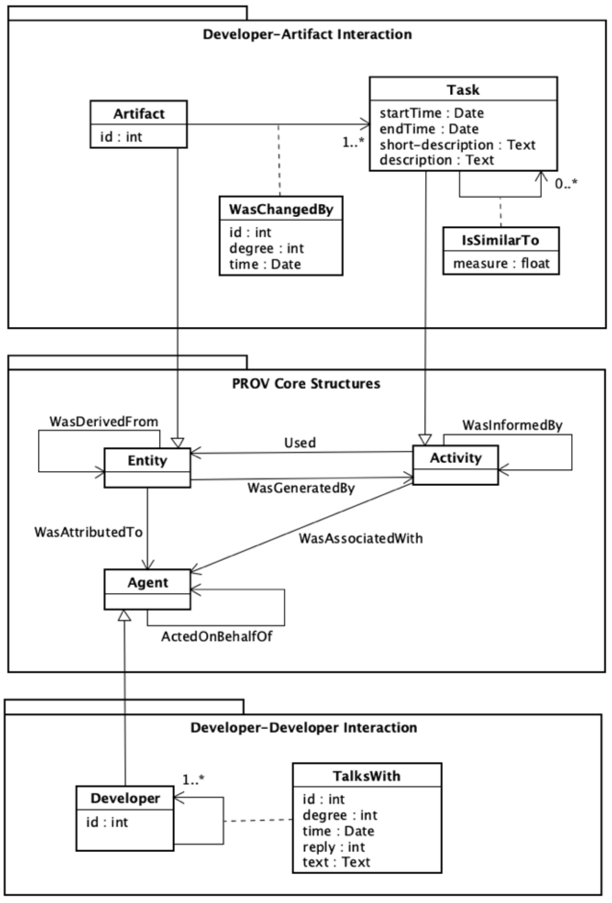


FIGURE 3. Class diagram of Developer-Artifact and Developer-Developer interactions.

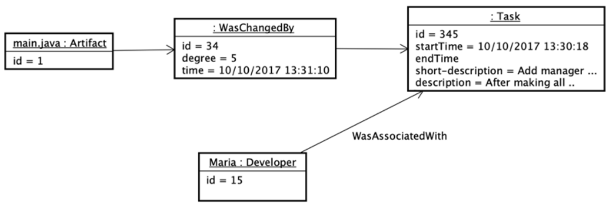

main.java : Artifact
id = 1
: WasChangedBy
id = 34
degree = 5
time = 10/10/2017 13:31:10
: Task
id = 345
startTime = 10/10/2017 13:30:18
endTime
short-description = Add manager ...
description = After making all ..
WasAssociatedWith
Maria : Developer
id = 15

FIGURE 4. A Developer-Artifact interaction example.

Mylyn [27]). Figure 4 shows an example of I_{da} interaction. The record of this interaction shows that developer *Maria* changed the *main.java* artifact while performing the task with id equal to 345. The change took place on 10/10/2017 at 1:31:10 pm with degree 5.

The I_{dd} interaction represents developers' conversations about performing each task. The class model in Figure 3 expresses that a developer can talk to one or more developers. A conversation is composed of communications between the participants, and includes a record of the degree, the time and the reply attribute. The degree in this case is a measure of how much information was exchanged in an interaction (e.g., in TalksWith). This information depends on the development environment in which the project is developed. In GitHub, for example, conversations are recorded in a text format based on message exchanges where the degree can be the number of words or the number of lines of each message. Figure 5 shows an example of I_{dd} interaction, the record of the conversation between Mary and John about performing task 345. Mary sends a textual message to John on 10/10/2017 at 1:33:17 PM and John answers her on 10/11/2017 at 10:13:10 AM.

The class model in Figure 3 also asserts that one task may have similarities to other tasks. In this article, project history tasks are organized into similar task groups based on the I_{da} interactions. Figure 6 illustrates the similarity between two tasks considering these interactions. In this figure a task is depicted by a triangle and an artifact by a square. An I_{da} interaction is depicted by an arrow from the task to the artifact. Thus, tasks $S239875$ and $S245189$ are defined to be similar with a weight equal to 3 because the developers performed I_{da} interactions on three artifacts ($A419$, $A425$, and $A1243$) while working on these tasks.

In what follows we will present our models in detail and describe how they are used to measure developers' knowledge about software project elements such as artifacts, tasks, similar tasks, and the whole software project.

A. DEVELOPER-ARTIFACT KNOWLEDGE MEASURE

The K_a model measures the developer's knowledge about an artifact using I_{da} interactions. We define $I_{da}(d, a, f_{endtime})$ as the degree of interaction between a developer (d) and an

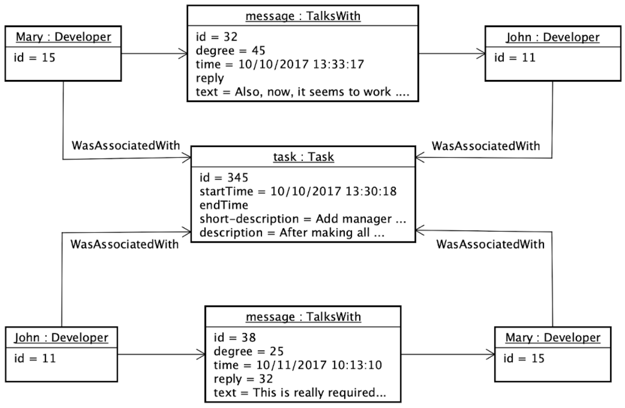


FIGURE 5. A Developer-Developer interaction example.

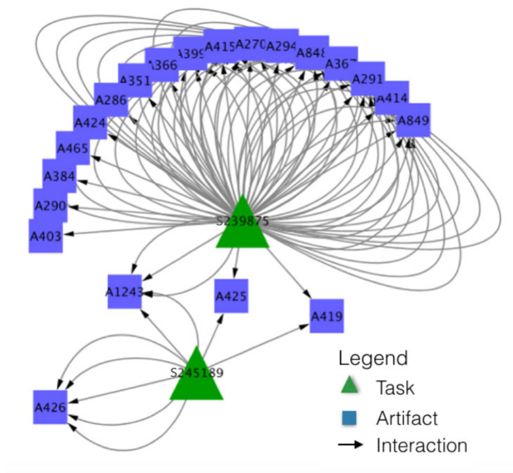


FIGURE 6. Similar tasks by interactions in artifacts.

artifact (a) during a specific time interval, recorded at the interaction end time ($f_{endtime}$). Because each artifact has a degree of production and maintenance difficulty, each interaction needs to be normalized by the current maximum interaction recorded for the artifact.

Equation (1) defines II_{da} as the normalized interaction of an I_{da} interaction. Let D be the set of all developers who have interacted by I_{da} with artifact (a) until time t. So $II_{da}(d, a, t)$ is the result of dividing $I_{da}(d, a, t)$ by the maximum I_{da} interaction among all developers for the artifact (a) until t. For example, according to Figure 7, the developer $d1$ had three interactions with the $a1$ artifact recorded in the times "2007-07-31 13:00:00", "2007-07-31 15:30:00", "2007-07-31 16:00", with degree values equal to 23, 15 and 18, respectively. The developer $d2$ had two interactions with $a1$ artifact, the first registered in "2007-07-31 15:45" with value 33 and the other in "2007-08-11 13:00:00" with value 50. The maximum interaction recorded was 50 and this interaction was performed by the developer $d2$. The measures of the interaction degree of $d1$ were $\{\frac{23}{50}, \frac{15}{50}, \frac{18}{50}\}$ and for $d2$ were $\{\frac{33}{50}, \frac{50}{50}\}$. Therefore, the interaction degree represents

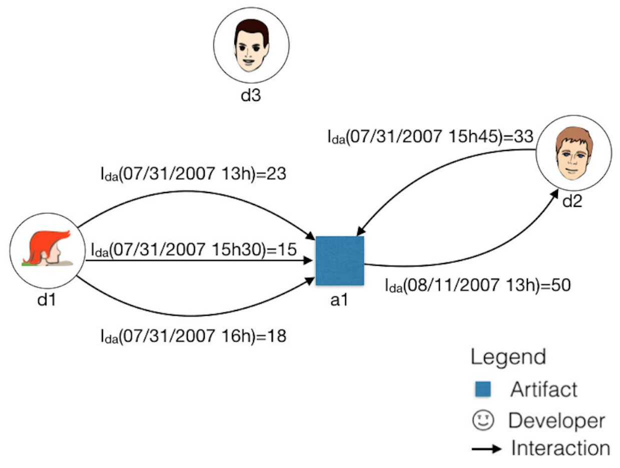


FIGURE 7. Example of Developer-Artifact interactions.

how intense the developer's interaction is with respect to the other interactions with the same artifact.

$$II_{da}(d, a, t) = \frac{I_{da}(d, a, t)}{\max_{d \in D, \forall t_i \leq t} I_{da}(d, a, t_i)} \quad (1)$$

The K_a model also considers the developers' forgetting and relearning. These aspects are modeled using a hyperbolic function. This function was chosen because it is a simple functional form that can model forgetting consistently [28], [29]. Equation (2) represents how much developers remember their interactions considering relearning, R_d. The forgetting constant b was set to 0.025 to calibrate knowledge depreciation based on an existing study [30]. This study indicates that developers, on average, forget about 50% of a file in 40 days if they don't revisit the file within that period. In addition, we used the information that developers tend to forget artifacts that were not manipulated by them for a while [20]–[22]. Let Δt be the number of days that have passed since the recording of an I_{da} interaction.

The function that models a developer's interaction with an artifact and its neighbors is defined as $G_d(n) = \frac{7}{e^n+6} \cdot b$, where n is the number of days. We have chosen to set n from 0 to 7 because previous studies (e.g., experiments by Fritz *et al.* [20], [21]) have shown that seven working days have produced the highest correlation between developers' knowledge and their interactions. In these days, the developer d produces I_{da} interaction: 1- with the artifact a; or 2- with artifacts related to tasks that are related through the I_{da} interaction to the artifact a (neighbors). The smaller G_d, the lower the depreciation of the developer's knowledge about artifact (a), and we consider that there was a relearning of the artifact-related tasks in that period. Equation (3) defines G_d and Fig. 8 shows its graph.

$$R_d(n) = \frac{1}{(G_d(n) \cdot \Delta t) + 1} \quad (2)$$

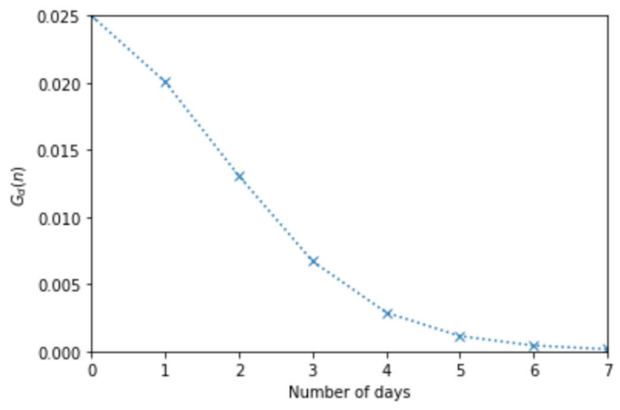


FIGURE 8. The values of G_d for n in [0-7].

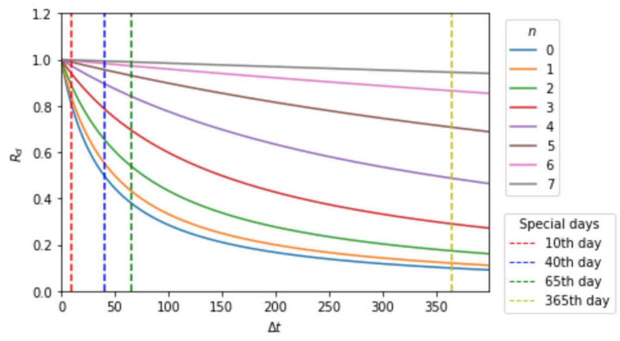


FIGURE 9. R_d graph with a highlight for the 10th, 40th, 65th, and 365th days.

$$G_d(n) = \frac{7}{e^n + 6} \cdot b$$
$$n \text{ is an integer such that } 0 \le n \le 7 \quad (3)$$

Figure 9 shows the values of R_d with a highlight for the 10th, 40th, 65th, and 365th days. Relearning has the same behavior for artifacts that have had early and late interactions over time. For n equal to zero, there was no relearning, and the curve has the smaller values for R_d. On the other hand, when the developer produces, in all the last seven days, I_{da} interaction with an artifact a or with task-related artifacts through I_{da} with artifact a, for n equal to 7, the values of the curve R_d will have larger values. The graph in Figure 9 also shows that the higher the value of n, the higher the values of R_d.

Table 1 shows the influences of the values of n and Δt on R_d. For n equal to zero, without relearning in the last 7 days, the developer remembers 97.6% of the I_{da} interaction performed in the previous day ($\Delta t = 1$). On the fortieth day ($\Delta t = 40$), the developer remembers 50% of the interaction and only 10% after one year. For n equal to 7, with relearning, the developer fully remembers the I_{da} interactions performed in the last 3 days. On the fortieth day, developer remembers 99.4% of the interaction and in one year he or she remembers 94.5%. Table 1 presents the R_d values for all values of n and some values of Δt ($\Delta t = 0, 3, 5, 7, 39, 40, 41, 63, 65, 67,$ and 365).

After briefly analyzing R_d, we define $K_a(d, a, t)$ to represent the knowledge measure of the developer d about artifact a at date-time t. The K_a model is composed of two factors. The first one calculates the normalized interaction (II_{da}) for each I_{da} interaction produced up to t. The second factor represents the forgetting of the interactions registered in the first factor, R_d. In this way, the developer's knowledge about an artifact is distributed over time, so that newer interactions contribute more than older ones. This is reasonable because developers tend to forget artifacts that have not been manipulated by them for a while [20]–[22].

Equation (4) formalizes $K_a(d, a, t)$ as the sum of the product of II_{da} by R_d for all I_{da} interactions produced by a developer d with the artifact a up to date-time t. The Δt_i calculates the number of days that have passed from the time the I_{da} interaction took place and the time t, where t is the date-time on which the developer's knowledge need to be estimated. According to K_a, the more I_{da} interactions a developer has with an artifact, the more knowledge he or she can have about it. This is true because II_{da} is always a positive number.

$$K_a(d, a, t) = \sum_{f_i \le t} II_{da}(d, a, f_i) \cdot R_d(n) \quad (4)$$

B. DEVELOPER-TASK KNOWLEDGE MEASURE

During a software development project, developers create and change artifacts through I_{da} interactions. Besides, developers interact with other developers to perform tasks, and this is captured by I_{dd} interactions. The K_s model measures the developer's knowledge of a task by making use of these interactions.

Figure 10 illustrates how developers interact with artifacts through I_{da} interactions to perform a task. The developer $d5$ interacted with artifact $a1188$ twice: the first interaction occurred at time 45 with degree 15, and the second interaction was in time 54 with degree 11. The developer $d5$ also interacted with artifact $a1475$ at times 60 and 80 with degrees equal to 26 and 18, respectively. According to data from the Bugzilla attachment field of the Mylyn project, more than one developer has contributed with I_{da} interactions to perform tasks, such as in the case of tasks IDs 166406, 210686, 234065 and 248490. Therefore, there is evidence that the knowledge measure about tasks should consider I_{da} interactions.

Figure 11 illustrates I_{dd} interactions among developers. The task was assigned to $d14$ and $d10$ has written about how the task needed to be done before the task was assigned to $d14$. This case shows that developer $d10$ has knowledge about this task, but he has not produced an interaction about artifacts related to this task through I_{da} interactions. This is an evidence that the knowledge measure about tasks should also consider I_{dd} interactions.

TABLE 1. The values of R_d for Δt in {0,3,5,7,39,40,41,63,65,67,365}.

	Δt in days										
n	0	3	5	7	39	40	41	63	65	67	365
0	1	0.930	0.889	0.851	0.506	0.500	0.494	0.388	0.381	0.374	0.099
1	1	0.943	0.909	0.877	0.561	0.555	0.549	0.442	0.434	0.426	0.120
2	1	0.962	0.939	0.916	0.662	0.657	0.651	0.548	0.541	0.533	0.173
3	1	0.980	0.968	0.955	0.793	0.788	0.784	0.703	0.696	0.690	0.290
4	1	0.991	0.986	0.980	0.899	0.896	0.894	0.846	0.842	0.838	0.487
5	1	0.997	0.994	0.992	0.958	0.957	0.956	0.933	0.931	0.929	0.707
6	1	0.999	0.998	0.997	0.984	0.983	0.983	0.974	0.973	0.972	0.865
7	1	1	0.999	0.999	0.994	0.994	0.994	0.990	0.990	0.989	0.945

TABLE 2. The values of K_c, in groups of up to 10 tasks, with K_s set to 1 for all tasks.

	m									
h	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0
1	1	0.50	0.33	0.25	0.20	0.17	0.14	0.13	0.11	0.10
2		2	1.33	1	0.80	0.67	0.57	0.50	0.44	0.40
3			3	2.25	1.80	1.50	1.29	1.13	1	0.90
4				4	3.20	2.67	2.29	2	1.78	1.60
5					5	4.17	3.57	3.13	2.78	2.50
6						6	5.14	4.50	4	3.60
7							7	6.13	5.44	4.90
8								8	7.11	6.40
9									9	8.10
10										10

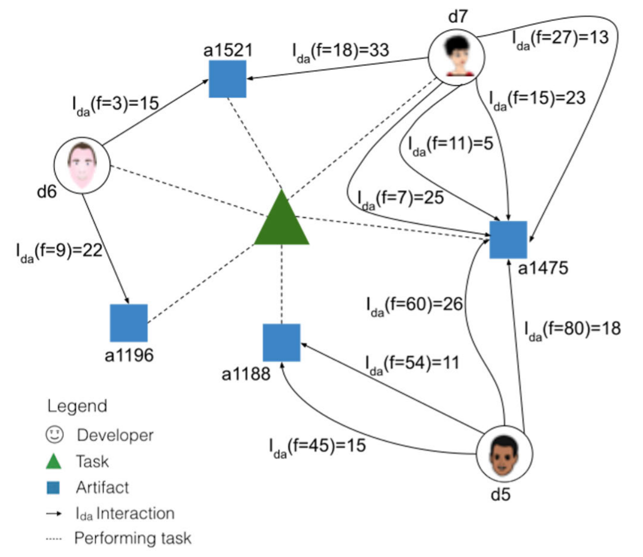
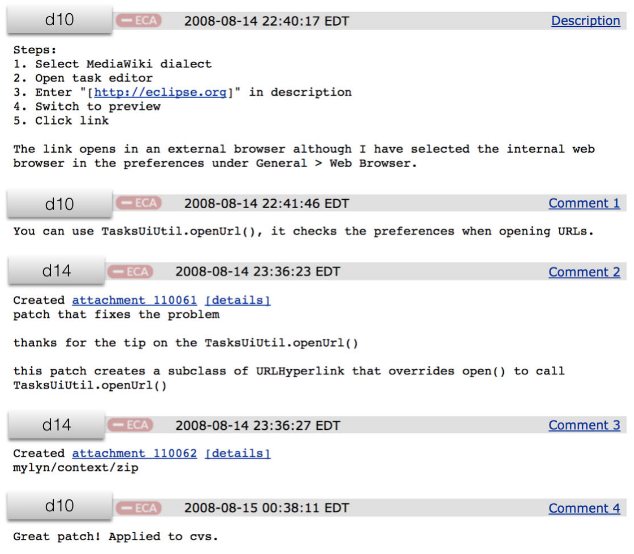


FIGURE 10. Example of Developer-Artifact interactions (I_{da}) to perform a task.


d10 ECA 2008-08-14 22:40:17 EDT Description

```
Steps:
1. Select MediaWiki dialect
2. Open task editor
3. Enter "[http://eclipse.org]" in description
4. Switch to preview
5. Click link

The link opens in an external browser although I have selected the internal web
browser in the preferences under General > Web Browser.
```

d10 ECA 2008-08-14 22:41:46 EDT Comment 1

```
You can use TasksUiUtil.openUrl(), it checks the preferences when opening URLs.
```

d14 ECA 2008-08-14 23:36:23 EDT Comment 2

```
Created attachment 110061 [details]
patch that fixes the problem

thanks for the tip on the TasksUiUtil.openUrl()

this patch creates a subclass of URLHyperlink that overrides open() to call
TasksUiUtil.openUrl()
```

d14 ECA 2008-08-14 23:36:27 EDT Comment 3

```
Created attachment 110062 [details]
mylyn/context/zip
```

d10 ECA 2008-08-15 00:38:11 EDT Comment 4

```
Great patch! Applied to cvs.
```

FIGURE 11. Interactions among developers.

In this scenario, the measure of the developer's knowledge about a task s is defined based on the measure of his knowledge of the artifacts related to s through I_{da} interactions combined with his contribution through I_{dd} interactions produced while performing s. Because each task has a degree of production and maintenance difficulty, each II_{dd} interaction needs to be normalized by the current maximum interaction recorded while developers perform the task. The maximum value of I_{dd} interaction is defined as the maximum degree value among all I_{dd} interactions concerning task s until time t. Equation (5) defines II_{dd} as the normalized interaction of an I_{dd} interaction. Let D be the set of all developers who have interacted through I_{dd} while performing task (s) until time t. Additionally, $II_{dd}(d, s, t)$ is the result of dividing $I_{dd}(d, s, t)$ by the maximum value of I_{dd} interaction. Equation (6) formalizes $K_s(d, s, t)$ as the sum of the developer's knowledge d about all task-related artifacts ($\sum_{a \in s} K_a$) and all normalized interaction (II_{dd}) for each I_{dd} interaction produced by the developer d in the execution of the task s up to the time t. The developer's knowledge about a task also undergoes

TABLE 3. Influence of $\frac{h}{m}$ factor on K_c measure.

	K_s					K_c	$K_c\ (h=m=1)$
	1ª	2ª	3ª	4ª	5ª		
$d1$	80	35	0	0	0	46	115
$d2$	7	9	6	13	8	43	43

depreciation over time - the first depreciation is inherited from (4), and the second one is caused by the fact that the second added of K_s uses the forgetting function R_d. In R_d, n is the number of days in the last seven days in which the developer d expressed knowledge about the task s through I_{dd} interactions. An example of this type of interactions is sending a message about the task s to every developer involved in performing it.

$$II_{dd}(d,s,t) = \frac{I_{dd}(d,s,t)}{\max_{d\in D,\forall t_i\le t} I_{dd}(d,s,t_i)} \quad (5)$$

$$K_s(d,s,t) = \sum_{a\in s} K_a(d,a,t) + \sum_{f_i\le t} II_{dd}(d,s,f_i)\cdot R_d(n) \quad (6)$$

C. SIMILAR-TASK GROUP KNOWLEDGE MEASURE

The $K_c(d,c,t)$ model is defined as the measure of the developer's knowledge about a similar-task group (c) at date-time t. K_c has two factors: the first is $\frac{h}{m}$, where h is the number of tasks related to the group that the developer belongs to, and m is the number of tasks belonging to the group. The $\frac{h}{m}$ factor quantifies the developer's involvement in group tasks (i.e., this factor penalizes the developer who only has knowledge about a few tasks). In the best case, when h equals to m, the developer has knowledge about all tasks in the group, $\frac{h}{m} = 1$. The second factor is the sum of the $K_s(d,s,t)$, the developer knowledge measure about each group task. Equation (7) shows $K_c(d,c,t)$ and Table 2 shows the values of K_c for groups that have up to 10 tasks, with K_s set to 1 for all tasks.

$$K_c(d,c,t) = \frac{h}{m}\cdot\sum_{s\in c} K_s(d,s,t) \quad (7)$$

Table 3 shows an example of how the $\frac{h}{m}$ adjustment factor can influence the K_c measure. For example, in a group with five similar tasks, the developer $d1$ has knowledge of two tasks with $K_s = \{80, 35\}$ and developer $d2$ has knowledge about all five tasks with $K_s = \{7, 9, 6, 13, 8\}$. According to K_c, considering the adjustment factor, developers have knowledge with close values: $K_c(d1) = 46$ and $K_c(d2) = 43$. However, when the $\frac{h}{m}$ factor is not considered, the developer $d1$ has a K_c value that is much higher than $d2$, even though he or she has no knowledge about 3 tasks in a group of 5 tasks.

D. DEVELOPER-PROJECT KNOWLEDGE MEASURE

The contribution of developers in software projects can be measured in different ways: by producing lines of code, by counting function points [31], and for open source project, by using measures based on the performed tasks and the number of commits.[1] Our K_p model measures the developers' knowledge about a software project considering the measure of developer's knowledge about similar-task groups, i.e., clusters of similar tasks (K_c). More specifically, $K_p(d,p,t)$ represents the developer's knowledge measure of a developer d about a software project p at a date-time t as shown in equation (8).

$$K_p(d,p,t) = \frac{r}{q}\cdot\sum_{i=1}^{q} K_c(d,c_i,t) \quad (8)$$

The first part of equation (8), the $\frac{r}{q}$ adjustment factor, determines the developer's participation in similar-task groups, where r is the number of groups that the developer knows about the software project and q is the total number of groups into which the software project was organized. If the developer participates in all groups, the factor is 1. This adjustment factor penalizes the developers who only know about a few similar-task groups. The influence of $\frac{h}{m}$ factor on the K_c measure, which is analyzed in Tables 2 and 3, is similar to the influence of $\frac{r}{q}$ factor on the K_p measure. The difference is that in K_c measure, the factor acts on K_s, whereas in the K_p, the factor acts on K_c. The second factor in equation (8) represents the sum of the developer's knowledge about similar-task groups (K_c).

IV. EVALUATION

The proposed evaluation method follows the guidelines of GQM (Goals Questions Metrics) approach [32]: Analyze some developers' edit interactions on software artifacts for the purpose of predicting developer's knowledge with respect to software projects from point of view of the project manager.

The general question: Q1 (Who knows about the software project?) can be answered using four measures. The first measure - the number of performed tasks - is a well-accepted measure to infer the production or participation of the developers in software projects, and is widely used in open source projects such as those that use Eclipse as an administration tool.[2] The second measure takes into account the number of edit interactions performed by developers to perform tasks. We note that in the study with Mylyn Docs project, both edit and commit interactions are considered and commit is assumed to be an inference of edit because of the lack of edit data. The third measure is the number of groups of similar tasks in which the developer has knowledge based on K_c model, i.e., the number of clusters that the developer has knowledge considering the whole software project. For example, a project with 100 tasks can be clustered in 5 groups of similar tasks. In this scenario, a developer that has done 30 tasks belonging to 3 similar groups can have knowledge about these 3 groups. The fourth measure, which is based on

[1] https://projects.eclipse.org/projects/eclipse/who
[2] https://www.eclipse.org

K_p model and illustrated in Section III-D, infers the developer's knowledge about software projects considering similar tasks captured by I_{da} and the developer's interaction degree with artifacts that takes into account forgetting. Analysis of these four measures led us to define three research questions to compare the similarities between the rankings, which are ordered lists from highest to lowest values, produced by these four measures:

RQ1: Is the ranking of developers by performed tasks (R_{pt}) similar to the ranking by their interactions with artifacts?

RQ2: Is the ranking R_{pt} similar to the ranking based on the K_c model?

RQ3: Is the ranking R_{pt} similar to the ranking based on the K_p model?

Our expectation is that these research questions show that the K_p model presents more similarity than the K_c model when compared to the R_{pt}. On the other hand, the K_p model also presents a result close to the comparison of the number of interactions with the R_{pt}. Thus, the K_p model can rank the developers of a software project in the same direction as by the number of interactions or the number of performed tasks.

In addition to these research questions, a sensitivity analysis of K_p is also performed to assess the influence of the parameters R_d, G_d, $\frac{h}{m}$ and $\frac{r}{q}$.

The following subsections are structured as follows. Next, Subsection IV-A describes how the data was collected. Subsection IV-B describes the steps for carrying out this evaluation study. Subsection IV-C presents the results that support the answers to questions RQ1, RQ2, RQ3 and describe the influence of parameters on K_p using data from the Mylyn Docs project. Finally, in Subsection IV-D, threats to the validity of this study are reported.

A. DATA COLLECTION

Lethbridge *et al.* [33] presented a taxonomy for the data collection techniques based on the degree of human contact. In Lethbridge and colleagues' taxonomy, our study is classified in the third degree because it requires access only to work artifacts using analysis techniques based on tool logs. The data was collected from the Mylyn Docs project , hosted at https://www.eclipse.org/. Initially, randomly, eight projects were selected from GitHub (https://github.com). The Homebrew/homebrew-formula-analytics project was discarded for having only six developers, and the brewsci/homebrew-science and kerl/kerl projects were discarded because the measures did not vary between trials. The selected projects from GitHub were: apache/commons-lang, brewsci/homebrew-bio, google/EarlGrey, Homebrew/homebrew-cask-drivers and iodide-project/iodide.

The Mylyn Docs project is a subproject of the Mylyn project involving an Eclipse IDE plugin that reduces the number of available artifacts to facilitate browsing artifacts relevant to a given task. The collected data was generated by the Git, Bugzilla and Mylyn tools organized in three observation perspectives: the first collects data from the Mylyn Docs project to generate the baseline of this evaluation study; the second collects the data recorded by the Mylyn plugin of the developer's interactions with artifacts; and the third perspective was produced by the Git tool that records the artifact submissions in the project repository. Commit actions were collected to infer edit actions given that, usually, a commit action submits edited files to a code repository. In addition, we have identified some performed tasks in the Mylyn Docs project that do not have Mylyn logging. Next, we will detail the data collection performed according to each perspective.

The first perspective, Performed Tasks, was generated from querying the Mylyn Docs data recorded by Eclipse Bugzilla.[3] The parameters for the query were: Classification = Mylyn and Product = "Mylyn Docs" and Component = EPUB or Framework or HtmlText or Wikitext and Status = RESOLVED and Resolution = FIXED. This query ensures that tasks marked as solved and tested by developers in the EPUB, Framework, HtmlText, and Wikitext components of the Mylyn Docs project that belong to Mylyn are returned after execution. The query was run on September 18, 2017, and generated an XML file from which the *bug_id*, *assigned_to*, and *endtask* information of 609 performed tasks (i.e., bug fixes and new features) were extracted.

The second observation perspective, Edit Data, was produced from the log generated by the tool Mylyn based on the execution of the tasks available in the attached files in Eclipse Bugzilla. The parameters for the query were the ones described in the query from the first perspective and additionally: and (Match ALL of the following separately→Attachment Description→contains the string→mylyn/context/zip). This search criterion returns only the tasks that have the interaction log registered by the Mylyn plugin. The query was run on April 05, 2017, and returned 345 tasks. After the query execution, all edition interactions (Kind="edit") were extracted from the attached files (mylyn/context/zip) in the tasks. The returned data are I_{da} interactions that use Mylyn's DOI as the degree of I_{da}. The DOI is defined as the degree of interest of a developer about an artifact in a specific time range. We use DOI inspired by the work of Fritz *et al.* that proposed the concept of a degree of knowledge [21]. For example, an event registered by Mylyn for bug 219939 shows that the developer associated with this bug performed an edit interaction that started on "2008-07-16 19:01:36.15 -04" and finished on "2008-07-16 19:34:54.312 -04" with the artifact */path/TaskEditorRichTextPart.java*, with a degree of interest equal to 82. The extraction resulted in data for 334 tasks with 49906 editing interactions performed by 6 developers with 1538 artifacts. The PDT time zone is equal to UTC-7.

The third and last observation perspective, Commit Data, collected the data recorded by Git in the Mylyn Docs project code repository. The Git commands were used to extract the commits until April 05, 2017. First, the *git clone* command

[3]https://bugs.eclipse.org/bugs/query.cgi

downloaded the data of the Mylyn Docs project. After the download, the *git log* command was used to extract *name*, *email*, *commit_date*, *commit_hash*, and *commit_msg*. Commits that identified the task at the beginning of *commit_msg* were selected to record the commit interaction. The *git diff-tree* command was used to extract the artifacts sent out by the commit. The degree of interest of a commit interaction was set to 1, and the start and end dates of the interaction were recorded with the commit date. The names of the developers registered in Git, in some cases, were different from those registered by Bugzilla, as we have adopted the names registered in Bugzilla. The extraction resulted in 918 commits performed by 33 developers on 5407 artifacts. Overall, the evaluation study for Mylyn Docs project used data produced according to the three aforementioned perspectives until April 05, 2017.

The data were collected until December 2019 according to the first and third observation perspectives using a library for accessing GitHub projects. PullRequest represents the set of updates that were necessary for the realization of an Issue (task), returning information about performing each task (issue = closed). Due to lack of data in the *assigned_to* field, we consider that if a developer has altered some artifact to assist in performing a task, then we attribute to him the accomplishment of the task. The interactions of non-human developers, known as bots, such as testing software and updating libraries were not considered. The degree of Developer-Artifact interaction was defined as the number of lines inserted and deleted in the artifact at each commit.

B. PLANNING AND EXECUTION

The experiment is organized in steps to generate rankings of developers who have knowledge about the projects selected for this evaluation. The rankings are generated in four different ways. The baseline is the ranking of the number of tasks performed by the developer using the data from the Performed Tasks observation perspective. The ranking by interactions is constructed from observation data from Edit Data and Commit Data based on the number of interactions the developer has made. The K_c ranking orders the developers by the number of groups (i.e., clusters) in which they have knowledge according to the K_c model. The K_p ranking uses the K_c model. Both models, K_c and K_p, use data from the Edit Data and Commit Data perspectives. This study uses a clustering technique to build K_c and K_p rankings.

Clustering is a computational technique for organizing data objects (elements) into clusters (groups). By providing data clusters, this technique facilitates information understanding. Similar elements tend to be in the same group, whereas less similar or non-similar elements tend to be in different groups [34], [35]. There are many algorithms for data clustering [36]. One of the most best-known and used clustering algorithms is K-means [37]. However, no algorithm gives optimal results for all data sets. In this article, the LNS-SMC algorithm [24] is used because it has presented a good efficiency when the modularization quality (MQ) measure was applied to the software module clustering problem. The MQ measure is used to measure the clustering quality of software modules (e.g., classes and source code) according to the cohesion and coupling between modules. Cohesion is the number of internal dependencies a source file has in relation to its package, whereas coupling is the number of external dependencies. For example, a dependency is described when importing a class in Java. The goal of clustering is to reorganize the software into packages that are highly cohesive and loosely coupled.

Equation (9) explains the MQ measure from the perspective of the clustering problem of similar tasks by edit interaction. Let C represent the result of a clustering algorithm with n clusters, that is, $C = C_1, C_2, C_3, .., C_n$. The MQ quality measure of a cluster C is defined as the sum of the quality of each cluster C_k, i.e., MF_{C_k}. For this experiment, we define an association between two tasks when exists at least one artifact that was edited by developers who performed these two tasks. The value of MF is a function of the number of internal (i) and external (j) associations of C_k tasks. For example, when all tasks in a C_k cluster do not have common artifacts with tasks in the same C_k, i is equal zero and MF is also zero, and this is the worst case. In the best case, all tasks in a C_k cluster are related to the same artifacts and do not have artifacts in common with tasks of other clusters. Thus, $i > 0$, j is zero, and MF has a maximum value equal to 1.

$$MQ = \sum_{k=1}^{n} MF(C_k) \quad MF(C_k) = \begin{cases} 0 & \text{if } i = 0 \\ \dfrac{i}{i + \frac{j}{2}} & \text{if } i > 0 \end{cases} \tag{9}$$

The LNS-SMC algorithm is classified as a heuristic to find an optimal solution by exploring a large neighborhood seach (LNS) method that has been applied to the software module clustering (SMC) problem. The MQ measure is used by LNS-SMC to compare solutions and choose the one with the highest MQ. This algorithm initially evaluates the MQ of a solution, choosing it as optimal, and then makes use of two operators to choose the neighbor of the optimal solution. The *destroy* operator is applied to the optimal solution to generate an incomplete solution, for example to cause the removal of some elements. The *repair* operator is then applied to the incomplete solution, generating a valid solution by, for example, inserting the removed nodes into other incomplete solution clusters. Then, it is checked if the MQ of the generated valid solution is higher than the current optimal solution. If so, the valid solution becomes the current optimal solution. These steps are performed until a stop condition is satisfied. In this study, the maximum number of trials to find a new optimal solution was set to 1000. For each trial, 10 trials of the LNS were performed for Mylyn Docs project, and the largest cluster MQ was chosen to be used in the K_c and K_p rankings calculations. We observed that MQ values are so close for each trial with Mylyn Docs, so only one trial of LNS was performed for each trial of the projects selected from GitHub.

The experiment was performed with a reference date for each first day of the month, and involved Mylyn Docs project from 2015-05-01 to 2016-05-01, with a total of 13 trials , and GitHub projects from 2019-02-01 to 2020-01-01, with a total of 12 trials. Each trial produced four ranking lists in descending order of developer scores based on: 1 - the performed tasks; 2 - the number of edit interactions with artifacts; 3 - the K_c model, and 4 - the K_p model. The rankings by interactions, K_c and K_p, were evaluated according to the degree of correlation with the performed task ranking using the Spearman correlation model [38] with a confidence level defined in 95%. The Spearman correlation is used to evaluate the similarity between rankings. The Spearman correlation coefficient is close to one when the rankings are very similar and close to zero when they are very different [39].

C. RESULTS AND ANALYSIS

Table 4 presents the results of the first trial of the experiment Mylyn Docs project for 2015-05-01. The ranking by "performed tasks" orders the 10 developers who performed the greater number of tasks until 2015-05-01. The developer in the first place performed 378 tasks and the one in the tenth performed 3. The developer in the first place in this ranking was also the first in the other rankings. The developer dev1039 was ranked tenth in the ranking of performed tasks, 22nd in the interaction ranking, 13th in the ranking based on K_c and 13th in ranking based on K_p. It is expected that to accomplish a task, a developer should perform one or more interactions, but that's not what happened to the developers dev1512 and dev1039. Dev1512 performed 6 tasks and 1 interaction. Dev1039 performed 3 tasks and 1 interaction. We conclude that these developers did not record all their interactions while performing their tasks. We opted to keep this information in the assessment of similarities among the rankings.

The only interaction of the developer dev1512 happened on 2009-05-26 with artifact 6135 and had degree 1, and it is only part of task 260483. This interaction applied to the K_a model results in a value very close to zero (0.02). Consequently, the models K_c and K_p deprecated this interaction by inferring that the developer completely forgot the task he or she performed. On the other hand, the only interaction of the developer dev1039 with artifact 382 was registered on 2015-03-10 with degree 1, and it is also part of 29 tasks. According to K_a, the knowledge of dev1039 about this artifact on 2015-05-01 results in 0.44. In this case, the model K_c inferred the knowledge of the developer based on 13 out of 124 clusters, with K_p equal to 0.13. For the developer dev1039 the K_p measure indicates a value closer to reality than the K_c value.

Table 5 shows the results of the similarity analysis between the performed tasks ranking (i.e., baseline) with ranking based on the number of interactions, and using the K_c and K_p models. The study with Mylyn Docs project was organized in 13 trials, starting on May, 2015, and ending on May, 2016, and was always performed on the first day of each month. The 13th trial was ruled out because it had a p.value greater than 5% for K_p. All other trials had a p.value smaller than 5%, indicating a 95% confidence in the rho values of the Spearman tests. In general, the ranking based on the K_p model presents on average a "strong similarity" with the ranking of performed tasks (72%).

Based on these results, we provide the answers for three research questions raised in this experiment in the following paragraphs. Schober *et al.* [40] discuss the interpretations of Spearman test results. In this study, the correlation between rankings can also be understood as similarity between rankings, and to simplify the interpretation of rho values, the following scale will be used: 0.00-0.10 as a "negligible similarity"; 0.10-0.39 as a "weak similarity"; 0.40-0.69 as a "moderate similarity"; 0.70-0.89 as a "strong similarity" and 0.90-1.0 as a "very strong similarity".

Table 6 shows the results of the similarity analysis between the performed tasks ranking (i.e., baseline) with ranking based on the number of interactions, and using the K_c and K_p models for five projects selected from GitHub. These studies were organized in 12 trials, starting on February, 2019, and ending on January, 2020, and was always performed on the first day of each month. Trials with p.value greater than 0.05 were removed. In general, the rankings based on the number of interactions, the K_c and K_p models present on average a "strong similarity" with the ranking of performed tasks.

1) RQ1 ANALYSIS

The research question RQ1 was defined to evaluate the similarity between the number of performed task rankings and the number of interaction rankings performed per developer. Table 5 shows that the degree of similarity tends to increase with time: on 2015-05-01 it was 66% (rho) and on 2015-08-01 it became 69% in the 4th trial. These rankings are similar in 68% on average. According to the Likert scale for similarity evaluation adopted in this study plan, the performed task ranking and the interaction ranking have a "moderate similarity". The study with Mylyn Docs project indicates that the number of performed task rankings (R_{pt}) and the number of interactions ranking have a moderate similarity of 68% on average.

Table 6 shows the results of five studies with projects selected from GitHub. According to the Likert scale for similarity evaluation, the performed task ranking and the interaction ranking have a "strong similarity" for three projects and a "very strong similarity" for two projects.

2) RQ2 ANALYSIS

The research question RQ2 evaluates the similarity between the performed task rankings and the ranking based on the K_c model. Table 5 shows that the degree of similarity tends to increase over time: on 2015-05-01 it was 70% (rho) and on 2016-03-01 it was to 71% in the 11th trial. These rankings are similar in 70% on average, a better result compared to RQ1 (68%). The p.value was smaller than those obtained for RQ1. According to the Likert scale for similarity evaluation, the performed task ranking and the ranking based on K_c model reveal a strong similarity, but very close to the range

TABLE 4. Results of the first trial conducted on 2015-05-01.

Performed Tasks			Interactions		Kc		Kp	
Ranking	Developer	Tasks	Ranking	Interactions	Ranking	Clusters	Ranking	Measure
1º	dev327	378	1º	50984	1º	121	1º	1802.02
2º	dev3331	31	3º	1397	2º	108	2º	63.75
3º	dev36	13	8º	32	4º	23	4º	3.61
4º	dev919	8	4º	742	3º	41	3º	5.64
5º	dev890	7	7º	33	19º	5	16º	0.06
6º	dev1512	6	23º	1	24º	0	24º	0
7º	dev3974	5	2º	1459	6º	18	7º	1.02
8º	commiter71	4	15º	7	7º	17	12º	0.19
9º	dev84	4	12º	10	11º	15	8º	0.63
10º	dev1039	3	22º	1	13º	13	13º	0.13

TABLE 5. Similarity analysis between the performed tasks ranking (baseline) and the interaction rankings based on K_c and K_p with Mylyn Docs project.

		Interactions Ranking		Kc Ranking		Kp Ranking	
trial	date	rho	p.value	rho	p.value	rho	p.value
1ª	2015-05-01	0.66	<0.045	0.70	<0.032	0.68	<0.036
2ª	2015-06-01	0.66	<0.045	0.70	<0.032	0.68	<0.036
3ª	2015-07-01	0.66	<0.045	0.70	<0.032	0.68	<0.036
4ª	2015-08-01	0.69	<0.036	0.70	<0.032	0.70	<0.032
5ª	2015-09-01	0.69	<0.036	0.70	<0.032	0.66	<0.045
6ª	2015-10-01	0.69	<0.036	0.70	<0.032	0.75	<0.019
7ª	2015-11-01	0.69	<0.036	0.70	<0.032	0.76	<0.016
8ª	2015-12-01	0.69	<0.036	0.70	<0.032	0.76	<0.016
9ª	2016-01-01	0.69	<0.036	0.70	<0.032	0.76	<0.016
10ª	2016-02-01	0.69	<0.036	0.70	<0.032	0.76	<0.016
11ª	2016-03-01	0.69	<0.036	0.71	<0.028	0.75	<0.019
12ª	2016-04-01	0.69	<0.036	0.71	<0.028	0.75	<0.019
	Mean	0.68		0.70		0.72	

TABLE 6. Similarity analysis between the performed tasks ranking (baseline) and the interaction rankings based on K_c and K_p with GitHub projects.

		Mean		
Project	Trials (trials removed)	Interaction	Kc Ranking	Kc Ranking
apache/commons-lang	12 trials from 2019-02-01 to 2020-01-01 (3,4,9 and 10)	0.74	0.72	0.74
brewsci/homebrew-bio	12 trials from 2019-02-01 to 2020-01-01 (6,7,8 and 9)	0.96	0.89	0.89
google/EarlGrey	12 trials from 2019-02-01 to 2020-01-01 (5, 6 and 7)	0.88	0.87	0.90
Homebrew/homebrew-cask-drivers	12 trials from 2019-02-01 to 2020-01-01 (-)	0.85	0.78	0.83
iodide-project/iodide	12 trials from 2019-02-01 to 2020-01-01 (-)	0.91	0.88	0.88

considered as moderate (i.e., rho value from 0.40 to 0.69). The study with Mylyn Docs project indicates that the R_{pt} ranking and the ranking generated by K_c model have a strong similarity of 70% on average.

Table 6 shows that the performed task ranking and the ranking based on the K_c model have a "strong similarity" for the five projects.

3) RQ3 ANALYSIS

The research question RQ3 was defined to evaluate the similarity between the performed task ranking and the ranking based on the K_p model. Table 5 shows that these rankings are similar in 72% of the cases on average, a better result in comparison with the results of RQ1 and RQ2, which were 68% and 70% of the cases on average, respectively. The p.value values also decreased significantly with respect to RQ1 and RQ2 for the 6th-10th trials, indicating a higher degree of confidence for similarity values (rho). According to the Likert scale for similarity evaluation, the performed task ranking and the ranking based on the K_p model revealed a strong similarity, which is higher than the upper limit of 0.69 of a moderate similarity (i.e., with value from 0.40 to 0.69). The study indicates that the R_{pt} ranking and the ranking generated by K_p model have a strong similarity of 72% on average.

Table 6 shows that the performed task ranking and the ranking based on the K_p model have a "strong similarity" for four projects and a "very strong similarity" for one project.

4) SENSITIVITY ANALYSIS

Table 7 shows the influence of R_d, G_d, $\frac{h}{m}$ and $\frac{r}{q}$ parameters on K_p, calculated on 11/11/2008 with data from the Mylyn Docs project. The better project organization with respect to similar-task groups resulted in 22 groups ($q = 22$). The second column presents the K_p measurements according to equation (8). The K_p measurements for the three developers in the third column, when calculated without considering forgetting ($R_d = 1$), have values greater than twice those in the second column, showing that modeling forgetting has a significant influence on the K_p measurements. The fourth column, with K_p values for $G_d = 1$, has lower values for the

TABLE 7. Sensitivity analysis of K_p related to the Mylyn Docs project on 11/11/2008.

	K_p	K_p ($R_d = 1$)	K_p ($G_d = 1$)	K_p ($h = m = r = q = 1$)
1º	2544	5135	2530	2545
2º	588	1643	578	603
3º	36	142	36	121

first two developers and equal values for the third developer. This indicates that the first two developers revisited artifacts in the last seven days and the third did not. The fourth column, with K_p measurements for h, m, r and q values equal to 1, shows values that are very close to the values in second column for the first developer, indicating that the first developer has knowledge about most tasks and most similar task groups (i.e., $r = 22$ for this developer). Finally, the third developer has a K_p value equal to 36 in the second column and 121 in the fourth column, which indicates an influence of these parameters on the K_p measurements. This developer has knowledge of only 7 ($r = 7$) project task groups within a total of 22 ($r = 22$).

Table 8 shows the influence of R_d, G_d, $\frac{h}{m}$ and $\frac{r}{q}$ parameters on K_p, calculated on 11/11/2011 with data from the Mylyn Docs project. We note that in this table the K_p measurements with $R_d = 1$ change the developers' ranking list when compared with K_p: the 3rd developer, according to the K_p model, appears in the 5th place (for K_p with $R_d = 1$); the 4th developer appears in 3th place; and the 5th developer appears in 4rd place. The third column, in which we have K_p without relearning, shows that only the first developer has a smaller measure in comparison with the first column, because this developer has interacted with artifacts in the last seven days to accomplish his or her tasks. Finally, the last column, in which K_p has the parameters h, m, r and q equal to 1, shows that all developers are negatively affected with respect to K_p when they do not have knowledge about all tasks and task groups in the project.

D. THREATS TO VALIDITY

According to [41], threats to validity can affect an experimental study in conclusion, construct, internal, and external threats. Conclusion refer to the relationship between treatment and outcome. The baseline of this study is the number of resolved tasks by developer and the other rankings have as their main input the developers' interactions with artifacts while performing tasks. Some developers of the Mylyn Docs project did not record the edit interactions, having recorded only the commit interactions. Therefore, we assign a degree equal to 1 to each commit interaction for all artifacts involved in a commit. Even so, among the 10 most accomplished developers, we have found out that two of them performed more tasks than artifact interactions. As a result, the correlations among rankings may be underestimated. Future work plans include the implementation of models and evaluations in additional software development projects in academia and industry.

Internal threats evaluate whether the relationship between treatment and outcome is causal or results from factors that the researcher cannot control. First, in a typical software development scenario where developers are working in the same room and participating in face-to-face interactions, knowledge exchanged about tasks and artifacts at first might not be captured or transmitted using an IT tool. As a consequence, interactions could occur without leaving a trace. However, we claim that these interactions are usually followed by interactions by those same developers using development and project management tools, which will update the information related to the artifacts and tasks, and reflect the decisions made during the face-to-face interactions. Second, interaction-based rankings cannot evaluate whether an interaction was positive, i.e., whether it contributed to the task or was dropped via undo-type commands. Third, there may be a distinction between the number of interactions of more experienced and less experienced developers in performing tasks with the same degree of difficulty. In the same way that personal questions (e.g., involving profiles or emotions) can influence how a developer accomplishes the task, it can also influence how he or she interacts with artifacts and with collaborators. One must investigate how the presented models can filter the noise associated with these questions.

Construct threats are associated with the relation between theory and observation, and are defined as to ensure that the treatment reflects the construct of the cause well and the outcome reflects the construct of the effect well. The K_c and K_p-based rankings did not use I_{dd}-type interactions, those observed among developers and predicted by the K_s model, because such interactions were not measured in the Mylyn Docs project. These interactions need to be modeled and considered in future studies.

Finally, external threats relate to the generalization of the observed results for other software projects. Our model is based on Developer-Artifact interactions (I_{da}), and Developer-Developer interactions (I_{dd}). These interactions are present in all software development projects. However, it is necessary to provide the projects with tools to support obtaining appropriate information about these interactions in other domains.

V. DISCUSSION

The proposed models for measuring developer's knowledge were evaluated to identify developers who are most knowledgeable about a software development project. The K_p model measures developer's knowledge about the entire project based on the K_c model, which measures developers' knowledge about a group of similar tasks using Developer-

TABLE 8. Sensitivity analysis of K_p related to the Mylyn Docs project on 11/11/2011.

K_p		K_p ($R_d = 1$)		K_p ($G_d = 1$)		K_p ($h = m = r = q = 1$)	
Ranking	Measure	Ranking	Measure	Ranking	Measure	Ranking	Measure
1º	2751	1º	40561	1º	2698	1º	2814
2º	118	2º	2924	2º	118	2º	128
3º	15.2	4º	71	3º	15.2	3º	66
4º	2.96	5º	4	4º	2.96	4º	26
5º	2.79	3º	87	5º	2.79	5º	15.4

Artifact interactions, and this is done based on the K_s model. The K_s model measures the developers' knowledge about a task, which in turn uses the K_a model, the model that measures the developers' knowledge about a artifact. Thus, the results of the K_p evaluation also represent an evaluation of the K_c, K_s and K_a models.

The study evaluates K_p based on the assumption that a developer who performed a task, also acquired knowledge about it. This premise is stated by Fritz *et al.* [6], who argue that authorship is a relevant information for measuring knowledge. The K_p evaluation used K_s with Developer-Artifact interactions and did not consider Developer-Developer interactions because of the lack of data. According to the experiment performed by Moraes *et al.* [9], Developer-Developer interactions can also contribute to the identification of experts. Thus, the inclusion of data from Developer-Developer interactions may improve the results of the evaluation models.

We should stress that the structure of our proposed models is based on fine-grained information. For example, in the K_a model an interaction between a developer and an artifact creates a link between these elements. Then, after performing a task, a developer tends to have multiple associations with the same artifact. These associations are also part of the other models (K_s, K_c and K_p) because they use K_a. A previous study conducted by the lead author of this present article suggested that these associations may contribute to the improvement of information quality [42]. In that study, the developer who most knew about the Mylyn project was the same in three observation perspectives: (i) the number of performed tasks according to the Mylyn project's official list; (ii) the number of performed tasks according to the data collected for a preliminary evaluation of the K_p model; and (iii) the K_p model applied to the collected data. In all three perspectives, the developer who ranked first performed 2,053 tasks in the official list, with 765 performed tasks in the collected data, and with the K_p model producing a value of 479,285. Specifically, according to this article, in the first trial of the experiment, which is presented in Table 4, the developer in first place was also the same, with 378 performed tasks, a total of 50,984 interactions (i.e., edition and commits) and a final value of K_p of 1802.02.

In this study, the baseline - the number of performed tasks - represents the knowledge of what was done because it indicates who did which task, assigning 1 to the developer for each performed task. The K_p model represents the knowledge of how it was done. Each performed task is associated with the artifacts by the Developer-Artifact interactions, and the value of knowledge attributed to the developer varies according to the interaction degree on artifacts over time. The baseline was compared to the number of interactions performed in the project. The average correlation degree with Mylyn Docs project was 68% with p.value values greater than 3.5% and less than 4.5% in 12 trials. However, the interaction rankings also do not take into account the developers' forgetting and relearning. The comparison of the baseline with the ranking generated by the K_c model presented, on average, a correlation of 70% with p.value values less than 3.2% in 12 trials. The K_c ranking presented a higher correlation than the interaction ranking besides inheriting from K_s and K_a the model of forgetting and relearning of developers. The comparison of the baseline with K_p showed a better result in 12 trials, the average correlation was 72% with most values of p.value below 2% for the 6th to the 10th trials. Furthermore, because the K_p model makes use of the K_c model, it also inherits the forgetting model from K_s and K_a.

The results of five studies with projects selected from GitHub presented an average similarity degree next to 90% with performed tasks. For these projects, the accomplishment of a task was attributed to every developer who performed at least one commit on the task's artifacts. This decision may have contributed to the fact that the rankings by interactions, K_c, and K_p had similar average values. In addition, commits capture the results of various interactions between developers and artifacts. This indicates that considering data from software such as Mylyn, which works by capturing Developer-Artifact interactions, can contribute to measure developers' knowledge.

The results show that the K_p model has presented more similarity than the K_c model when compared to the R_{pt}. On the other hand, K_p model has also presented results close to the comparison of the number of interactions with the R_{pt}. So, these results are evidence that the K_p model is suitable to assess the evolution of the developers' knowledge for finding who knows more about specific elements of a software project and how this knowledge decreases over time.

There is no evidence that only the number of performed tasks can accurately express the developer's knowledge about a software project. As previously stated, Developer-Developer interactions are also relevant in this aspect. However, the measured number of performed tasks can be taken as a useful measure of knowledge because those who perform more tasks also tend to participate in the creation and evolution of software artifacts, when the explicit knowledge of developers is recorded. Thus, the correlation of K_p with the number of performed tasks provides some evidence that the presented knowledge-oriented models contribute to the research direction that aims at measuring developers' knowledge appropriately.

DOK is the closest work related to our knowledge measurement models [6]. Therefore, it could be a candidate to be a basis for comparison in the evaluations of the presented models. However, Fritz *et al.* showed that the DOK model is not suitable for measuring knowledge about APIs [6], [21]. For this reason, in this study we adopted performed tasks as the most appropriate measure to measure knowledge about a software development project. Besides, assigning knowledge about a task as a result of its execution is a well-accepted practice adopted by the software community, and, as already mentioned, it is also a form of authorship, an information that contributes to the measurement of knowledge.

Our models do not use authoring information directly as the DOK model does, but it does so in an indirect way. The author of an artifact is anyone who contributed to its creation. The models punctuate these contributions, thus also indicating a degree of developer authorship related to artifacts, tasks, similar tasks, and the whole project. On the other hand, software often grows in size over time and, as a result, the artifacts also grow in size with changes caused by error correction and the addition of new features. Therefore, the author of an artifact is not only the developer who created it, but also all the developers who contributed to its current version. In this sense, our models consider the authorship information related to the developers and use this information in an appropriate way to measure developers' knowledge.

Overall, knowledge-oriented models are a complement to information about the number of performed tasks to identify who knows more about the whole or parts of the software development project.

VI. RELATED WORK

Several works extract knowledge from developers' interactions with software artifacts. Table 9 shows some related works from three observation perspectives: the source, the model and the target of the selected approaches. The first column shows how the approach captures information about the developers' interactions with artifacts. The modeling perspective, depicted in the second column, identifies whether the approach explicitly modeled the human forgetting and relearning. Finally, the target observation perspective, in the third column, shows whether the approach infers knowledge about artifacts, tasks, similar tasks, and software projects.

Robbes and Röthlisberger [43] infer the developers' knowledge from their editing, selection, and commit interactions with artifacts over time. They define two models that take into account the developers' forgetting. One model deals with knowledge about artifacts and the other with knowledge about tasks. Robbes & Röthlisberger's models neither consider developers' relearning nor infer developers' knowledge about similar tasks and software projects.

McDonald and Ackerman [44] defined the "Line 10 Rule" heuristic. According to the result of their study, the specialist in a software module is assumed to be the developer who made the last change in the module. The Expertise Recommender (ER) is based on this heuristic and can recommend an expert on an artifact or software module and also help to identify a module associated with a reported error. ER does not model the developer's relearning or infer the developers' knowledge about similar tasks and software projects.

Mockus and Herbsleb [45] defined expertise atoms as elementary units of expertise used to measure the degree of expertise of a developer. They have defined expertise in an object as the set of these elementary units associated with this object. In software development, the elementary unit of expertise is the available difference among versions of a file, which in many code repositories are the updated rows. The Expertise Browser (EA) is an implementation of this expertise measurement model: it lists the experts on an object (file) based on the number of changes (i.e., expertise atoms) made by the developers. EA can list the experts by code, documentation, functionality or product. According to Table 9, EA infers knowledge from the editions in the artifacts, does not consider the developers' forgetting and relearning, acts on artifacts, tasks and software projects, and does not infer the knowledge about similar tasks.

Girba *et al.* [46] defined a model to determine the owner of an artifact in function of the current owners of the artifact lines. They stated that the developer who last modified an artifact line is the current owner of the line. In this way, it is possible to infer that the current owner of an artifact is the developer who owns the most artifact lines. Therefore, Girba *et al.* can infer the owners of the software artifacts, and also produce a preview of the software project from the perspective of the artifact owners. This approach infers knowledge based on artifact commits and does not take into account the developers' forgetting and relearning. The model developed by Girba and colleagues can not be applied to tasks or similar tasks.

Vivacqua and Lieberman [47] have explored the use of artifacts by developers working in the Java language domain to infer knowledge according to a Likert scale (novice - beginner - intermediate - advanced - expert). The use of an artifact is measured using the parser on the Java files by analyzing which libraries, classes and methods were used and how often they were used. It is thus possible to infer the developers' knowledge about the artifacts from information about their use. This approach neither takes into account developers' forgetting or relearning nor it can be applied to similar tasks and software projects.

Mylyn captures a developer's interest in specific artifacts based on the developer's interaction with the environment [27]. In Mylyn, when a new task is started, the initial context is the same as that of the set of artifacts available in the project, and as the developer interacts with the environment,

TABLE 9. Works that extract knowledge from developers' interactions with software artifacts.

		Model		Target			
Approach	Source	Forget	Relearn	Artifact	Task	Similar Tasks	Software Project
[43]	Edits, selections, and commits in artifacts	yes	no	yes	yes	no	no
[44]	Commits in artifacts	no	no	yes	yes	no	no
[45]	Edits in artifacts	no	no	yes	yes	no	yes
[46]	Commits in artifacts	no	no	yes	no	no	yes
[47]	Artifact usage	no	no	yes	yes	no	no
[27]	DOI model: Visualization, editing, selection, command, propagation and prediction	no	no	yes	no	no	no
[21]	DOK = DOI + Authorship	no	no	yes	no	no	no
[48]	Edits in artifacts	yes	no	yes	no	no	no
[49]	Commits in artifacts	no	no	yes	no	no	yes
Our approach, see Section III	Edits in artifacts	yes	yes	yes	yes	yes	yes

Mylyn will reduce the number of artifacts that can be viewed to minimize the developer's effort in searching for artifacts. Mylyn is based on the Degree Of Interest (DOI) model. DOI infers the developer's interest in an artifact over time based on visualization, editing, selection, command, propagation, and prediction events. The DOI model does not represent explicitly developers' forgetting and relearning about artifacts. It also does not predict interest levels directly based on tasks, similar tasks, and software projects.

Fritz *et al.* [21] proposed the Degree Of Knowledge (DOK), a model for inferring developers' knowledge about artifacts. This model is based on the authorship of an artifact combined with the DOI model. DOK does not explicitly model developers' forgetting and relearning, but inherits from the DOI model the ability to represent developer's interest or disinterest in specific artifacts. Therefore, DOK models can implicitly be used to represent forgetting and relearning. Besides, DOK does not act on a task, similar tasks, and software projects.

Hattori *et al.* [48] defined a successful change of an artifact as any error-free compilation event of the artifact executed after at least one edit. They assume that whoever has made the most changes to an artifact becomes the developer who knows the most about the artifact. Hattori and colleagues model the concept of forgetting by considering that more recent changes in time contribute more to knowledge than older changes, and similarity to Girba *et al.*'s work [46], they propose a visualization map of the artifacts and their respective owners. In this way, through the visualization map, it is possible to visualize the whole software project. This approach does not consider the concept of relearning of past tasks and the knowledge model does not apply to tasks.

The approach proposed by da Silva *et al.* [49] infers the developer's knowledge about a project, package, file, class or method based on the analysis of the editions made by him or her over time. The analysis is done by extracting commit information from code repositories, and associating each commit to a developer, package, file, class, and method in a project. Da Silva and colleagues do not consider forgetting and relearning, and also do not infer knowledge about task and similar tasks.

Finally, our knowledge-oriented models infer knowledge from the developer's edit interactions about software artifacts. In contrast to other models, our models consider the developers' forgetting to infer knowledge about artifacts, tasks, similar tasks, and software projects over time. So far, we could not identify any related work that models both forgetting and relearning, and acts based on the targets mentioned in Table 9.

VII. CONCLUSION AND FUTURE WORK

The main goal of this article was to present and evaluate a set of knowledge-oriented models based on forgetting and relearning to measure developer knowledge about artifacts, tasks, similar tasks and the whole software projects by analyzing developers' interactions while they perform tasks. Our models take into account the complexity of the tasks, the human trait of forgetting and relearning, the task similarity based on interactions, and the distribution of developers' knowledge in a software project. The knowledge-oriented models can also be seen as a novel solution to describe knowledge about how software is produced by linking developers, artifacts and tasks over time.

The evaluation of the models was performed based on information related to the interaction degree provided by the Mylyn Eclipse Plugin. In the literature, there are other ways of measuring interaction, such as the one described in the work by Omoronyia [50]. The models proposed in this article depend on measures of the interaction degree and, for this reason, the choice of the specific form of measurement may interfere with the results produced by the models. Besides, models are restricted to a particular software project. For example, knowledge that is recorded in an artifact of the same name in another software project is not considered. Future work will further investigate II_{dd} in the K_s model.

Personal issues (e.g., profile, emotions) can interfere with how a developer performs the tasks and also how he or she interacts with the artifacts and their collaborators. We need to investigate the influence of these issues on the presented models that aim at measuring developers' knowledge. Moreover, the evaluation needs to be extended by interviews or surveys to compare results from the knowledge-oriented models with the perception of developers about the experts.

A general limitation of our models is that the computation (especially the normalized computations) of the values in the model need to be done with respect to a specific time. We understand the proposed approach can work in general in two ways: (i) a static way, in which, the information is generated in a batch mode, and computations are performed with respect to a specific time based on this data; and (ii) a dynamic way, which takes into account the streaming nature of the information that is generated as a function of the interactions of developers with artifacts and other developers, and provides recommendations in real time. The proposed approach currently supports solution (i) and, because of this limitation, we provide an overview in this paragraph on how this approach can be extended to support solution (ii). A possible way to address this limitation is by using regression and correlation [51]. We suggest replacing t (time) by a time period constant in our knowledge models to associate developers with elements directly, i.e., $K_a(d, a)$, $K_s(d, s)$, $K_c(d, c)$, and $K_p(d, p)$. These extended models must be executed according to the time period chosen, adding or removing a degree of knowledge to the developer. This strategy was used by Fritz *et al.* [6].

We have performed a preliminary survey involving some GitHub projects to evaluate how well developers know: 1- the content of the source code (i.e., artifact) about which they have made commits; and 2- closed issues (i.e., tasks) associated with these artifacts. Currently, we are preparing a new survey with project data related to a specific software organization. In addition, our relearning modeling only considers the last 7 days, but we can not ascertain that the eighth, the ninth, or the subsequent days are not relevant. Future studies also need to address this issue.

Further, the proposed knowledge-oriented models can also be used in other domains with some adaptations. For example, in the Web domain users interact with online sites to achieve a goal. In this case, the equations of the presented knowledge-oriented models could be adapted to estimate the users' interest in products, services or subjects. In the scenario of distance education, where students interact with online systems and tutors to fulfill tasks, in order to acquire knowledge, there is also an opportunity to apply the proposed models. One possible application could be to estimate the degree of student involvement in terms of effort per course and associate this degree with student grades.

Finally, these models contribute to the burgeoning literature on developers' interactions as a source for new and significant discoveries related to the software development process.

REFERENCES

[1] I. Mistrík, J. Grundy, A. van der Hoek, and J. Whitehead, "Collaborative software engineering: Challenges and prospects," in *Collaborative Software Engineering* I. Mistrík, J. Grundy, A. Hoek, and J. Whitehead, Eds. Berlin, Germany: Springer, 2010, pp. 389–403.

[2] R. E. Fairley, J. M. Glabas, and R. H. Thayer, *IEEE Standard for Software Project Management Plans*, document ISO/IEC/IEEE 16326, 1988, pp. 1–28.

[3] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proc. 29th Int. Conf. Softw. Eng. (ICSE)*, May 2007, p. 344–353.

[4] H. L. Roediger, III, "Relativity of remembering: Why the laws of memory vanished," *Annu. Rev. Psychol.*, vol. 59, no. 1, pp. 225–254, Jan. 2008.

[5] W. Thalheimer. (2010). *How Much do People Forget*. Accessed: Nov. 31, 2017. [Online]. Available: http://www.work-learning.com/catalog.html

[6] T. Fritz, G. C. Murphy, E. Murphy-Hill, J. Ou, and E. Hill, "Degree-of-knowledge: Modeling a developer's knowledge of code," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, Apr. 2014.

[7] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, and A. Mockus, "Quantifying and mitigating turnover-induced knowledge loss: Case studies of chrome and a project at avaya," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 1006–1016.

[8] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "DRETOM: Developer recommendation based on topic models for bug resolution," in *Proc. 8th Int. Conf. Predictive Models Softw. Eng.*, 2012, p. 19.

[9] A. Moraes, E. Silva, C. da Trindade, Y. Barbosa, and S. Meira, "Recommending experts using communication history," in *Proc. 2nd Int. Workshop Recommendation Syst. Softw. Eng.*, 2010, p. 41.

[10] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *Proc. 20th Work. Conf. Reverse Eng. (WCRE)*, Oct. 2013, pp. 72–81.

[11] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *J. Syst. Softw.*, vol. 117, pp. 166–184, Jul. 2016.

[12] J. Zhu, B. Shen, and F. Hu, "A learning to rank framework for developer recommendation in software crowdsourcing," in *Proc. Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2015, pp. 285–292.

[13] Z. Wang, H. Sun, Y. Fu, and L. Ye, "Recommending crowdsourced software developers in consideration of skill improvement," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Oct. 2017, pp. 717–722.

[14] J. M. J. Murre and A. G. Chessa, "Power laws from individual differences in learning and forgetting: Mathematical analyses," *Psychonomic Bull. Rev.*, vol. 18, no. 3, pp. 592–597, Jun. 2011.

[15] J. M. J. Murre, A. G. Chessa, and M. Meeter, "A mathematical model of forgetting and amnesia," *Frontiers Psychol.*, vol. 4, p. 76, Feb. 2013.

[16] J. M. J. Murre and J. Dros, "Replication and analysis of Ebbinghaus' forgetting curve," *PLoS ONE*, vol. 10, no. 7, Jul. 2015, Art. no. e0120644.

[17] D. C. Rubin, S. Hinton, and A. Wenzel, "The precise time course of retention.," *J. Experim. Psychol., Learn., Memory, Cognition*, vol. 25, no. 5, pp. 1161–1176, 1999.

[18] D. C. Rubin and A. E. Wenzel, "One hundred years of forgetting: A quantitative description of retention.," *Psychol. Rev.*, vol. 103, no. 4, pp. 734–760, Oct. 1996.

[19] J. T. Wixted and E. B. Ebbesen, "On the form of forgetting," *Psychol. Sci.*, vol. 2, no. 6, pp. 409–415, Nov. 1991.

[20] T. Fritz, G. C. Murphy, and E. Hill, "Does a programmer's activity indicate knowledge of code?" in *Proc. 6th Joint Meeting Eur. Softw. Eng. Conf.*, 2007, pp. 341–350.

[21] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng. - ICSE*, 2010, pp. 385–394.

[22] W. Maalej, *Intention-based Integration of Software Engineering Tools*. München, Germany: Verlag Dr. Hut, 2010.

[23] D. W. Hubbard, *How to Measure Anything: Finding the Value of Intangibles in Business*. Hoboken, NJ, US: Wiley, 2007.

[24] M. C. Monçores, A. C. F. Alvim, and M. O. Barros, "Large neighborhood search applied to the software module clustering problem," *Comput. Oper. Res.*, vol. 91, pp. 92–111, Mar. 2018.

[25] W. B. Frakes and K. Kang, "Software reuse research: Status and future," *IEEE Trans. Softw. Eng.*, vol. 31, no. 7, pp. 529–536, Jul. 2005.

[26] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes, "PROV-DM: The PROV data model," L. Moreau and P. Missier, Eds., World Wide Web Consortium, Project Rep., 2013. [Online]. Available: https://eprints.soton.ac.uk/356851/

[27] M. Kersten, "Focusing knowledge work with task context," Ph.D. dissertation, Univ. British Columbia, Vancouver, BC, Canada, 2007.

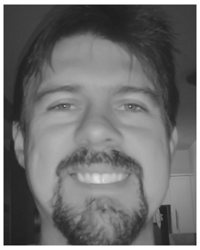
[28] A. Chessa and J. Murre, "Spurious power laws of learning and forgetting: Mathematical and computational analyses of averaging artifacts," in *Proc. Annu. Meeting Cognit. Sci. Soc.*, vol. 31, pp. 1175–1179, Jul. 2009.

[29] M. D. Lee, "A Bayesian analysis of retention functions," *J. Math. Psychol.*, vol. 48, no. 5, pp. 310–321, Oct. 2004.

[30] J. Krüger, J. Wiemann, W. Fenske, G. Saake, and T. Leich, "Do you remember this source code?" in *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 764–775.
[31] S. D. Sheetz, D. Henderson, and L. Wallace, "Understanding developer and manager perceptions of function points and source lines of code," *J. Syst. Softw.*, vol. 82, no. 9, pp. 1540–1549, Sep. 2009.
[32] V. Basili, "Software modeling and measurement: The goal/question/metric paradigm," Univ. Maryland, College Park, MD, USA, Tech. Rep. CS-TR-2956, UMIACS-TR-92-96, Sep. 1992.
[33] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical Softw. Eng.*, vol. 10, no. 3, pp. 311–341, Jul. 2005.
[34] J. Han, M. Kamber, and J. Pei, "10—Cluster analysis: Basic concepts and methods," in *Data Mining*, J. Han, M. Kamber, and J. Pei, Eds. Burlington, MA, USA: Morgan Kaufmann, Jan. 2012, pp. 443–495,
[35] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999.
[36] R. Xu and D. Wunsch, II, "Survey of clustering algorithms," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 645–678, May 2005.
[37] T. Boongoen and N. Iam-On, "Cluster ensembles: A survey of approaches with recent extensions and applications," *Comput. Sci. Rev.*, vol. 28, pp. 1–25, May 2018.
[38] D. J. Best and D. E. Roberts, "Algorithm AS 89: The upper tail probabilities of Spearman's rho," *J. Roy. Stat. Soc. C*, vol. 24, no. 3, pp. 377–379, 1975.
[39] A. A. Goshtasby, "Similarity and dissimilarity measures," in *Image Registration: Principles, Tools and Methods*, A. A. Goshtasby, Ed. London, U.K.: Springer, 2012, pp. 7–66.
[40] P. Schober, C. Boer, and L. A. Schwarte, "Correlation coefficients: Appropriate use and interpretation," *Anesthesia Analgesia*, vol. 126, no. 5, pp. 1763–1768, May 2018.
[41] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic, 2000.
[42] E. M. Lucas and T. C. Oliveira, "Um modelo de percepção para indicar o contexto de novas tarefas em projetos de software," in *Proc. VI Workshop Theses Diss.*, 2016, pp. 10–18.
[43] R. Robbes and D. Rothlisberger, "Using developer interaction data to compare expertise metrics," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 297–300.
[44] D. W. McDonald and M. S. Ackerman, "Expertise recommender: A flexible recommendation system and architecture," in *Proc. ACM Conf. Comput. Supported Cooperat. Work*, 2000, pp. 231–240.
[45] A. Mockus and J. D. Herbsleb, "Expertise browser: A quantitative approach to identifying expertise," in *Proc. 24th Int. Conf. Softw. Eng.*, 2002, pp. 503–512.
[46] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse, "How developers drive software evolution," in *Proc. 8th Int. Workshop Princ. Softw. Evol.*, 2005, pp. 113–122.
[47] A. Vivacqua and H. Lieberman, "Agents to assist in finding help," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2000, pp. 65–72.
[48] L. P. Hattori, M. Lanza, and R. Robbes, "Refining code ownership with synchronous changes," *Empirical Softw. Eng.*, vol. 17, nos. 4–5, pp. 467–499, Aug. 2012.
[49] J. R. da Silva, E. Clua, L. Murta, and A. Sarma, "Niche vs. Breadth: Calculating expertise over time through a fine-grained analysis," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Mar. 2015, pp. 409–418.
[50] I. Omoronyia, "Enhancing awareness during distributed software development," Ph.D. dissertation, Dept. Comput. Inf. Sci., Univ. Strathclyde, Glasgow, U.K., 2008.
[51] J. Lee Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *Amer. Statistician*, vol. 42, no. 1, pp. 59–66, Feb. 1988.

EDSON M. LUCAS received the B.S. degree in computer science from the IM–Federal University of Rio de Janeiro (UFRJ), in 1998, and the M.S. and Ph.D. degrees in systems engineering and computer science from COPPE/UFRJ, in 2013 and 2019, respectively. He is currently a Researcher at Polytechnic Institute (IPRJ), Rio de Janeiro State University (UERJ), Brazil. He has experience in computer science, with an emphasis on software engineering, focusing on software reuse, frameworks, object-oriented design and programming, collaborative processes, and recommendation systems in software engineering.

TOACY C. OLIVEIRA received the bachelor's degree in electrical engineering and the M.Sc. and Ph.D. degrees from the Pontifical Catholic University of Rio de Janeiro, Brazil, in 1992, 1997, and 2001, respectively. He worked as a Postdoctoral Researcher with the University of Waterloo, Canada, for three years. He is currently an Associate Professor with the Computing and Systems Engineering Program (PESC/COPPE), Federal University of Rio de Janeiro, Brazil. He is also an Adjunct Professor with the David R. Cheriton School of Computer Science, University of Waterloo. He has authored or coauthored more than 70 refereed publications. Besides being a leading investigator in Brazilian research projects supported by CAPES and CNPq, he has also exercised entrepreneurship by founding several companies in Brazil. His current research interests include understanding the complexities of knowledge intensive processes and developing new tools and techniques to help domain experts, a.k.a knowledge workers, achieving their goals. He has been a member of program committees of numerous conferences and workshops.



DANIEL SCHNEIDER (Member, IEEE) received the B.S. degree in computer science from the IM–Federal University of Rio de Janeiro (UFRJ), in 2001, and the M.S. and Ph.D. degrees in systems and computer engineering from COPPE/UFRJ, in 2004 and 2015, respectively.

From 1999 to 2011, he worked as a Software Consultant and also as the Leader of the Development of a Data Warehouse in the Brazilian Navy, through the Coppetec Foundation, Rio de Janeiro, before joining the Federal Rural University of Rio de Janeiro and acting as a Professor since 2011. Since 2017, he has been an Assistant Professor at the Federal University of Rio de Janeiro (UFRJ). He has been working at Tércio Pacitti Institute of Computer Applications and Research (NCE) and the Postgraduate Program in Informatics (PPGI/UFRJ). He is also an inventor and the TI Leader of the Acropolis Platform, a social curation environment specially designed to engage citizens in complex or long-running news stories. He has experience in databases, CSCW, and social and crowd computing systems. He has published more than 40 peer-reviewed articles in conferences and journals in the area of computer supported cooperative work and human–computer interaction. His current research interests include designing and implementing crowdsourcing systems to solve problems in a myriad of application areas.

PAULO S. C. ALENCAR (Member, IEEE) is currently a Research Professor with the David R. Cheriton School of Computer Science, University of Waterloo, and also an Associate Director of the Computer Systems Group. He has authored or coauthored more than 200 refereed publications and has been a member of program committees of numerous highly regarded conferences and workshops. His recent research in information technology and software engineering has focused on high-level software architectures, design, components, and their interfaces, software frameworks and application families, software processes, automated workflows, work graphs, and evolution, Web-based approaches and applications, open and big data applications, context-aware and event-based systems, software agents, machine learning, cognitive chatbots, artificial intelligence, and formal methods. He has received international research awards from organizations, such as Compaq and IBM. He has been a Principal or a Co-Principal Investigator in projects supported by NSERC, ORF-RE, IBM, SAP, CITO, CSER, Bell, and funding agencies in Canada, U.S., Brazil, Germany, and Argentina. He is a member of the Association of Computing Machinery, the Association for the Advancement of Artificial Intelligence, the Waterloo Water Institute (part of the Global Water Futures initiative), and the Waterloo Artificial Intelligence Institute.