Journal of the
Brazilian Computer Society

RESEARCH **Open Access**

MylynSDP — Process - aware artifact filtering based on interest

Ivens Portugal[1,2*], Toacy Oliveira[1,2], Paulo Alencar[2] and Donald Cowan[2]

*Correspondence:
iportugal@uwaterloo.ca
[1]Federal University of Rio de Janeiro, Av. Pedro Calmon, 550 - Cidade Universitária, Rio de Janeiro, 21941-901, Brazil
[2]University of Waterloo, 200 University Avenue West, Waterloo, N2L 3G1, Canada

Abstract

A software development process is used by software engineers to guide their activities during all phases of the software product development. When executing a software development process, software engineers may lose time and effort while searching for artifacts or changing contexts. This happens, for example, when they need to search for a specific code file in a list of hundreds of files or when they interrupt an activity to execute another but forget specific details and need to re-execute searches related to the previous activity. This impacts their productivity negatively, because extra time and effort are spent into non-productive work. Therefore, automated assistance is required to mitigate or avoid these issues. The Degree of Interest (DOI) function infers an element's importance in a context, helping software engineers to handle many artifacts. Mylyn, an Eclipse IDE plugin, uses a DOI function on Java documents to assist programmers when looking for code documents during development. However, Mylyn's DOI function is limited to the implementation phase of software processes and relies on manual task creation. This paper presents MylynSDP, a software Process-aware extension to Mylyn's DOI function. MylynSDP's DOI function infers an artifact's importance during an activity and filters uninteresting artifacts, reducing the time taken to search items and improving productivity. Mylyn code was augmented, and an evaluation study was performed. Seven subjects executed a software process with many artifacts. Exercise times were recorded for productivity analysis. Subjects answered a Technology Acceptance Model (TAM) questionnaire. New task and artifact creation wizards link tasks and artifacts to specification activities and artifacts, respectively. A new interaction event handles context creation, and the DOI function was extended to other software process phases. Exercise time reduction shows a productivity increase. TAM questionnaire answers show a positive overall willingness to adopt MylynSDP and provide evidence that using a DOI function in different software process phases increases productivity. This work advances the state of the art in software engineering by providing additional methods to support artifact search and discovery, context change management, and artifact relevance mechanisms.

Keywords: Software development process, Software process specification, Software process execution, Mylyn, DOI function, Task context

Introduction

Software development is a complex task. Besides implementing the logic that governs the function of computer software, there are many other challenges including incomplete user requirements, changes in specifications, and inter- and intra-team communication issues [15, 19, 58]. These challenges contribute to the likelihood of errors and, as a result, increase the chances of software failure. Depending on the software, a failure can be costly and even life-threatening [1, 21, 64]. For these reasons, both academic and business organizations focus on avoiding software failures. Research on avoiding errors in software has shown that the use of a software development process usually leads to improvement in the quality of the final software product [5, 22, 59].

Software development processes originated from the notion of the software lifecycle, which was a concept from the 1960s and 1970s that described the lifetime of a software project [23]. The main objective of the software development process is to guide the work of software engineers during the development of a software product towards the improvement of the quality of the final deliverable [46]. This result is achieved by defining a common set of policies, structures, procedures, activities, and artifacts that must be followed or manipulated when developing software [23].

Once a software development process is specified, the software engineers effectively perform the activities of the process following a given order and select, open, read, edit, and close suitable software artifacts [8, 20]. Software artifacts are documents manipulated to complete the execution of an activity, such as requirement specifications, bug reports, use cases, or source code [6, 9, 28]. Depending on the organization and the way it manages software processes, software artifacts can be located in the file system or in a shared repository [45]. Software engineers have several ways to search and access artifacts. The most common ways are to search manually, (i) by looking into folders in a hierarchical structure and opening documents as needed; (ii) by doing a query search using natural language [63] or a regular expression [26], and browsing through the search result; or (iii) by skimming through a summary of the software artifacts, which is most useful for documentation artifacts [41]. Software artifact searching is a significant concern among practitioners and researchers because as the software process advances, the number of software artifacts increases [2, 39, 53]. As an example, the Rational Unified Process (RUP)[1] specification describes more than 100 types of artifacts, which means that software engineers executing a RUP-based software process would normally deal with a large number of artifacts, especially in the latter half of a software project.

Managing several software artifacts during the execution of a software process increases the complexity of the process and is likely to lead to more errors as the software engineers could become confused [43]. For example, a software engineer that needs to access a test case document and a requirement document to validate a requirement may find it difficult if he or she has to browse manually through hundreds, perhaps thousands, of software artifacts. Even if an automatic search can be performed, some complications may arise: documents may have changed names, or been updated to a new version, leading to searches without meaningful results.

Three problems can be identified with respect to software artifact management. First, searching and accessing software artifacts affect a software engineer's productivity

[1]http://www.ibm.com/software/rational

because effort is being expended on a secondary task, rather than focusing on the primary activity of creating error-free software. The artifact search problem is a major problem faced by software engineers as it is error-prone and can be time-consuming [2]. Generally, software engineers perform a search for suitable artifacts to execute a particular activity. The set of suitable artifacts is, in most cases, a subset of all available software artifacts. This subset of relevant artifacts for the execution of an activity is defined as the context of that activity [32, 35].

Second, in addition to the artifact search problem, software engineers frequently face the context change problem [40, 48]. This problem arises when they need to interrupt the execution of an activity and start a new activity probably with a higher priority leading to a context change. During a context change, a software engineer often forgets the details of the first activity because of the interruption. Moreover, he or she will need to perform another search for suitable artifacts when he or she decides to return to the first activity.

A third problem relates to relevance of an artifact to a given activity when software engineers handle several software artifacts. It is difficult to define the relevance or importance of an artifact to the execution of an activity to build support for software engineers [18, 42] as each activity has its own set of suitable artifacts, and each of these artifacts evolves with time, by having its name changed, or being split, merged, or even deleted.

These three issues of searching, context change, and relevance lead to the following set of research questions (RQs):

RQ 1. How can a search for software artifacts be supported to maximize a software engineer's productivity?

RQ 2. How can context changes be supported while maximizing a software engineer's productivity?

RQ 3. How can an approach be defined that addresses the relevance or importance of software artifacts?

In this paper, we propose a mechanism based on artifact relevance to aid a software engineer's productivity in artifact search and activity context change. Here, one's productivity is a function of the number of artifacts produced and the time. Improvements to productivity can be done either by increasing the number of artifacts produced at each unit of time or by reducing the time it takes to produce a given number of artifacts. The mechanism described in this paper chooses the latter way. The proposed relevance mechanism consists of an extension of Mylyn's [30] DOI function, is called MylynSDP, and allows the DOI function to deal with software process phases other than implementation. An evaluation study has been performed to assess the concepts discussed in this paper.

Mylyn is an Eclipse plugin that assists programmers by filtering out less important Java code documents based on their interaction with the files during the execution of a task. The more a Java file is used in a task, the higher its likely relevance. However, Mylyn's DOI function has some limitations. As the function was designed for the implementation phase of a software development process, it does not take into account details found in artifacts used at other phases of software development, such as modeling or testing. Further, because Mylyn is restricted to the implementation phase, it is not possible to recommend artifacts in all other phases of software development processes. Therefore, programmers lack support to assist them in finding artifacts of interest in these remainder phases. Another limitation is that tasks must be manually defined, since there is no underlying software process to support task creation. Mylyn has some restrictions in this regard, including (i) an explicit software development process based on activities and artifacts is not supported by the tool; (ii) the initial contexts of the tasks are not explicitly defined; (iii) the DOI function cannot take advantage of the explicit relationships between activities, artifacts, and initial task contexts; and (iv) programmer productivity can be significantly compromised when the DOI function cannot automatically support process-related activities (e.g., task creation, artifact search).

MylynSDP's DOI function expands the task × Java code document relationship with the activity × artifact one. Based on the original Mylyn's DOI function, the new MylynSDP's DOI function assigns an interest value with each artifact associated with an activity. For each activity, MylynSDP then filters out uninteresting artifacts based on their interest value. Thus, the new DOI function makes the most interesting elements for an activity easily accessible for software engineers. The main difference from the original function is that the new DOI function has the ability to calculate each artifact's interest value and filter them based on the software development process and on the interactions with the artifacts performed by the software engineer. It should be noted that an artifact can be related to different activities and can have a different interest value for each associated activity. The MylynSDP's DOI function calculates the interest value according to the activity in execution.

Similar to the original function, MylynSDP's DOI function saves the context of an activity (i.e., the activity itself and its related artifacts). By saving the context, MylynSDP's DOI function aids software engineers when a context change takes place. Later, the previous activity context can be retrieved quickly and effortlessly, which allows software engineers to continue their work from the point where they had stopped. The use of a DOI function can help software engineers to be more productive by allowing them to focus on effective work and not on additional and unnecessary work such as searching for suitable artifacts [31].

This paper is organized as follows. The "Related work" section presents related work. In the "MylynSDP" section, MylynSDP and the new DOI function are introduced, described, and explained. The "Evaluation study" section contains the details of the case study. The "Conclusion and future work" section concludes this work and discusses future work.

Related work

The approach described in this paper is related to three research areas: (i) artifact search and discovery, (ii) context change management, and (iii) artifact relevance mechanisms. To create MylynSDP and its DOI function for these areas, our research investigates how frameworks deal with artifacts. Descriptions of related research are discussed in the next subsections.

Artifact search and discovery

A useful approach for discovering artifacts related to an activity is their traceability. Generally, the main objective of a traceability approach in software engineering is to analyze change, maintenance, and evolution effects that could happen to a software product during its lifecycle [60]. By doing this analysis, software engineers aim to improve the quality of the final software product.

Traceability is also important in comparing new and known software requirements, aiding in artifact reuse, and serving as a basis for software testing and inspection [60]. Thus, artifact traceability may be used for clear communication between users and developers, as well as for improving documentation and increasing the chances of software acceptance [60]. According to [61], the best way to improve software artifact traceability is by using a traceability matrix to represent the relationship between activities and artifacts. Nevertheless, the use of such a traceability matrix has disadvantages. First, the spreadsheet that holds the information is usually created manually, which often requires a substantial amount of time. Second, minimal computer support in creating the matrix increases the chances of failure or inaccurate information. Third, gathering the data in the matrix, even when automated, may require considerable time given the sheer number of artifacts and activities in a software process. Finally, software process flexibility indicates that new artifacts may be created or activities may be re-executed, thus adding new information thereby requiring matrix reprocessing and additional time [60].

A second approach is the use of a process-centered software engineering environment (PSEE). Some research effort in the 1980s focused on the specification of a public interface to be used as the basis for the construction of software engineering environments (SEE). Projects such as the Portable Common Tool Environment (PCTE) [10] were created as a way to reduce the cost of building software engineering tools. Since then, research in the area resulted in the construction of many different software engineering tools, including process-centered software engineering environments. A PSEE is formally defined as a software engineering environment in which there is an explicit definition of the process to be followed during software development [24]. PSEEs usually allow the modeling and execution of software processes, as well as their improvement. The use of a PSEE also enhances the communication between stakeholders, the reuse of some parts of the process, the automatic collection of data to generate reports, and the control and improvement of software processes [3, 38]. Thus, the research focused on looking for suitable PSEEs that could help software engineers searching for a large number of artifacts.

WebAPSEE [54] is a PSEE whose objective is to provide automatic support for software development process management. The PSEE maintains three key structures: a modification control module to register all changes to the process being executed, an artifact meta-data repository, and an artifact versioning repository. WebAPSEE allows software engineers to upload artifacts to its repository and set which artifacts will be visible to which member of the software development team. By doing this, WebAPSEE simplifies the work of some software engineers dealing with software process execution by avoiding the display of an excessive amount of information. However, the artifact allocation process is manual, which can take time and be error-prone.

MoDErNE [36, 37, 47] is a PSEE that improves model-driven development (MDD) [57] process specification and enactment. It contains two modules: Process Editor, which uses UML notation for modeling MDD processes, and Process Executor, which executes the MDD process in a semi-automated way, showing task status and displaying reminders to software engineers. Spider-PE [49] is a PSEE that aims to assist software organizations in the implementation of the capability maturity model integration for development (CMMI-DEV)[2]. The project defines its own language for process execution, xSPIDER_ML, and is divided into three components: Management, Process Management, and Process Execution. The first module is responsible for importing an XML file containing the process model. The second module monitors the process and checks whether the process adheres to good practices described in CMMI-DEV. The third module allows software engineers to allocate resources and executes the process. The work in [27] describes a change-aware PSEE aimed at managing changes systematically. The project achieves that by introducing three constructs: a Process Dependency Graph (PDG), a Change Observer process, and a Change Analyzer component. The first one is used to represent dependencies among running processes instances. The second one identifies changes and updates the PDG. The third one reasons on the PDG and extracts the impacts of these changes. Note that, in all three approaches, artifacts are manually assigned to activities during process modeling, which is time-consuming and highlights the lack of an automated approach.

Context change management

One project dealing with the management of document contexts in an activity-centric manner is the Presto Project [16], from the Xerox Palo Alto Research Center (PARC). The underlying assumption is that the structures of the hierarchical schemes used to organize documents are rigid and not suitable for the more fluid nature of everyday practices. Therefore, scientists at Xerox PARC came up with a new approach to document management that aims to provide users with new ways of organizing, structuring, managing, and interacting with document collections. The focus is not where in the system the document is located, but to what it is related, such as projects, tasks, and meetings. The goal was to build a document system organized around document properties, so that users could group documents associated with a particular event based on the context of an activity. The main drawback of this approach is that the system required people to categorize their entire collection of files manually.

TaskTracer [17] is a system aimed at categorizing user's events while they execute some general tasks. The system then uses this data to build a profile for the task. The main goal is to help workers during the process of interruption discovery, which is the moment an interruption occurs in the execution of a general task because of a switch between tasks. TaskTracer adopts the idea that workers organize their work into discrete units, usually called tasks. Therefore, TaskTracer collects information about user interaction with Microsoft Office, Visual Studio, and Internet Explorer applications during an ongoing task and builds a task profile. When users return to a task they have previously interrupted, TaskTracer can restore all applications being used for that task. However, at the initial stage of data collection, users have to specify manually which tasks they are doing, meaning that there is not an automated support for software development processes and their activities.

The UMEA [29] scheme uses a project-centered approach to help workers in the retrieval of suitable documents, as well as helping with the context change problem. UMEA, which stands for User-Monitoring Environment for Activities, makes a clear distinction between the file system, a hierarchically organized storage system of information

[2]https://cmmiinstitute.com/cmmi/dev

and documents, and the desktop, a workspace where documents and applications necessary to accomplish a task can be placed to make them easily accessible. While taking this organizational principle into consideration, UMEA creates the concept of project spaces, where documents related to a particular project are placed to separate them from non-relevant documents. Project Spaces, which are the same concept as activity contexts, are automatically built based on user interactions with Microsoft Office documents. The history of interactions is saved, so that UMEA can manage which files are related to a particular project. Once a user needs to change projects, all open windows are saved; thus, the user can continue to work from the point where he or she left. UMEA's approach to the management of task contexts is similar to TaskTracer's approach. The main difference is that UMEA can work in a background mode, which is transparent to the user. The main deficiencies of the UMEA software are that projects, which are similar to activities, must be manually defined, and UMEA is limited to Microsoft Office documents.

Artifact relevance mechanism

The work in [62] describes the use of regression analysis [12] to identify domain-dependent predictors of file importance. The authors retrieve common files used in an academic environment (e.g., homework assignments) and use domain information (e.g., due date) to predict the importance of the file. Although the regression function correctly identifies homework assignments close to the due date as the most important ones, the choice of these domain-dependent predictors is strongly dependent on domain experts, which hinders the development of an automated approach.

The work in [35] describes a method called FDA to calculate an artifact's relevance with respect to an artifact context. The name of the method is based on the three parameters that are used in the calculations: frequency, duration, and age. As expected, frequency relates to the rate of use of an artifact based on the number of interaction events, duration describes the amount of time in seconds that all interaction events took, and age is the time passed since the last interaction with artifact a. When using FDA, the relevance of an artifact a for a task t is calculated as shown in Eq. (1), where i is the type of interaction events. One major drawback of this method is the use of the age in the denominator. Depending on the task, some artifacts such as diagrams may be used for inspection and have no interaction event detected. After some time, the denominator will be so large that the artifact may be considered less important, even though it is not.

$$Rel(a,t) = \sum_i \frac{Frq(a,i) \cdot Dur(a,i)}{Age(a,i)} \tag{1}$$

Degree of Interest (DOI) trees [11] are used in an Attention-reactive User Interface (AUI) to calculate the user's interest in the items being displayed. The assumption is that, for many tasks, information can be structured hierarchically and therefore can be displayed using a tree, where tree nodes are data items (e.g., nested folders in a file browsing application). The authors hypothesize that some data items are on the focus of the user (e.g., when the item is selected) and should be displayed, whereas other data items can be hidden. This idea is based on the combined use of Focus and Context Trees described in [25]. The work in [25] calculates the interest value of a data item using Eq. (2). In the equation, the *Intrinsic Importance* of a data item present in a tree node is the distance from that node to the tree root and the *Distance from a focus node* is the number of tree nodes that should be traversed in a path from the node with the item to the node with the focus item. The difference between the trees described in the two works is that a DOI tree considers that data items present in sibling nodes, i.e., they have the same distance from a focus node and the same parent, can be ordered and, therefore, receive a different DOI value based on this order. The concept of a Degree of Interest (DOI) is further explored by the project Mylyn in the domain of software development.

$$\textit{DOI of a node} = \textit{Intrinsic Importance} - \textit{Distance from a focus node}. \tag{2}$$

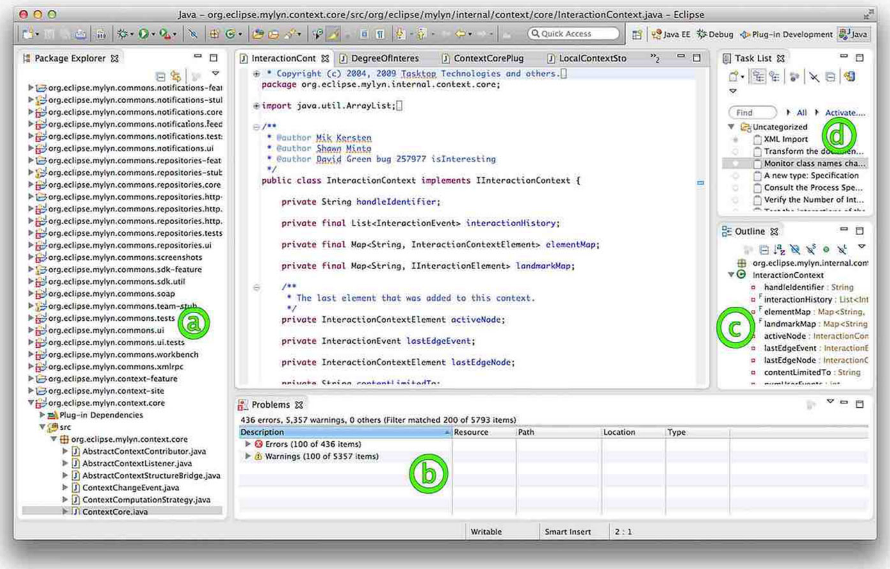
Mylyn [30] is a plugin for the Eclipse Integrated Development Environment (IDE) that helps programmers improve their productivity. Mylyn is the result of research by scientists at the University of British Columbia, in Canada. According to the Mylyn's official page[3], Mylyn is downloaded one million times per month on average, making it the most popular IDE tool for application lifecycle management (ALS). Mylyn's goal is to help programmers focus their work on the code related to a task. This is achieved using Mylyn's Degree of Interest (DOI) function. Mylyn's DOI function filters Java classes, methods, and variables based on programmers' interaction events that indicate how interesting these classes, methods, and variables are to the task currently being executed. In addition, the DOI function saves the context of a task, which allows programmers to continue their work later in the case of a task execution interruption. The Mylyn project was essential to the development of MylynSDP, and so Mylyn's interface and architecture are explained in some depth in the following subsections.

Interface

Mylyn has Eclipse views for storing all Java projects (Fig. 1 (a)), code problems list (Fig. 1 (b)), code outline (Fig. 1 (c)), and available tasks (Fig. 1 (d)). Mylyn starts with the creation of a task. The programmer uses Mylyn's Task Creation wizard and fills in data about the new task such as its name, observations, and deadline. When finished, Mylyn displays the new task on the task view. No initial task context is created. Following task creation, the programmer starts a Java code document creation, by creating a Java class. He or she uses Eclipse's class creation wizard to set the new class name and link it to a suitable Java package. As the programmer interacts with classes, some classes become more interesting than others.

As an example of the use of Mylyn, consider a large software project being developed. Java projects and code files are displayed in the view on Fig. 1 (a). Suppose a programmer needs to code a new functionality. To do this, the programmer invokes a Task Creation wizard and creates a task with a particular title (say "Add new button"). The new task is then displayed in the view on Fig. 1 (d). The new task can be toggled on or off to indicate whether the programmer is working on that task. If a task is active (toggled on), every interaction that the programmer has with the files, such as selecting, opening, or editing a file, is captured by Mylyn. These interactions will later be used to discover the most interesting code files for the task, namely its task context. At some point during development, Mylyn's DOI function filters out some files that it deems not interesting for the current task from the view on Fig. 1 (a). This assists the programmer on focusing only on the most interesting files.

[3]https://www.eclipse.org/mylyn/


Java – org.eclipse.mylyn.context.core/src/org/eclipse/mylyn/internal/context/core/InteractionContext.java – Eclipse
Package Explorer
InteractionCont
DegreeOfInteres
ContextCorePlug
LocalContextSto
Task List
Outline
Problems
a
b
c
d

```
 * Copyright (c) 2004, 2009 Tasktop Technologies and others.
package org.eclipse.mylyn.internal.context.core;

import java.util.ArrayList;

/**
 * @author Mik Kersten
 * @author Shawn Minto
 * @author David Green bug 257977 isInteresting
 */
public class InteractionContext implements IInteractionContext {

    private String handleIdentifier;

    private final List<InteractionEvent> interactionHistory;

    private final Map<String, InteractionContextElement> elementMap;

    private final Map<String, IInteractionElement> landmarkMap;

    /**
     * The last element that was added to this context.
     */
    private InteractionContextElement activeNode;

    private InteractionEvent lastEdgeEvent;

    private InteractionContextElement lastEdgeNode;
```

```
436 errors, 5,357 warnings, 0 others (Filter matched 200 of 5793 items)
Description                         Resource   Path   Location   Type
▶ Errors (100 of 436 items)
▶ Warnings (100 of 5357 items)
```

Fig. 1 Mylyn's interface

Technical details and DOI function

The Mylyn plugin implementation is divided into more than 30 Java projects, each containing more than 10 Java packages and even more Java classes. To simplify the overall understanding of Mylyn, Fig. 2 exposes the main classes and their relationships to one of the key Java projects that constitutes Mylyn. Class properties and methods are omitted for clarity. At the top of the figure, the class `ContextCorePlugin` is the one that specifically deals with Mylyn's ability to handle Java classes, user interactions, and interest values. On the left, the `InteractionContextScaling` class stores the interest contribution that each interaction event will add to a class' interest value at the moment of the interaction. The management of events, classes, and classes' interest values is performed by the `InteractionContextManager` class, which also stores data about task contexts. The class can be seen in the middle of the figure. Below, the `InteractionContext` class represents a task context and it may have one or more `InteractionEvent` classes, i.e., interaction events. Finally, task contexts contain elements (e.g., classes, methods, and variables), represented by the `InteractionContextElement` class, and each of these elements has its own related Degree of Interest function, or an instance of the `DegreeOfInterest` class, displayed at the bottom of the figure.

As mentioned earlier, Mylyn's DOI function calculates an interest value for elements of a Java project by monitoring the programmer's interactions with the code. A class' interest value increases according to the number of interactions the class receives and decreases based on the total amount of interactions performed by the programmer in any class. To calculate the interest value, the DOI function gathers some data about each interaction event performed on the classes, methods, and variables already created. Table 1 shows the main data collected from these interactions.

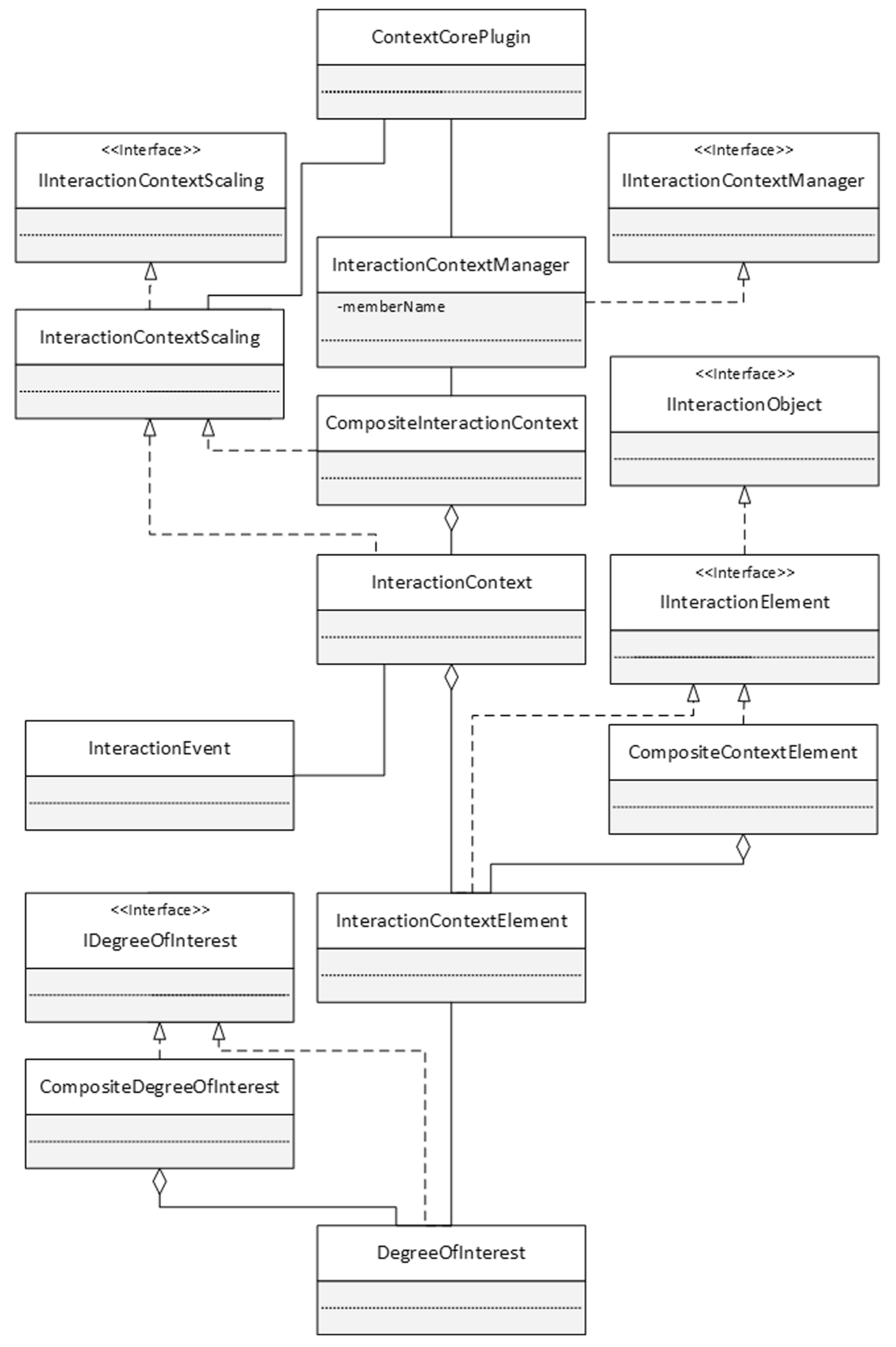


Fig. 2 Mylyn's class diagram

Table 1 Interaction event data. Adapted from [32]

Data	Description
Time	The time when the event occured.
Kind	The type of event occurred (Table 2).
Origin	Identity of the tool that caused the event (e.g., new class wizard, keyboard).
Handle	An identifier for the target element.

Mylyn's DOI function recognizes five types of interactions: Selection, Editing, Command, Propagation, and Prediction. Each is briefly explained in Table 2. Table 2 also shows the scores each interaction contributes to a particular class's interest value.

Mylyn calculates the interest value of a class c according to Eqs. (3) and (4). First, as shown in the summation of Eq. (3), it sums the contributions given by each interaction events e whose target is the class c; then, it subtracts a decay value. This decay value is calculated based on the number of interactions that were performed since the creation of the artifact. Its calculation is shown in Eq. (4). The function $curr\_event()$ returns a number representing the ordinal number of the most recent interaction event, and the function $create\_event(c)$ returns a number representing the ordinal number of the event performed at the creation of class c. As a consequence, the difference of these two numbers is the number of events performed since the creation of class c. The $decay\_constant$ is a weight on this difference that helps calculate the final decay value.

This is a description of how the interest value changes during class creation and manipulation. Table 3 provides a small but correct example of how Mylyn's DOI function calculates the interest value for an arbitrary class called `ClassA`. The first line of the table states that the first interaction was a selection on `ClassA`. The second line of the table represents the next 10 interactions, which were of the type selection performed on classes other than `ClassA`. Note that the class was selected two times (i.e., in interactions #1 and #12). Therefore, its interest value is innitially calculated as 2 selections $\times$ 1 $=$ 2. However, since a class's interest value decreases according to the number of interactions performed over time, a decay value has to be subtracted. In this example, the decay value is (current interation$-$first interaction on class)$\times decay_constant$. Mylyn uses $decay\_constant = 0.017$. Thus, $decay\_value = (22 - 1) \times 0.017 = 0.357$ needs to be subtracted from `ClassA`'s interest value according to Eq. (4). In the end, the class's final interest value is $2 - 0.357 = 1.643$ according to Eq. (3), which is positive, meaning that this class is still interesting for the task being executed.

$$InterestValue(c) = \left(\sum_e count(c,e) \cdot contrib(e)\right) - (decay\_value(c)) \tag{3}$$

$$decay\_value(c) = (curr\_event() - create\_event(c)) \cdot decay\_constant \tag{4}$$

Although Mylyn's DOI function positively affects programmers on the improvement of their productivity (an experimental study can be found in [32]), it has three major disadvantages. The first is that it does not account for characteristics of artifacts used at other phases of software development, including frequency of use or file type; the second is the lack of support for other phases of software development, such as modeling or

Table 2 Interaction event type. Adapted from [32]

Interaction Event type	Contribution	Description
Selection	1.0	Interaction that occurs by selecting files with mouse or keyboard.
Editing	0.7	Interaction that occurs when editing files.
Command	1.0	Operations such as saving or compiling.
Propagation	1.0	Interaction that occurs when other elements are indirectly affected by another interaction event.
Prediction	1.0	Capture of potential future interaction.

Table 3 An example of Mylyn's DOI function calculating the interest value for a class. Most recent interaction at the bottom

Index	Quantity	Event	Target
#1	1	Selection	`ClassA`
#2–#11	10	Selection	Other classes
#12	1	Selection	`ClassA`
#13–#22	10	Selection	Other classes

testing; the third is that tasks may be manually defined. That would not happen if Mylyn's DOI function considered a software development process, since process's activities could be translated into Mylyn tasks. Thus, it can be said that Mylyn's DOI function is not process-aware.

MylynSDP

In the previous section, we have described several related works that attempt to assist software engineers, or just regular users, in dealing the problems related to artifact search, context change, and artifact relevance, which were described in the first section. After each description, we briefly discussed some drawbacks. We now summarize these drawbacks to motivate the introduction of our solution, MylynSDP. All projects show some manual functionality that should be automated. The traceability matrix is manually created, whereas PSEEs and the Presto Project have a manual allocation of artifacts (files) to activities. Activities in TaskTracer, UMEA, and Mylyn are manually defined based on the user's needs at the time of their creation. Lastly, Mylyn focuses on the development phase of software development. We therefore aimed at developing an approach that solves the artifact search, context change, and artifact relevance problems by automatically creating activities and artifacts, defining their relationships (allocation of artifacts to activities), and provides support for other phases of a software development process. Solving these issues was the main driver for introducing our MylynSDP approach described in this section.

Concept description

Overview

In this section, MylynSDP (Mylyn + software development process) [50–52], a process-aware and intent-based artifact filtering approach, is described. MylynSDP is an extension of the Mylyn software and is an Eclipse plugin. MylynSDP, which added new functionality to Mylyn, has as its primary role to help software engineers in any phase of software development in the areas previously described, namely artifact search and discovery, context change management, and artifact relevance. MylynSDP therefore is not restricted to the implementation phase of a software development process (with the search of code files) and is able to assist software engineers in other phases such as requirements and testing (with the identification and search of use case and test case documents). It extends the DOI function and introduces the awareness of a software process. This software process is assumed to be the one the software engineer or the organization is already using. It contains a rich set of information that, when inspected by MylynSDP, will be used to assist software engineering during tasks. In general, MylynSDP works as illustrated in Fig. 3.

After creating activities and artifacts, a software engineer performs the activities by interacting with the artifacts (Fig. 3 (1)). While activities are performed, MylynSDP calculates interest values for artifacts with the DOI function based on the interactions that happen and on the relationship “activities $\times$ artifact” present in the software process (Fig. 3 (2)). Interest values are used to determine which artifacts should be highlighted. Since artifact interest values are activity-specific, i.e., calculated based on the activity being executed, a second software engineer that works on the same project is able to execute a different activity and benefit from MylynSDP features (Fig. 3 (3)). Currently, MylynSDP is intended to be used by software engineers individually, who may share the same workplace. However, there are other initiatives aimed at the study of software artifact manipulation in a collaborative approach [44].

Based on how Mylyn works, MylynSDP associates an interest value for each artifact in an activity context. This value calculation is based on the software engineer’s interaction with artifacts and also on the underlying software process. All information is saved in a separate file for later context reconstruction. The main part of MylynSDP is its DOI function, because it is responsible for interest calculations for each artifact in each activity context. A detailed description is presented in the following paragraphs and sections.

MylynSDP deals with three areas: artifact search and discovery, context change management, and an artifact relevance mechanism. To help in search and discovery, MylynSDP’s novelty is that it bases its execution on the underlying software process to map which artifacts are related to a particular activity, and thus assumes that those artifacts are more interesting when executing that activity. Furthermore, artifacts that are not interesting to the current executing activity might be omitted from the software engineer’s view in a similar way that Mylyn omits Java code documents, helping him or her to focus on more relevant artifacts. MylynSDP’s novelty on context change management is related to introducing and saving a new type of interaction in the history of interactions of a context for future reference and context reconstruction. Thus, every time a software engineer changes context, information about the initial importance of artifacts for each activity of the software development process is saved alongside all information about the software engineer’s interactions. The third area is the artifact relevance mechanism. MylynSDP’s novelty is a new formula to calculate artifact interest values based on the software engineer’s interaction and the underlying software project. Just as in

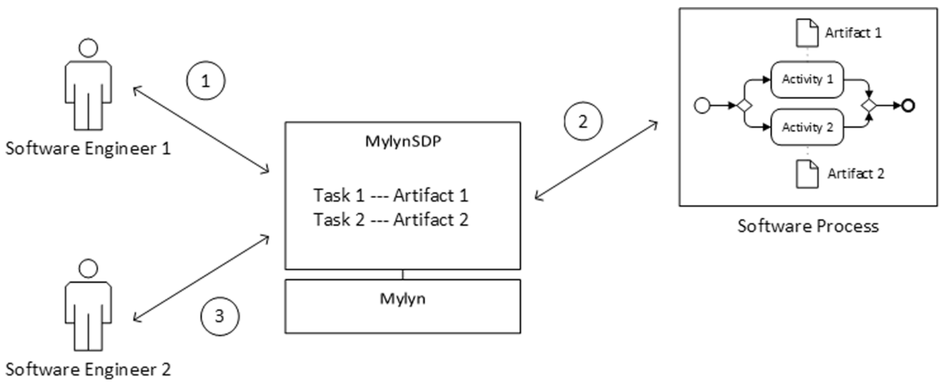


Fig. 3 MylynSDP functionality

Mylyn, the more interaction with an artifact, the larger is its interest value. An artifact's interest value decreases gradually based on the number of interactions that has been performed, especially those whose target is not the artifact in question. In addition, other mechanisms such as the Saving mechanism and the Restore mechanism described in the "Architecture" section help the DOI function to calculate interest values correctly. All of MylynSDP's components are explained in subsequent sections.

During development, some naming conventions have been defined. Software engineers often use the term "activity" to name each unit of work to be performed in the current software process specification. MylynSDP uses the same term to refer to units of work in the *specification* of a software process. However, during the execution of a software process, an activity can be performed more than once. In addition, each activity related to the execution of the software process may have its own set of required artifacts needed to finish its execution. For these reasons, during the *execution* of software processes, activities are instantiated into "tasks." Although an activity can be executed several times, and thus deploy several tasks, a single task belongs to only one activity.

A similar situation happens with artifacts. In an object-oriented system, artifacts represented in a software process *specification* act like a class, whereas artifacts manipulated during the *execution* of software processes perform as instances of this class. Thus, it is said that software process artifacts are *types* of the artifacts available to the execution of the software process. The difference from the "activities × tasks" relationship, though, is that the name "artifact" is used for both elements: the software process specification artifact and the execution artifact.

The MylynSDP interface is divided into three views (Fig. 4): artifact, task, and working area. The artifact view (Fig. 4 (a)) holds all existing artifacts of the current project, regardless of their location in the system. The task view (Fig. 4 (b)) displays tasks to be executed during the software process and includes ongoing tasks, as well as finished tasks. The task view is also where software engineers can start or finish the execution of a task, by clicking a button on the side of the name of the task. Finally, the working area view (Fig. 4 (c)) is where the contents of the artifacts can be created, queried, and edited. There are not major modifications to the views when comparing to Mylyn.

MylynSDP has several steps. It starts by importing a software process specification, which tells MylynSDP the activities and artifacts to be created, as well as their relationship. In our evaluation study, the software process specification used is a BPMN process created in the Bizagi BPMN Modeler[4] and exported as an XML file. Once the specification is obtained, the software engineer is able to create tasks and artifacts based on the activities and artifacts of that specification. While tasks and artifacts are being created, MylynSDP's DOI function grades artifact relevance with regard to a task if the two are related. Finally, the more an artifact is manipulated, the greater its interest for the current task.

As an example of the use of MylynSDP, consider a large software project being developed. Usually, a software development process has been specified with activities and artifacts. Suppose that the first activity on this process is "Create Use Case Documents" and that it consumes artifacts of the type "Use Case Specifications" to produce artifacts of the type "Use Case Documents." This software development process is then imported

[4]https://www.bizagi.com/en/platform/modeler

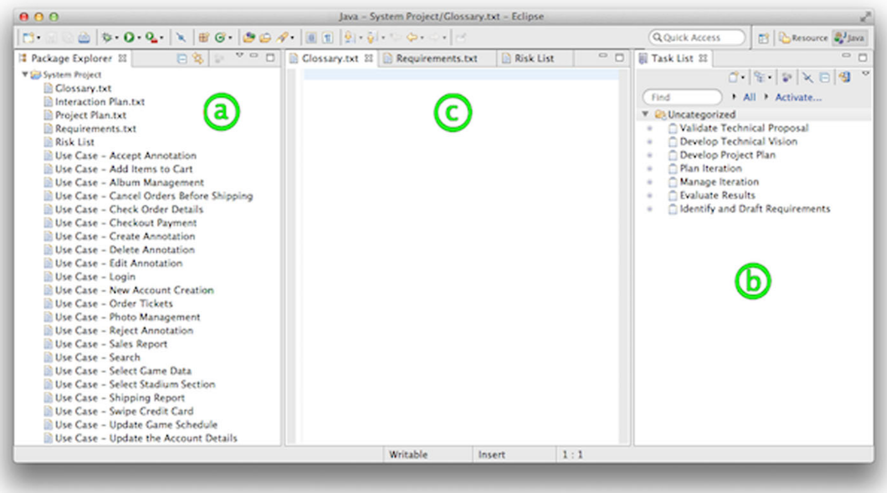

Java – System Project/Glossary.txt – Eclipse
Package Explorer
System Project
Glossary.txt
Interaction Plan.txt
Project Plan.txt
Requirements.txt
Risk List
Use Case – Accept Annotation
Use Case – Add Items to Cart
Use Case – Album Management
Use Case – Cancel Orders Before Shipping
Use Case – Check Order Details
Use Case – Checkout Payment
Use Case – Create Annotation
Use Case – Delete Annotation
Use Case – Edit Annotation
Use Case – Login
Use Case – New Account Creation
Use Case – Order Tickets
Use Case – Photo Management
Use Case – Reject Annotation
Use Case – Sales Report
Use Case – Search
Use Case – Select Game Data
Use Case – Select Stadium Section
Use Case – Shipping Report
Use Case – Swipe Credit Card
Use Case – Update Game Schedule
Use Case – Update the Account Details
a
Glossary.txt
Requirements.txt
Risk List
c
Task List
Find
All
Activate...
Uncategorized
Validate Technical Proposal
Develop Technical Vision
Develop Project Plan
Plan Iteration
Manage Iteration
Evaluate Results
Identify and Draft Requirements
b
Writable
Insert
1 : 1

Fig. 4 MylynSDP's interface

to MylynSDP so that the framework can identify activities and artifacts. After importing, the software engineer can create artifacts and tasks based on the artifacts and activities of the software process. When creating a new artifact or task, the software engineer is queried about the type of the artifact or task, which links it to a process artifact or activity. New artifacts, such as use case documents, code files, or test cases, will be displayed in the view on Fig. 4 (a), and new tasks are shown in the view on Fig. 4 (b). Tasks can be toggled on or off to indicate whether the software engineer is working on it. When a task becomes active (toggled on), MylynSDP and its DOI function can filter out artifacts that are not interesting to the task based on the software process. Further interactions with the remaining artifacts will help shape this task's context. These ways in which MylynSDP is used assist on focusing only on the most relevant artifacts used in specific projects.

Note that the use of MylynSDP during the execution of a software development process is similar to the use of Mylyn during development. The main differences are in the need to import a software process before use, in the association of project artifacts to process artifacts and project tasks to process activities, and in the calculations made by the DOI function.

Context creation

The process of creating task contexts starts whenever a new task is created; this procedure is illustrated in Fig. 5. As previously explained, when a task is created by the software engineer, it is then associated with an activity from the software process specification (Fig. 5 (1)). As a result of that association, the DOI function is able to ask MylynSDP to identify what specification artifacts are related to the activity being analyzed (Fig. 5 (2)). Once the DOI function is provided with this information, it can look for execution artifacts that originated from those specification artifacts (Fig. 5 (3)). This final set of execution artifacts is then considered relevant to the context of the task being created. As a final action, MylynSDP's DOI function needs to increase the interest value of these relevant execution artifacts to make sure they will take part in the context of the task being

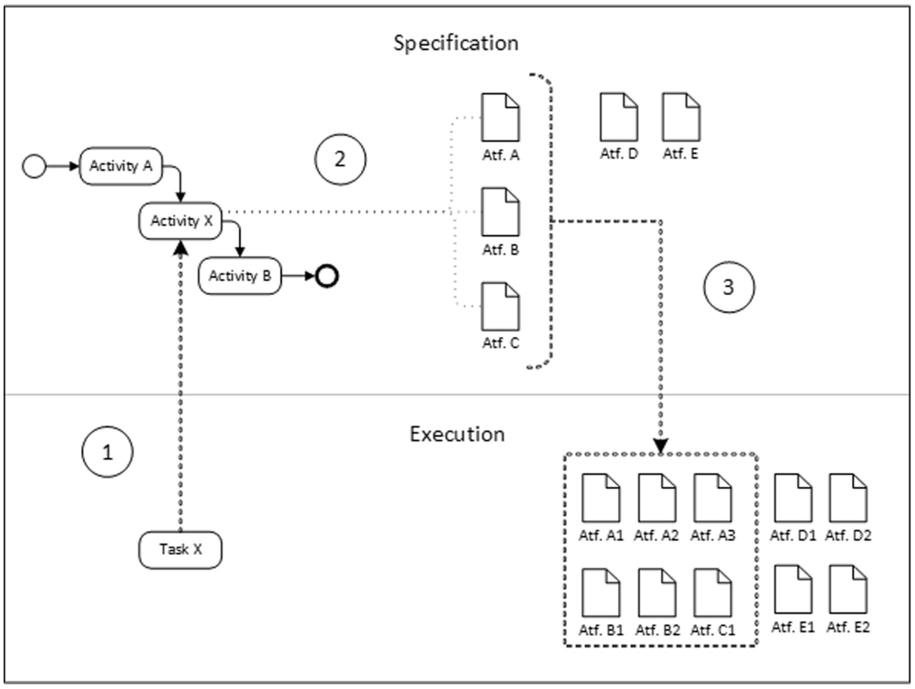


Fig. 5 MylynSDP's path to a context creation. The top of the figure shows a software process specification with its activities (top left) and related artifacts (top right). The bottom half displays a software process execution and its tasks (bottom left) and artifacts (bottom right). Relationships between elements (e.g., activities and artifacts) are made implicit for clarity

created. For that reason, the DOI function performs a special interaction event called Specification. This interaction event is explained in the next section.

Interactions

Once initial task contexts are created and the software process is being executed, artifacts will be manipulated. Each interaction with an artifact is mapped by the DOI function, which is then able to infer the importance of an artifact in relation to a particular task. In addition to Mylyn's original interactions, MylynSDP's DOI function supports a new type of interaction called Specification (Table 4). This new type of interaction is necessary to handle the case when an execution *artifact* that is being created is relevant to the task currently executing. In this case, the new artifact should receive a high interest value at the moment of creation. As the new artifact has not yet been manipulated, the DOI function performs a specification interaction event to increase this artifact's interest value.

Another example of the use of the new interaction event is when a new *task* is being created and some of the existing artifacts are relevant to its execution. This set of execution artifacts has not been manipulated during the execution of the new task because the task has just been created. In this case, the DOI function should assign a high value of

Table 4 MylynSDP's new interaction event type

Interaction event type	Contribution	Description
Specification	5.0	Supports tasks' initial context creation.

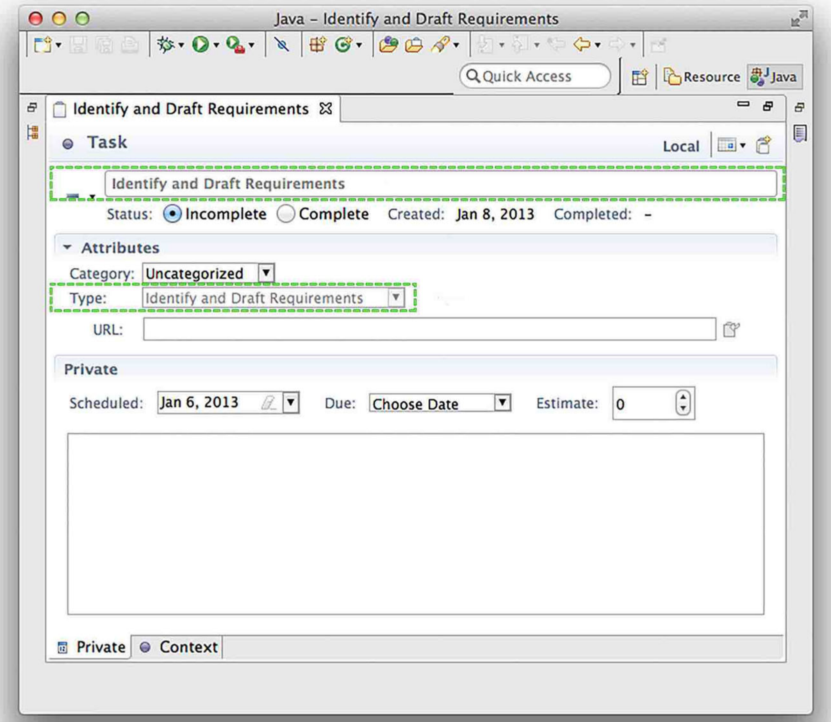
interest for these artifacts to assure they will not be removed from the context easily. This is done by automatically performing a specification interaction over the suitable artifacts.

Tool and architecture

Wizard

To create tasks and artifacts in MylynSDP, the software engineer makes use of wizards. There are two new wizards: the Task Creation wizard and the Artifact Creation wizard. Figure 6 shows the Task Creation wizard. This wizard is a modified version of the Mylyn's Task Creation wizard, where there is a field ("Type") to describe the activity related to the task. A task must also have a name ("Identify and Draft Requirements" in the figure), which can be different from the name of the associated activity. The field for a task name did not undergo major modifications.

The Artifact Creation wizard, shown in Fig. 7, enables the software engineer to create a file with any extension by leveraging the Eclipse file creation mechanism. This totally new wizard has an option to import an artifact that has already been created in MylynSDP. After naming the new artifact, the software engineer then associates it with one of the artifacts recognized in the software process specification (using the field "Type" in the foreground window). As tasks and artifacts are created, Mylyn's DOI function has the data for building initial task contexts.


Java – Identify and Draft Requirements
Quick Access
Resource
Java
Identify and Draft Requirements
Task
Local
Attributes
Private
Private
Context

Identify and Draft Requirements

Status: [✓ Incomplete] Complete Created: Jan 8, 2013 Completed: –

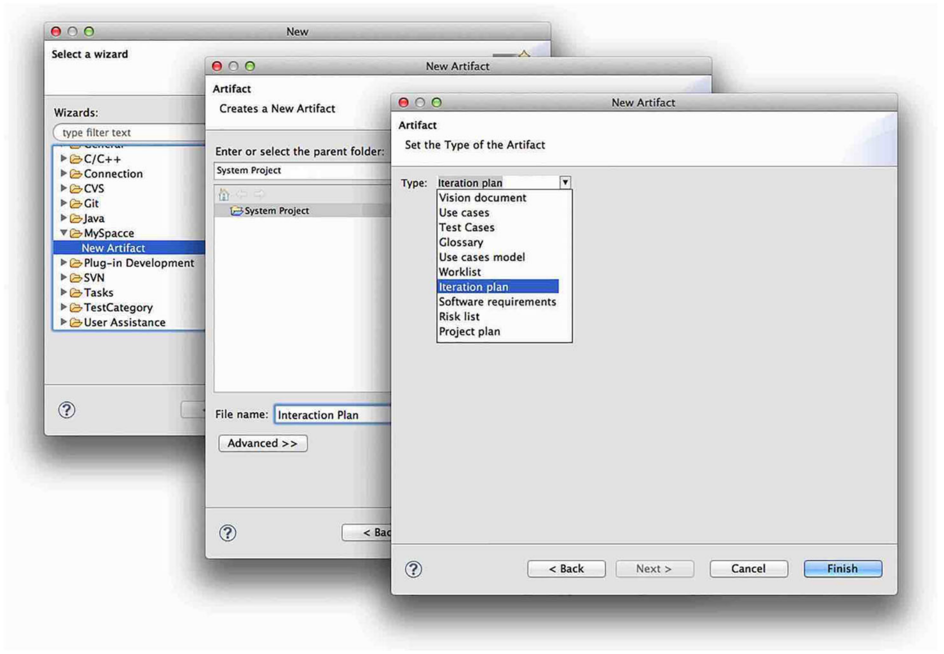
Category: Uncategorized

Type: Identify and Draft Requirements

URL:

Scheduled: Jan 6, 2013 Due: Choose Date Estimate: 0

Fig. 6 The Task Creation wizard


Enter or select the parent folder: System Project

File name: Interaction Plan

Type: Iteration plan

Fig. 7 The Artifact Creation wizard. The wizard has three steps (shown from the background window to the foreground window)

Architecture

Mylyn's original source code is comprised of more than 200 Java projects, and even more Java classes. Some parts of the code were inspected when implementing MylynSDP. To explain the MylynSDP code, it has been divided into four parts, based on functionality, namely: the Software Process Specification Import mechanism, the Restore mechanism, the Saving mechanism, and the DOI function.

Software Process Specification Import mechanism

The Software Process Specification Import mechanism is responsible for importing software processes, as its name implies. It is the first mechanism to be used when dealing with MylynSDP, since a software process specification should be imported before any other interaction. This totally new mechanism imports software processes in three steps. First, it processes an XML file (containing the software process specification) to gather information about activities and artifacts. Second, it copies the software process specification (i.e., the actual information about activities and artifacts) to the Eclipse workspace for later use (e.g., to check available activities when creating tasks). Third, it transforms the XML file to a format that the underlying Mylyn software can understand, so it is able to continue the importation and create the space needed for the management of tasks and artifacts.

Restore mechanism

The Restore mechanism expands the way Mylyn deals with context recovery by saving new task and artifact properties, such as their names and types. Moreover, the Restore mechanism saves the relationship between activities and tasks, as well as the relationship between software process specification artifacts and execution artifacts, which is not done by Mylyn. By doing this, the DOI function is able to identify which artifacts initially belong to a particular task context based on the software process specification that was imported earlier.

Saving mechanism

The Saving mechanism saves all interactions made by a software engineer along with the interest contribution for each interaction. When the DOI function wants to calculate the interest value for a particular artifact in relation to the current task, it consults the interaction history file deployed by the Saving mechanism. Modifications made to this mechanism are small and mainly related to the ability of saving a new type of interaction in the interaction history.

DOI function

The Degree of Interest (DOI) function is the mechanism used to calculate the interest value of each artifact during the execution of a particular task. The calculation is based on the imported software process specification and on the interactions performed by a software engineer. Similar to how Mylyn works, an artifact's interest value is increased whenever that artifact is subject to an interaction. Along with that artifact's interest value increasing, other artifacts' interest values are decreased by a small amount. Every interaction event has an ordinal number associated with it. The comparison between (i) the ordinal number associated with the most recent event and (ii) the ordinal number related to the interaction event performed at the creation of the artifact allows the DOI function to calculate the actual interest value, as well as the decay value, for an artifact at runtime. A concrete example of how the DOI function calculates the interest and decay values is provided at the end of this subsection.

Some Mylyn classes shown in Fig. 2 were changed. Figure 8 illustrates those changes by highlighting them. The class `InteractionContextScaling` had the interest decrease parameter modified from 0.017 to 0.2 to filter out unnecessary artifacts with less interactions. As explained in the next paragraph, the modification of the decay parameter is made to take into account the number of interactions with the different types of software artifacts (e.g., code files, use case documents). Moreover, the refresh rate was updated so MylynSDP's views may reflect changes rapidly. Note that a new type of interaction was introduced, and a few changes were made to `InteractionEvent` and `InteractionContext` classes. The `DegreeOfInterest` class had its interest calculation changed. The first modification is related to the ability to increase the interest value according to the Specification interaction event. The DOI function's second modification allows the first interactions to be of higher importance than the others. The details of the DOI function's implementation are provided next.

Note that some artifacts are more used than others and this impacts the calculation of the interest value. For example, code documents may be opened and edited for a long time, and use case documents may be opened once for a quick inspection. In contrast with Mylyn, MylynSDP deals not only with code files but also with other software artifacts such as use case documents and test cases. Therefore, its DOI function should try to accommodate these new interaction behaviors. Figure 9 guides our discussion on how to deal with these new behaviors and how to define a new decay parameter. The

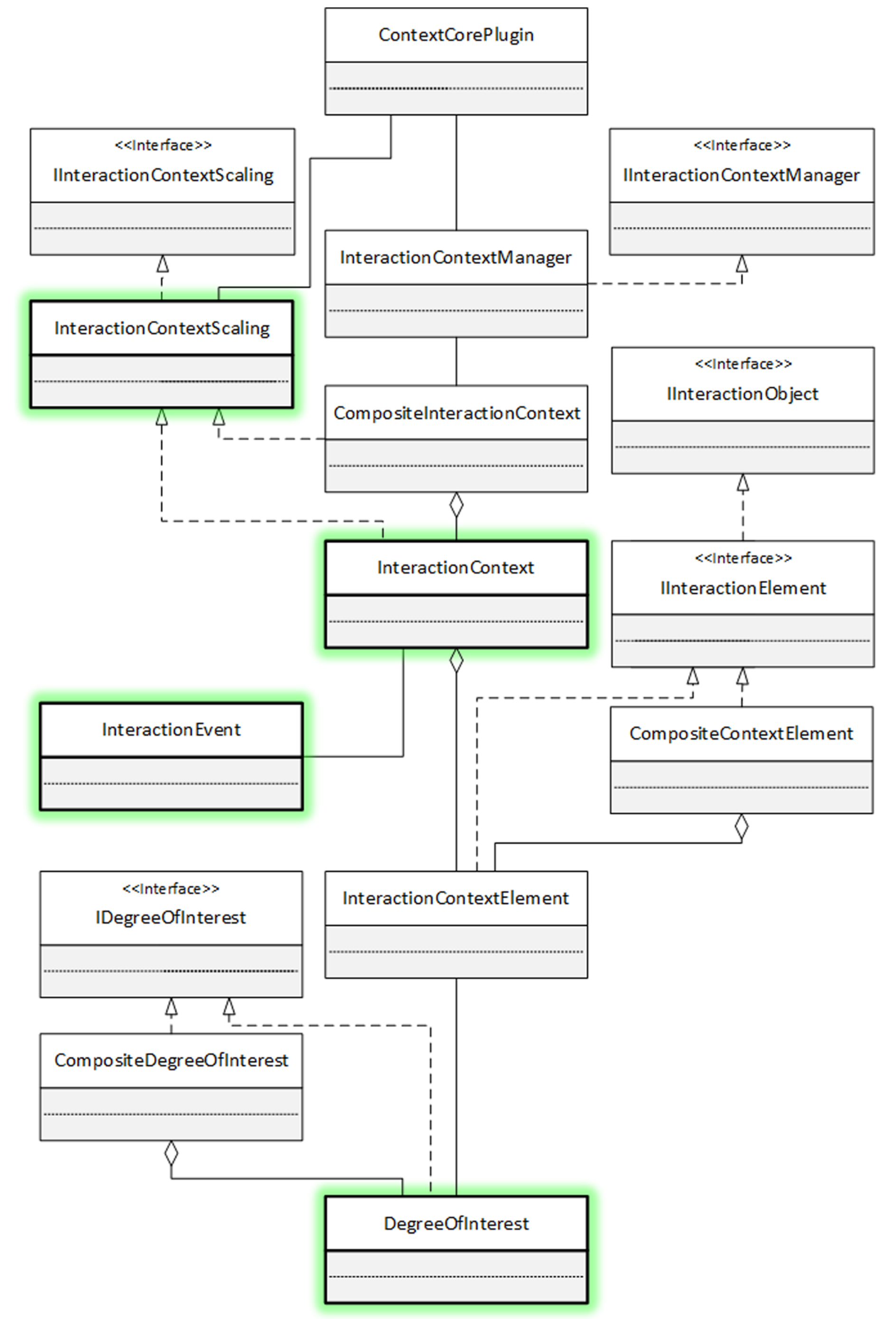


Fig. 8 MylynSDP's class diagrams. Classes highlighted were subject to significant modifications

original Mylyn's DOI function requires several interactions to happen before it starts filtering out artifacts. Consider an artifact that has been selected once (see the line with circles in Fig.9a). According to Mylyn's DOI function, the artifact's interest value, which has the value 1 so far, will decrease at a rate of 0.017 at each new interaction, regardless of the target of the interaction. Note that this artifact's interest value is negative

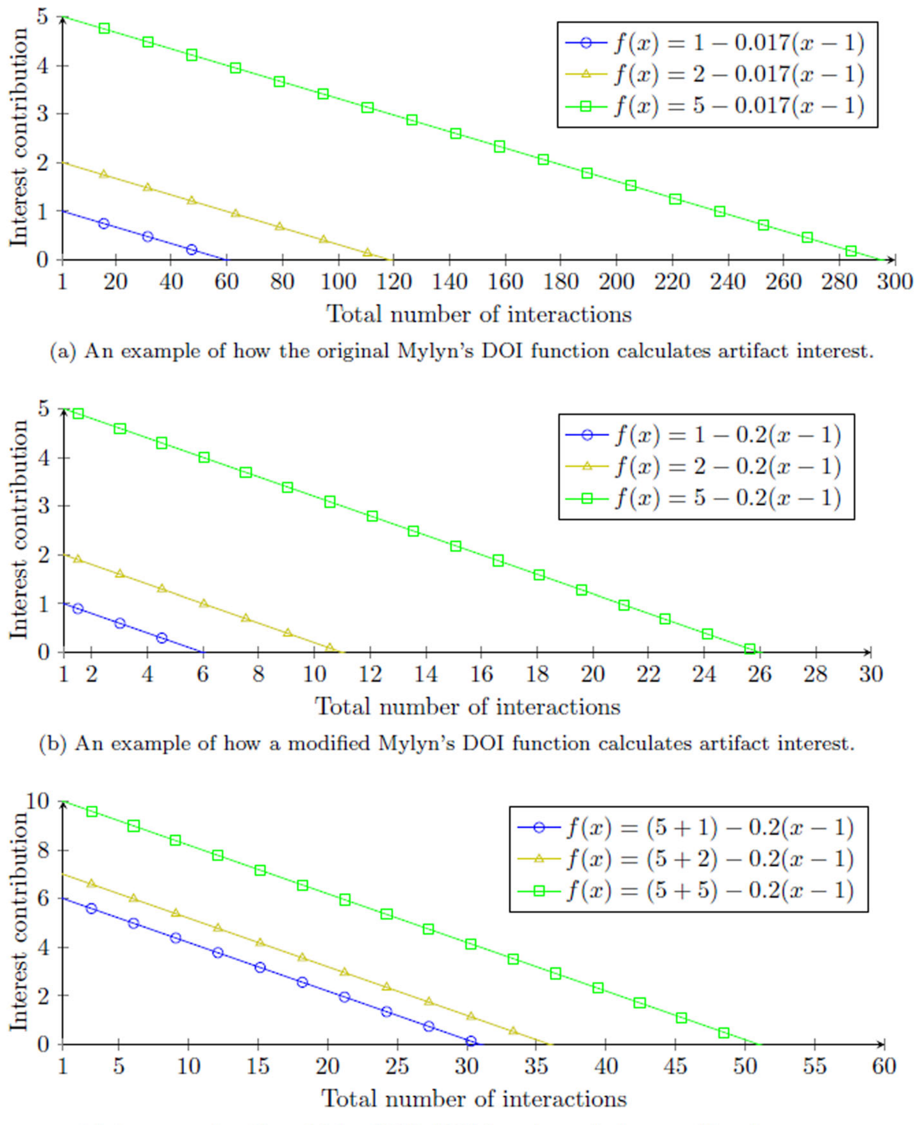


Fig. 9 An example of how different versions of a DOI function calculate artifact interest

after 60 interactions. The other two lines in Fig.9a show that more than 119 interactions are required to filter out an artifact that has been involved in an interaction twice, and almost 300 interactions are required to filter out an artifact that was involved in only five interactions. This implies that it takes a substantial number of interactions for Mylyn to update task contexts and this may not be desirable for all types of software artifacts. Because the filtering out process for different artifacts heavily depends on the decay parameter, MylynSDP modifies this parameter from 0.017 to 0.2 so that task contexts do not require neither too many nor too few interactions to be updated. Figure 9b shows what happens when the decay parameter is changed to 0.2. In this case, 6 interactions would be required to filter out an artifact that was involved in an interaction once (see the line with circles in Fig. 9b). This change to the decay parameter sped up the decay considerably. The other two lines in this figure also exhibit the same effect by showing that 11 and 26 interactions are required to filter out artifacts that were involved in an interaction twice or five times, respectively. Because in Fig. 9a too many interactions and in Fig. 9b too few interactions were needed to filter out an artifact, we decided to investigate a middle ground solution. This solution moves the curve of interest decay up by a constant (see Fig. 9c). This constant is the Specification interaction event contribution value calculated at the creation of any artifact, which is 5. In this case, 31 interactions would cause an artifact interest value to be negative if the artifact was involved in an interaction only once, and 51 interactions would be required to filter out artifacts that involved in five interactions. This is how the MylynSDP's DOI function ensures that uninteresting artifacts are quickly filtered out while preserving interesting artifacts.

The MylynSDP DOI function pseudo-algorithm is presented in Listing 1. It is divided into three main parts: event registration, partial interest value calculation, and decay value calculation. Whenever an interaction happens targeting an artifact, a method (not shown in the figure) is called to register the event. Its work is to increment a counter of selections, editions, or any other interaction event type. Later, when needed, the DOI function calls the `getValue()` method to retrieve the interest value for an artifact in relation to the running task context.

Listing 1 Mylyn's DOI function pseudoalgorithm.

```
function getValue() {
  double value = 0
  value = calculatePartialInterestValue()
  value = value − calculateDecayValue()
  return value
}

function calculatePartialInterestValue() {
  double value = 0
  value = value + (numSelections   * getConstantSelection())
  value = value + (numEdits        * getConstantEdits())
  value = value + (numCommands     * getConstantCommands())

  value = value+(specificationBool*getConstantSpecification())

  return value
}

function calculateDecayValue() {
  double decayValue = 0
  double numberOfInteractions=eventCount−eventCountOnCreation
  return numberOfInteraction * getConstantDecay()
}
```

The `getValue()` method calculates a partial interest value (line 3) and then subtracts a decay value (line 4) from this partial interest value. The main modifications from Mylyn's DOI function are in each of the two other methods. Interest value calculations are initiated in the method named `calculatePartialInterestValue()`. The method calculates the number of occurrences of a particular type of interaction times a constant associated with that type of interaction (i.e., its contribution to the interaction value). This is done for selection (line 10), edition (line 11), and command (line 12) interaction event types that already exists in the original Mylyn. For instance, if a particular artifact has been selected five times, its partial interest value is $5 \times 1 = 5$, because selection contributions are worth one each. The new Specification event is captured by the `specificationBool` variable. The variable is a zero-or-one value that indicates if the artifact initially belongs to the current task context based on the software process specification. If this is true, then the value five is added to the artifact's interest (line 14). The value five is the contribution made by the Specification interaction event. The value is arbitrary and was reached after testing the code and calibrating it. The `calculateDecayValue()` method is then called to calculate an offset of the interest value, referred to as the decay value in a way similar to how Mylyn does it. The decay value of a particular artifact is based on how many interactions were performed in that task context since the creation of that artifact (see Eq. (4)). As explained earlier, the number of interaction events performed since the creation of a particular artifact (line 21) is multiplied by a decay constant (line 22). The novelty, in the case of MylynSDP, relates to the inclusion of the number of Specification interactions and the new decay constant in the calculations. An example may clarify these calculations. Suppose an artifact was created during interaction #10 and five other interactions happened to other artifacts, meaning that the most recent interaction event is #15. The decay value is now calculated as $(15 - 10) \times 0.2 = 1$. This value is the result of the `calculateDecayValue()` method to the `getValue()` method (line 22) as that artifact's decay value. Finally, the `getValue()` method calculates and returns the interest value for a particular artifact in a task context (line 5).

MylynSDP usage starts with either the creation of a task or the creation of an artifact. When creating a task, a software engineer uses the MylynSDP Task Creation wizard and names the new task. At this point, no initial task context has yet been created. By the time the software engineer associates the new task with an activity from the software process specification, the DOI function searches for artifacts that should belong to the new task context, based on the software process specification, assuming that one has been previously imported. Once these artifacts are found, the DOI function performs a Specification interaction event on them, which increases their interest value, and creates the initial task context.

Artifact creation starts with the use of the MylynSDP Artifact Creation wizard. After providing a name for the artifact, the software engineer needs to associate it with one of the artifacts present in the software process specification. If a task is active at this moment, it means that the new artifact is relevant for the task. In that case, the new artifact receives a Specification interaction event and its interest value is increased by five. An example of the new interest calculation is shown in Table 5. Suppose a software engineer creates an artifact `Artifact A` using the appropriate wizard. After setting its properties, the artifact's interest value is 5. Now suppose that the software engineer performs a series of interaction events on other artifacts, as illustrated in Table 5. In this example, the interest value is the sum of the contributions of the Specification and the Selection events; that is, $(1 \times 5) + (2 \times 1) = 7$. Following the example, the decay value is calculated as follows:

Table 5 An example of MylynSDP's DOI function calculating the interest value for an artifact. Most recent interaction at the bottom

Index	Quantity	Event	Target
#1	1	Specification	`Artifact A`
#2	1	Selection	`Artifact A`
#3–#12	10	Selection	Other classes
#13	1	Selection	`Artifact A`
#14–#23	10	Selection	Other classes

$((23-1)\times 0.2) = 4.4$. Thus, the final interest value of that artifact is $(1\times 5)+(2\times 1)-((23-1)\times 0.2) = 2.6$, which is positive, meaning that it is still of interest for the current task's execution.

Evaluation study

An evaluation study was conducted to assess the results of the use of MylynSDP and its DOI function. The case study involved several subjects who were asked to perform tasks from a real software development process with the help of MylynSDP, and a questionnaire in which opinions about the use of the DOI function were gathered. The impact of the MylynSDP's DOI function has on the execution of a software development process is evaluated by the extra functionality that Mylyn does not include. This section presents more details about the evaluation study design, execution, results, and validity.

Overview

In this paper, we describe the benefits of the use of DOI function in the implementation phase of a software development process and the likelihood that these benefits could be applied to the whole software process. To understand the implications of the use of DOI function in practice, it was decided to perform an evaluation study and investigate the use of a DOI function in the management of artifacts during an execution of a software development process. Note that each contribution described in the previous section, namely the use of a predefined software development process, the changes of the wizards, the creation of the Specification interaction event, and the use of a new decay value, is required for defining the new DOI function in MylynSDP. Therefore, all of these contributions have to be taken into account and their impact is analyzed as a whole in our evaluation study, which includes an assessment of productivity improvement. The study focused mainly on a software engineer's productivity. We define productivity as being the number of artifacts produced at each unit of time. We believe that by reducing the amount of time spent in non-productive work (i.e., searching artifacts to start a task or in consequence of a context change), software engineers may have more time to perform productive work, generate more artifacts, and be more productive. According to [55], this study is of an exploratory nature because the intention is to discover the consequences of the use of DOI function and generate insights and new hypotheses. Based on the classification in [33], the study adopted a positivist research perspective as it follows the classic research model based on evidence that measures variables, tests hypotheses, and draws inferences. This study conforms to a fixed design process as characterized in [4], which means that parameters such as interview questions and data analysis procedures are defined at the beginning of the study and do not change as the study develops. This evaluation study is intended to address the following question:

- What is the impact of the use of DOI function in managing artifacts in the execution of a software development process?

To answer this question, it was decided to use subjects to simulate the execution of a software development process in a laboratory setting. Running a simulated study in a laboratory allows scientists to gather data better from the subjects, but the setting also decreases the reality of the study. For that reason, it was decided to use a real software development process with real artifacts. The software process that served as the case study comes from the SIGA-EPTC project[5], whose objective is to allow the management of academic data, such as student and professor profiles, teaching rooms, grades, and academic transcripts, originating from public federal educational institutions in Brazil. The software process, which can be seen in Fig. 10, describes activities from the modeling and specification phases of software development, where requirements are turned into concrete design documents and database entities. There are more than 300 artifacts to be used during the execution of the software process and they have different nature: use case descriptions, requirements and test case documents, SQL scripts, and class and database diagrams. During the execution, software engineers are expected to access a list of previously gathered requirements and business rules, and read and turn them into specifications and screen designs. Specification and screen design documents can then be used to create class diagrams, by identifying important nouns, verbs, or other elements, and test cases, by analyzing the scope of specifications and reasoning on what is outside it. Specifications documents, class diagrams, and screen designs are then used to create a database and SQL scripts.

The evaluation study was performed with seven software engineering students, six Ph.D. students, and one M.Sc. student, who were enrolled in the PESC/COPPE software engineering program at the Federal University of Rio de Janeiro, Brazil. Five subjects reported professional experiences in industry that ranges from 4 months to 10 years working with software development. Some subjects have worked with software process design in industry for 1 year up to a decade. Two subjects have taught courses on software process design in the academic environment for at least one term, and another subject has studied software processes in his or her doctorate degree.

The evaluation study was divided into three sequential stages: training, exercises, and the final questionnaire. The first stage was the training stage. Most of the subjects had never seen MylynSDP or the software process. The aim of the training stage was to explain to all the subjects the concepts of MylynSDP, its DOI function, and the details of the software process and artifacts. Those details were the project, its activities and artifacts, the nature of the project, the modules of the project, and the naming conventions for the project's artifacts. The training was performed in the form of a presentation, and subjects were free to ask questions.

The second stage was the exercise stage, in which the execution of the software process was simulated and each subject was asked to solve five common tasks. During the execution of the tasks, subjects interacted with MylynSDP's interfaces, as well as multiple artifacts. The five exercises, which were executed as MylynSDP tasks, were created

[5]https://softwarepublico.gov.br/social/siga

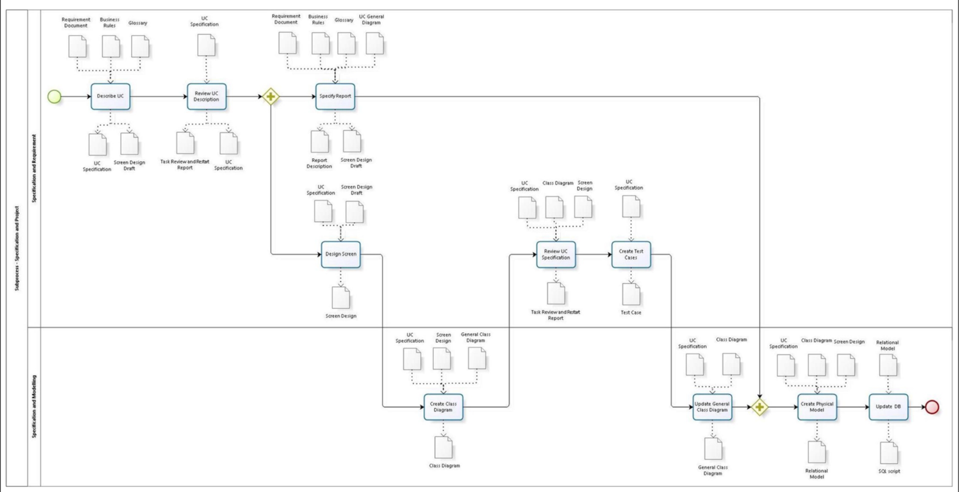


Fig. 10 Software process used on the evaluation study

based on the activities of the software process. Each exercise was designed to observe the reaction of the subjects when executing common software engineering tasks, which MylynSDP, along with its DOI function, was expected to help solve. The tasks included the manipulation of artifacts such as a use case description or a glossary. Table 6 describes each exercise and their goals. Comments left by subjects explain the difficulties they had during the exercise and attempt to clarify the reasons why they took a particular decision when interacting with MylynSDP.

The third stage was the final questionnaire. After being trained and interacting with a simulated application of MylynSDP and its DOI function, subjects were asked to answer a questionnaire about their experiences when dealing with MylynSDP. The questionnaire is based on the Technology Acceptance Model (TAM) [13, 14], which consists of two scales for two variables: perceived usefulness and perceived ease of use. Both variables are considered fundamental determinants of user acceptance. According to [13], perceived usefulness is related to one's belief that a given technology will help him or her to perform a job better, whereas perceived ease of use refers to the degree to which one believes that using a given technology will minimize effort. The questionnaire, which was comprised of 12 statements, had the first six statements related to perceived usefulness and the remaining six statements related to perceived ease of use.

Table 7 shows the 12 statements. Each statement has seven possible answers ("I completely disagree," "I partially disagree," "I slightly disagree," "I do not agree, nor disagree," "I slightly agree," "I partially agree," and "I completely agree"), which are used as a Likert scale [34].

The Goal-Question-Metric (GQM) approach [7] defined for this evaluation study is as follows:

Goal:	To investigate the use of a DOI function during the execution of a software development process focused mainly on, but not limited to, the software engineer's productivity.
Question:	Does the use of a DOI function improve the software engineer's productivity during the execution of a software process?

Table 6 Evaluation study's exercises

Exercise	Description	Purpose
1	Subjects are expected to locate a particular use case document among many others and add a brief description for that use case.	This exercise executes the activity "Describe UC" and deals with the case in which low filtering is applied, because the software engineer has not interacted enough with the artifacts.
2	Subjects are expected to locate a particular use case document among others and update its description by replacing acronyms and abbreviations by their respective expanded meanings. The latter is described in another artifact named glossary. Also, subjects should add an explanatory note beside a particular business rule present at one of the business rule documents.	This exercise executes the activity "Describe UC" and has as its goal to observe subject's reaction when the task context is small enough to show only a few artifacts to the software engineer.
3	Subjects are expected to read a note left by another software engineering asking the subject to review a use case that is not initially part of the subject's work. Subjects should locate the corresponding use case and fix any grammatical or typing mistakes.	This exercise executes the activity "Review UC Description" and is designed to analyze what subjects do when they do not find an artifact in a task context, either by incorrect filtering or by incorrect software process modeling.
4 and 5	Subjects are expected to locate a particular test case document and write a new test case for a given use case based on the specification of this use case. When the subject is about to start writing, the subject is presented with a new high priority exercise. From that moment, subjects are expected to update part of a particular SQL script based on the syntax of other similar SQL scripts. Subjects are then expected to resume the previous exercise.	These exercises execute the activities "Create Test Cases" and "Update DB" and simulate a context change. During the execution of exercise 4, the subject is presented with exercise 5, which is said to be high priority. Then, the subject has to change the context of the task he or she is performing.

Table 7 Evaluation study's twelve statements

Number	Statement
1	Using the DOI function in my job would enable me to accomplish tasks more quickly.
2	Using the DOI function would improve my job performance.
3	Using the DOI function in my job would increase my productivity.
4	Using the DOI function would enhance my effectiveness on the job.
5	Using the DOI function would make it easier to do my job.
6	I would find the DOI function useful in my job.
7	Learning to operate the DOI function would be easy for me.
8	I would find it easy to get the DOI function to do what I want it to do.
9	My interaction with the DOI function would be clear and understandable.
10	I would find the DOI function flexible to interact with.
11	It would be easy for me to become skillful at using the DOI function.
12	I would find the DOI function easy to use.

Metrics:	• Subjective opinion about their execution
	• Time of the execution of each task

Note that although the study was focused on investigating productivity, which is defined here as the number of artifacts produced at each unit of time, we measured the time of the execution of each task and not the number of artifacts produced. The assumption is that the less time to produce an artifact, the more artifacts can be produced per unit of time.

Based on this GQM, this evaluation study collected both qualitative and quantitative data, which characterizes it as a mixed methods study. Moreover, as explained, data was collected with the researcher's observations and as a questionnaire. Both methods are described as a direct method or first-degree techniques, since the researcher and the subject were in contact and data is gathered in real time. The use of a questionnaire to allow subjects to express their opinion about the use of the DOI function is a type of interview usually classified as a single interview, as opposed to a group interview. This interview had closed questions, because it offers a limited set of answers, and used a fully structured approach, in which questions were planned ahead of time and were asked in the same order for all subjects.

This evaluation study took two measures to alleviate ethical concerns. Before the study, every subject was made aware that data was gathered for research purposes and that results from this study would be reported anonymously. Second, SIGA-EPCT granted access to its software development process and artifacts for research purposes.

Analysis

Following the description of the evaluation study, we now analyze its results. Specifically, we investigate the impact of the DOI function on productivity, provide evidence for an improvement, and discuss the questionnaire results. For example, we have compared productivity in environments in which low and high filtering are applied (E1 and E2), and provided evidence that MylynSDP's high filtering capabilities improve productivity. Additional supporting evidence for productivity improvement comes from the fact that, for most subjects, the time taken to complete two tasks (E4/E5) using MylynSDP's filtering capabilities is similar to the time taken to complete one task (E1) that relies on low filtering.

Table 8 shows the time each subject took to finish the proposed exercise. E1 and E2 were designed to observe the impacts on productivity that the introduction of the DOI function had. The two exercises were relatively simple and similar, with the difference

Table 8 Execution times for subjects in each exercise

Subject	E1	E2	E3	E4 and E5
S1	19 min 13 s	13 min 09 s	06 min 42 s	17 min 19 s
S2	12 min 48 s	06 min 57 s	04 min 09 s	11 min 20 s
S3	06 min 40 s	05 min 11 s	05 min 03 s	10 min 13 s
S4	12 min 06 s	06 min 56 s	05 min 36 s	18 min 26 s
S5	11 min 15 s	08 min 28 s	04 min 22 s	21 min 00 s
S6	09 min 41 s	04 min 28 s	03 min 04 s	05 min 45 s
S7	14 min 20 s	12 min 49 s	07 min 40 s	14 min 45 s

that E2 included an advanced aid of the DOI function. One may note that all subjects reduced the time to execute E2, mainly because of the DOI function assistance. However, one skilled subject, S2, did not have a significant time reduction.

Additionally, it is worth to compare the execution times of exercises E1, and E4 and E5. Subjects S1, S2, S6, and S7 were able to reduce the execution time from the first to the two last exercises. This means that, after using the DOI function, one can execute two exercises with the same time that one used to execute only one exercise. These results may be influenced by external factors, such as the difficulty or length of the exercises. However, E1 and E4 are believed to be similar, since subjects had to write document descriptions based on other artifacts. Subjects who increased the execution time from E1 to E4 and E5 did not leave any significant comments about it, and more studies should be done to investigate the use of the DOI function when handling a context change.

The two previous comparisons show that the use of the DOI function in MylynSDP assists software engineers to reduce their times when executing activities, leaving more free time to start working on another activity. This also indicates that software engineers' productivity is positively affected and improved with the use of the DOI function.

The results of the questionnaire are shown in Fig. 11. The values are the percentage of answers given by the seven subjects in each statement. In general, results point to a positive adoption of the concept of MylynSDP and its DOI function in all phases of a software process execution. As can be seen, all percentages related to an "I completely agree" answer are greater than 50%, except for statement #5, which also had a high rate of acceptance. However, some points are worth mentioning.

In statement #6, one subject said that the DOI function might not be useful in his or her work. The same subject left a comment at the end of the questionnaire explaining that he or she was working on coding tasks strictly in his or her job when taking part on this study and did not feel that the DOI function would useful in his or her work. Perhaps, the changes made to the DOI function made it less suitable to that particular area of software development, which can be assumed to be true when generalizing any concept.

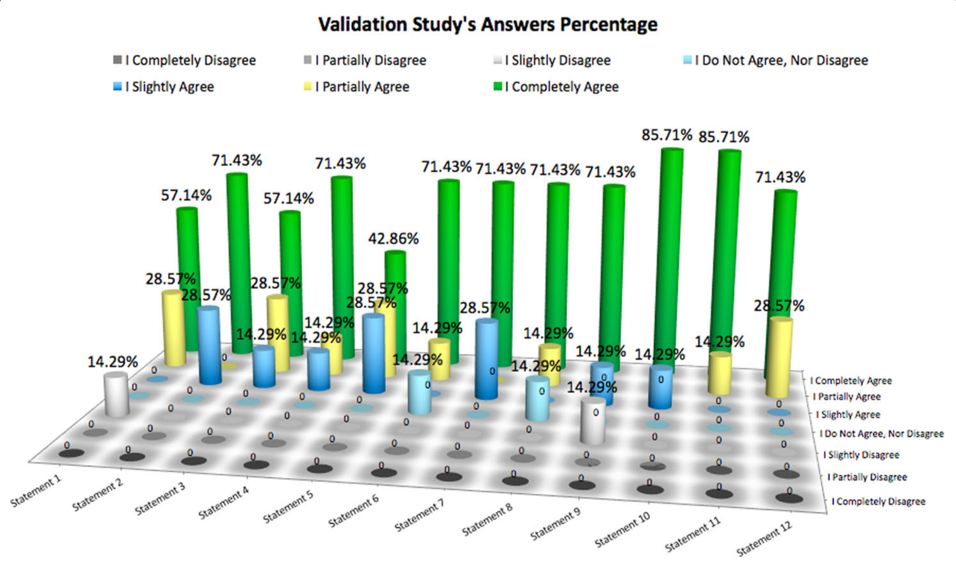


Fig. 11 Evaluation study results. Numbers are in percentage

The second point to note refers to statement #8. One of the answers did not consider the DOI function as being easy enough to use to accomplish what the software engineer desired. The subject had difficulties with the Mac operating system's interface, such as scrolling, minimizing, and closing documents.

The third point to emphasize relates to the low score statement #1 received from one of the subjects. Unfortunately, no explanation was left at the end of the questionnaire regarding this evaluation, but the overall percentage of answers was high enough to cause the low score to be considered an outlier.

The fourth point deals with statement #9. One of the subjects slightly disagreed that the DOI function implemented in MylynSDP is clear and comprehensible. Although no explanation was left in the comment section at the end of the questionnaire, one suggestion was made. The subject suggests that filtering could be improved with the use of keywords, either by looking for particular words in the name of the files and within their contents or by tagging artifacts based on the needs of the software engineer.

Finally, the fifth point concerns statement #2, which deals with productivity. Two of the subjects slightly agreed that the DOI function improves their productivity. Although it is a positive result, no comments were left to explain the reasons for the lack of a higher score.

Threats to validity

Three threats to external validity were identified. The first one is that, for convenience, subjects of the evaluation study were software engineering graduate students who had a close academic relationship with some of the authors. Some studies [56] investigate the applicability of students in software engineering evaluation studies and conclude that the differences between graduate students and professionals working in the field are minor. Even though the subjects of this evaluation study have different levels of experience and background in software engineering, the representatives of this population may not be ideal for a complete generalization of the study. The second threat is that the study had seven subjects. Although some trends in the answers from the TAM questionnaire could be observed and some conclusions could be drawn, it is known that seven is a low number of subjects and this can affect the generalization of the results of the case study. The third is that the software process used in the evaluation study may not be representative of all possible cases found in the industry. For example, it does not include feedback flows. Therefore, our results represent evidence of the benefits of the use of a DOI function on a software engineer's productivity, but more studies are required.

In addition, subjects may have been subject to internal validity threats, where a factor not identified by the researcher may affect the results of the study. Three internal validity threats can be described. During the training stage of the evaluation study, all subjects were introduced to the software process and the related software. Therefore, results of the evaluation study can be affected because each subject is totally new to the project. Second, most of the subjects complained about the naming convention used for the artifacts of the software project simulated in the case study. Although they said it was confusing and disorganized, nothing could be done to avoid this situation because a real software project with its own details was used. Third, the evaluation study was conducted on an iMac. Six of the seven subjects were not familiar with the Mac operating system, and they had minor problems during the execution of the case study, such as minimizing a window or scrolling with the mouse. These subjects were given quick instructions of how to perform such actions, and these difficulties were not observed anymore. This minimizes the impact of these difficulties on the final result.

Conclusion and future work

Conclusion

During the execution of a software process, activities are performed and artifacts are manipulated to produce new artifacts. To execute a particular activity, a software engineer searches suitable artifacts among several other non-relevant artifacts. The search can be time-consuming, can be error-prone, and can lead to confusion. Moreover, if a higher priority activity interrupts the execution of the current activity, a new search must be performed to find the artifacts for the new activity, which may lead to more confusion. In the end, the software engineer spends a substantial amount of time and effort in the search for suitable artifacts for the activity execution that could be spent on the work he or she is supposed to perform.

Apropos those problems, the solution of the Degree of Interest (DOI) function is described as a mechanism that rates elements according to a predefined rule. A DOI function can aid software engineers to be more productive by grading artifacts of a software process according to their relevance to the activity being executed. An implementation of a DOI function is found in Mylyn, a plugin for Eclipse. However, Mylyn's DOI function is aimed only at the coding phase of a software process. This work then expands the DOI function to allow it to help software engineers in all phases of a software process.

The main contributions of this work start with the expansion of Mylyn to consider an underlying software process being executed. Two wizards for task and artifact creation are introduced to collect information (e.g., the type) that can help to link tasks and artifacts to their specification counterparts (activities and specification artifacts, respectively). Moreover, this work presents the new Specification interaction event, which is essential in the management of activity contexts. Lastly, some adjustments on the DOI function formula, including a new decay value, adapt it to a broader class of activities in a software process execution that is not limited to implementation activities. The final implementation is named MylynSDP.

An evaluation study has been conducted to analyze the positive and negative points associated with this approach. Seven subjects were invited to interact with MylynSDP in a simulated environment, and at the end of the study, they were asked to answer a TAM questionnaire. Comparison between the time taken to complete study exercises showed that as the project progresses, and software engineers interact more with artifacts, less time is needed to complete tasks. The reduction is a result of the MylynSDP's assistance in artifact search and context change. Answers to the TAM questionnaire showed that MylynSDP's concepts and its DOI function are likely to be used by software engineers in their work.

Future work

Some new features can be developed to improve MylynSDP's DOI function and how it works. A better log file can be deployed to be consulted later, either manually or by an application, generating better reports of usage. The current log file displays a long list of interaction events that happened during the execution of a task. Some of the entries in the log file are grouped to speed up calculations when returning to a task. This log file can be consulted to identify, for example, the most important files of the project, or a particular document that has been used multiple times.

In addition, the DOI function's grading mechanism can be improved to accept not only interactions with documents, but also diagrams or images. Note that basic interactions such as opening or closing an image file are currently captured by MylynSDP's DOI function. Here, it is desirable that a richer set of interactions is captured, such as color change, rendering, or line drawing. This is valuable because at early phases of a software development process, such as system design, diagrams are often used to express ideas.

An evaluation of the effort required to use MylynSDP compared to Mylyn could be undertaken using quantitative or qualitative approaches. Quantitative approaches may rely on time (e.g., work hours), the skill level of the software engineers, and the quality of the task results. Qualitative approaches could be based on subjective assessments such as questionnaires and surveys.

Also, change management is a critical research avenue. Processes may change during execution and MylynSDP does not support this kind of change, and a new import would be necessary. However, non-conflicting changes (e.g., adding a new activity without changing previous relationships or flows) can be quickly identified by MylynSDP. Furthermore, activities and processes may have versions. Currently, an activity can be executed as different tasks multiple times, but no further support is provided. To support process versioning, a new process should be imported if a new version of the process is to be used, and in this case, new versioning capabilities are required to further assist software engineers during software process execution.

Moreover, it is interesting to allow the DOI function's records to be consulted by another DOI function to improve the filtering for software engineers that perform similar tasks, or even to improve filtering for software engineers working at the same task. For example, if two software engineers are working on two different tasks that came from the same activity, it is reasonable to expect some overlap between these task contexts. Research on this avenue may encourage collaboration within MylynSDP.

Abbreviations
ALS: Application lifecycle management; AUI: Attention-reactive User Interface; DB: Database; DOI: Degree of Interest; CMMI-DEV: Capability maturity model integration for development; GQM: Goal-Question-Metric; IDE: Integrated Development Environment; MDD: Model-driven development; PARC: Palo Alto Research Center; PCTE: Portable Common Tool Environment; PDG: Process Dependency Graph; PSEE: Process-centered software engineering environment; RUP: Rational Unified Process; SDP: Software development process; SEE: Software engineering environment; TAM: Technology Acceptance Model; UC: Use case

Acknowledgements
Not applicable.

Authors' contributions
Not applicable. The authors read and approved the final manuscript.

Authors' information
Not applicable.

Funding
This work was supported by the CAPES Foundation and NSERC. We thank both organizations for their support and investments in education and development of science.

Availability of data and materials
Not applicable

Competing interests
The authors declare that they have no competing interests.

Received: 28 August 2019 Accepted: 8 July 2020
Published online: 10 August 2020

References

1. Abdul-Rahman H, Mohd-Rahim F, Chen W (2012) Reducing failures in software development projects: effectiveness of risk mitigation strategies. J Risk Res 15(4):417–433. https://doi.org/10.1080/13669877.2011.634520
2. Afanaseva T (2019) Search of software artefacts based on the project quantitative characteristics. In: ACM International Conference Proceeding Series, Association for Computing Machinery, vol Part F148261, pp 26–30. https://doi.org/10.1145/3318236.3318252
3. Ambriola V, Conradi R, Fuggetta A (1997) Assessing process-centered software engineering environments. ACM Trans Softw Eng Methodol 6(3):283–328
4. Anastas JW, Macdonald ML (1994) Research design for social work and the human services, 1st edn.. Jossey-Bass Inc, San Francisco
5. Baharom F, Yahaya J, Deraman A, Hamdan A (2013) Software process certification: a practical model for maintaining software quality. Int J Inf Process Manag 4(3):51–61. https://doi.org/10.4156/ijipm.vol4.issue3.5
6. Bai X, Huang L, Zhang H (2010) On scoping stakeholders and artifacts in software process. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6195 LNCS:39–51. https://doi.org/10.1007/978-3-642-14347-2_5
7. Basili V, Weiss D (1984) A methodology for collecting valid software engineering data. IEEE Trans Softw Eng SE-10(6):728–738. https://doi.org/10.1109/TSE.1984.5010301
8. Bendraou R, Sadovykh A, Gervais MP, Blanc X (2007) Software process modeling and execution: the uml4spm to ws-bpel approach. In: EUROMICRO 2007 - Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2007, pp 314–321. https://doi.org/10.1109/EUROMICRO.2007.55
9. Bigliardi L, Lanza M, Bacchelli A, Dambros M, Mocci A (2014) Quantitatively exploring non-code software artifacts. In: Proceedings - International Conference on Quality Software, IEEE Computer Society, pp 286–295. https://doi.org/10.1109/QSIC.2014.31
10. Boudier G, Gallo F, Minot R, Thomas I (1989) An overview of pcte and pcte+. ACM SIGPLAN Not 24(2):248–257. https://doi.org/10.1145/64140.65026
11. Card S, Nation D (2002) Degree-of-interest trees: a component of an attention-reactive user interface. In: Proceedings of the Workshop on Advanced Visual Interfaces AVI, pp 231–245. https://doi.org/10.1145/1556262.1556300
12. Chatterjee S, Simonoff J (2013) Handbook of regression analysis. John Wiley and Sons. https://doi.org/10.1002/9781118532843
13. Davis F (1989) Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q Manag Inf Syst 13(3):319–339
14. Davis F (1993) User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. Int J Man Mach Stud 38(3):475–487. https://doi.org/10.1006/imms.1993.1022
15. Defranco J, Laplante P (2017) Review and analysis of software development team communication research. IEEE Trans Prof Commun 60(2):165–182. https://doi.org/10.1109/TPC.2017.2656626
16. Dourish P, Edwards W, LaMarca A, Salisbury M (1999) Using properties for uniform interaction in the presto document system. In: Proceedings of the 12th annual ACM symposium on User interface software and technology - UIST '99. ACM. https://doi.org/10.1145/320719.322583
17. Dragunov A, Dietterich T, Johnsrude K, McLaughlin M, Li L, Herlocker J (2005) Tasktracer: a desktop environment to support multi-tasking knowledge workers. In: International Conference on Intelligent User Interfaces, Proceedings IUI, pp 75–82. https://doi.org/10.1145/1040830.1040855
18. Drechsler A, Dörr P (2014) What kinds of artifacts are we designing? An analysis of artifact types and artifact relevance in is journal publications. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8463 LNCS:329–336. https://doi.org/10.1007/978-3-319-06701-8_23
19. Drury-Grogan M, Conboy K, Acton T (2017) Examining decision characteristics & challenges for agile software development. J Syst Softw 131:248–265. https://doi.org/10.1016/j.jss.2017.06.003
20. Eisty N, Thiruvathukal G, Carver J (2019) Use of software process in research software development: a survey. In: ACM International Conference Proceeding Series, Association for Computing Machinery, pp 276–282. https://doi.org/10.1145/3319008.3319351
21. Feng Y, Jones J, Chen Z, Fang C (2018) An empirical study on software failure classification with multi-label and problem-transformation techniques. In: Proceedings - 2018 IEEE 11th International Conference on Software Testing, Verification and Validation, ICST 2018, Institute of Electrical and Electronics Engineers Inc., pp 320–330. https://doi.org/10.1109/ICST.2018.00039
22. Fleming I (2016) Defining software quality characteristics to facilitate software quality control and software process improvement. Elsevier Inc. https://doi.org/10.1016/B978-0-12-802301-3.00003-X
23. Fuggetta A (2000) Software process: a roadmap. In: Proceedings of the Conference on The Future of Software Engineering, ACM, New York, NY, USA, ICSE '00, pp 25–34. https://doi.org/10.1145/336512.336521
24. Fuggetta A, Ghezzi C (1994) State of the art and open issues in process-centered software engineering environments. J Syst Softw 26(1):53–60. https://doi.org/10.1016/0164-1212(94)90095-7
25. Furnas G (1999) The fisheye view: a new look at structured files. In: Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann Publishers Inc., San Francisco. pp 312–330. isbn = 1558605339
26. Haiduc S, Bavota G, Oliveto R, De Lucia A, Marcus A (2012) Automatic query performance assessment during the retrieval of software artifacts. In: 2012 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012 - Proceedings, pp 90–99. https://doi.org/10.1145/2351676.2351690
27. Hajmoosaei M, Tran H, Percebois C, Front A, Roncancio C (2015) Towards a change-aware process environment for system and software process. In: ACM International Conference Proceeding Series, Association for Computing Machinery, vol 24-26-August-2015, pp 32–41. https://doi.org/10.1145/2785592.2785596

28. Hasan M, Stroulia E, Barbosa D, Alalfi M (2010) Analyzing natural-language artifacts of the software process. In: IEEE International Conference on Software Maintenance, ICSM, pp 1–5. https://doi.org/10.1109/ICSM.2010.5609680
29. Kaptelinin V (2003) Umea: translating interaction histories into project contexts. In: Conference on Human Factors in Computing Systems - Proceedings, pp 353–360. https://doi.org/10.1145/642611.642673
30. Kersten M (2007) Focusing knowledge work with task context. PhD thesis. University of British Columbia, Vancouver. http://dx.doi.org/10.14288/1.0302110
31. Kersten M, Murphy G (2005) Mylar: a degree-of-interest model for ides. In: AOSD 2005: 4th International Conference on Aspect-Oriented Software Development - Conference Proceedings, pp 159–168. https://doi.org/10.1145/1052898.1052912
32. Kersten M, Murphy G (2006) Using task context to improve programmer productivity. In: Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp 1–11. https://doi.org/10.1145/1181775.1181777
33. Klein H, Myers M (1999) A set of principles for conducting and evaluating interpretive field studies in information systems. MIS Q Manag Inf Syst 23(1):67–94
34. Likert R (1932) A technique for the measurement of attitudes. Arch Psychol 22(140):55
35. Maalej W, Ellmann M, Robbes R (2017) Using contexts similarity to predict relationships between tasks. J Syst Softw 128:267–284. https://doi.org/10.1016/j.jss.2016.11.033
36. Maciel R, Gomes R, Magalhães A, Silva B, Queiroz J (2013) Supporting model-driven development using a process-centered software engineering environment. Autom Softw Eng 20(3):427–461. https://doi.org/10.1007/s10515-013-0124-0
37. Maciel R, Magalhães Mascarenhas A, Gomes R, De Queiroz J (2014) Supporting model-driven development: key concepts and support approaches. IGI Global. https://doi.org/10.4018/978-1-4666-6026-7.ch009
38. Matinnejad R, Ramsin R (2012) An analytical review of process-centered software engineering environments. https://doi.org/10.1109/ECBS.2012.11
39. Meedeniya D, Rubasinghe I, Perera I (2019) Traceability establishment and visualization of software artefacts in devops practice: a survey. Int J Adv Comput Sci Appl 10(7):66–76
40. Melo G, Alencar P, Cowan D (2019) Context-augmented software development in traditional and big data projects: literature review and preliminary framework. In: Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019, Institute of Electrical and Electronics Engineers Inc., pp 3449–3457. https://doi.org/10.1109/BigData47090.2019.9006245
41. Moreno L (2014) Summarization of complex software artifacts. In: 36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings, Association for Computing Machinery, pp 654–657. https://doi.org/10.1145/2591062.2591096
42. Murakami N, Masuhara H, Aotani T (2014) Code recommendation based on a degree-of-interest model. In: 4th International Workshop on Recommendation Systems for Software Engineering, RSSE 2014 - Proceedings, Association for Computing Machinery, Inc, pp 28–29. https://doi.org/10.1145/2593822.2593828
43. Murphy G (2009) Attacking information overload in software development. In: SBES 2009 - 23rd Brazilian Symposium on Software Engineering, p 15. https://doi.org/10.1109/SBES.2009.36
44. Omoronyia I, Ferguson J, Roper M, Wood M (2010) A review of awareness in distributed collaborative software engineering Vol. 40. pp 1107–1133. https://doi.org/10.1002/spe.v40:12
45. Ossher J, Sajnani H, Lopes C (2012) Astra: bottom-up construction of structured artifact repositories. In: Proceedings - Working Conference on Reverse Engineering, WCRE, pp 41–50. https://doi.org/10.1109/WCRE.2012.14
46. Osterweil L (1987) Software processes are software too. In: Proceedings - International Conference on Software Engineering. Institute of Electrical and Electronics Engineers Inc., New York. pp 2–13
47. Pitangueira Maciel R, Magalhães Mascarenhas A, Gomes R, De Queiroz J (2017) Supporting model-driven development: key concepts and support approaches. IGI Glob. https://doi.org/10.4018/978-1-5225-3923-0.ch016
48. Ploesser K, Janiesch C, Recker J, Rosemann M (2009) Context change archetypes: understanding the impact of context change on business processes. In: ACIS 2009 Proceedings - 20th Australasian Conference on Information Systems. Monash University, Melbourne. pp 225–234
49. Portela C, Vasconcelos A, Oliveira S, Silva A, Elder S (2014) Spider-pe: a set of support tools to software process enactment. In: Proceedings of the 9th International Conference on Software Engineering Advances
50. Portugal I (2014) Aiding software process execution with artifact filtering, degree of interest function and task context. Master's thesis. Federal University of Rio de Janeiro, Rio de Janeiro
51. Portugal IDS, Oliveira TC (2013) Introducing software process specification to task context. In: Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering, SEKE, Vol. 2013. Knowledge Systems Institute Graduate School, Skokie. pp 22–25
52. Portugal IS, de Oliveira TC (2014) Using task contexts to improve software process execution. In: CIBSE 2014: Proceedings of the 17th Ibero-American Conference Software Engineering. Universidad de la Frontera, Temuco, Araucanía. pp 109–122
53. Rastkar S, Murphy G, Murray G (2010) Summarizing software artifacts: a case study of bug reports. In: Proceedings - International Conference on Software Engineering, vol 1, pp 505–514. https://doi.org/10.1145/1806799.1806872
54. Reis R, Lima Reis C, Schlebbe H, Nunes D (2002) Automatic verification of static policies on software process models. Ann Softw Eng 14(1-4):197–234. https://doi.org/10.1023/A:1020509809235
55. Robson C, McCartan K (2016) Real world research, 4th edn. John Wiley & Sons, Inc, Hoboken
56. Salman I, Misirli A, Juristo N (2015) Are students representatives of professionals in software engineering experiments?. In: Proceedings - International Conference on Software Engineering, IEEE Computer Society, vol 1, pp 666–676. https://doi.org/10.1109/ICSE.2015.82
57. Selic B (2003) The pragmatics of model-driven development. IEEE Softw 20(5):19–25. https://doi.org/10.1109/MS.2003.1231146
58. Sievi-Korte O, Beecham S, Richardson I (2019) Challenges and recommended practices for software architecting in global software development. Inf Softw Technol 106:234–253. https://doi.org/10.1016/j.infsof.2018.10.008

59. Singh B, Gautam S (2017) The impact of software development process on software quality: a review. In: Proceedings - 2016 8th International Conference on Computational Intelligence and Communication Networks, CICN 2016, Institute of Electrical and Electronics Engineers Inc., pp 666–672. https://doi.org/10.1109/CICN.2016.137
60. Spanoudakis G, Zisman A (2005) Software traceability: a roadmap. World Scientific Publishing Co. https://doi.org/10.1142/9789812775245_0014
61. Sundaram S, Hayes J, Dekhtyar A, Holbrook E (2010) Assessing traceability of software engineering artifacts. Requir Eng 15(3):313–335. https://doi.org/10.1007/s00766-009-0096-6
62. Wong K (2013) A domain-dependent approach to determining file importance. Simul Ser 45:1–6
63. Yan S, Yu H, Chen Y, Shen B, Jiang L (2020) Are the code snippets what we are searching for? A benchmark and an empirical study on code search with natural-language queries. In: SANER 2020 - Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering, Institute of Electrical and Electronics Engineers Inc., pp 344–354. https://doi.org/10.1109/SANER48275.2020.9054840
64. Zhu YM (2017) Software failure mode and effects analysis. In: Failure-Modes-Based Software Reading. Springer. pp 7–15. https://doi.org/10.1007/978-3-319-65103-3_2

Publisher's Note