Using Task Contexts to Improve Software Process Execution

Ivens da S. Portugal[1] and Toacy C. de Oliveira[1]

[1] PESC/COPPE – Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil

Abstract. During a software process execution, software engineers often deal with a sheer number of artifacts to be consulted while only a small set of them is needed to perform a given activity. Thus, the search for suitable artifacts to perform an activity, usually referred to as an activity context, may be tiring, time consuming, error-prone and may demand the software engineer additional effort. This behavior leads to a lower productivity. A Degree of Interest (DOI) function is a mechanism that scores elements according to a particular rule specified earlier. It can be used to evaluate artifact's interest value in relation to each of the software process' activities. Mylyn is an Eclipse plugin that implements a DOI function. It is aimed at providing support for programmers in the search of Java classes during coding task's executions. However, Mylyn's DOI function does not take into consideration particularities of other phases of the execution of a software process such as use case or test case description. In addition to this lack of support, Mylyn's DOI function does not use the underlying software process to infer task contexts. For that reason, this work expands this DOI function so (i) software engineers may be assisted in the search for artifacts no matter which activity is being executed and (ii) the software process is considered in the calculation of a given artifact's interaction value. The final implementation was named MylynSDP. A validation study was conducted to assess the concepts described here.

Keywords. Software Development Process, Software Process Specification, Software Process Execution, Degree of Interest (DOI) function, Task Context

1 Introduction

Software processes have been used to guide the development of software products since the last century [7] [18]. The underlying assumption is that the use of a software process during the development of a system reduces the chances of system failure and increases the overall quality of the final system. Some studies point to the same direction [3] [1].

During the execution of a software process, software engineers involved in the development of the system perform activities and manipulate artifacts. In most cases, they have a sheer number of artifacts available to be manipulated whereas he only needs a small subset of them. For instance, RUP from IBM [11] has more than a hun-

dred artifacts in its specification, but only a relatively small number of these artifacts are needed to perform each of its activities. Due to this, a search for the suitable artifacts for each activity must be done. As the software engineer performs this search over a great number of artifacts, it can be tiring, error-prone and time consuming. Once the needed artifacts are at hand, the execution of the activity then starts.

The search problem arises again when changing activities. A change of an activity's execution may be caused by either an interruption by a high-priority activity or by the decision to finish that activity's execution later. In both cases, a new search for the suitable artifacts will have to be performed again, which brings all of its negative effects early mentioned into consideration again. The subset of the suitable artifacts related to one activity's execution is often named the context of that activity. For that reason, the problem of changing activities is referred to as the context change problem.

Both the search and the context change problems negatively affect the software engineer's productivity [16]. A solution for these problems is the utilization of a Degree of Interest function. A DOI function is a mechanism that scores and ranks elements based on a predefined rule. Software processes' artifacts may be given an interest value based on their importance to the current activity's execution. Thus, whenever a software engineer starts the execution of an activity, the most important artifacts are highlighted for him, which drastically reduces time and effort spent on the context search.

An implementation of a DOI function can be found on Mylyn [15]. Mylyn is a plugin for Eclipse IDE, aimed at helping software developers finding Java classes and methods suitable for the task being performed. Mylyn's DOI function associates an interest value to each class of the Java project based on the programmer's interaction with the classes. The interest value association is done in such a way that the more manipulated a class is, the more interesting it is to the task execution.

Mylyn's DOI function effectively solves the search and the context change problems for the code implementation stage. However, it is focused for that specific stage only, which makes it not suitable for other software process' stages. Moreover, the definition of programmers' tasks is manually done without any aid that could support or justify its creation.

Regarding this scenario, Mylyn's DOI function was extended in order to help software engineers solve the search and context change problems in all phases of the software process during its execution. In addition to it, the new DOI function consults the software processes modeled to infer the importance of artifacts in relation to a given activity and to aid the software engineer in the definition of the activities to be performed. The new DOI function's implementation was named MylynSDP.

A validation study was conducted in order to assess the concepts described in this paper. Participants executed a software process in a simulated environment and then answered a technology acceptance questionnaire. The overall acceptance of MylynSDP was positive. Details of the validation study are described at the end of this work.

This paper is organized as follows. Section 2 describes some related works that helped in the development of this paper. Section 3 presents and describes in detail MylynSDP and its new DOI function. Section 4 is related to the validation study con-

ducted and its analysis. Section 5 concludes this paper with suggestions for future work.

2 Related Work

One of the first attempts to manage the context of a task to by pointing out a subset of suitable documents related to that task come from Placeless and Presto projects from Xerox Palo Alto Research Center (PARC) [5]. According to them, conventional approaches emphasize the location of documents of a project when organizing them rather than their usage. For that reason, project documents are stored in a hierarchical structure of folders and subfolders and not stored according to what they relate to (e.g. project, priority, level of expertise needed). Presto is a prototype system that allows users to apply properties to documents and organize them according to these properties. Placeless was a project inside Xerox used to assess the concepts proposed. One major drawback of Presto and Placeless projects is that a user must manually categorize all of his files, which still requires an elevated amount of time and effort to be spent.

Researchers from Oregon State University developed a software system to help highly multitasking workers in the categorization and search of daily work documents. The system was named TaskTracer [6]. TaskTracer divides work in discrete units called tasks. When the user starts a task, TaskTracer records what is being done in every task to build a task profile, which is very similar to a task context. If the user has his work interrupted, he can restore his applications later and return to what he was working with at the time of the interruption with low effort. Nonetheless, TaskTracer is limited to Microsoft Office, Visual Studio and Internet Explorer applications, which does not fully represent all the applications a software engineer deals during his work.

In Sweden, researchers from Umeå University created a system called UMEA, which stands for User-Monitoring Environment for Activities [12]. UMEA uses the concept of project spaces to split up the execution of tasks from each other. Each project space is a separate work environment to help the user organize resources according to his desired way. Moreover, UMEA monitors user activities to automatically add new resources to a given project space. UMEA reduces the time and effort needed to search suitable documents and automatically organizes them into context, or project spaces, to aid the user on a context change. However, there is not a support for a process to guide the execution of tasks. A process can help define which tasks should be done and help the user even more.

Most of the work aimed at helping on the management of the documents and task contexts do not consider a process as the underlying guide to the execution of the tasks. Thus, a research on Process-Centered Software Engineering Environments (PSEE) had to be performed. A PSEE is an environment that provides software engineers several services such as process modeling, process execution, software engineer team coordination, deadlines monitoring and creation of reports [8] [10]. Not all PSEEs offer all possible services, but most of them help software engineers to manipulate software process artifacts [15] [2].

One of the PSEE that should be pointed out is WebAPSEE [17]. Developed by researchers of Federal University of Pará, Brazil, WebAPSEE is a PSEE aimed at providing automation and flexibility of software processes at both modeling and execution times. This PSEE stores information about several projects, such as their activities and artifacts, as well as data about software engineers and their roles. A software engineer can be allocated to work on a given task with some artifacts, and the association can be edited at execution time. WebAPSEE has an artifact repository, which not only stores the artifacts to be manipulated by software engineers but also works as a version control system. However, the association between a software process activity and artifacts is done manually even though most of the information needed to do this automatically is accessible in the underlying software process.

2.1 Mylyn

Although Mylyn is not a PSEE, its contributions to documents categorization and search are interesting. Mylyn [13] [14] was developed by researchers of University of British Columbia, Canada, in the form of an Eclipse IDE plugin. Eclipse is one of the most used IDEs for software development and, for that reason, Mylyn and its Degree of Interest function, is aimed at helping programmers in their daily coding work.

Mylyn's initial interface is comprised of five views: Mylyn Package Explorer, Mylyn Problem List, Mylyn Outline, Mylyn Task List and the working area. Mylyn Package Explorer stores Java Projects, packages and classes that will be manipulated during the development of a system. Mylyn Problem List shows Java syntax error and warnings to help programmers fix their codes. Mylyn Outline summarizes a Java document by showing its classes, methods and variables in a concise way. Mylyn Task List displays all the tasks alongside a button to start or finish the execution of each of them. The working area is where Java classes will be created and edited. The look of Mylyn's interface is similar to the one presented in Figure 1.

The more the programmer manipulates Java classes, the more they become more important to the task being executed, and thus, the more it should belong to that particular task context. Other Java classes, that are not constantly used are considered of low interest and later are filtered out from the programmers view. For this to work properly, Mylyn defined five types of interaction events that a programmer may perform when dealing with Java classes. Table 1 shows each of these interactions and briefly explains them. Mylyn's DOI function scores Java classes based on the interactions they receive. Table 1 also describes what points each of the interactions contribute to that Java class' interest value.

Table 1. Mylyn's interaction events and their scores. Each interaction event contributes to an artifact's interest value with the score indicated on this table

Interaction Event	Description	Score
Selection	Selection of artifacts with the mouse or the keyboard.	1 point
Edition	Edition of the contents of artifacts.	0.7 point
Command	Commands such as saving or compiling	1 point
Propagation	Indirect interactions on elements (e.g. renaming a meth-	1 point

	od name affects all classes that executes that method).	
Prediction	Capture of future interactions based on previous ones.	1 point

Mylyn's DOI function usage starts with the creation of the artifact. The programmer uses the Eclipse's Java class creation wizard, specifies some parameters and finishes the creation. At this point, a new Java class is created, without any interest value associated. Once a class is created, the programmer can manipulate it, i.e. interact with it either by selecting it, editing it, saving it or by any other way specified in Table 1. If a selection is performed at that new Java class, that information is saved in a log file. The interactions performed on other classes affect the final interest value of the classes that were not interacted. In this example, the programmer performed 10 selections on other classes. All of these interaction events are saved to the log file as expected. Whenever Mylyn's DOI needs to calculate one Java class' interest value, it consults the log file and looks for the set of interaction events targeting that particular class. In this example, there is only one interaction event: the selection one. Once Mylyn's DOI function has gathered all interaction for one class, it can start calculating its interest value. Mylyn's DOI function multiplies each interaction by its score and then subtracts from it a decay value. The decay value is a small value that is subtracted from the interest value for a class at each programmer's interaction with other classes. For that reason, less manipulated class are eventually excluded from a task context. The default decay value is 0.017. Therefore, in this example, the class' interest value is 1 x 1 - (10 * 0.017) = 0.83.

The creation of a task is straightforward. The software engineer creates a task using a Mylyn wizard either by browsing the menu or by clicking "new task" button on Mylyn's Task view. Once the task is created, no initial context task in built. It will be built, though, from the moment the software engineer starts interacting with the existing classes. The software engineer finishes the task creation by naming the new task.

Although Mylyn deals with software engineering field, it has two major drawbacks. The first one is the lack of support for other phases of the software process. As Mylyn is aimed at providing help for software development, other phases, such as use case description or definition of test cases, do not benefit of Mylyn's features. The second drawback relates to the disregard of the software process. It is said that Mylyn is not process-based. When taking the software process into consideration, the task creation can be guided and each task context can be initially inferred.

3 MylynSDP

3.1 Overview

MylynSDP is an Eclipse plugin whose objective is to help software engineers in the search of software artifacts during a software process execution. The name MylynSDP comes from the fact that it is an extension of Mylyn project intended to work for software development processes. MylynSDP aids software engineers by scoring the im-

portance of each artifact based on (i) the software engineer's interaction with them and (ii) their relationship with software process' activities, which is presented in the software process specification. Once each artifact has its interest value set in relation to each activity, MylynSDP's DOI function can group artifacts, thus creating a context for each activity, and filter out low scored artifacts from the software engineer's view.

Each unit of work in a software process is called an activity. In MylynSDP, each software process' activity is transformed into a task. Thus, a task is nothing more than an instance of an activity. This allows a software engineer to re-execute an activity without losing the information of a previous execution. The same approach is used between software process' artifacts and MylynSDP's artifacts, but both are called artifacts. From this point in this paper, this naming convention will be used.

Figure 1 shows the initial view of MylynSDP. It has three main views: the Artifact view (Figure 1-A), the Task view (Figure 1-B) and the Working Area view (Figure 1-C). The Artifact view, on the left, is named "Package Explorer" on Figure 1. It is the place where all artifacts from one project are displayed to the software engineer. By browsing it, the software engineer may select or open artifacts. If a task in currently in execution, the Artifact view will only display artifacts that belong to that task context. The Task view, or "Task List" on the right column of Figure 1, stores all tasks created by the software engineer. By clicking in a small rounded button next to each task, the software engineer is able to start and finish the execution of a task, as well as change the task being executed to another. In the middle, there is the Working Area view, where artifacts can be manipulated during the execution of a given task. It currently displays the contents of an empty Glossary file.

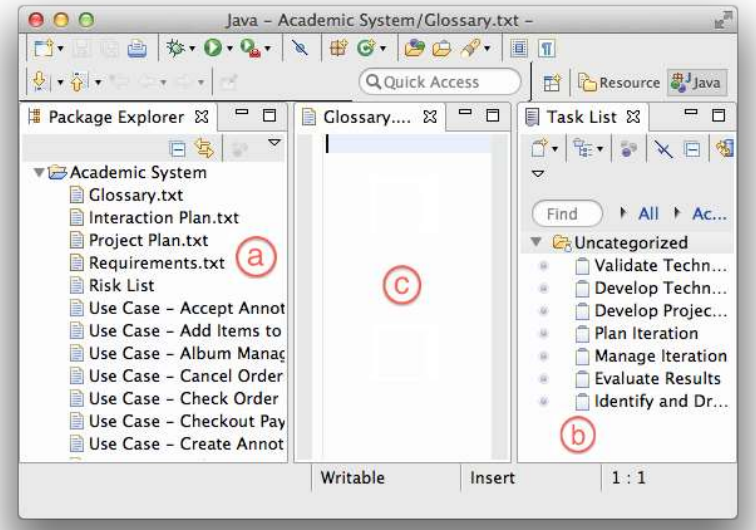

Fig. 1. MylynSDP's initial view. Artifacts are displayed in the left and tasks are displayed on the right. Once a task is selected, artifacts are filtered. The central area is the working area

In order to (i) associate each artifact to the suitable task, according to the software process and (ii) to allow MylynSDP's DOI function to infer an initial context for each task created, MylynSDP features a new type of interaction: the specification interaction event. This interaction is automatically performed by MylnSDP's DOI function in three situations. The first is during the creation of an artifact when a task is being executed and that artifact, according to the software process, is supposed to belong to that task context. The second situation is also at the creation of an artifact, but when the new artifact does not belong to that task context, according to the software process. In both situations, MylynSDP's DOI function performs a specification interaction event and adds the new artifact to the current task context. The third situation is at the creation of a task. When a task is created, MylynSDP's DOI function consults the software process and checks what artifacts should initially belong to that task context. Next, it scans all artifacts already created and automatically performs a specification interaction on the suitable artifacts. Thus, each artifact interacted will have its interest value increased. Moreover, at the creation of both an artifact and a task, the software engineer uses a suitable wizard, which helps him to associate the new task with its activity and the new artifact with the software process artifact that based its creation.

3.2 DOI function

DOI function is the mechanism that calculates interest values for all artifacts in relation to activities, which allows artifacts to be grouped into task contexts. These sets of relevant artifacts then are displayed to the software engineer as the most important ones, whereas other artifacts are hidden from his view.

An artifact's interest value is calculated as shown in MylynSDP's DOI function algorithm in Figure 2. There are three main steps: the event registration, the addition and the subtraction steps. First, each interaction event occurrence that aims a particular artifact is registered in a variable that has the name of the interaction. This step is relatively simple and it was omitted from Figure 2 due to space limitation. Whenever an artifact's interest value is necessary, "getValue()" method, on line 1, is called. The addition step is represented by "getEncodedValue()" method on line 8. MylynSDP's DOI function multiplies the number of each interaction event's occurrence ever performed to that artifact to its score (lines 10, 11 and 12). The result is then saved into "value" variable. The "specificationBias" variable is a zero-or-one value that represents if that artifact belongs to the actual task context. If so, "value" variable is increased by 5 points (line 14). The third and final step is related to the decay value, which represents how much uninteresting an artifact is to the current task context. This is a subtraction step. DOI function calculates the difference between (i) the ordinal number associated with the interaction event that created the artifact and (ii) the ordinal number related to the last interaction event. The result is then multiplied to its score (lines 24, 25 or 27, 28). If the current task context's numbers of occurrence of interaction events are below a given initial threshold (line 22 and 23), the decay value

then returned. Otherwise, half of the decay value's original value is returned. This is done in order to better filter out unnecessary artifacts from the software engineer's view since the start of MylynSDP's usage.

```
public float getValue() {
    float value = getEncodedValue();
    value += predictedBias;
    value += propagatedBias;
    return value;
}

public float getEncodedValue() {
    float value = 0;
    value += selections * contextScaling.get(InteractionEvent.Kind.SELECTION);
    value += edits * contextScaling.get(InteractionEvent.Kind.EDIT);
    value += commands * contextScaling.get(InteractionEvent.Kind.COMMAND);

    value += specificationBias * 5;

    value -= getDecayValue();
    return value;
}

public float getDecayValue() {
    if (context != null) {
        if ((context.getUserEventCount() - eventCountOnCreation) <
                initialEventThreshold) {
            return (context.getUserEventCount() - eventCountOnCreation)
                    * contextScaling.getDecay();
        } else {
            return (context.getUserEventCount() - eventCountOnCreation)
                    * contextScaling.getDecay() * 0.5f;
        }
    } else {
        return 0;
    }
}
```

Fig. 2. MylynSDP's DOI function code. During getEncodedValue() method's execution, the interest value of an artifact is calculated, taking into consideration whether the artifact is supposed to be on that context or not. The getDecayValue() method calculates the decay value for that artifact based on how many interaction events happened since the creation of the artifact. After an initial threshold of interaction events, the decay value is reduced to the half of the original one.

Here is an example of how an artifact is created and interacted with. For the sake of simplicity, it is considered that a software process has been imported and a task has already been created. At the beginning, the software engineer creates a brand new artifact using a suitable wizard. That wizard will help him associate the new artifact to a software process specification's artifact. At the moment that the software engineer finishes the creation of the artifact, MylynSDP's DOI function (i) performs a specification interaction event on the new artifact and (ii) it moves the artifact to the current task context. As the execution of the software process continues, the software engineer will eventually select the new artifact. In that case, that artifact will have been targeted

by two interaction events: 1 specification and 1 selection events. At that moment, its interest value will be calculated by multiplying the number of occurrence of each interaction event by its score. A specification event has a score of 5 and a selection event has a score 1. Thus, that artifact's interest score calculation is as follows: (1 x 5) + (1 x 1) = 6.

The decay value has not yet been calculated, though. It can be considered that 10 interaction events happened since the creation of that artifact. Number 10 is arbitrary and chosen for example purposes. As explained, to calculate the decay value of an artifact's interest, MylynSDP's DOI function multiplies the number of interaction events since artifact's creation to a decay constant, which is 0.4. Thus, the decay value in that situation is 10 * 0.4 = 4. Final interest value is calculated by subtracting decay value from partial interest value. Therefore, final interest value is 6 - 4 = 2.

The creation of a task starts with MylynSDP's Task Creation wizard accessed by the menu or the "new task" button on MylynSDP's Tasks view. The software engineer then names the task. After it, he must inform the software process activity on which that task was based on. There is a list where he selects the type. Once a task type is selected, it cannot be changed later. After the setting of the type of that task, MylynSDP's DOI function consults the software process searching for the activity that based the creation of the new task and then checks the context of that activity, i.e., the types of artifacts that are supposed to be used in that activities execution. Thereafter, MylynSDP's DOI function scans all software process' execution artifacts searching for artifacts whose type matches that activities' context. At each artifact found, the DOI function performs a specification interaction event in order to add that artifact to the new task's context. After scanning all artifacts, the new task creation is finished and an initial task context for the new task is set.

4 Validation Study

4.1 Overview

In order to assess the feasibility of MylynSDP in a software engineer's work, a validation study has been conducted. It was decided to run the validation study under a simulated environment due to the complexity and the time needed to observe the execution of a software process and the usage of MylylnSDP, along with its DOI function, in a real environment. However, the software process chosen to the simulation was a real one, and the participants were experienced in the software process field. Details of the execution of the validation study are described in the next paragraphs.

The validation study consisted of two major stages: the simulation and the questionnaire. The simulation stage started with a presentation with some instructions. Participants were taught the concepts of MylynSDP and how to operate it. They also learned about the nature of the software process and its naming conventions, types of activities and artifacts. Once finished, participants simulated the execution of a software process by executing five exercises. Table 2 shows the objective of each of the exercises.

The questionnaire stage, as the name implies, demands participants to answer a 12 items questionnaire. The questionnaire comes from Technology Acceptance Model (TAM) [4], which analyses a given technology under two perspectives: usefulness and ease of use. The questionnaire participants answered is comprised of 12 statements, shown in Table 3, and 7 possible answers (I completely disagree, I partially disagree, I slightly disagree, I do not agree, nor disagree, I slightly agree, I partially agree, I completely agree). At the end of the questionnaire, there was a free commentary area on which participants were able to justify their answers, criticize the concepts or suggest modifications.

Table 2. The 5 exercises of the validation study alongside with their objective. Each exercise was designed to observe participants when dealing with a particular situation

Exercise	Objective
Exercise 1	Observe participants when dealing with low DOI function's filtering due to the initial usage of MylynSDP
Exercise 2	Observe participants' performance when dealing with advanced filtering by the DOI function
Exercise 3	Observe participants' reactions when an artifact is needed during a task execution but it does not belong to that task context because of modeling error
Exercises 4 & 5	Observe how participants interact with tasks and artifacts during a context change. During the execution of exercise 4, they were interrupted with a high-priority exercise 5.

Table 3. Each of the twelve statements presented in the TAM questionnaire. Participants were asked to answer them with one of the seven possible answers ranging from "I completely disagree" to "I completely agree"

Number	Statement
Statement #1	Using MylynSDP's DOI function in my job would enable me to accomplish tasks more quickly.
Statement #2	Using MylynSDP's DOI function would improve my job performance.
Statement #3	Using MylynSDP's DOI function would increase my productivity.
Statement #4	Using MylynSDP's DOI function would enhance my effectiveness on the job.
Statement #5	Using MylynSDP's DOI function would make it easier to do my job.
Statement #6	I would find MylynSDP's DOI function useful in my job.
Statement #7	Learning to operate MylynSDP's DOI function would be easier for me.
Statement #8	I would find it easy to get MylynSDP's DOI function to do what I want it to do.
Statement #9	My interaction with MylynSDP's DOI function would be clear and understandable.

Statement #10	I would find MylynSDP's DOI function to be flexible to interact with.
Statement #11	It would be easy for me to become skillful at using MylynSDP's DOI function.
Statement #12	I would find MylynSDP's DOI function easy to use.

Invitations to take part in the validation study were sent to 1 MSc and 6 PhD Software Engineer students who are enrolled in the Software Engineer course at PESC/COPPE in Federal University of Rio de Janeiro, Brazil. They all have dealt with software processes during their academic or professional life.

The software process used in the validation study comes from SIGA-EPCT project. This projects aims at managing academic information from public universities in Brazil. Data collected and managed by SIGA-EPCT range from information about students, professors and subjects to enrollments, universities and teaching rooms. The software process dealt with the specification and creation of use cases, interfaces, class diagrams, test cases and data base scripts. More than 250 artifacts were available to use in the execution of 10 activities.

4.2 Analysis

Percentages of answers were represented in a 3D chart, which is shown in Figure 3.

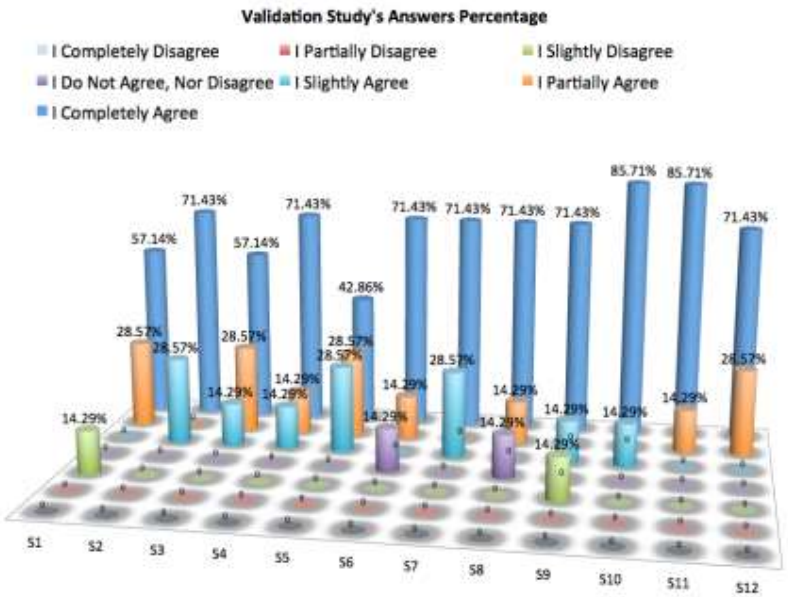

Fig. 3. The percentage of answers for each questionnaire's statement. Statements are represented by the letter "S" followed by its number.

It can be seen that the overall acceptance of MylynSDP's concepts is high as the "I completely agree" answer's occurrence was more than 50% for almost all of the statements. The only exception was statement #5, which also features a high percentage of "I agree" answers. Although MylynSDP was highly accepted, some points worth mentioning.

Statements #6 and #8 have received "I do not agree, nor disagree" answers. The participant who evaluated statement #6 with that answer, justified his opinion with a commentary at the end of the questionnaire which says that he does not work with software development anymore, and thus he could not evaluate the application of MylynSDP's concepts in his own work. Another participant also evaluated statement #8 with the same answer as statement #6. This participant did not consider MylynSDP along with its DOI function to be easy to do what he wanted, or controllable. It should be pointed out that this participant had problems with Mac commands such as scrolling, minimizing and closing documents, which could slightly affect his performance.

Statements #1 and #9 have been evaluated with an "I slightly disagree" answer. Statement #1's agreement was high, which makes this low score an outlier. The participant did not leave a comment to explain his answer. The same situation happened with the participant who scored statement #9 with a low score. Although he did not leave any commentary, a suggestion was made. He suggested that more ways to improve the search of artifacts should be implemented, as for example, the use of keywords.

4.3 Threats to Validity

Although it could be observed that, according to the validation study, software engineers accept MylynSDP's concepts, it should be pointed out that the study was conducted with only 7 participants. This number is low for a complete study and this represents a threat to the validity of the study. Moreover, it was the first time that participants dealt with this software process. The training before the study was performed to give a better explanation of the project and the software process. However, it is known that no previous experience with the project may affect the performance of the participants.

Several participants complained about the naming convention used for artifacts of the project. They did not consider the name of the artifacts intuitive. There was nothing to be done to mitigate this threat to validity because artifacts used in the software process simulation were the same used by real workers during the real execution of the process.

The simulation of the execution of the software process aided by MylynSDP and its DOI function was performed on an iMac. Six out of seven participants were not familiar with Mac operating system and its interfaces. Some of them had minor problems with scrolling, minimizing, closing documents and double clicking. It is believed,

though, that these issues have not negatively affected the overall's performance of the participants.

5 Conclusion

In this paper, we have noted two major problems that negatively affect a software programmer's productivity: the search for artifacts and the context change problem. Both problems require the software engineer to spend more time and effort on the search task than the task he was expected to be performing, thus reducing his productivity. The solution proposed was the utilization of a Degree of Interest (DOI) function. It scores elements based on a predefined rule and was able to score artifact's interest in relation to the task being performed based on the software engineer's interaction with the artifacts.

A DOI function has been implemented in Mylyn, an Eclipse plugin aimed at helping Java programmers to search Java classes better within a huge pool of classes, packages and projects. Mylyn's DOI function was limited to the coding phase of a software process and was not able to infer task contexts because it did not considered the software process as one of the elements capable of helping the programmers.

Then, Mylyn's DOI function was extended in order to recognize the underlying software process and to support software engineer's tasks throughout the whole execution of the software process. The final implementation was named MylynSDP. A validation study was conducted which showed a high overall acceptance of MylynSDP's concepts among experienced software engineers.

However, two limitations or improvements may be highlighted. MylynSDP's log file does not easily provide useful information about the stream of events happened during the execution of a software process. Such information may be used to investigate the way the software process execution tasks and artifacts are being used and it is useful for insights about forms in which a software process execution can be improved.

In addition to it, MylynSDP's DOI function aids the work of one software engineer at a time. The development of a system, nevertheless, is generally a collaborative work between groups of software engineers. Research in collaboration area for software processes have been performed [9] and the support for collaboration could be profitable when joined with MylynSDP's concepts.

References

1. Aalst, W.M.P.: Trends in Business Process Analysis: From Verification to Process Mining. In: International Conference on Enterprise Information System. Vol. AIDSS. (2007) IS13–IS22
2. Arbaoui, S., Derniame, J.C., Oquendo, F., Verjus, H.: A Comparative Review of Process-Centered Software Engineering Envirnments. In: Annals of Software Engineering, Vol. 14. (2002) 311–340

3. Barjis, J.: The Importance of Business Process Modeling in Software Systems Design. In: Science of Computer Programming, Vol. 71. (2008) 73–87
4. Davis, F.D.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. In: MIS Quarterly, Vol. 13. (1989) 319–340
5. Dourish, P., Edwards, W.K., LaMarca, A., Salisbury, M.: Using Properties for Uniform Interaction in the Presto Document System. In: Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology. (1999) 55–64
6. Dragunov, A.N., Dietterich, T.G., Johnsrude, K., Mclaughlin, M., Li, L., Herlocker, J.L.: TaskTracer: A Desktop Environment to Support Multi-Tasking Knowledge Workers. In: Conference on Intelligent User Interfaces. (2005) 75–82
7. Fuggetta, A.: Software Process: A Roadmap. In: Proceedings of the Conference of the Future of Software Engineering. (2000) 25–34
8. Fuggetta, A., Ghezzi, C.: State of the Art and Open Issues in Process-Centered Software Engineering Environments. In: Journal of Systems and Software, Vol. 26. (1994) 53–60
9. Grambow, G., Oberhauser, R., Reichert, M.: Enabling Automatic Process-aware Collaboration Support in Software Engineering Projects. In: Proceedings of the 6th International Conference on Software and Data Technologies, Vol. 303. (2013) 73-88
10. Gruhm, V.: Process-Centered Software Engineering Environments: A Brief History and Future Challanges. In: Annals of Software Engineering Vol. 14. (2002) 363-382
11. IBM: IBM Rational Unified Process (RUP). (2013) retrieved from http://www-01.ibm.com/software/awdtools/rup.
12. Kaptelinin V.: UMEA: Translating Interaction Histories into Projects Contexts. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. (2003) 353–360
13. Kersten, M., Murphy, G.C.: Mylar: A Degree-of-Interest Model for IDEs. In: Proceedings of the 4th International Conference on Aspect-Oriented Software Development. (2005) 159–168
14. Kersten, M., Murphy, G.C.: Using Task Context to Improve Programmer Productivity. In: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. (2006) 1–11
15. Matinnejad, R., Ramsim, R.: An Analytical Review of Process-Centered Software Engineering Environments. In: Proceedings of the 19th IEEE International Conference and Workshops on Engineering of Computer Based Systems. (2012) 64–73
16. Murphy, G.: Attacking Information Overload in Software Development. In: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing. (2009) 4
17. Reis, C.A.L.: Uma Abordagem Flexível para Execução de Processos de Software Evolutivos. Doctoral Thesis, Rio Grande do Sul Federal University
18. Scacchi, W.: Process Models in Software Engineering. In: Marciniak, J.J. (eds.): Encyclopedia of Software Engineering. 2nd edn. John Wiley and Sons, Inc, New York (2002)