The Journal of Systems and Software 000 (2017) 1–18

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Using contexts similarity to predict relationships between tasks

Walid Maalej [a,*], Mathias Ellmann [a], Romain Robbes [b]

[a] Department of Informatics, University of Hamburg, Hamburg, Germany
[b] Computer Science Department, University of Chile, Santiago de Chile, Chile

ARTICLE INFO

Article history:
Received 23 September 2015
Revised 16 November 2016
Accepted 23 November 2016
Available online xxx

Keywords:
Task context
Task dependencies
Task management
Developer productivity
Empirical study
Recommender systems

ABSTRACT

Developers' tasks are often interrelated. A task might succeed, precede, block, or depend on another task. Or, two tasks might simply have a similar aim or require similar expertise. When working on tasks, developers interact with artifacts and tools, which constitute the contexts of the tasks. This work investigates the extent to which the similarity of the contexts predicts whether and how the respective tasks are related. The underlying assumption is simple: if during two tasks the same artifacts are touched or similar interactions are observed, the tasks might be interrelated.

We define a task context as the set of all developer's interactions with the artifacts during the task. We then apply Jaccard index, a popular similarity measure to compare two contexts. Instead of only counting the artifacts in the intersection and union of the contexts as Jaccard does, we scale the artifacts with their relevance to the task. For this, we suggest a simple heuristic based on the Frequency, Duration, and Age of the interactions with the artifacts (FDA). Alternatively, artifact relevance can be estimated by the Degree-of-Interest (DOI) used in task-focused programming.

To compare the accuracy of the context similarity models for predicting task relationships, we conducted a field study with professionals, analyzed data from the open source task repository Bugzilla, and ran an experiment with students. We studied two types of relationships useful for work coordination (dependsOn and blocks) and two types useful for personal work management (isNextTo and isSimilarTo). We found that context similarity models clearly outperform a random prediction for all studied task relationships. We also found evidence that, the more interrelated the tasks are, the more accurate the context similarity predictions are.

Our results show that context similarity is roughly as accurate to predict task relationships as comparing the textual content of the task descriptions. Context and content similarity models might thus be complementary in practice, depending on the availability of text descriptions or context data. We discuss several use cases for this research, e.g. to assist developers choose the next task or to recommend other tasks they should be aware of.

1. Introduction

A task is an atomic and well-defined work assignment for an individual or a team (Bruegge and Dutoit, 2009; Kersten and Murphy, 2006). In software projects, tasks typically describe what developers should do and might include additional information such as the work priority, the resources needed, or the component affected. Tasks can be defined, e.g., in issue trackers, in project backlogs, or in to-do lists but can also be informal and loosely defined in emails, personal notes, or in the minds of people (Maalej, 2009).

Both individual developers and development teams use tasks for organizing their work (Kersten and Murphy, 2006; Blincoe et al., 2013). Developers use task lists to manage their personal productivity. They keep track of what is done, how much is still to do, and what should be done next. Development teams break down the work into manageable tasks, distribute the tasks among the members, and use them to synchronize and coordinate the work (Bruegge and Dutoit, 2009; Blincoe et al., 2013).

Tasks are often interrelated. For instance, a task might follow (i.e. be the *next* on the list), might *block*, or might *depend on* an-

* Corresponding author.
E-mail addresses: maalej@informatik.uni-hamburg.de (W. Maalej), ellmann@informatik.uni-hamburg.de (M. Ellmann), romain.robbes@gmail.com (R. Robbes).

http://dx.doi.org/10.1016/j.jss.2016.11.033

other task. Two tasks might also be *similar*, requiring similar expertise, modifications, or tools. Automatically identifying task relationships would be beneficial to the individual developers and for the teams. For instance, previous studies have shown that task switches are frequent (Mark et al., 2005; Ko et al., 2007; Maalej, 2009) and expensive (Parnin and Gorg, 2006; Iqbal and Horvitz, 2007), as developers get blocked, interrupted, or simply must handle multiple tasks at once. With every task switch, developers have to adjust their mindset to the new task (Mark et al., 2005) and locate relevant artifacts to work on (Kersten and Murphy 2006). Identifying similar tasks to work on next might help developers to manage their personal work and handle task switching more efficiently.

Moreover, predicting related tasks in a project can help developers become aware of other "pieces of work" which might depend on their changes (e.g. who is waiting for their work to be completed) and of other pieces they are dependent on (e.g. to resume postponed tasks when blockers are solved). Task relations are also crucial for collaborative activities such as code reviewing, testing, integration, or handoff. Predicting related tasks would thus help identify with whom developers should synchronize to handle dependencies, avoid conflicts, and share knowledge (Blincoe et al., 2013).

The goal of this research is study whether *relationships* between tasks can be predicted based on the similarity of the corresponding contexts. We focus on two types of relationships useful for work coordination (*dependsOn* and *blocks*) and two types useful for personal work management (*isNextTo* and *isSimilarTo*). The main underlying idea is simple: if "the most relevant artifacts" used in two tasks are similar or are used similarity, the corresponding tasks might be related. To identify relevant artifacts (e.g., a document, a class, a method, or a tool) for a task, we observe developer's interactions during the task and define this as its context.[1] Then, to calculate the relevance of the artifacts to the task, we use a simple heuristic called FDA, which combines the *Frequency* of interactions with the artifacts, the interactions' *Duration*, and their *Age*. To calculate the context similarity, we extend the well-known Jaccard similarity model with the artifact relevance. Instead of only counting the artifacts in the intersection and union of two tasks, we scale these artifacts with their relevance values. This context similarity model constitutes the **first contribution** of the paper.

In a series of simulations with Bugzilla data, a field study, and a user study with students, we empirically investigated the accuracy of the context similarity model to predict various types of task relationships. We compared various configurations including the pure Jaccard index and the popular degree-of-interest (DOI) model (Kersten and Murphy, 2006) for calculating artifact relevance. We also compared context similarity to predict task relationships with text similarity applied on the descriptions of the tasks. This quantitative evaluation of different models in different settings constitutes the **second contribution** of the paper. Finally, inspired by the experiment results and additional qualitative evaluation, we discuss and summarize insights for researchers and tool vendors on using context similarity models to build task recommendation systems. This is the paper's **third contribution**.

The remainder of the paper is structured as follows. Section 2 represents the foundation of the work. It introduces the concept of context, context relevance models FDA and DOI, and the model to calculate contexts similarity. Section 3 reports on the study design to evaluate how well context similarity can predict task relationships, including the research questions, data, and methods. Then, Sections 4–6 report respectively on the results of the field study, the simulations, and the experiment for predicting the following relationships between two tasks: *isSimilarTo, dependsOn, blocks*, and *isNextTo*. We summarize the quantitative and qualitative results and the comparison of the different similarity models. Section 7 discusses the findings and their implication for researchers and practitioners, including the limitations and the steps required to build task recommenders. Finally, Section 8 summarizes related work while Section 9 concludes the paper.

2. Context similarity models

When working on their tasks developers interact with tools and artifacts. They might edit source code in the code editor, debug it in the debugger, run tests in a testing tool, create a model using a modeling tool, or send a clarification request using an email client. Developers "consume and produce" artifacts of different types, including requirements, models, source code, test specifications, or emails. Each artifact contains information that is useful for a particular task. Developers might modify the artifacts. They might also read documents, reference APIs, use tools, or search information. All interactions and the concerned artifacts constitute the context of the tasks.

Definition 1. A task *context* consists of all *interactions* and the concerned *artifacts* a developer performs to work on a particular task (Maalej et al., 2014a).

Maalej et al. (2014a) defines interactions as: "the actions (i.e. the interaction events) taken by a developer, such as the clicks of a specific buttons, the changes to code entities, or views of documentation pages." We adopt this definition. To monitor the context, all interactions with tools and artifacts need to be captured. This is typically done by a background process that collects data from various sensors while each sensor captures specific interaction events of the developer with the system (Maalej and Happel, 2008; Maalej et al., 2014a). Since developers switch back and forth between tasks, a single task might be performed in multiple work sessions.

Not every interaction with a tool or an artifact is similarly *relevant* to the task and characterizes its context. A developer might change a method by mistake, experiment with several search keywords until the right results are identified, or quickly answer a chat message during a bug fix, while the project is building.

Definition 2. Context *relevance* is a model which quantifies the importance of each artifact to the context compared to other artifacts. The more relevant an artifact is, the better it describes the context and the more likely it contributes to accomplish the task (inspired by Hjorland and Sejer Christensen, 2002).

2.1. Context relevance with FDA model

In order to calculate the relevance of an artifact within a context, we propose a *Frequency-Duration-Age (FDA)* Relevance Model. The model is based on a simple assumption from daily life: The more and the longer we interact with "objects" or with people, the more relevant they become to the current context. The older our interactions with these objects are, the less important they become for our context (Maalej, 2010). A similar concepts called "edit wear and read wear" was introduced first by Hill et al. (1992), including the use of time spent on particular lines in a text editor. The idea of wear was applied to software development by DeLine et al. (2005) to facilitate the understanding of programs through wear-based filtering.

The FDA model ranks the relevance of different artifacts used in a task. An artifact is more important for a task the more often it is used by a developer (*frequency*). The longer a developer interacts

[1] In the following, we use the term "context" to denote "task context".

with an artifact, the more important it becomes in a context (*duration*). In contrast, the older interactions with an artifact according to current time are, the *less* important they are for the current task (*age*). Age and duration are both relative to the task. That is, an artifact that is used toward the end of a task has a smaller age and gets a higher relevance score than artifacts touched at the beginning of a task. Similarly, we consider the duration within the task.

For calculating the relevance based on the FDA assumption, we define:

$\mathbb{T}$ the set of all tasks t considered.
$\mathbb{IN}(t)$ the set of observed developer interactions while performing task t.
$\mathbb{AR}(t)$ the set of all artifacts concerned by the interactions in $\mathbb{IN}(t)$.

Additionally, we define the following constants for each task:

D the total duration of all work sessions of a task in seconds ($D > 0$).
F the total number of interactions that occurred in all work sessions.
A the total age of all artifacts in seconds i.e., the total elapsed time since each artifact was most recently concerned.

Let $e_{i,a} \in \mathbb{IN}(t)$ be the interaction event of type i concerning the artifact $a \in \mathbb{AR}(t)$ for a task t. Let d_e be the duration of the interaction event e and s_e be the elapsed time since an interaction event e happened. Let further $\mathbb{IN}(a,t) = \{e_{i,b} \in \mathbb{IN}(t) | a = b\}$ be the set of all interaction events related to a task t that concern an artifact a. For a task t, the relevance of a given artifact a is defined as follows:

$$Rel(a,t) = \sum_{e \in \mathbb{IN}(a,i)} \frac{\mathbf{F}rq(a,i) \cdot \mathbf{D}ur(a,i)}{\mathbf{A}ge(a,i)} \quad (1)$$

where

$Frq(a, t)$ denotes the normalized frequency of all interactions with the artifact a related to task t:

$$Frq(a,t) = \frac{\sum_{e \in \mathbb{IN}(a,t)} \cdot 1}{F} \quad (2)$$

$Dur(a, t)$ denotes the normalized time in seconds of all interactions with the artifact a related to task t:

$$Dur(a,t) = \frac{\sum_{e \in \mathbb{IN}(a,t)} d_e}{D} \quad (3)$$

$Age(a, t)$ denotes the normalized time in seconds since the last interaction with the artifact a:

$$Age(a,t) = \frac{\max_{e \in \mathbb{IN}(a,t)}(s_e)}{A} \quad (4)$$

Rel is proportional to *Frq, Dur* while inversely proportional to *Age*. An artifact relevance *Rel* enables us to interpret, e.g., which artifacts developers have used often and for a long time, and which tool they have used most recently. The calculation of the artifact relevance via the FDA model is done as follows: First, we determine the normalized frequency, duration, and age for the given artifact. Then, we calculate the artifact relevance in all related interactions as defined by $Rel(a, t)$. Calculating the duration of interactions depends on the interaction type. For interactions that can be observed over a period of time such as writing or editing, the duration can be observed. For other interactions such as the selection of an artifact or the execution of a command, we assume that these events last until another interaction is observed, unless an inactivity event is fired. We define a time threshold to fire inactivity events (if no mouse movements or keystrokes were observed). The duration of an interaction is then calculated based on the start time of the concerned interaction event and the next observed event.

The interaction types are determined by the context monitoring system used. Per default, we expect such a system to recognize the following types:

- *Selection*: artifact selections via mouse or keyboard (including selecting and using tools and views).
- *Command*: general operations such as saving, copying and pasting, preference setting, as well as specific development operations such as building a project, running a test, or setting a breakpoint.
- *Edit*: textual and graphical changes such as adding, removing, writing, editing text or a code element.

Depending on the granularity of the monitoring system, other interaction types can be included, such as Search, Read, Refactor, Inactivity, Configure. Maalej et al. (2014a) discuss common interaction types used in recommendation systems and how they can be monitored or processed.

2.2. Context relevance with DOI model

Kersten and Murphy (2006) introduced a model called "degree-of-interest" (DOI) for calculating the relevance of each artifact used in a task. The DOI value for a certain artifact is continuously calculated and updated based on the *frequency* and *decay* of interactions with the artifact. The frequency denotes the number of interaction events referring to an artifact. The decay is proportional to the position in the event stream of the first interaction with the artifact (Kersten and Murphy, 2006). Over time, if the artifact is not selected or edited, its interest value decays. Each selection also leads to decaying the interest values of other artifacts (Kersten and Murphy, 2005). At any point in time, the interest values of the artifacts reflect their relevance ranking to a particular task (Kersten and Murphy, 2006).

By iterating over the sequence of interaction events concerning a given artifact, the **interest** value of the artifact (denoting its relevance) is incremented successively based on the type of the current interaction event. If the interest is not higher than the decay, the decay is reset to start at the last interaction with the artifact. This ensures that artifacts which have decayed to a negative interest get positive interest values when a developer interacts with them again (Kersten and Murphy, 2006).

Kersten and Murphy used the DOI model to filter artifacts irrelevant to the current task (Kersten and Murphy, 2006). The set of relevant artifacts can then be highlighted or reopened when a task is resumed. Mylyn, a tool which implements this approach for the Eclipse IDE is a standard plugin within the Eclipse community (The Eclipse Foundation, 2011; BZ Media LLC., 2008).

Although both relevance models look similar, there are several differences in the technical details.

Frequency. Both FDA and DOI relevance models implement the idea of a frequency defined by the number of interaction events with an artifact. Both models consider a linear, proportional dependency between the frequency and the relevance of an artifact: The more often a developer interacts with an artifact, the more important the artifact becomes.

Duration. In contrast to DOI, the FDA relevance model additionally considers the duration of interaction events. The duration is to measured by calculating the time between observing the event and observing the next event while considering an inactivity threshold. This assumes that an event is active as long as we do not observe a new event. If for a certain period of time (e.g. 5 min) nothing happens (i.e. no mouse move, no keystroke) an inactivity event is

Please cite this article as: W. Maalej et al., Using contexts similarity to predict relationships between tasks, The Journal of Systems and Software (2017), http://dx.doi.org/10.1016/j.jss.2016.11.033

Table 1
Recency represented by age vs. decay.

Aspect	Decay (DOI model)	Age (FDA model)
Representation of time	Since the interaction duration is not considered, interactions happen independently from real time. Instead, "time" is represented by an integer that counts the number of events between two interactions.	Real time is used to determine the age of an interaction. Different durations of interactions are assumed and represented by real time.
Events needed for calculation	Two interaction events (start and end of a decay) are used to calculate the artifact decay. The decay's start and end are adjusted dynamically according to the artifact's interest and the interaction event.	Only the last interaction event is used to calculated the age of an artifact.
Impact on relevance	Linear with subtraction.	Linear with division.
Influence by other artifacts	The time and the frequency of interaction events concerning other artifacts influence an artifact's decay.	Only the time of interaction events concerning other artifacts influence an artifact's age.

fired. The longer an artifact is concerned the more relevant it becomes within a context. For example, if two artifacts are edited with the same frequency but one of them is edited for a shorter period, then this one is less important for the task than the one edited for a longer period.

Recency. A major difference between both models is the representation of an artifact's recency as shown on Table 1. FDA represents an artifact's recency by its age, the time since the last interaction with this artifact happened. The idea is, that if an artifact was edited several days before and another artifact is just being edited, the latter is likely to be more important than the older one. In contrast, Kersten and Murphy (2006) define recency by a decay as mentioned before. The idea is that if an artifact is not selected or edited, its interest value decays over time. Additionally, each interaction also has the effect of decaying the interest values of the other artifacts in the model.

2.3. Context similarity

We define context similarity, introduce common models for calculating the similarity of sets or vectors, and extend these models by including the relevance calculation as described in the previous section.

Definition 3. Context similarity is a model which quantifies how similar two contexts and the corresponding tasks are. Two tasks are similar, if a developer would work on them in parallel or after each other, because the effort needed to switch focus and restore new information and tools is small.

Comparing the similarity of two contexts is equivalent to comparing the similarity of two artifact collections. There are established measures for comparing the similarity of sets and vectors (Pang-Ning et al., 2006), which we discuss in the following.

Perhaps the most well-known measures for calculating set similarity is the Jaccard Index. The **Jaccard Index** J is defined as the size of the intersection of two sets divided by the size of the sets' union:

$$\mathcal{J}(T_1, T_2) = \frac{|\, T_1 \cap T_2 \,|}{|\, T_1 \cup T_2 \,|} \tag{5}$$

When applied to contexts, J is the number of artifacts in the intersection of two tasks divided by the number of artifacts in the union of these tasks.

Jaccard only consider whether an element is present in the set or not (i.e., an artifact is used in the task). It treats all artifacts equally, ignoring their relevance to the task. Unlike in information retrieval, where e.g., all characters in a string have the same relevance for a similarity calculation, in the task similarity comparison, it is crucial to use the relevance of the single artifacts. A context might include a lot of "noise artifacts", which might be identified in the intersection of various tasks but does not increase their similarity. Therefore, we extend Jaccard with the relevance models discussed above. We define the *Relevance-aware Jaccard* $\ddot{J}$ of two tasks A and B as follows:

$$\ddot{\mathcal{J}}(T_1, T_2) = \frac{\sum_{x \in |T_1 \cap T_2|} \mathrm{Rel}(x, T_1) + \mathrm{Rel}(x, T_2)}{\sum_{x \in |T_1|} \mathrm{Rel}(x, T_1) + \sum_{x \in |T_2|} \mathrm{Rel}(x, T_2)} \tag{6}$$

In the numerator (i.e., the intersection set) we consider all agreement artifacts (i.e., artifacts found in both contexts). Instead of just counting the artifacts, we sum up their relevance values within each context. The more an artifact is relevant in the two contexts being compared, the more it increases the contexts similarity. In the denominator, we also sum up the relevance values for all artifacts in both tasks. This way, if two contexts A, and X contain 1000 artifacts each, but only have two highly relevant artifacts in common, and context B and X also contain 1000 artifacts each, but have 100 relevant artifacts in common, the similarity score of A and X will be lower than B and X. The relevance values can be calculated by the two relevance models presented previously, either with FDA or with DOI. We call the resulting similarity models FDA Jaccard and DOI Jaccard respectively.

3. Research design

Our research empirically studies the various models of context similarity (Jaccard, FDA Jaccard, and DOI Jaccard) and explores to which extent these models can predict task relationships in various situations. The main goal is to gain insights into how context similarity can be applied to predict related tasks in specific situations.

We focus on the following specific research questions:

1. *Models effectiveness and accuracy:* Can the context similarity models predict relationships between tasks, and which model is most accurate?
2. *Task relationships:* How well can various types of task relationships be predicted based on context similarity?
3. *Context vs. content similarity:* Which approach can predict task relationships better: context-based or content-based similarity?
4. *Qualitative insights:* Do our assumptions about context similarity match with the way developers work and which additional factors are important for measuring the context similarity?

We focus on studying the following particular relationships between tasks:

- *blocks*: "Task T1 *blocks* task T2" means that T1 stops T2 from being performed, e.g., because T2 requires a deliverable from T1.

Table 2
Overview of research questions and methods.

Study	RQs	Studied task relationships
Field study	1, 2, 3	*isSimilarTo*
Simulation with Bugzilla data	1, 2, 3, 4	*dependsOn, blocks, isNextTo*
Experiment with students	1, 4	*isSimilarTo*

- *dependsOn*: "Task T1 *dependsOn* task T2" means that the work of T1 cannot be done unless T2 is being done.
- *isNextTo*: "Task T2 *isNextTo* T1" means that a developer chose to consequently work on T2 after working on T1.
- *isSimilarTo*: "Task T1 *isSimilarTo* T2" means that the work conducted in both tasks is similar with respect of what should be done and how it should be done. It is important to note the difference between task similarity and context similarity. The task similarity is a subjective measure "manually" defined by developers independently from the observed context elements, while context similarity is calculated based on the similarity of artifacts and interactions observed in a task.

For the model accuracy (RQ1), we aim to compare the context similarity models introduced above not only between each other, but also with a random model and a text-based similarity model as a baseline. Comparing the similarity of the textual descriptions of two tasks might be an alternative way for identifying related tasks. That is, the more overlapped terms are used to describe two tasks, the more related the tasks might be. For this we use Latent Semantic Indexing (LSI) (Deerwester et al., 1990). LSI is a popular statistical model that analyzes the relationships between a set of text documents and the terms they contain by producing a set of concepts (also called topics) related to the documents and terms.

We performed three different studies to answer the research questions: a field study with professional developers, various simulations with data from the Bugzilla repository of the Eclipse community, and an experiment with students including an online survey. Table 2 gives an overview about the research questions and methods.

3.1. Field study with professionals

In the field study, we focused on *isSimilarTo* relationship between tasks. We collected context data of tasks performed in different real development projects. We also collected the subjective assessments of the developers for the similarity of the tasks (as defined in Section 3) and compared the subjective assessments with the values predicted by the models described in Section 2.

We asked each subject to define about 20 tasks that he or she plans to process in the next 1 or 2 weeks. Subjects should work on their tasks as usual, including interruptions, definition of new tasks, issues, etc. The tasks should be defined either in Mylyn or in an issue tracker and imported via a Mylyn connector in Eclipse. Each task should also have a short textual description. Each time subjects start working on a task, they should activate it in the "Task List" view in Eclipse. This enables the tracking of the context data including subjects' interactions with tools and artifacts.

Mylyn is a task management tool that implements the degree-of-interest (DOI) model (Kersten and Murphy, 2006) for the Eclipse development environment. It is the closest mature tool to our work. Using Mylyn instead of an ad-hoc implementation minimizes tool acceptance and usability biases for the evaluation of DOI and FDA. Moreover, comparing FDA to DOI with data collected by DOI's implementation (Mylyn) is the approach that is fairest to the DOI, leading to more reliable results than if other data collection tools were used. Since the analyzed tasks were different in status, we simplified the notion of work session by simply using the available context for the given task as the time of analysis. This might correspond to the whole task if the task is closed or only to part of it if the task is postponed or interrupted.

Subjects were also asked to maintain a *task similarity sheet*, in which they specify for each task the first, second, and third most similar tasks. After each work day, subjects should indicate the tasks similarities in the sheet as they still remember this information. We collected the interaction histories from all subjects together with their similarity assessments, but did not have access to the full source code as these were closed source projects.

Overall we recruited 13 subjects, who worked between September 2011 and August 2012. Nine of them submitted useful and complete data, including 111 isSimilarTo relationships for 64 unique tasks with their contexts. All subjects had at least 2 years of development experience and were familiar with Mylyn.

We also compared the results to a random similarity prediction, and to the text-based similarity model based on LSI ran on the short textual description provided by the subjects. LSI assumes that terms that are close in meaning will occur in similar pieces of text. To remove all English stop words from the task description, we used the Natural Language Toolkit NLTK.[2] We used the python library Gensim[3] to create the corpus of the tasks being studied and calculate the task similarity. The main benefit of this library is that it takes the whole text corpus (i.e., the set of all available task descriptions) into account to generalize the semantics of the text descriptions. LSI requires defining the number of topics that should be extracted. For this we choose 300 because this is often recommended as best-practice for mid-sized documents (Bradford, 2008; Řehůřek and Sojka, 2010).

3.2. Simulations with Bugzilla data

The second study focused on studying *dependsOn, blocks*, and *isNextTo* relationships between tasks. The study consisted of a series of simulations (Walker and Holmes, 2014) with task data collected from the Bugzilla repository of the Eclipse community.[4] In a simulation, a model is evaluated by first collecting reference data (e.g., perfect data, often called a golden standard), which includes the input data and output data of the model. The model is then run with the perfect input data and its output is compared against the perfect output.

Identifying an open repository with all the needed data, including the tasks, their interaction data (contexts), and information about the tasks similarity turned out to be a challenging task. The Eclipse Bugzilla repository partly includes this data. This repository is used to track bugs and other development tasks for the Eclipse products. Some product teams consistently use Mylyn and attach the Mylyn Context files (including the interaction data) to the task descriptions to allow other developers to reproduce the context. Moreover, Bugzilla provides a field to link two kinds of related tasks: *dependsOn* and *blocks*. Many teams tend to enter and maintain this information for their tasks in Bugzilla.

We downloaded and crawled the task and context data from Bugzilla using the Python library BeautifulSoup.[5] We first downloaded all tasks that have Mylyn Context data attached. This resulted in 6650 unique tasks. In this list, there were only 2605 tasks which had at least one related task (dependsOn or Blocks). We then filtered each related task pair, in order to keep only the pairs in which both tasks had Mylyn Context data attached. This step led to 679 eligible tasks which have 928 related tasks (since a task can be related to more than one task). The related tasks included 430

[2] http://www.nltk.org/book/ch02.html.
[3] http://radimrehurek.com/gensim/tut3.html.
[4] https://bugs.eclipse.org/bugs/.
[5] https://pypi.python.org/pypi/beautifulsoup4.

unique *dependsOn* tasks and 292 unique *blocks* tasks. The sample included the task ids, the task descriptions, the task metadata (status, comments, creation time, etc.), the ids of the related tasks, and the context data describing the interaction of the developers when working on the task. Most of the eligible tasks (59%) included only one related task, 9% included two, and 3% three related tasks or more.

In addition, we also reconstructed and studied *isNextTo* relationships between the tasks. For each task in the Eclipse Bugzilla dataset that had a context attached, we identified the developer that attached the context. Since the Mylyn data is time stamped, we could infer when developers start working on a task and when they stop working on it: these are the time stamps of the first and the last event in the Mylyn trace. Based on this information, we could order the tasks of a developer in time and detect when the same developer stopped working on a task and started working on the next task. If this duration is less than 24 h, we assume that the developers switched to a subsequently ordered task consciously. Overall 105 developers in our data-set have worked on 4076 different tasks that fulfilled this criteria. As for *blocks* and *dependsOn*, we also evaluated how well the various models can predict the *isNextTo* relationship and compared the context-based prediction to the prediction based on the textual description of the tasks as in the field study. In this study we used the task titles as input for LSI.

Finally, we manually analyzed and labeled a subset of these tasks to check how semantically related the task switches actually are. Since *isNextTo* is a hypothetical relationship that is not explicitly defined by developers and based on our assumption that consecutive tasks should be related, it is worth checking if the relationships are also meaningful and if the prediction models work better for the *isNextTo* tasks which are semantically related.

3.3. Experiment with students

The third study focused on *isSimilarTo* relationship between tasks. We conducted an experiment with 152 software engineering students at the Technische Universität München (TUM) in end of 2011. The study included two phases. First, we defined real development tasks and let the students work on them. Second, we asked the students to assess the similarity of these tasks and explain their decisions. We then compared the similarity assessments of the students with the similarity which we foresee for the tasks based on our models for relevance and similarity.

The students were randomly grouped into 37 teams of 4–5 members and randomly assigned to one of two projects. The first project, IM, was an instant messenger for students. The second, POLL, was a poll tool to vote for course-related questions and evaluate the course sessions.[6] Students had to implement the systems as client-server applications. For IM we defined four mandatory tasks and for POLL three. We did not give any constraints on how and when to implement the tasks, but we had assumptions on their similarity.

After finishing the tasks, we asked the students to rank the similarity of the predefined tasks and explain their rationales using an online questionnaire. We then compared the questionnaire results with our assumptions.

4. Results of the field study

We report on the results of our field study, in which we compared how well different context similarity models can predict the similarity between tasks as seen by developers (*isSimilarTo* relationship). Overall, the nine subjects of the field study worked on a total of 64 unique tasks and identified 111 isSimilarTo relationships between those. We first applied the similarity models on the tasks of each subject separately (within-subjects analysis) and searched for the similar tasks over all collected data (between-subjects analysis). We report on the hit ratio, normalized distance performance measure, and mean absolute error, often used to evaluate prediction accuracy (Shani and Gunawardana, 2011).

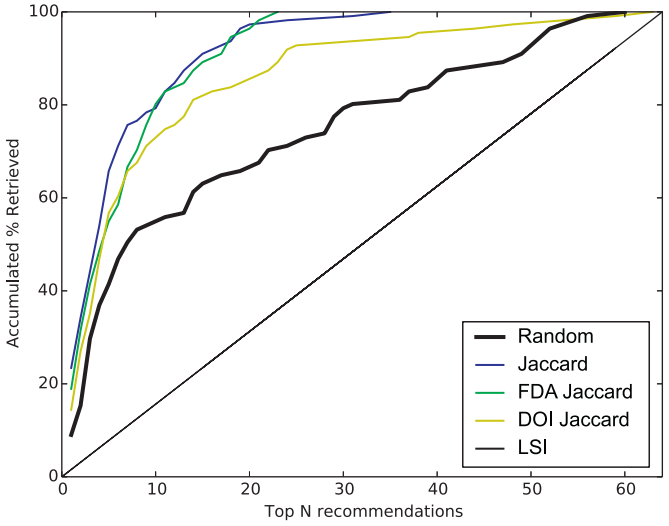

Fig. 1. Hit ratios of the context similarity models for predicting similar tasks (*isSimilarTo* in field study between subjects analysis. Search space with N=63 tasks.

4.1. Hit ratios

Hit ratios are useful for evaluating recommendation models in particular if the recommendation items are ranked in a given order (Dumitru et al., 2011). A hit ratio computes the probability that a given item is predicted as part of the top N recommendations produced by the model. That is, for the items that were recommended, it shows how many were ranked first in the recommendation list, how many were ranked as the second, third, and so on. Typically, hit ratio values are plotted against different values of N. The hit ratio curves plot the accumulated percentage of correctly retrieved results against the number of recommendations made (Dumitru et al., 2011; Hariri et al., 2014). The straight diagonal indicates what one can expect of a random performance, i.e., a random algorithm would in the long run tend to recommend the correct task roughly in the middle of the list.

Fig. 1 shows the prediction ability of the models if we search for similar tasks **in the entire dataset**. This means that all the 63 tasks are included in the hit ratio computation, regardless of the subject. The Normal Jaccard and FDA algorithms exhibit the best performance, managing to recommend all the similar tasks by recommending at most 22 tasks in the worst case.

Next, we calculate for each task in the sample similar tasks based on the similarity of the descriptions. Overall, developers worked on 64 unique tasks that have a summary, for example “Find and eliminate Bugs”. 61 tasks have a unique text description. Fig. 1 shows that the text-based similarity is not as accurate as the context based similarity calculation based on the Jaccard distances. One possible reason is that most task descriptions in the sample were short: the shortest task description included 12 characters, and the longest, 89 characters. This finding is consistent with previous studies which showed that developers often encounter dif-

[6] For the description of the projects and tasks see http://www.teamweaver.org/wiki/index.php?title=Next.

Table 3
Accuracy of ordered similarity prediction for the field study (*isSimilarTo*).

	Within subjects N=9			Between subjects N=63			
Eval. metric	DOI Jacc.	FDA Jacc.	Jacc.	DOI Jacc.	FDA Jacc.	Jacc.	LSI
MAE in positions	2.17	2.23	2.04	8.23	5.24	4.68	15.42
NDPM	0.44	0.60	0.55	0.44	0.60	0.55	0.43
NDPMs with 0.0 in %	33.33	25.64	41.03	33.33	25.64	41.03	43.59

ficulties to write a meaningful description for their tasks (Maalej, 2009; Maalej and Happel, 2010). Many of the tasks (about half of developers' work) are informal and do not have a meaningful text description (Maalej, 2009).

4.2. Rank-aware prediction

Since the field study subjects ordered the similar tasks (most similar, 2nd. most similar, 3rd. most similar), we also calculated additional measures which are often used to evaluate the prediction accuracy with ranked items: Normalized Distance Performance Measure (NDPM), and Mean Absolute Error (MAE) in rankings. Table 3 shows the metrics and the results when searching for similar tasks in the data of the individual developers (within subject analysis) and when searching for similar tasks in the data of all subjects (between subjects analysis).

Mean Absolute Error. The Means Absolute Error (MAE) shows how close are the system's recommendations to our reference recommendations. It summarizes the differences in ranks, taking into account all the tasks in the list. In the ideal case, the most similar task should occupy rank 1, the second most rank 2, and the third most rank 3. These ranks are compared with the actual ranks produced by the recommenders.

The results in Table 3 show that the models missed the valid position of the similar tasks less with the Jaccard distances. If we take the case of the between subject analysis, on average the Jaccard distances miss by 5 to 8 ranks, while the text similarity misses by 15. These results agree with the hit ratio shown on Fig. 1.

Normalized Distance Performance Measure. The Normalized Distance Performance Measure (NDPM) assesses the ordering in which the items are shown (Luostarinen and Kohonen, 2013; Yao, 1995). It only considers the items for which the ordering is given, i.e., the ordering of the 3 most similar tasks. The marginal values of the NDPM are as follows:

- 0.0 when the ratings are ordered perfectly (best ordering);
- 0.5 when the ordering is random;
- and 1.0 when the ratings are ordered in reverse (worst ordering).

These values are averaged for all the tasks and presented in Table 3. Note that we only calculated the NDPM of the 39 tasks (out of 55) that have at least two similar tasks. We observe that the performance ranges from 0.44 to 0.60, meaning that there is a degree of similarity with the original ordering, but not a very high one. Interestingly the context similarity models predict the tasks independently of the number of tasks provided in the same order of tasks. Note that to ease the interpretation of NPDM metrics, we also rescale them in a percentage scale, with 0% being the perfect case (0). The Jaccard and the LSI matches in 40% of the cases the expected order.

4.3. Threats to validity of the field study

The results of our field study should be interpreted within the context of the study. The analysis to evaluate the performance of the context models is based on a relatively small sample (64 unique tasks and 111 relationships). Obviously, a larger sample would help to better generalize the results and gain a deeper understanding for which model works best when. We recruited 13 professional developers working in different domains and different companies. To get a realistic dataset, we purposefully refrained from using the contexts of students' tasks e.g. in an experiment. Nine of the 13 developers submitted useful data. Two of the original 13 cancelled the study due to privacy reasons (since much information about the source code must be shared), while two others did not use Mylyn all the time which led to partial, unreliable, and thus unusable results. The remaining nine subjects submitted data describing a "long-enough" period of development ($\sim$ 2 weeks) from real projects. While all developers worked in Germany, they have different cultural backgrounds (Eastern and Western Europe, Asia, and South America). Overall, we feel that the results – while certainly not representative to all developers – give initial indications about the prediction ability of similar tasks for developers working with Eclipse.

Concerning the internal validity, we assumed that the subjects submitted correct data, in particular the task similarity relationships and the sessionization of the interaction data when starting to work on a different task. Originally, we also asked subjects to submit an assessment of the artifact relevance. However, after several dry runs and during the real field study we observed that this information was not intuitive and difficult to assess, and thus complicated the study. Therefore, we decided to focus on the similarity evaluation.

Since the overall submitted task list per subject included 10–15 tasks, we assumed that the task similarity assessments were correct. The task lists were rather small and the similarities can be well assessed manually. Moreover, all subjects performed this exercise without major clarification requests.

Subjects might have indicated a similar task on which they worked on a different day. This might lead to the potential risk that they do not remember that task with the same level of details as tasks from the current day. We think that this threat is negligible. None of the developers had more than 15 tasks for the whole period of the study. Moreover, none of the developer reported on work that lasted for longer than 2 weeks. We think that this size and dependencies of the tasks can be handled well. We also think that all participating subjects with useful data were fairly well committed and assessed the similarity of the tasks to the best of their knowledge.

Finally, we used Mylyn for data collection and part of the data analysis. While Mylyn is a mature tool and the probability of mistakes in the data is rather small, this tool was certainly less mature at the time of conducting the study in 2011. Moreover, using Mylyn means that we had to restrict ourselves to the types and granularity of data about interactions and artifacts. In dry runs and self experiments, we used other monitoring tools and observed more accurate results. However, we think that using Mylyn in the field study is the right choice to minimize usability bias of our less mature implementation and to allow for a fair comparison between the DOI and FDA relevance models.

Please cite this article as: W. Maalej et al., Using contexts similarity to predict relationships between tasks, The Journal of Systems and Software (2017), http://dx.doi.org/10.1016/j.jss.2016.11.033

Table 4
Average precisions P and recalls R of the context similarity models to predict *dependsOn* and *blocks* relationships Bugzilla. The k in P_k and R_k indicates that the model is evaluated using the k first predicted tasks.

	Jaccard				FDA Jaccard				DOI Jaccard			
Relationship	P_4	P_{10}	R_4	R_{10}	P_4	P_{10}	R_4	R_{10}	P_4	P_{10}	R_4	R_{10}
dependsOn & *blocks*	13.2	7.0	41.4	53.8	9.0	5.3	28.1	40.1	3.6	3.8	11.8	30.1
dependsOn	15.2	7.9	43.5	55.5	10.6	6.0	30.5	40.8	3.7	4.0	11.5	28.7
blocks	10.6	5.7	39.8	53.3	7.0	4.3	26.6	39.9	3.2	3.2	11.7	30.3

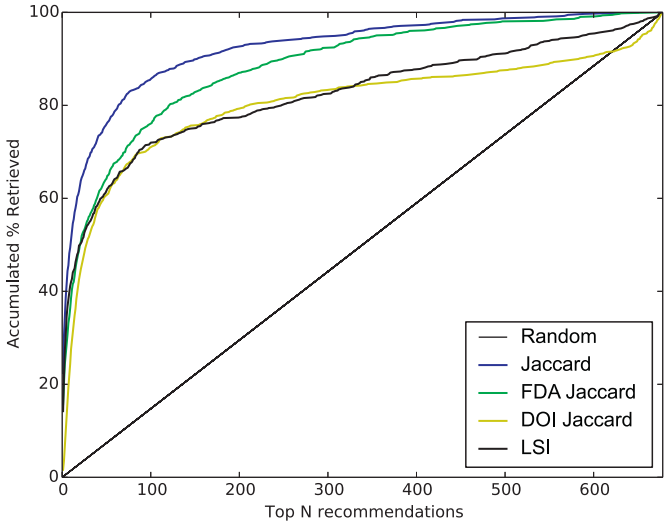

Fig. 2. Hit ratios of the context similarity models for predicting *dependsOn/blocks* task relationships (Bugzilla data, search space with N = 678 tasks).

5. Results of the simulation with Bugzilla data

We report on the results of the simulation series conducted based on data collected from the Bugzilla repository of the Eclipse community. We compared the context similarity models to first predict *dependsOn* and *block* relationships. Then, we constructed *isNextTo* relationships from the timestamps and studied those. Finally, we manually assessed 500 randomly selected *isNextTo* relationships to check the corresponding tasks are actually related.

5.1. Predicting dependsOn and blocks relationships

Table 4 shows the k-precision and k-recall values of the similarity models to predict the three types of task relationships studied with the Bugzilla data. We report on $k = 4$ and $k = 10$ as these are common in practice. For instance, Google search autocomplete shows 4 predicted items in the browser search field and 10 items in the webpage. Other recommender systems as Amazon also typically shows 4 to 10 recommendation items to the user. Overall, the table shows that Jaccard has the highest precision and recall, followed by FDA Jaccard, followed by DOI Jaccard. The precision values are rather small but still greater than random (which was around 1%). The recall values are more encouraging and go up to 55% for Jaccard to predict *dependsOn* tasks.

The results of the **hit ratios** for all *dependsOn* and *blocks* task pairs in Bugzilla that have context data are summarized in Fig. 2. Overall, there is a slight difference between the performances of the similarity models. The pure Jaccard seems to perform best, followed by the FDA Jaccard, followed by DOI Jaccard, and then by LSI. These results are similar to our field study. The Jaccard model performed best and was able to predict >60% of the related tasks up to the 20th position. We think that this is an encouraging result since the search space is much bigger in this study.

Recommending to a developer out of hundreds 20 tasks which others are working on and which he should be aware of might be acceptable, even if more accurate predictions and shorter lists is clearly desired in practice. This prediction accuracy can be reached if developers are using context data or if the summary and product component name are available in the task description. Systems like Google and Amazon render about 10 results to the users per defaults. Stack Overflow provides up to 15 potential questions to be answered by other developers.

The pure Jaccard performed better than the context aware Jaccard. In particular, Jaccard identifies the correct similar tasks in an earlier recommendation position than the DOI Jaccard. In the Bugzilla dataset, the relevance of the artifacts seems to rather have a secondary importance for the similarity. This sounds reasonable, in particular for "blocks" tasks. Two tasks might block each other even if they share artifacts which are not relevant for both tasks (e.g., one single API call in common).

We found that the text-based similarity model performed as good as the DOI Jaccard but less accurate than the pure Jaccard and the FDA Jaccard models as depicted on Fig. 2. This shows that it is possible to identify related tasks by comparing the similarity of their textual descriptions (i.e. the summary fields of the bug report).

We found that adding the metadata about the product and component names (affected by the task) to the task descriptions significantly increases the prediction accuracy of the text-based similarity model (LSI). By using only the summary as an input, the text-based similarity model discovered 65% of the tasks until the first 100th position. By adding the metadata (product and component names) the model discovered 85% of the tasks and performed overall as good as the pure Jaccard.

We also ran a differentiated analysis for the *dependsOn* vs. *blocks* relationships. We found that the hit ratios for both relationships are almost identical with the hit ratio of the FDA Jaccard slightly better for *dependsOn* than for *blocks* (a slightly sharper until position 100).

5.2. Predicting isNextTo relationships

Table 5 shows the k-precision and k-recall values of the similarity models to predict isNextTo relationships studied with the Bugzilla data. Again, the table shows that Jaccard has the highest precision and recall, followed by FDA Jaccard, followed by DOI Jaccard. The values are smaller than *blocks* and *dependsOn* but still greater than random.

We also calculated the hit ratios for the different similarity models to predict *isNextTo* relationships, as shown on Fig. 3. We choose only the developers who switched at least 3 times within a day to another task. This resulted in 23 developers who worked on a total of 1201 tasks (average: 52.26, median: 16.0, std: 88.45). The developers switched 848 times to another task within a day (average: 36.87, median: 11.0, std: 67.062). For one single developer, there was at most 439 predicted related tasks, which correspond to the maximum of the hit ratio graph on Fig. 3.

Again, all models clearly outperformed random and the pure Jaccard performed best. However, for this type of relationship, there was no significant difference between FDA Jaccard and DOI

Table 5
Average precisions (P) and recalls (R) of the context similarity models to predict *isNextTo* tasks in the Bugzilla study. The k in P_k and R_k indicates that the model is evaluated using the k first predicted tasks.

Jaccard				FDA Jaccard				DOI Jaccard			
P_4	P_{10}	R_4	R_{10}	P_4	P_{10}	R_4	R_{10}	P_4	P_{10}	R_4	R_{10}
4.4	2.7	17.1	25.9	3.2	2.2	12.5	21.0	3.1	2.1	12.2	20.8

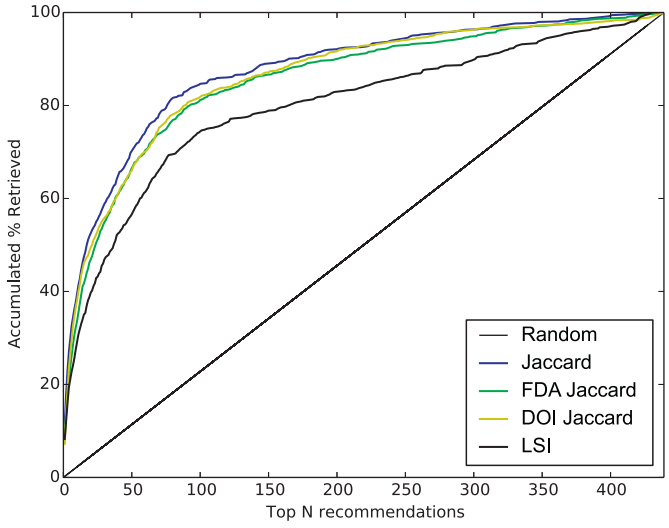

Fig. 3. Hit ratios of the context similarity models for predicting *isNextTo* tasks in the Bugzilla study.

Jaccard. Moreover, overall, the increase of all hit ratios is smaller than for dependsOn and blocks. It seems that, on average, the context similarity can better predict work dependencies than time dependencies of tasks.

Severity and task switches. We were nevertheless curious as to why the developers do not switch to similar tasks, and formulated the hypothesis that there may be more urgent tasks to do. In order to look into this, we investigated whether developers tended to switch to more urgent tasks in same-day tasks, compared to tasks on other days.

Normally tasks in Eclipse are prioritized by the Eclipse community to clarify how and when a task or a feature request should get fixed. The severity clarification of a task (e.g. "critical") are used to define a task as important and inform the responsible development team working on a software product and component. We have redefined the seven text-based severity levels[7] to a scale of 1–7. A severity level of 1 means a task blocks some component or the complete software system and should get fixed immediately and should have a high priority in the community.

We suppose that developers switch from one to another task within a day when the following, next task is from a high interest or priority and should get closed as soon as possible, even if it is not similar to the previous task. To assess this, we divide task switches in a subset of our dataset, concerning 57 developers, according to duration of the task switches: less than a day (941 switches), and more than a working day (3524 switches).

We find that the average severity rating of the tasks where work started after a short task switch was lower (4.62), than the ones when the task switch was longer (4.80). This means that the short task switches involve in general tasks that are more severe, yielding a degree of support to our hypothesis that developers may switch to important tasks even if they may be dissimilar. A Welch's *t*-test (p-value:8.95e-5) shows a significant difference between the severities of the two types of tasks. However, the difference in severities is relatively small, which tells us that there are a variety of other factors at play.

Similarity in products and components We found evidence that developers try to focus on similar tasks, at a coarser level of granularity. In the dataset of 941 task switches that happened on the same day, we find that 57 developers work on 46 different software products and 105 different software components. These 57 software developers, when switching to tasks at longer time intervals (3524 switches), work on 86 different software products and 204 software components.

We find that for short task switches (a day or less), developers normally keep their focus on the same software products (82.6% of the cases), while for longer task switches, they focus more frequently on different software products (the product stays the same in 78% of the cases). This difference is significant according to a Welch's *t*-test (p-value: 0.0125).

The same analysis at the level of software components finds that during short task switches, developers keep working in 67% of the cases on the same software component. For longer task switches, the proportion drops to only 48% of the cases. The difference is also significant (Welch's *t*-test, p-value: 3.37e-30).

Conclusions. We find that in a subset of our Eclipse Bugzilla dataset, developers are overall working on dissimilar tasks, even on the same day. These larger context switches may reduce their productivity. A possible reason for this is that they may have to switch to important defects to fix, although the observed effect is small; other factors may be at play. Despite this, we find that developers are more likely to try to stay on the same product and component during 24 h. Thus at a high level they try to minimize the number of their context switches. These results show that there is potential for a recommender system that recommends similar tasks in a fine-grained fashion.

5.3. Manual analysis of isNextTo task pairs

We conducted a detailed manual analysis on a subset of the dataset, focusing in particular on developer task switches, i.e., on how developers transition from working on a task to the next one, based on their activity as recorded in Bugzilla. The intent of this analysis was threefold:

- Assess how often developers switch to tasks that are related to each other in our dataset. This is important to check our assumptions that developers might be interested to work on related tasks consecutively.
- Find out from the documented activities how developers use context in practice, i.e., why they attach Mylyn context data to their tasks.
- To build an initial dataset of tasks that are manually found to be related, and to evaluate the performance of our various models on it.

The process we employed to build this new dataset was to first select the task switches in the PDE dataset, restricting our selection to the task switches containing pairs of tasks that both had Mylyn context files attached. We chose the PDE dataset as it is one of

[7] https://www.mediawiki.org/wiki/Bugzilla/Fields.

the largest, but is not so large as becoming unmanageable. In total, this dataset contains 497 task switches, which is enough to form a good understanding of the data and the entities featured in it, without being too large.

Once the data was selected, we inspected it. The previous step produced a list of pairs of tasks with their URLs on Bugzilla, so that we could easily inspect them. We reviewed each pair of tasks with two goals: the first was to determine to which degree the tasks described in the bug reports were related, while the second was to identify how Mylyn contexts were used, highlighting any specific comment that could help with that overall goal. For the first use case, the entire task description was considered, while for the second one, only the comments related to Mylyn were considered.

We determined that two tasks were related based on the language use in the task description or the comments. We tried to answer the question whether the tasks appear to concern similar parts of the software, either explicitly (are the same entities mentioned?) or implicitly (are the same concepts mentioned?). As part of that we also considered the stack traces that were occasionally included in the comments. We used three degrees of relationships between the tasks: if we found strong evidence that the tasks concern similar parts of the software, we would rate the tasks as "related". If the evidence was weaker, we would rate them as "maybe related". In other cases, we would rate the tasks as "not related".

All in all, we rated 342 task switches as "not related" (i.e. 68.8%), 101 as "maybe related" (i.e. 20.3%) , and 54 as "related" (i.e. 10.9%). This means that we found around one third of the task switches related or maybe related, while two thirds are not related. Thus, it seems more common in the aggregate that developers switch to unrelated tasks, which is aligned with the findings of the previous section. Going into the details, however, we found differences between developers. Some developers are much more focused than others, usually working on a single component, and spend the majority of their times on switches to related or maybe related tasks. On the other hand, other developers switch to unrelated tasks more often, and skew the average since these developers have the most task switches. Further, a closer inspection of those developers shows that they often have the role of integrator of functionality (external contributions), rather than developing it themselves.

As far as the usages of the Mylyn context goes, we find several usages, of which two are predominant:

- Use Mylyn task contexts to support code reviewing. In the eclipse projects, code is systematically reviewed by another project member or a project integrator before being accepted. In this scenario, the context file can be opened by the reviewer in order to focus the reviewing activity on the entities the developer worked on, while still being in the IDE. This is an advantage compared to the conventional process of reviewing a contribution as a patch, where only the changes are highlighted.
- Use Mylyn as a handoff mechanism in bug triage. When an expert is notified of a bug, the expert may elect to delegate the bug to another project member. In order to transfer some of their expertise, the expert may create a context model with the code entities that they believe need to be changed or understood in order to perform the task. After loading the context file in their IDEs, the developers will find the IDE focused on these entities, which should ease their program comprehension efforts, as they do not have so many entities to consider, which can be a big issue in a large system.

Finally, we observe large variations in performance in the different subsets of tasks and their degree of relationships. For each task switch in the dataset, we run our context similarity models with the first task as input and measure the rank of the second task in the result list. We repeat this procedure for each of the model: i.e. Jaccard, FDA Jaccard, DOI Jaccard, as well as for LSI applied on the tasks' descriptions. The results are shown on Fig. 4.

We found that, in all cases, the average rank of the predicted task in the list of results is the lowest (i.e. most precise) in the subset of tasks that are rated as related (the average rank for "related" tasks varies between 52 for LSI, and 114 for DOI), with intermediate performance in the subset of "maybe related" tasks (the average rank for "maybe related" tasks varying between 119 for Jaccard and 162 for DOI). The lowest performance is in the group of "unrelated" tasks, with average ranks varying from 197 for Jaccard, to 216 for DOI Jaccard. This shows that – as expected – the context similarity approaches perform better on more related tasks, with varying degrees of performance. We also run paired Wilcoxon rank sum tests between the various average ranks and found that all differences between "related", "may be related" and not related are statistically significant with $p \leq 0.01$.

The DOI Jaccard is the model with the highest ranks in average, and thus the lowest performance (114–162–216). The three other variants are closer together, with the FDA Jaccard performing slightly worse than the other two (78–132–212), with no clear winner on the top two: the LSI model (52–131–282) performs better for the "related" tasks, while the Jaccard model performs better for the "maybe related" category (69–119–197).

5.4. Threats to validity of the simulation with Bugzilla data

The external validity of the Bugzilla study is rather high, since the study data include many tasks of many developers from different Eclipse products (i.e., teams). Therefore, we think that the results generalize well, at least to the Eclipse community. One limitation, however, is that the data represent only developers who use Mylyn and share their context data in Bugzilla and only Eclipse teams that maintain the *dependsOn* and *blocks* relationships in the issue trackers. Other developers might have different work habits and patterns. As we identified in the manual labeling there are recurrent work patterns in the tasks with context data attached as developers often share their context data with specific goals in mind.

While the heterogeneity of the studied projects, task types, and developers increase the external validity of the study, these different settings (e.g. size of contexts, type of work done, different developers role, products etc.) might have influenced the prediction results acting as "hidden variables". To mitigate this threat, we checked several obvious variables for correlations. We did not find any correlation between the size, the project, the severity, the developers, and the performance of the models. The tasks studied were heterogeneous lasting from a few minutes to days.

We cannot completely exclude the possibility of script and analysis code errors, since this was a multi-year lasting project and includes multiple levels of analysis including multiple models, data, studies, and relationships. We took special care and conducted code reviewing for the crucial computations.

When exploring the *isNextTo* relationships, our analysis of severity mapped severity levels to an ordinal scale (from 1 to 7). While we cannot assure that the difference between the different levels, e.g. between "blocker" and "critical" is the same as between "trivial" and "enhancement", there is a clear semantic order of the levels. Using a nominal scale (which does not assume that the distance between items is similar), may slightly change the results. We were not interested in quantifying the severity level of the next tasks, but rather checking if, on average, developers tend to switch to more sever tasks if the switching time is shorter.

Finally, the manual labeling to assess how related *isNextTo* tasks are might include mistakes or limitation to the representativeness. We had to create a sample that is large enough but also manage-

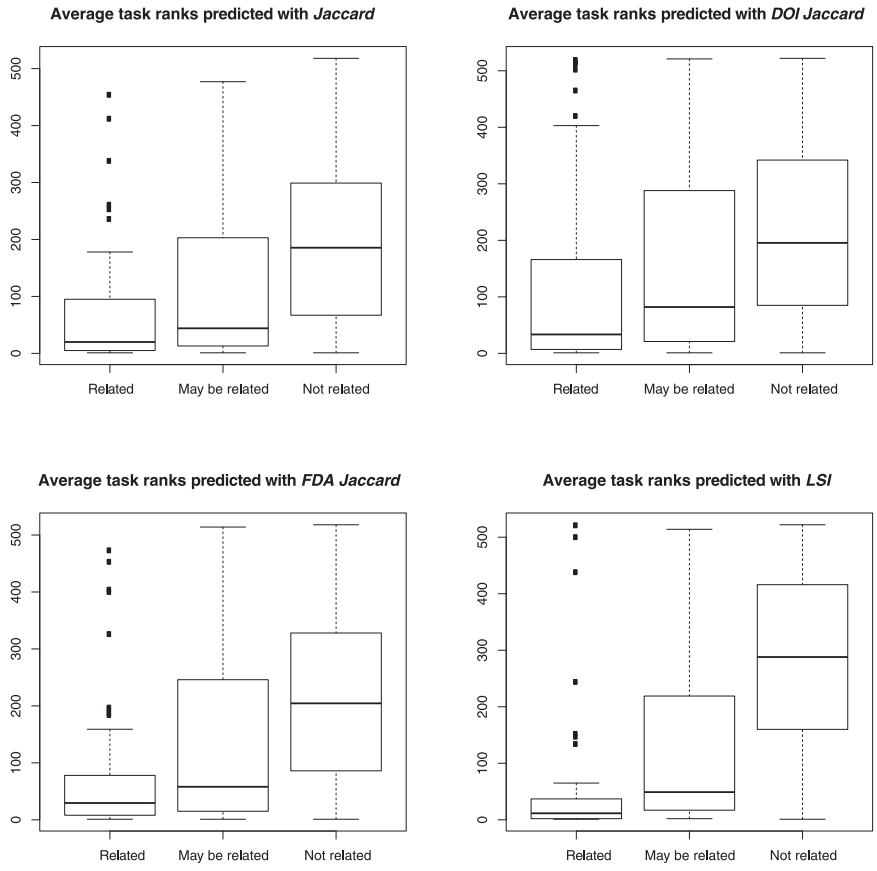

Fig. 4. Boxplots showing the average ranks of the predicted task for the three labeled categories: related, may be related, and not related (yes, may be, and no). N=497 assessed *isNextTo* relationships of the Eclipse PDE project.

able since the labeling task is time intensive. It requires consulting each task in the pair, reading the descriptions, comments, and if necessary looking at additional data such as the changeset or additional documentation in the internet. For this reason, we also decided to limit the labeling to one single project. This enabled the labeler to acquire knowledge about the project and use it for all pair labeling. Each task pair was labeled independently twice by two authors (the second and third). There was a disagreement only in a few cases (4 pairs) which does not influence the results. A broader representative sample would certainly lead to more reliable results. However, the consistent results across the different prediction models, as well as the good performance of the text similarity increases our confidence about the results.

6. Experiment results with students

We analyzed the results of the task similarity assessment questionnaire we handed the students after they worked on the tasks we pre-defined, and compared their similarity assessment with ours. In the experiment we checked the following hypothesis: Hypothesis H_0: Our task similarity definition based on context similarity is intuitive for developers. Hypothesis H_1: Developers understand task similarity in a different way. Overall, 127 students (84%) answered the questionnaire. A full matching of similarity means that a student ranks all tasks exactly in the expected order. More than half of the ranks exactly match with the expected positions. A Chi-Squared test rejected H_1 with 95% confidence.

6.1. Qualitative analysis

Each time students ranked the tasks similarity, they were asked to explain the reason for their decision (open question in the questionnaire). We manually analyzed the justifications, in an iterative, independent, peer-conducted process, focusing on additional similarity drivers. 92 of the 127 submitted questionnaires included justifications. The analysis resulted in interesting observations summarized in Table 6.

Table 6
Summary of qualitative analysis (N=92).

Observation	Count
Functionality and architecture indicate task similarity	55
Different technologies let tasks appear dissimilar	32
Participants think about tools instead of artifacts	21
Participants think about artifact types instead of artifact instances	18

Functionality and architecture indicate task similarity. 55 participants considered the similarity of the functionality instead of the artifacts as indicator for tasks similarity. Even though we defined two tasks as similar, if they use the same tools and artifacts, the functionality, i.e. the functional requirement that needs to be implemented, played the most important role in the justifications. One student stated: “the functionality of sending a message is pretty much the same like login in or log out. The user triggers an event and you tell the server he sent a new message. Receiving messages is a bit different... ”. One other student wrote: “task 3 is most similar because most code has to run on the client side... ”. Students often compared tasks on a conceptual level, relating the design, architecture, or workflow of the desired solution.

Different technologies let tasks appear dissimilar. 32 participants claimed that if the technologies (e.g., framework or programming language) involved in the tasks are different, both tasks were considered as completely dissimilar. In this case tools and ar-

tifacts involved in the tasks seem to recede into the background. One student stated: "working on [internal behavior] is totally different than working on gui stuff". One other wrote: "tasks 1 and 2 comprise editing java code [...] using a database backend on the server. Task 3, by contrast, mainly involves modifying cascading style sheets which are exclusively relevant for the client."

Participants think about tools instead of artifacts. We counted 21 participants who compared tasks based on the tools that were used, but rarely referred to concrete artifacts created, modified, or managed with those tools. Instead of explaining differences between tasks based on different artifacts (e.g., index.html, or Message.java), they explained the differences by the tools they used (HTML editor vs. Java editor). They also tend to use specialized tools for specific types of artifacts rather than trying to solve the tasks with the IDE. One student justified his decision: "while implementing task 1 and task 2 [we used] the Source Code Editor and for the beautifying task [we used] another editor." Another wrote: "my most important tool [for this task] was Adobe Fireworks. It is dissimilar by definition".

Participants think about artifact types instead of artifact instances. Instead of explaining differences between tasks based on concrete artifacts (e.g., index.html, main.css, or Message.java), 18 participants rather see the differences based on the artifact types (HTML, CSS, or Java documents). Concrete file names seem to play a subordinate role. One student wrote: "for implementing functionalities we mostly used source code editor and java files, for beautifying an application we need to change the css file".

6.2. Threats to validity of the experiment with students

As for the experiment with the students, we exclude nonrespondent bias (i.e., due to missing data sets) as the majority of students have answered the questionnaire. One threat might be the lack of experience with task management in software projects. As the study was designed to be more close to real situations than a lab setting, students might have also worked on different tasks than the predefined ones, and developed their position about task similarity. To minimize interventions biases, nothing was enforced during the experiment. Only the submission of a working system led to a successful completion of the projects.

The number of tasks assessed by students was rather small. In everyday's work answering the question on "what's next" might include the similarity assessment of a bigger number of tasks. However, a larger number of tasks would have complicated the study leading to difficult and probably imprecise assessments by the students. For the field study, the assessments concerned a larger number of tasks up to 10 tasks. Predicting the similarity of a larger number of tasks might lead to a different performance. However, we think that this does not bias the results due to the high number of identical rankings predicted by our model.

Our study only compares development tasks excluding other tasks as design, requirements, or managements. Therefore, even if we think that our assumption and heuristic for task similarities apply for different types of tasks, the evaluation results should be interpreted within implementation settings and not generalized to the full spectrum of developer's work. Finally, asking experienced developers instead of students might lead to additional qualitative insights.

We note that the fact that our results are going in similar directions in the three datasets (from the field study, the simulation, and the experiment) gives us increased confidence in our results.

7. Discussion of implications

We revisit the research questions of this work and discuss the implications and limitations of our findings.

7.1. Effectiveness of context similarity for predicting task relationships

The first and perhaps the most import finding from our studies is that the various context similarity models significantly outperformed a random prediction for all datasets and task relationships discussed in the paper. This finding clearly answers the first part of the first research question and confirms our main assumption: that the contexts similarity reveals the existence of a relationship between the corresponding tasks. This context-based approach seems to work at least for two types of relationships useful for work coordination (*dependsOn* and *blocks*) and two types useful for personal work management (*isNextTo* and *isSimilarTo*). Additional types of relationships (e.g. subtask or incorporates) (Thompson et al., 2016) should be studied separately. We think that our assumption holds in these relationships as well. Also, the context similarity prediction worked similarity well or better than predicting task relationships by comparing the textual descriptions of the tasks.

In our first and second research questions, we also aim at comparing the context similarity models discussed in Section 2: in particular, which one predicts more accurately certain types of task relationships. The results of the Bugzilla study and the field study suggest that the simplest model Jaccard outperform the DOI Jaccard and FDA Jaccard. However, this should be interpreted with care, since the differences between the various models in particular within the first 20–50 predicted tasks were rather small. Therefore, we think that there is no clear winner. This result might still question the need for extending well known similarity metrics such as Jaccard by context relevance.

Overall FDA Jaccard generally outperformed DOI Jaccard. The difference between Jaccard and FDA Jaccard was smaller in the field study. This is not surprising since the Bugzilla study included more task data points.

We also checked whether the size of context data in a single task influences the prediction accuracy. We clustered the context data into small, medium, large, and very large contexts and conducted a Chi-Squared test of independence. The test rejected the hypothesis that the accuracy is dependent on the size for all similarity models ($p < 0.02$).

We observed a small difference between the relevance models DOI and FDA in the evaluation. The hit ratios of the FDA model were slightly better than those of the DOI model. Interestingly, the pure Jaccard model and the text-based similarity performed almost always best for predicting similar tasks. A scaling function that weights the interaction events (e.g., selects, edits, views, etc.) in the context data might change the prediction accuracy of the relevance models. neither Moreover, we originally designed the FDA model to distinguish between tools and other artifacts. In the evaluation, we had to use existing Mylyn data which does not include tool interaction. We think that interaction with tools contributes to defining the context and comparing its similarity and might lead to improve the prediction accuracy. This is in line with the qualitative findings shown in Section 3.3. Two tasks appear to be similar, when they require the same tools, even for different artifacts. When determining the relevance of a tool, the usage duration becomes important, since a single tool might be used to manage multiple artifacts, so that switching between tools happens infrequently (compared to artifacts). The less tools and artifacts developers need to solve a task, the more important the duration becomes instead of the frequency.

7.2. Context-based vs. content-based similarity

Using the textual descriptions of the tasks as indicator for the task similarity seems to work roughly as well as using the context. One major advantage of the context-based approach is that it does not require the task descriptions. Previous studies have confirmed

Please cite this article as: W. Maalej et al., Using contexts similarity to predict relationships between tasks, The Journal of Systems and Software (2017), http://dx.doi.org/10.1016/j.jss.2016.11.033

that a large number of developers' tasks are not explicitly documented or described and rather exist in the head of the developers (Maalej, 2009; Mark et al., 2005; Maalej and Happel, 2010; Bachmann et al., 2010). Characterizing and identifying the relationships between these rather informal tasks based on context might help developers better manage their work.

One disadvantage of our approach is that the developers need to start working on a task in order to have a first set of interactions which characterize the context. Certainly, our approach of context similarity cannot handle completely new tasks. This is a major limitation. Nevertheless, as developers' work is often fragmented (Sanchez et al., 2015) there is a plenty of ongoing parallel tasks: some defined as tasks with descriptions, some only informal or with quick references. The context of these ongoing tasks can be analyzed to identify relationships between them. LaToza et al. (2006) surveyed Microsoft developers and found that 62% agreed that they have "to switch tasks often because of requests from my teammates or manager", more frequently than switching work because the task is blocked. We think that a combination of the context-based (interaction-based) and text-based task is needed for such a similarity model to be applicable in practice.

To predict the tasks' relationships based on contexts, the tasks do not have to be completed but they have to be started. That is, context data must be available for the comparison at the prediction time. Context data would be available for interrupted or postponed tasks. Predicting task relationships can be done at any of these points or simply "on demand" during the work on a task. Our results show that context data are available in the studied projects and often used during code review work (Section 5.3)—independently from whether the code review is defined as a separate task, conducted as a part of the original development task itself, or sometime not even defined as a task. We observed similar scenarios for testing and integration work.

Several previous studies (e.g. Maalej, 2009 and Bachmann et al., 2010) showed that a large portion of developers work ($\sim$ 50%) is rather informal and neither strictly defined as task nor assigned in issue trackers and task lists. This is the case even in professional software development teams (or perhaps especially for these teams). For instance, Bachmann et al. (2010) hired a very experienced Apache developer to analyze committed Apache source code and trace it to bugs/tasks in the project's issue tracker. The authors found that only 48% of the commits are documented in the issue tracker while the remaining code changes are informal. Maalej (2009) surveyed 782 professional developers and found very similar results. Only 30% of respondents confirmed that the majority of their tasks are defined in issue trackers. The remaining 70% claimed that a half or more of the work is rather informally specified without a task description. A further example is of agile teams who often use sticky notes instead of tasks in issue trackers.

We think that the context of a task includes additional dimensions which we did not consider in this paper. Our evaluation results show that developers (at least beginners) think, that the context similarity depend on other factors than the interactions, artifacts, and tools. For instance, the similarity of functionality to be implemented in the tasks and the similarity of architecture components and technology used are two strong indicators for context similarity. These dimensions partly represent the developers' mental models (Singer et al., 1997). In addition to their behaviors externalized in interactions, tools, and artifacts, the thoughts about what is the functionality, how to associate the work into the architecture, as well as how to use the technology seem to be an important part of the context. Neither FDA nor DOI use these dimensions. We think that considering them (e.g., by mining or associating keywords to context) might improve the prediction accuracy of a context similarity model.

7.3. Application areas for context similarity models

One main motivation of this work was to evaluate whether related tasks can be recommended to developers based on the context of current task and context of other tasks. Our research results are encouraging but also show that hybrid models would probably be needed in practice. Also additional research is still needed to achieve more accurate results with higher precision and recall values. Overall, we expect two main areas of applying context similarity models to predict task relationships: developer's productivity and collaboration and awareness.

7.3.1. Developer's productivity: recommending similar tasks

Development work is frequently interrupted requiring developers to switch between tasks back and forth (Parnin and Rugaber, 2011; Maalej, 2009). Each time developers postpone tasks and resume others their time is wasted by reestablishing the contexts in their mind. We envision a recommendation system that suggests to the developers the tasks they should work on next where the effort needed for the switch is minimized. If one would apply the principles of Getting Things Done (Allen, 2001) to the domain of software engineering by comparing the contexts of the tasks and recommending the tasks with most similar contexts, this would probably reduce context switches, save time, and increase productivity. The context might consist of the interaction of the developers with the tools (such as Debuggers and Web browsers) and the artifacts (such as source code elements, documentation pages, and communication threads). But context might also include additional information to reflect the mental state including technology or architecture standards used. Two contexts are similar if their most relevant context information, including artifacts and tools, are the same.

In order to check whether developers tend to switch to similar, related tasks in practice, we ran the Jaccard similarity index on the text descriptions of a random sample of 1000 "*isNextTo* task pairs" from our sample Bugzilla dataset. We found that developers tend to switch to relatively dissimilar untreated tasks on the same day: the overall average Jaccard similarity index was only 0.19. This result might indicate that there is potential for recommendation systems that assist developers to switch to similar tasks to what they are currently doing.

However, the context is quite complex and choosing the next task to work on depends on several additional factors—also according to the principles of Getting Things Done. First, the priority of the tasks is clearly important for the work management. Urgent tasks might have a priority. The following is a challenging question: how similar, how related, and how complex should a task to the current work be, to be more meaningful for a developer to work on it than on an other urgent task?

Finally, in addition to a task recommender, it might also be useful to explore analytics tools that simply show the similarity or dependencies of the tasks a developer worked on together with other personal productivity and performance metrics such as the number of tasks closed, number of edits, overall satisfaction etc. This could lead to a better reflection and learning of developers about the "optimal" work management decisions in future.

7.3.2. Collaboration and awareness: recommending dependent tasks

Recommending others' tasks that are related to the current work of the developer and she should be aware of is certainly useful in collaborative settings. A simple use case might be to recommend creating and maintaining *dependsOn* and *blocks* dependencies in issue trackers—something currently being done manually, and certainly not perfect. A developer might need to be aware of the status of tasks that blocks her current work. Alternatively, she might be blocking others' tasks and this might help identifying

Please cite this article as: W. Maalej et al., Using contexts similarity to predict relationships between tasks, The Journal of Systems and Software (2017), http://dx.doi.org/10.1016/j.jss.2016.11.033

priorities. Predicting tasks relationships can thus support collaboration and coordination, setting work priorities, assignments and work plans.

We think that identifying and recommending related tasks based on their contexts can also support knowledge sharing among developers. From a related task, a developer can e.g., learn from the discussion what is important to note, where to find additional reference information, or simply whom to ask if one has a questions (Maalej et al., 2014b). Also recommending similar tasks might tighten the knowledge sharing in how to solve similar issues.

Task relations are crucial for collaborative activities such as code reviewing, testing, integration, or handoff. Predicting related tasks would help identify with whom developers should synchronize to handle dependencies, avoid conflicts, and share knowledge (Blincoe et al., 2013). We noticed in our qualitative analysis of *isNextTo* tasks, that a popular usage of Mylyn contexts is supporting code reviewing and task handoffs—two highly collaborative scenarios (see Section 5.3). Context similarity prediction can thus support such scenarios so that, e.g., code integrators can review similar contributions, instead of switching to contributions on different areas of a large software system.

8. Related work

We focus the related work discussion to approaches of (a) personal productivity management based on task context, (b) approaches to quantify relevance and similarity of artifacts and contexts in software engineering, and (c) approaches for identifying similar, related tasks for development teams.

The conceptual cornerstone of our work is David Allen's methodology Getting Things Done (Allen, 2001). There are several task management tools that implement GTD. Things (Cultured Code GmbH, 2011), MyLifeOrganized[8] and Remember The Milk (RTM)[9] are few examples. Common features include organizing tasks according to task categories, define contexts of the tasks or group tasks by tags, and managing areas of responsibility. While Allen recommend to organize the tasks by their contexts to increase productivity, none of these tools monitors the interaction of users to characterize the context and calculate the task similarity.

Several approaches have been proposed to support knowledge workers in their personal productivity management by using the context. Dragunov et al. proposed TaskTracer, a tool that helps multitasking knowledge workers to rapidly resume past activities and restore contexts (Dragunov et al., 2005). Similarly, Shen et al. introduced a similar tool called TaskPredictor (Shen et al., 2006), and later TaskPredictor2 (Shen et al., 2009), which both use machine learning approaches to detect task switches and to predict the user's current task. TaskTracer, TaskPredictor and TaskPredictor2 also monitors developers' interactions and associates interaction events with tasks. However, these systems aim at supporting developers to quickly restore contexts when resuming tasks (e.g., reopening the files), while we aim at predicting and retrieving related tasks according to their contexts.

Rattenbury and Canny (2007) proposed a task representation and visualization tool called CAAD (Context-Aware Activity Display). It automatically gathers information about the user current task and processes this information to infer the context. By monitoring the user interactions with tools and artifacts and applying pattern mining, CAAD detects structures that encode the task context. It uses these context structures to predict artifacts relevant to the user's current task and to visualize the user's work behavior. Brdiczka (2010) propose a similar approach to construct a task representation based on artifact interactions. The author suggests to monitor a user's desktop activities and leverages artifact usage information without intervening with the content of the artifacts. Using a spectral clustering algorithm, they groups artifacts into tasks using a similarity matrix.

In software engineering, Kersten and Murphy introduced the degree-of-interest model (DOI) for a task-focused programming. Mylyn implements the DOI and is today one of the most recognized task management tools for the Eclipse IDE (The Eclipse Foundation, 2011; BZ Media LLC., 2008). Mylyn monitors a developer's interactions in Eclipse and filters source code elements, which are irrelevant for the current task. The goal is to increase the developer's productivity by reducing information overload (Kersten and Murphy, 2006). Mylyn does not predict similar tasks based on the context information. Switch! (Maalej and Sahm, 2010) is a context-aware artifact recommender, assisting developers in switching artifacts based on interaction history and types of development tasks. While Switch! assists developers to switch to the next artifact needed in a task, in this paper we discuss the recommendation of the next task based on task similarity. A similar concept called "edit wear and read wear" to DOI was introduced first by Hill et al. (1992), including the use of time spent on particular lines in a text editor. The idea of wear was applied to software development by DeLine et al. (2005) to facilitate the understanding of programs through wear-based filtering.

With TASKREC, Vo et al. aim at recommending relevant tasks for users according to their situation and environment capabilities (Vo et al., 2009). Assuming that "people accomplish similar tasks in similar situations" they propose a measure for task-based situation similarity based on context attributes. This approach is similar to our since it is also based on the same assumption that task similarity can be predicted based on the context of executing the task. However, the domain, definition of tasks, and similarity models are different. Vo et al. focus on physical smart spaces with sensors and actuators, while we focus on knowledge workers and their interaction history as context.

Finally, UMEA (Kaptelinin, 2003) also monitors user interactions with artifacts. It uses a strategy for creating and managing task-related work contexts based on activity theory. It supports the user in organizing resources relevant to a (higher-level) task in order to make them easily available when the user resumes this task. We focus on organizing tasks based on their context similarity rather than task hierarchies. Our models are tailored to typical situations in developer's work, as it considers particular artifact and interaction types.

Both relevance and similarity are two well-studied concepts in statistics, machine learning, and information retrieval. Some of the assumptions of this work are based on these results such as Jaccard. In software engineering several authors have studied the relevance and similarity. For instance, Parnin and Gorg (2006) looked at various ways to compute file relevance for a task and used (including recency and frequency). Sridhara et al. (2008) discussed six different state-of-the-art similarity techniques applying on software. They find out that all of them did not perform well on a higher level. Grechanik et al. (2010) developed a search engine for solution domains to find highly relevant applications. They discovered that there is need for a system that can find relevant applications faster and more precise than other search engines. Jeh and Widom (2002) proposed SimRank to measure the similarity of the structural context in which objects occur based on their relationships with other objects. Two objects are similar if they are related to similar objects. For a given domain, SimRank can be combined with other domain-specific similarity measures. In this work we

[8] http://www.mylifeorganized.net/products/my-lifeorganized/GTD-Getting-Things-Done.htm.
[9] http://www.rememberthemilk.com

focused on a specific view of task similarity based on comparing the developers' interaction histories when working on the task.

Rastkar and Murphy (2009) studied the computed similarity of 700 bug reports from the Eclipse Mylyn project. They compared bug reports found similar through a comparison of changesets to the set of bug reports found similar through contexts. The authors used a normalized cosine similarity metric for both approaches. While the aim of this work is similar to ours (predicting task relationships) the authors studied only computed relationships (similarity) and did not compare the predicted relationship to the manually identified relationship by developers. Moreover, the author used textual names of the artifacts to uniquely identify the context elements and not the interaction. Finally, the authors focused on comparing the context without relating it to the tasks (i.e. bug reports).

Rocha et al. (2015) described an approach to predict similar bugs a developer should work on next to reduce context switches. They used the text descriptions as the summary of the bug and a cosine algorithm to predict a next, similar bug (see also Jalbert and Weimer, 2008). We also identified related, including similar bugs, by using context-based approaches as well as text-based approaches (LSI) to predict similar bugs based on their textual description and extend their research. Our goal was rather to study and compare multiple approaches than to suggest and evaluate a particular approach.

Alipour et al. (2013) suggested a new approach to find related especially duplicate bugs by using contextual information. They derived this information from different sources as commit log comments, bug reports, and the category of the bug to find duplicate Android bugs. We did not study the duplicate relationships but investigated the isSimilarTo as indicated by developers. Moreover, the definition of context according to Alipour et al. is different to ours since we focus on context including the interactions of the developer when fixing a bug. In practice, the various contextual information might be complementary.

Lamkanfi et al. (2011) used four classification algorithms to predict the severity of a bug. To predict bugs they used also meta information of the bug reports as the product and component name as well as the bug description. In our study we found that developers often switch to bugs have a high severity that have not consequently be contextual similar to each other. We expect additional research to find significant factors a NextBug has a high severity. Researchers could learn from a history of bug context data and could study what artifacts are touched frequently when fixing a critical bug, a block bug or a trivial bug. A trivial bug might be a UI bug as changing the color of a UI element.

Finally, there is a large body of research about coordination and collaboration issues of development teams, e.g., studying developers social networks (Zanetti et al., 2013), mining of the revision histories (Zimmermann, 2009), or analyzing the text comments of the bug reports (Runeson et al., 2007). Our work brings a new, complementary perspective to identify when developers should synchronize and of which changes they should be aware of by comparing the context of tasks (i.e., the interaction history). Perhaps the most related work is of Blincoe et al. (2013), who analyzed the information needs of developers working on tasks in parallel using the Bugzilla repository. The authors identified several issues coordinating tasks between developers in large projects. We hope that the context similarity models and the differentiated evaluation contribute to identifying and recommending tasks on which the developers should coordinate.

Ying and Robillard (2011) investigated the differences of occurrences of an interaction event (edit) in a programming task. They investigated that there are significant differences in the number of occurrences of the interaction event working on enhancement tasks, minor, and major bug fixes. These results can be used to scale the context similarity models and reach a higher accurate prediction rate, by enabling the weighting of the interaction events for edit if a specific kind of task has to be recommend in the upper front positions. Recently, Soh et al. (2015) studied noises in context data, and found that Mylyn context data can miss on average about 6% of the time spent performing a task and contain on average about 28% of false edit-events. The authors conclude that interaction traces must be carefully cleaned before being used in research studies. We had the same assumption when we introducing the FDA relevance models. The results of this study might be used to further remove noise when calculating the contexts similarity.

9. Conclusion

Development tasks have a broad variety of task relationships between them (Thompson et al. 2016). A task might e.g. follow, block, depend on, or be part of another task. Tasks might also be similar on what their output should be and how they should be performed. Automatically identifying task relationships would be thus useful to developers in various scenarios.

This work investigates how context data, extracted from developer interactions with development artifacts, can be used to predict relationships between tasks. We took a broad approach and evaluated several context similarity models (Jaccard, DOI Jaccard, FDA Jaccard) on two types of task relationships useful for work coordination (*dependsOn* and *blocks*) and two types useful for personal work management (*isNextTo* and *isSimilarTo*). We also compared the results with relationships prediction based on mining the task descriptions. We evaluated our assumption in different scenarios: a field study with professionals, an array of simulations using Eclipse Bugzilla data (including manual analysis of task relationships on a subset of the data), and an experiment and a survey with students.

Overall the results are encouraging. The context similarity models and task relationships evaluated point in the same direction in the three studies: context can be used to predict task relationships—significantly better than a random prediction and at least as good as mining the textual descriptions of tasks. It seems also, the more related the tasks are, the better the context similarity predicts the relationships. As for the accuracy of the context similarity models evaluated there was no clear winner in our studies.

We made several interesting observations. For instance, developers in the Bugzilla study tend to switch to tasks with dissimilar contexts in the same work day, even if they attempt to keep working on the same software product or component. This indicates that a task recommenders based on context similarity might be useful, as developers are successful in staying focused on the same broad components, but not so much on similar fine-grained tasks. Our manual analysis shows that contexts are used for knowledge sharing and coordination, e.g. as support for code reviewing when having a similar context may be useful as well.

We think that our results can be used to build task recommenders, yet the accuracy of the prediction models evaluated is certainly far from being perfect to be applied in practice. From the students survey we identified dimensions that might make task contexts more accurate such as the technology and tools used. We also think that a task recommender will need to consider other information such as the text description, discussions and annotations tasks, as well properties such as the task severity, complexity or simply the level or concentration and remaining work time (Allen, 2001).

Certainly, our approach of context similarity cannot handle completely new tasks. Context data must be available for the comparison at the prediction time. This is a major limitation. To predict the tasks' relationships based on contexts, the tasks do not have to

Please cite this article as: W. Maalej et al., Using contexts similarity to predict relationships between tasks, The Journal of Systems and Software (2017), http://dx.doi.org/10.1016/j.jss.2016.11.033

be completed, but they have to be started. In contrast, previous research has also shown that about half of development work is rather informal and not documented as tasks with textual descriptions. Identifying related tasks based on text similarity would not be possible in this case too. Typically, there is a plenty of ongoing pieces of work in development projects: some defined as tasks with descriptions, some only informal. In practice, we think a hybrid approach of context- and content-based prediction of task relationships might be the most appropriate.

Acknowledgements

We are grateful to the JSS reviewers for their feedback. We also would like to thank Vincenz Doelle and Amel Mahmuzic for helping with the implementation of the FDA model and execution of the field study and the experiment with students. We are thankful to all study participants in particular in the field study. Part of this work was funded by the German Academic Exchange Service A1401226 DAAD. R. Robbes is partially supported by FONDECYT project #1151195.

References

Alipour, A., Hindle, A., Stroulia, E., 2013. A contextual approach towards more accurate duplicate bug report detection. In: Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, pp. 183–192.

Allen, D., 2001. Getting Things Done. The Art of Stress-Free Productivity. Penguin, New York, New York, USA.

Bachmann, A., Bird, C., Rahman, F., Devanbu, P.T., Bernstein, A., 2010. The missing links: bugs and bug-fix commits. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7–11, 2010, pp. 97–106. doi:10.1145/1882291.1882308.

Blincoe, K., Valetto, G., Damian, D., 2013. Do all task dependencies require coordination? The role of task properties in identifying critical coordination needs in software projects. In: Proceedings of the 2013 Ninth Joint Meeting on Foundations of Software Engineering. ACM, pp. 213–223.

Bradford, R.B., 2008. An empirical study of required dimensionality for large-scale latent semantic indexing applications. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management. ACM, pp. 153–162.

Brdiczka, O., 2010. From documents to tasks: Deriving user tasks from document usage patterns. In: Proceedings of the 15th International Conference on Intelligent User Interfaces. ACM, New York, NY, USA, pp. 285–288. doi:10.1145/1719970.1720012.

Bruegge, B., Dutoit, A.H., 2009. Object-Oriented Software Engineering, third ed. Prentice Hall.

BZ Media LLC., 2008. 5th Annual Eclipse Adoption Study. Technical Report November. BZ Media LLC.

Cultured Code GmbH, 2011. Things. URL http://culturedcode.com/things

Deerwester, S.C., Dumais, S.T., Landauer, T.K., Furnas, G.W., Harshman, R.A., 1990. Indexing by latent semantic analysis. J. Am. Soc. Inf. Sci. Technol. 41 (6), 391–407.

DeLine, R., Khella, A., Czerwinski, M., Robertson, G., 2005. Towards understanding programs through wear-based filtering. In: Proceedings of the 2005 ACM Symposium on Software Visualization. ACM, New York, NY, USA, pp. 183–192. doi:10.1145/1056018.1056044.

Dragunov, A.N., Dietterich, T.G., Johnsrude, K., Mclaughlin, M., Li, L., Herlocker, J.L., Hall, D., 2005. TaskTracer: a desktop environment to support multi-tasking knowledge workers. In: Proceedings of the 10th International Conference on Intelligent User Interfaces—IUI '05. ACM Press, New York, New York, USA, p. 75. doi:10.1145/1040830.1040855.

Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., Mirakhorli, M., 2011. On-demand feature recommendations derived from mining public product descriptions. In: Proceedings of the 33rd International Conference on Software Engineering. ACM, New York, NY, USA, pp. 181–190. doi:10.1145/1985793.1985819.

Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., Cumby, C., 2010. A search engine for finding highly relevant applications. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—Volume 1. ACM, New York, NY, USA, pp. 475–484. doi:10.1145/1806799.1806868.

Hariri, N., Castro-Herrera, C., Cleland-Huang, J., Mobasher, B., 2014. Recommendation systems in requirements discovery. In: Recomemndation Systems in Software Engineering. Springer Verlag, pp. 455–476.

Hill, W.C., Hollan, J.D., Wroblewski, D., McCandless, T., 1992. Edit wear and read wear. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, New York, NY, USA, pp. 3–9. doi:10.1145/142750.142751.

Hjorland, B., Sejer Christensen, F., 2002. Work tasks and socio-cognitive relevance: a specific example. J. Am. Soc. Inf. Sci. Technol. 53 (11), 960–965.

Iqbal, S.T., Horvitz, E., 2007. Disruption and recovery of computing tasks. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems—CHI '07. ACM Press, New York, New York, USA, p. 677. doi:10.1145/1240624.1240730.

Jalbert, N., Weimer, W., 2008. Automated duplicate detection for bug tracking systems. In: Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on. IEEE, pp. 52–61.

Jeh, G., Widom, J., 2002. Simrank: a measure of structural-context similarity. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York, NY, USA, pp. 538–543. doi:10.1145/775047.775126.

Kaptelinin, V., 2003. UMEA. In: Proceedings of the Conference on Human Factors in Computing Systems—CHI '03. ACM Press, New York, New York, USA, p. 353. doi:10.1145/642611.642673.

Kersten, M., Murphy, G.C., 2005. Mylar : a degree-of-interest model for IDEs. In: Proceedings of the Fourth International Conference on Aspect-oriented Software Development. ACM Press, New York, NY, USA, p. 159—-168. http://doi.acm.org/10.1145/1052898.1052912.

Kersten, M., Murphy, G.C., 2006. Using task context to improve programmer productivity. In: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering—SIGSOFT '06/FSE-14. ACM Press, New York, New York, USA, pp. 1–11. doi:10.1145/1181775.1181777.

Ko, A.J., DeLine, R., Venolia, G., 2007. Information needs in collocated software development teams. In: 29th International Conference on Software Engineering—ICSE' 07. IEEE Computer Society, Washington, DC, USA, pp. 344–353. doi:10.1109/ICSE.2007.45.

Lamkanfi, A., Demeyer, S., Soetens, Q.D., Verdonck, T., 2011. Comparing mining algorithms for predicting the severity of a reported bug. In: Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. IEEE, pp. 249–258.

LaToza, T.D., Venolia, G., DeLine, R., 2006. Maintaining mental models: a study of developer work habits. In: Proceedings of the 28th International Conference on Software Engineering. ACM, New York, NY, USA, pp. 492–501. doi:10.1145/1134285.1134355.

Luostarinen, T., Kohonen, O., 2013. Using topic models in content-based news recommender systems. In: Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013), pp. 239–251.

Maalej, W., 2009. Task-first or context-first? Tool integration revisited. In: Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering—ASE '09. IEEE Computer Society, Washington, DC, USA, pp. 344–355. doi:10.1109/ASE.2009.36.

Maalej, W., 2010. Intention-Based Integration of Software Engineering Tools. Technische Universität München Ph.D. thesis.

Maalej, W., Fritz, T., Robbes, R., 2014. Recommendation Systems in Software Engineering. Springer, Ch. Collecting and Processing Interaction Data for Recommendation Systems. pp. oftware Engineering. Springer, Ch. Collecting and Processing Interaction Data for Recommendation Systems. pp. 173-197.

Maalej, W., Happel, H.-J., 2008. A lightweight approach for knowledge sharing in distributed software teams. In: Proceedings of the Seventh International Conference on Practical Aspects of Knowledge Management—PAKM'08. Springer-Verlag, Berlin, Heidelberg, Germany, pp. 14–25.

Maalej, W., Happel, H.-J., 2010. Can development work describe itself? In: 2010 Seventh IEEE Working Conference on Mining Software Repositories (MSR 2010), pp. 191–200. doi:10.1109/MSR.2010.5463344.

Maalej, W., Sahm, A., 2010. Assisting engineers in switching artifacts by using task semantic and interaction history. In: Proceedings of the Second International Workshop on Recommendation Systems for Software Engineering—RSSE '10. ACM Press, New York, NY, USA, pp. 59–63. doi:10.1145/1808920.1808935.

Maalej, W., Tiarks, R., Roehm, T., Koschke, R., 2014. On the comprehension of program comprehension. ACM Trans. Software Eng. Method. 23 (4), 31:1–31:37. doi:10.1145/2622669.

Mark, G., González, V.M., Harris, J., 2005. No task left behind? Examining the nature of fragmented work. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems—CHI '05. ACM Press, New York, NY, USA, pp. 321–330.

Tan, P.N., Steinbach, M., Kumar, V., 2006. Introduction to Data Mining. Pearson Addison Wesley.

Parnin, C., Gorg, C., 2006. Building usage contexts during program comprehension. In: Proceedings of the 14th IEEE International Conference on Program Comprehension. IEEE Computer Society, Washington, DC, USA, pp. 13–22. doi:10.1109/ICPC.2006.14.

Parnin, C., Rugaber, S., 2011. Resumption strategies for interrupted programming tasks. Softw. Qual. Control 19 (1), 5–34. doi:10.1007/s11219-010-9104-9.

Rastkar, S., Murphy, G.C., 2009. On what basis to recommend: changesets or interactions? In: Proceedings of the 2009 Sixth IEEE International Working Conference on Mining Software Repositories. IEEE Computer Society, Washington, DC, USA, pp. 155–158. doi:10.1109/MSR.2009.5069494.

Rattenbury, T., Canny, J., 2007. CAAD. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems—CHI '07. ACM Press, New York, New York, USA, p. 687. doi:10.1145/1240624.1240731.

Řehůřek, R., Sojka, P., 2010. Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. ELRA, Valletta, Malta, pp. 45–50. http://is.muni.cz/publication/884893/en.

Rocha, H., Oliveira, G.d., Marques-Neto, H., Valente, M.T., 2015. Nextbug: a Bugzilla extension for recommending similar bugs. J. Softw. Eng. Res. Dev. 3 (1), 1–14. doi:10.1186/s40411-015-0018-x.

Runeson, P., Alexandersson, M., Nyholm, O., 2007. Detection of duplicate defect reports using natural language processing. In: Software Engineering, 2007. ICSE 2007. 29th International Conference on, pp. 499–510. doi:10.1109/ICSE.2007.32.

Sanchez, H., Robbes, R., González, V.M., 2015. An empirical study of work fragmentation in software evolution tasks. In: 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2–6, 2015, pp. 251–260. doi:10.1109/SANER.2015.7081835.

Shani, G., Gunawardana, A., 2011. Evaluating recommendation systems. In: Recommender Systems Handbook. Springer, pp. 257–297.

Shen, J., Irvine, J., Bao, X., Goodman, M., Kolibaba, S., Tran, A., Carl, F., Kirschner, B., Stumpf, S., Dietterich, T.G., 2009. Detecting and correcting user activity switches: algorithms and interfaces. In: Proceedings of the 14th international conference on Intelligent user interfaces (IUI '09). ACM, New York, NY, USA, pp. 117–126. doi:10.1145/1502650.1502670.

Shen, J., Li, L., Dietterich, T.G., Herlocker, J.L., 2006. A hybrid learning system for recognizing user tasks from desktop activities and email messages. In: Proceedings of the 11th International Conference on Intelligent User Interfaces—IUI '06. ACM Press, New York, New York, USA, p. 86. doi:10.1145/1111449.1111473.

Singer, J., Lethbridge, T., Vinson, N., Anquetil, N., 1997. An examination of software engineering work practices. In: Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON '97. IBM Press, p. 21.

Soh, Z., Drioul, T., Rappe, P.A., Khomh, F., Gueheneuc, Y.G., Habra, N., 2015. Noises in interaction traces data and their impact on previous research studies. In: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–10. doi:10.1109/ESEM.2015.7321209.

Sridhara, G., Hill, E., Pollock, L., Vijay-Shanker, K., 2008. Identifying word relations in software: a comparative study of semantic similarity tools. In: Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension. IEEE Computer Society, Washington, DC, USA, pp. 123–132. doi:10.1109/ICPC.2008.18.

The Eclipse Foundation, 2011. Eclipse Integrated Development Environment. URL http://www.eclipse.org.

Thompson, C.A., Murphy, G.C., Palyart, M., Gašparič, M., 2016. How software developers use work breakdown relationships in issue repositories. In: Proceedings of the 13th International Conference on Mining Software Repositories. ACM, New York, NY, USA, pp. 281–285. doi:10.1145/2901739.2901779.

Vo, C.C., Torabi, T., Loke, S.W., 2009. Towards context-aware task recommendation. In: 2009 Joint Conferences on Pervasive Computing JCPC, pp. 289–292. doi:10.1109/JCPC.2009.5420173.

Walker, R.J., Holmes, R., 2014. Simulation—a methodology to evaluate recommendation systems in software engineering. In: Recomemndation Systems in Software Engineering. Springer Verlag, pp. 301–332.

Yao, Y., 1995. Measuring retrieval effectiveness based on user preference of documents. JASIS 46 (2), 133–145.

Ying, A.T., Robillard, M.P., 2011. The influence of the task on programmer behaviour. In: Program Comprehension (ICPC), 2011 IEEE 19th International Conference on. IEEE, pp. 31–40.

Zanetti, M.S., Scholtes, I., Tessone, C.J., Schweitzer, F., 2013. Categorizing bugs with social networks: a case study on four open source software communities. In: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 1032–1041.

Zimmermann, T., 2009. Changes and bugs; mining and predicting development activities. In: Software Maintenance, 2009. ICSM 2009. IEEE International Conference on, pp. 443–446. doi:10.1109/ICSM.2009.5306296.

Please cite this article as: W. Maalej et al., Using contexts similarity to predict relationships between tasks, The Journal of Systems and Software (2017), http://dx.doi.org/10.1016/j.jss.2016.11.033

Walid Maalej is a professor of informatics at the University of Hamburg where he leads the Applied Software Technology group. His research interests include data-driven software engineering, context-aware adaptive systems, e-participation and crowdsourcing, and software engineering's impact on society. He received his Ph.D. in software engineering from the Technical University of Munich. He is an editorial board member of the Journal of Systems and Software and a Junior Fellow of the German Computer Science Society (GI).

Mathias Ellmann is a Ph.D. at the Applied Software Technology research group of the University of Hamburg in Germany. He studied electrical engineering and information technology (B. Eng.) as well as industrial engineering and management (M.Sc.). He previously worked at different companies such as Fraunhofer Institute and Siemens Corporate Research. His research focuses on context sensitive systems, data analytics, and empirical software engineering.

Romain Robbes is an associate professor at the University of Chile (Computer Science Department), in the PLEIAD research lab. He earned his Ph.D. in 2008 from the University of Lugano, Switzerland and received his Master's degree from the University of Caen, France. His research interests lie in Empirical Software Engineering, including Mining Software Repositories. He authored more than 60 papers on these topics, including top software engineering and programming languages venues such as ICSE, FSE, ASE, EMSE, ECOOP, or OOPSLA, received best paper awards at WCRE 2009 and MSR 2011, and was the recipient of a Microsoft SEIF award 2011. He has served in the organizing and program committees of many software engineering conferences and serves on the Editorial Board of EMSE.

Please cite this article as: W. Maalej et al., Using contexts similarity to predict relationships between tasks, The Journal of Systems and Software (2017), http://dx.doi.org/10.1016/j.jss.2016.11.033