



MYLYNSDP: AIDING SOFTWARE PROCESS EXECUTION WITH ARTIFACT  
FILTERING, DEGREE OF INTEREST FUNCTION AND TASK CONTEXT

Ivens da Silva Portugal

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Toacy Cavalcante de Oliveira

Rio de Janeiro  
Junho de 2014

MYLYNSDP: AIDING SOFTWARE PROCESS EXECUTION WITH ARTIFACT  
FILTERING, DEGREE OF INTEREST FUNCTION AND TASK CONTEXT

Ivens da Silva Portugal

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO  
LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE)  
DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS  
REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM  
CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

---

Prof. Toacy Cavalcante de Oliveira, D.Sc.

---

Prof<sup>a</sup>. Cláudia Maria Lima Werner, D.Sc.

---

Prof. Leonardo Gresta Paulino Murta, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

JUNHO DE 2014

Portugal, Ivens da Silva

MylynSDP: Aiding Software Process Execution with Artifact Filtering, Degree of Interest Function and Task Context / Ivens da Silva Portugal – Rio de Janeiro: UFRJ/COPPE, 2014.

XII, 119 p.: il.; 29,7 cm.

Orientador: Toacy Cavalcante de Oliveira.

Dissertação (Mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2014.

Referências Bibliográficas: p. 96-102.

1. Processo de Software. 2. Execução de Processo. 3. Gerência de Execução do Processo de Software. 4. Automação de Execução do Processo de Software. I. Toacy Cavalcante de Oliveira II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

# Acknowledgements

I would like to thank my parents Alberto and Miriam and my brother Erlon for supporting me throughout all of the years of this Master's Degree. They highly contributed to the conclusion of this study by encouraging me on the good moments and assisting me on the difficult ones. Their valuable teachings about responsibility, perseverance and friendliness definitely made the difference during the most significant phases passed on this Master's Degree study.

Moreover, I would like to thank the relatives of my family. They have always taken part in my entire life and they have certainly collaborated on my education and wellness. These two key factors were unquestionably decisive during several ups and downs that I went through during the study of this Master's Degree.

Furthermore, I would like to thank my advisor Prof. Toacy for patiently guiding me towards this milestone in my academic life. Indubitably, his vision and experience led me to the right path when hope for a solution seemed to vanish or when the best next step in the study was yet unknown. His peculiar orientation, which is a mixture of responsibility, entertainment and creativity, has indeed taken part on the efforts required to finish this Master's Degree.

In addition, I would like to thank every professor that directly or indirectly contributed to my education, especially the ones I got in touch with during the classes and studies for this degree. The concepts taught helped me not only in academic life, but also in professional and even on personal life. I assure that these concepts will remain within my experience for an indefinite amount of time.

Last, but not least, I would like to thank every single friend I interacted with during this Master's Degree years. This includes friends I met during Bachelor's Degree as well as the ones I met during this Master's Degree. This also includes friends related to academic life and other friends that I fortunately met in the course of my life. Every conversation, discussion and giggle was totally useful to either relieve the stress caused by this study or enhance my strengths to keep studying until the end of this Master's Degree.

## **Agradecimentos**

Eu gostaria de agradecer aos meus pais Alberto e Miriam e meu irmão Erlon por ter me apoiado durante todos os anos do Mestrado. Eles fortemente contribuíram para a conclusão desse estudo me incentivando nos bons momentos e me ajudando nos mais difíceis. Seus valiosos ensinamentos sobre responsabilidade, perseverança e amizade definitivamente fizeram a diferença durante as fases mais significantes do estudo deste Mestrado.

Além disso, eu gostaria de agradecer aos meus familiares. Eles sempre participaram de toda a minha vida e eles certamente colaboraram na minha educação e bem-estar. Esses dois fatores foram inquestionavelmente decisivos durante vários altos e baixos que eu tive durante o estudo desse Mestrado.

Ainda, eu gostaria de agradecer ao meu orientador Prof. Toacy por pacientemente me orientar para esse marco na minha vida acadêmica. Sem dúvidas, sua visão e experiência me levaram para o caminho correto quando a esperança por uma solução parecia sumir ou quando o melhor próximo passo do estudo ainda era desconhecido. Sua orientação singular, que é uma mistura de responsabilidade, divertimento e criatividade, contribuiu de fato nos esforços necessários para terminar esse Mestrado.

Adicionalmente, eu gostaria de agradecer a todo professor que direta ou indiretamente contribuiu para a minha educação, especialmente aqueles que eu tive contato durante as aulas e estudos para esse Mestrado. Os conceitos ensinados me ajudaram não somente na vida acadêmica, mas também na vida profissional e até na vida pessoal. Eu tenho certeza que esses conceitos permanecerão comigo por tempo indeterminado.

Por último, mas não menos importante, eu gostaria de agradecer a todo amigo com quem eu interagi durante os anos de Mestrado. Isso inclui amigos que eu conheci durante a Graduação bem como amigos que eu conheci durante o Mestrado. Isso também inclui amigos relacionados a vida acadêmica e outros amigos que felizmente eu conheci no curso da minha vida. Toda conversa, discussão e risada foi totalmente útil para aliviar o estresse causado pelo estudo ou para aumentar minhas forças para continuar estudando até o fim desse Mestrado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

MYLYNSDP: AUXILIO A EXECUÇÃO DE PROCESSO DE SOFTWARE COM  
FILTRAGEM DE ARTEFATOS, FUNÇÃO DE GRAU DE INTERESSE E CONTEXTO  
DE TAREFA

Ivens da Silva Portugal

Junho/2014

Orientador: Toacy Cavalcante de Oliveira

Programa: Engenharia de Sistemas e Computação

Executar processos de software pode ser difícil quando o número de artefatos é alto. Nesse caso, ao executar uma atividade, engenheiros de software devem procurar determinados artefatos entre muitos outros disponíveis. O conjunto de artefatos relacionados a uma atividade é chamado contexto de atividade. Sua busca pode ser exaustiva, propensa a erro e demorada. Além disso, a execução de uma atividade pode ser interrompida por outra prioritária ou por execução em paralelo, resultando em uma troca de contexto. Esse problema afeta a produtividade do engenheiro de software pois ele investe tempo e esforço adicionais em trabalhos de suporte em vez da execução da atividade. Uma função de grau de interesse (DOI) é um mecanismo que pontua e destaca elementos de acordo com regras predefinidas. Ela é útil para descobrir o contexto de atividade. A implementação de uma função DOI pode ser encontrada em Mylyn. Porém, a função DOI do Mylyn é voltada apenas para tarefas de implementação e não considera um processo de software. Então, essa Dissertação de Mestrado propõe uma modificação na execução de processo de software com a utilização de uma função DOI para auxiliar engenheiros de software na localização de artefatos relevantes para uma atividade. A função DOI proposta é uma extensão da função DOI do Mylyn e lida com atividades e artefatos de todas as fases do processo. Além disso, ela é sensível ao processo pois considera o processo de software em seu funcionamento. A implementação final foi nomeada MylynSDP. Um estudo de validação foi conduzido para avaliar os conceitos discutidos nesse trabalho.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

MYLYNSDP: AIDING SOFTWARE PROCESS EXECUTION WITH ARTIFACT  
FILTERING, DEGREE OF INTEREST FUNCTION AND TASK CONTEXT

Ivens da Silva Portugal

June/2014

Advisor: Toacy Cavalcante de Oliveira

Department: Computer Science and Systems Engineering

Software process executions may be complex when the number of artifacts is high. In that case, to execute a software process activity, software engineers must search for suitable artifacts among several other available ones. The set of artifacts related to the execution of an activity is called activity context. The search for an activity context may be tiring, error-prone and time consuming. Moreover, activity execution may be interrupted by high priority activities or parallel execution, which results in a context change. That problem affects software engineers' productivity because they spend additional time and effort on support work rather than activity execution. A Degree of Interest (DOI) function is a mechanism that scores and highlights elements according to predefined rules. It is useful to discover the context of an activity. An implementation of a DOI function can be found on Mylyn. However, Mylyn's DOI function is aimed at implementation tasks only and it does not take into consideration the underlying software process that guides the development of the software product. Thus, this Master's Degree Dissertation proposes a modification in software process execution with the use of a DOI function in order to help software engineer better locate artifacts relevant to a software process execution activity. The proposed DOI function is an extension of Mylyn's DOI function and deals with activities and artifacts from all phases of software process. Moreover, the proposed DOI function is process-aware because it takes into consideration the executing software process in its workings. The final implementation was named MylynSDP. A validation study has been conducted to assess the concepts discussed in this work.

# TABLE OF CONTENTS

1	Introduction .....	1
1.1	Motivation .....	1
1.2	Problem .....	2
1.3	Goals .....	5
1.4	Work Methodology .....	6
1.5	Organization .....	6
2	Theoretical Foundation .....	8
2.1	Introduction .....	8
2.2	Software Process .....	8
2.3	Software Process Representation .....	11
2.3.1	BPMN .....	12
2.4	Conclusion .....	15
3	Related Work .....	16
3.1	Introduction .....	16
3.2	Presto and Placeless Projects .....	17
3.3	Task Tracer .....	18
3.4	UMEA .....	19
3.5	PSEE .....	20
3.6	WebAPSEE .....	21
3.7	TABA Station .....	23
3.8	Software Traceability .....	25
3.9	Mylyn .....	26
3.9.1	Overview .....	26
3.9.2	Characteristics .....	28
3.10	Conclusion .....	39
4	MylynSDP .....	40
4.1	Introduction .....	40
4.2	Overview .....	40
4.3	Concepts .....	42
4.4	Characteristics .....	44
4.4.1	Interface .....	45
4.4.2	Software Process Specification Import Mechanism .....	48
4.4.3	Recovery Mechanism .....	50
4.4.4	DOI function .....	54
4.4.5	Saving Mechanism .....	70
4.5	Conclusion .....	72
5	Validation Study .....	74
5.1	Introduction .....	74
5.2	Software Process .....	75
5.3	Participants .....	76
5.4	Training .....	77
5.5	Validation Study Exercises .....	78
5.6	Technology Acceptance Model .....	81
5.7	Analysis .....	82
5.8	Threats to Validity .....	87
5.9	Conclusion .....	88
6	Conclusion .....	89

6.1	Conclusions .....	89
6.2	Limitations.....	93
6.3	Future Work.....	94
BIBLIOGRAPHIC REFERENCES .....		96
APPENDIX A – SIGA EPCT SOFTWARE PROCESS AUTHORIZATION OF UTILIZATION AND SPECIFICATION .....		103
A.1.	SIGA EPCT Software Process Authorization of Utilization .....	103
A.2.	SIGA EPCT Software Process Specification.....	104
APPENDIX B – VALIDATION STUDY DOCUMENTS .....		105
B.1	Consent Form.....	105
B.2	Characterization Questionnaire.....	106
B.3	Time Record Form .....	107
B.4	Validation Study Exercises.....	111
B.5	Final Questionnaire .....	115

# INDEX OF FIGURES

Figure 2.1 - A process modeled using BPMN notation.....	15
Figure 3.1 – Presto’s interface. Image extracted from (DOURISH <i>et al.</i> , 1999).....	18
Figure 3.2 - Task Tracer's interface. Image extracted from (DRAGUNOV <i>et al.</i> , 2005). .....	19
Figure 3.3 - UMEA's interface. Image extracted from (KATPELININ, 2003). ....	20
Figure 3.4 - Main Interface of Manager Console and Task Agenda. Image extracted from (REIS & REIS, 2007).....	23
Figure 3.5 - AdaptPro - a software process adaptation aid tool. Image extracted from (ROCHA <i>et al.</i> , 2005). ....	24
Figure 3.6 - A traceability matrix being used to trace software requirements.....	25
Figure 3.7 - Mylyn's components and their relationship. ....	29
Figure 3.8 - Mylyn's interface.....	30
Figure 3.9 - An example of Mylyn's DOI function usage. A task interaction process example is three steps on the left and a task interaction process example in two steps on the right.....	34
Figure 4.1 - An extract of an example of a software process. The same artifact symbol is used to specify several documents from the execution of a software process. ....	42
Figure 4.2 - MylynSDP's components and their relationship. ....	45
Figure 4.3 - MylynSDP's interface. ....	46
Figure 4.4 - New task creation wizard. ....	46
Figure 4.5 - New artifact creation wizard. ....	47
Figure 4.6 - An example of an importable software process specification XML document.....	49
Figure 4.7 - An example of a " <i>restore.xml</i> " document. ....	53
Figure 4.8 - Explanation of the concepts used to create an initial task context. ....	57
Figure 4.9 - Class diagram of a part of MylynSDP. Some classes are highlighted. ....	58
Figure 4.10 - An example of Mylyn's DOI function usage. A task interaction process example in four steps on the left and a task interaction process example in three steps on the right.....	67
Figure 4.11 - Folder structure managed. ....	70
Figure 5.1 - Percentage of answers given by participants on the questionnaire. ....	84

## INDEX OF TABLES

Table 2.1 - Main elements from EPC notation.....	11
Table 2.2 – Main elements from SPEM notation. ....	12
Table 2.3 - Main elements of BPMN notation.....	13
Table 3.1 - Data about one interaction event captured by Mylyn .....	35
Table 3.2 - Types of interaction captured by Mylyn.....	36
Table 4.1 – Main methods of MylynSDPRestoreXML class .....	51
Table 4.2 - MylynSDP's types of interaction event. ....	55
Table 4.3 - Interaction event types and their scores.....	57
Table 4.4 - Summary of the description of the six algorithms characterized on this section. ....	69
Table 5.1 - Exercises of validation study. ....	80
Table 5.2 - Statements of the validation study alongside the metrics observed.....	82
Table 5.3 - First goal of validation study. ....	83
Table 5.4 - Second goal of validation study.....	83
Table 5.5 - Execution times for each of the participants on each of the exercises.....	86
Table 6.1 - MylynSDP's features and related work's drawbacks put in a nutshell for comparison purposes. ....	90

## INDEX OF ALGORITHMS

Algorithm 3.1 - Algorithm of Mylyn's Degree of Interest (DOI) function.....	34
Algorithm 4.1 - Recovery Mechanism code snippet. ....	54
Algorithm 4.2 - InteractionContextScaling class' code snippet.....	59
Algorithm 4.3 - InteractionContextManager class' code snippet.....	60
Algorithm 4.4 - InteractionContext class' code snippet.....	61
Algorithm 4.5 - InteractionEvent class' code snippet.....	62
Algorithm 4.6 - InteractionContextElement class' code snippet.....	63
Algorithm 4.7 - DegreeOfInterest class' code snippet. ....	64
Algorithm 4.8 - Code snippet with the registration of a Specification interaction event.	72

# 1 Introduction

*This chapter introduces the context of this work, the motivation for this research and the research question. It also introduces the objectives, the methodology and the organization of this text.*

## 1.1 Motivation

Nowadays, computer software plays an important role in society daily life. At homes, word processors help students to write school papers, spreadsheets help families to create household monthly budget and games entertain people of all ages. In the case of Academic or Business Organizations, computer systems help them with task automation, report generation and staff control, for example. The dependence on computer software became high and beneficial. Complex activities, which used to demand a great deal of effort, high level of attention, sheer number of employees and significant financial resources, are currently easily performed with some clicks due to the ease provided by technology and suitable software. Moreover, the number of workers allocated to a given task lowered, due to the fact that several tasks may now be automated (PARASURAMAN & RILEY, 1997). This also led to a low usage of financial resources. The advantages of the use of computer software are numerous. However, software development may not be an easy task. Depending on the functionality that will be offered by the computer software, software development can be slow, complex and difficult to be tested. These factors contribute to software failures and make users' job harder instead of helping them. If a database management system reports a flaw and fails to record an Organization's major sell, all Organization's income and outcome calculations are compromised, which leads to employees rework and stress. While regarding software complexity and possible negative scenarios that a bad software behavior may cause, software engineers aimed their efforts to define policies to ensure high quality software development and thus lower the chances of software faults. One of the research directions, focused to increase software quality, is related to the software process on which software development is based.

In order to increase the chances of developing software with quality, a software process is often used. A software process can be defined as the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are

needed to conceive, develop, deploy and maintain a software product (FUGGETTA, 2000). In other words, a software process is comprised of a set of activities, the relationship between them and associated artifacts, which are useful to the activities' execution. Software processes are described, modeled and, once they are finished, they are executed, that is, each activity is carried on. The work performed on each activity, their order and the use and production of the suitable artifacts yield a software product. As an example, a software process activity can be "collect system requirements" and it can have the artifacts "requirements questionnaire" and "system development contract" as associated artifacts. From the execution of this activity, an artifact is produced: "system requirement list".

A software process is not always simple and easy to be executed. Some software process executions are highly complex and difficult to deal with. This requires additional attention from the software engineer, the specialist charged to manage software process executions. One of the factors that increase the complexity of software process executions is the number of existing activities and artifacts, either consumed or produced, which can be too high. For example, the Rational Unified Process (RUP), owned by IBM, defines more than a hundred types of artifacts (IBM, 2012). Complex cases like this one makes software process executions difficult to be managed, increases the chances of failure and negatively affects the quality of the final software product (ANNOSI *et al.*, 2008).

## **1.2 Problem**

When executing one of the activities of a software process, software engineers access and use some suitable artifacts. The set of artifacts related to the execution of an activity, either because artifacts were needed or produced during the execution, is called an activity context, or just context (KERSTEN & MURPHY, 2006). Usually, an activity context is a small subset of all available artifacts. In order to find an activity context during a software process execution, software engineers need to perform a search on several available artifacts. However, two concerns should be addressed. The first one relates to artifact search. As artifacts that belong to an activity context are not distinguished from other artifacts that are not important at the given moment, software engineers must deal with an excess of artifacts on his work environment when building the context for an activity, which can be tiring, error-prone and time consuming. This is an important issue for software engineers because it may affect their productivity. When browsing a sheer number of available artifacts in order to find the suitable ones for the execution of an activity, software engineers may spend additional time and effort

on the search of artifacts rather than on the work of the activity itself. Once they have all or some of the suitable artifacts, they can then start the execution of that activity.

A second concern during the execution of a software process relates to the change of the activity being executed, and thus the change of its context. The execution of a given activity may be interrupted if a high priority activity arises. The artifacts a software engineer is using may be closed, in order to give place to the new activity context. When the software engineer returns to the previous activity, he then needs to perform a search to recover the artifacts he was using. That requires time and effort that are not used to the execution of the activity, but to supporting tasks. Whenever a context change happens, the search for an activity context is necessary, and then the productivity of the software engineer is negatively affected.

A possible solution for the artifact search and the context change problems is the use of a degree of interest (DOI) function (KERSTEN, 1999). A DOI function is a mechanism that scores elements according to some predefined rules. By doing this, a DOI function is able to distinguish which elements are relevant from other elements that are not and thus helps its user in different ways. Software engineers may benefit from a DOI function during the search of suitable artifacts for the execution of a particular activity. A DOI function is able to score artifacts with interest points that indicate that a particular artifact is interesting for the execution of an activity. After scoring artifacts, DOI function is able to select artifacts that have the highest points. They are considered to be relevant to the execution of that activity. Once DOI function has the set of interesting artifacts, it can highlight them, which then creates the context of that activity, and show them to software engineers.

Moreover, the DOI function helps Mylyn to persist each set of artifacts related to a given task to disk. As a result, software engineers may be able to switch from one activity to another without the need to worry about difficulties in restoring the activity context that was being used before the interruption. The assistance provided to software engineers avoids wasting time with support work and increases the time they spend working on the effective work of the activity, which improves his productivity.

An implementation of a DOI function can be found on a work done by researchers of the University of British Columbia, Canada. They foresaw both artifact search and context change problems on the software implementation field. Therefore, the group of researchers launched on 2005 an open-source application aimed at aiding programmers that use the Eclipse platform. The application was named Mylyn (KERSTEN & MURPHY, 2005). Mylyn is an Eclipse plugin, which implements a DOI function that deals with Java class search and task change problems in the implementation step of a software development. Mylyn uses the concept of task

context. A task context is the set of all classes, either used or created, during the execution of a coding task. For instance, in order to change the functionality of a class method, a programmer needs to check and edit some other classes on which that method is used. The set of classes edited in this task is said to be the context of that task.

Mylyn's DOI function automatically creates a task context based on how programmers interact with the code. Initially, a programmer selects the task that will be performed among all other tasks that can be performed, in an Eclipse view. During the execution of the task, several classes and methods are created, edited, accessed or deleted. Based on the frequency of selection or edition of those classes, Mylyn's DOI function calculates their interest value in relation to the execution of the current task. In other words, if the programmer selects a given class, its interest value is increased, which means that this class is important to that task. The same behavior applies to class edition during a task execution. When all artifacts of the project have an interest value associated, DOI function is able to select the most important classes for that task. By creating the set of classes most relevant to a task execution, DOI function creates the context of that task.

Coding tasks may also be interrupted by another task, which forces the programmer to change the task being executed and thus to lose the current task context. Mylyn also persists every task context to disk. So, whenever a task change happens, programmers are able to recover the interrupted task context effortlessly and within a shorter amount of time than if they had to recover it without the aid of Mylyn's DOI function.

The context of a task is used by Mylyn to facilitate the search and recovery of suitable classes related to the execution of a particular task. Mylyn's interface aims to display classes in a way to simplify their visualization and in a task-based way (GOTH, 2009).

Nevertheless, Mylyn and its DOI function solve the class search and the context change problems for the implementation phase of the development of a system. Other phases such as planning, modeling, requirements definition and testing, are not supported by the concepts used by that mechanism. In addition to that, task definition is done in an ad-hoc way. That means that every task is created as the need for that task arises. A more interesting approach is to consider the software process on which the development of the system is based as the starting point of the discovery of tasks to be executed as well as what artifacts are related to an activity (or task). This is possible because a software process usually relates activities and artifacts, thus highlighting the importance of some project artifacts on the execution of activities. In

other words, the tasks or activities performed by the programmer may be based on the software process.

### **1.3 Goals**

This Master's Degree Dissertation proposes a way to execute a software process by aiding software engineers in the search for artifacts to execute a software process activity with the use of a DOI function. A DOI function is able to solve artifact the search problem and the context change problem in all phases of the execution of software development processes by scoring artifacts during execution of activities based on the interaction software engineers have with them. The more an artifact suffers interactions, the more it is interesting to the context of the current activity. Thus, from any phase of the software process execution, activities will have their most important artifacts highlighted from the other artifacts by a DOI function in order to facilitate artifact search. Moreover, context of activities may be persisted in order to aid software engineers during the recovery of the most important artifacts after a context change. These two features, scoring artifacts and persisting context activities, help software engineers to focus on the activity to be executed rather than support work (such as the search of suitable artifacts). In addition to the two features just mentioned, this work also aids software engineers during the execution of a software process with the introduction of the ability to be process-based to the DOI function. This means that DOI function is able to take into consideration the relation between activities and artifacts already described on software processes. Hence, the initial definition of activity contexts can be drawn from the underlying software process being executed.

In short, this Dissertation aims at improving productivity of software engineers with three main features:

- A solution to the artifact search problem with the score of software process artifacts done by DOI function extended from Mylyn
- A solution to the context change problem by persisting activity contexts for later retrieval done by the same DOI function
- The ability to access software process specification to DOI function in order to better create activity contexts.

An implementation of the concepts discussed in this work was made in the form of an Eclipse plugin. The proposed DOI function is an extended version of Mylyn's DOI function and the final implementation was named MylynSDP. In order to validate the concepts introduced in this work, a validation study has been conducted.

## **1.4 Work Methodology**

The study and the work proposed and done on this Dissertation followed a methodology that is divided in problem identification, problem investigation, solution, validation and a final Dissertation to conclude the studies (PEFFERS *et al.*, 2007). The problem, which relates to the sheer number of artifacts available to deal with when executing a software process and how it affects the work of software engineers, was identified based on artifact handling problems reported in (ANDERSON *et al.*, 2002, ANTONIOL *et al.*, 2002, ASUNCION *et al.*, 2010, MURPHY, 2009). Following this, an informal review of literature was performed in order to find out projects that have dealt with similar problems. Projects found were significantly important because they helped to get insights about the problem and its possible solutions as well as ideas for future developments. During literature review, one project is pointed out: Mylyn project. When studying this project, it was acknowledged that it partially solved the problems identified for this Dissertation with the use of a Degree of Interest (DOI) function. Therefore, some opportunities for extension on Mylyn work along with its DOI function were identified and the work of this Dissertation begun. Next steps were to access and download Mylyn's code and study it. This phase was particularly meticulous due to the size of Mylyn project, in terms of code, which is comprised of more than 200 Java projects and even more Java packages. After identifying suitable parts of the code appropriate to be edited and to directly help software engineers, a new code was implemented and tested. The first publication of an international short paper was done, right after that step, explaining the work done and the implications of the study. Next, a validation study was conducted with some software engineers in order to assess concepts introduced and to get feedback from people with different experiences. Once again, an international full paper was prepared and published with the results of the validation. Finally, this Dissertation was written with details from the research studied, the work done and the results concluded.

## **1.5 Organization**

The remaining of this Dissertation is divided in five more chapters. In Chapter 2, this work explains the concepts that comprise the theoretical foundation needed to understand this document. It is mainly focused on the underlying explanation of a software process, its history, its importance and notations used to create one. In Chapter 3, related works that helped in the conception of this Master's Degree work are presented. Also, in Chapter 3, there is an image of the interface of each related work describing them. Moreover, a paragraph is dedicated to the limitations of each project

found. Due to its importance, Mylyn project is described in a separate section of this Chapter. In Chapter 4, the work of this Master's Degree, named MylynSDP, is presented and detailed. Modifications made to the original Mylyn are explained and how each component Interacts with each other. Also in this chapter, DOI function is introduced, its code is displayed and it is explained with examples. In Chapter 5, the validation study is presented and discussed. Therefore, this chapter includes details about participants, documents, exercises performed and how the validation study has been conducted. Finally, in Chapter 6, a conclusion is draw and presented. In addition to it, a final consideration is made and limitations of this work are discussed. Future work associated with concepts studied is also listed on this chapter with expectations of further contributions to the software engineering field.

## 2 Theoretical Foundation

*On this chapter, the theoretical foundation for comprehension of the concepts to be discussed in this Dissertation is introduced. The definition of a software process, its importance, main existing notations and their characteristics are mostly discussed.*

### 2.1 Introduction

Software processes are created and executed everyday by software engineers on Organizations following some guidelines. However, according to (FUGGETA, 2000), software processes date back to the late 70s when computer specialists were discussing the creation of a software lifecycle. The importance of software processes on the development of software is understood when studying the steps that were performed in the history until the creation and popularization of software processes among software engineers. This is particularly important for this Dissertation because its main focus is on the role software engineers play when executing a software process having their work based on software process specifications. Concepts such as activities and artifacts are presented from the perspective of three main notations used to create software processes: EPC, SPEM and BPMN. This chapter is divided in three other sections. Section 2.2 shows the history, importance and definition of software processes, the underlying concepts of this Dissertation. Section 2.3 describes the main elements from three notations used to model software processes. Section 2.4 concludes this Chapter.

### 2.2 Software Process

Some decades ago, the majority of daily life activities were performed without the aid of a computer. For instance, in order to make cash withdrawal, clients had to wait in a line for some time, and after that, they should ask a bank representative to withdraw the money. The bank representative then checked some paper profiles with personal information about clients, organized alphabetically in files and, after checking that no problem was found on the profile of those clients, he allowed the withdrawal to be made by giving the money to the clients along with information about the new account balance. All of these actions used to take a lot of time and few of them were

automated. Similarly, a project approval process inside an Organization used to be costly in terms of time and financial resources, because it used to take time and thus delay the financial income. The manager needed to sign a lot of papers, send several documents to numerous departments, which sometimes were geographically distant. Besides, a staff member, specifically allocated for the matter, carefully used to do the document archive and management process.

Technology development allowed the arrival of computers, and alongside it, computer systems that could help practitioners to better organize work information and automate some repetitive tasks. Nowadays, cash withdrawals can be made relatively quickly on one of the several automated teller machines (ATMs) spread all over the city. Starting from the decision of a client's decision to make cash withdrawal, the ATM embedded system check the client's bank account data included on the ATM card's chip and send them, through a secure internet connection to a centralized system situated miles away. The distant system authenticates client's data and allows the money to be withdrawn. The entire process, which used to be manually performed, is now fully automated and done in seconds. A project's approval process also benefited from the arrival of specific computer aided systems. With a single click, a manager automatically sends documents to his workers, no matter where they actually are, without leaving his office. Furthermore, digital signatures allow managers to sign and authenticate documents in a simple and efficient way. Other existing computer systems on Organizations also aid staff members on document organization, archive and recover. Within some seconds, a work report made for a given month is persisted in a database, accessible by stakeholders and easily recoverable.

The benefits introduced by technology evolution are countless and they really changed they way society deals with information. Gadgets, such as computers, mobile phones or tablets, with computer software running on them, help the execution of daily tasks, which slowly makes society dependent on these systems.

However, in order to maintain computer systems as a tool that provides easy and quick ways to perform tasks, professionals that develop them, i.e. software engineers, must have a well developed logical thinking in order to find the best solution to task automation. They also have to have good perception on user's needs to develop intuitive interfaces and have discipline to implement the software code using a programming language. Aiming at organizing the development of a software product, researches made in the 60's and 70's were focused on the creation of a software lifecycle (FUGGETTA, 2000, MARCINIAK, 2002).

A software lifecycle defines each different step a system may undertake during its development and usage, or lifecycle. Usually, these steps are Requirements

Analysis and Specification, Design, Implementation, Verification and Validation, Delivery, Maintenance and Retirement (FUGGETTA, 2000). These steps consist of both a high level guide and the principles that software engineers follow when executing software development activities.

Nevertheless, a software lifecycle does not define in details the flow of execution of tasks, tools, participants, restrictions, documenting policies, software delivery and information exchange between Organization and client during the development of software. Software lifecycle defines the steps to build a software product, but not the effective work that should be done in order to actually develop it with quality.

In addition to the lack of orientation, software lifecycles must deal with the software complexity problem. In most cases, the client does not know what he wants or needs, which makes the requirements gathering a difficult task. Moreover, the understanding of the software that is about to be developed may be complex, which results in the difficulty of elaboration of the description of its components, through either text or images. Implementation requires technical knowledge and advanced technologies, sometimes unavailable. Meeting performance requirements and other software metrics, although essential, demands an extra effort from software engineers. In short, two factors, the high complexity of computer software and the lack of orientation of its development, contribute to the unexpected and undesired behavior of computer software. In other words, the system may fail (BARJIS, 2008).

This problem has several negative consequences. A computer system that monitors the arrival and depart time of the staff of a company, when in failure, does not record important data about the work time of their workers. Thus, the calculation of the work time during the life of a project may be affected, positively or negatively, which results in financial complications to the Organization (in salary payments, for instance). Some of the loss caused by a software failure, for example, may happen in financial area, time management, communication of the Organization's members, denial of important data registration, or even the death of workers (when a failure happens in a system that monitors large structures, such as platforms).

As a result, researchers and industry professionals explore topics related to software quality with the aim of lowering the chances of a software failure, and as a consequence, lower the risks inherent to this event. One of the research directions is studying software development processes (also named software processes). It is believed that a well-developed and well-specified software process, which describes tasks, their relationships and the people involved by making use of a level of detail

understandable enough by all stakeholders, increases the chances of developing computer software with high quality (ASHRAFI, 2003, GREEN *et al.*, 2005).

According to (FUGGETTA, 2000) and (PRESSMAN, 2010), a software process may be defined as the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product. In other words, a software process is a wide and understandable concept of the activities, their relationships, the workflow and organizational factors that are related to the development of a computer system. The organization of the development of software in a software process view helps managing each one of the executed tasks and using suitable corrective measures in order to solve problems.

The efforts in defining software processes resulted in studies in the best practices area and frameworks to design software processes (AALST, 2007). Researchers created specific languages to deal with the modeling and specification of software processes, named notations (CAMPOS & OLIVEIRA, 2012). Furthermore, another group of researchers aimed at the definition of generic software processes, which are suitable to several cases, such as RAD (MARTIN, 1991), SCRUM (SCRUM.org, 2011) and RUP (IBM, 2012).

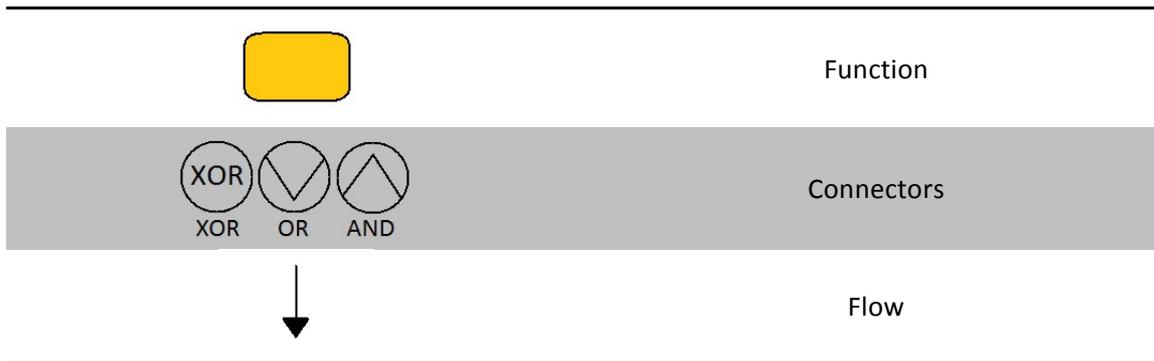
### 2.3 Software Process Representation

A software process is useful to specify the activities and their relationships, which will be executed to develop a software product. It is based on a software process that an Organization, its workers, and the client must understand each other in relation to the steps executed during the development of the software. The Academy created some notations and the most important ones are described here.

Event-driven Process Chain (EPC) (AALST, 1999) was created focused to general processes rather than specifically to software processes. This notation allows the representation of the entire Organization's structure as well as project elaboration and execution steps through drawings. To display them, a framework named ARIS (Architecture for Integrated Information Systems) is used. Table 2.1 presents the main elements used in process modeling by EPC.

Table 2.1 - Main elements from EPC notation.

Image	Name
	Event



SPEM (Software and Systems Process Engineering Meta-Model) notation (OMG, 2008) was created with the objective of defining software processes and their components. Differently from EPC, SPEM is aimed at a specific area: the software engineering domain. Therefore, SPEM is aimed at aiding software engineers on software process specification and development. In 2002, Object Management Group (OMG), the international regulator of open patterns to object-oriented applications, adopted SPEM and defined utilization patterns that are used until now. OMG defines SPEM as a UML profile, that is, a generic extension mechanism to customize domain-specific UML models. The main elements used by SPEM notation are illustrated on Table 2.2.

Table 2.2 – Main elements from SPEM notation.

Image	Name
	Work Product
	Work Definition
	Activity
	Process Role
	Document

### 2.3.1 BPMN

Software Processes are created and managed, so Institutions develop their work with efficiency, clarity and in an organized way (AALST *et al.*, 2003). They specify

the existing participants, their projects and their responsibility. This is quite similar to business processes and their management. For that reason, a business process feature used on the creation and management of this kind of process should be studied: its notation.

In order to aid the specification of a business process, the Business Process Management Initiative (BPMI) developed the Business Process Model Notation (BPMN) (OMG, 2011). Later, in 2002, OMG defined a standard for BPMN. Nowadays, BPMN is owned by OMG because in 2005, BPMI and OMG merged together. Since March 2011, the available release of BPMN is version 2.0.

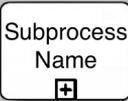
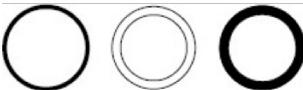
As it is written on the BPMN specification document, its main objective is to offer a notation that is easily comprehensible by all business users, from business analysts, which create the first version of processes, to technical developers, which are responsible for implementing the technology that will execute these processes, and finally to business professionals, which manage and monitor processes (OMG, 2011).

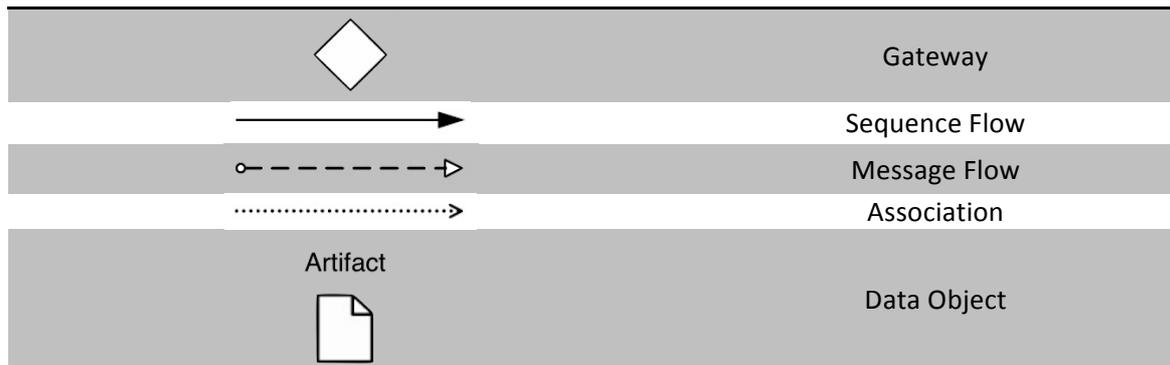
General process modeling is comprised of tasks and workflow. In addition, there are elements to represent departments and Organizations. Studies show that BPMN is one of the most suitable notations to represent software processes due to its simplicity and intuitiveness to model and understand processes (PILLAT *et al.*, 2012).

Thanks to the importance and ease of representing general processes using BPMN, studies are performed with the objective to provide resources not yet available to BPMN, but existent in other notations. It is the case of the refinement of processes, which is featured in SPEM notation (PILLAT *et al.*, 2012).

Table 2.3 exposes the main components of BPMN notation.

Table 2.3 - Main elements of BPMN notation.

Image	Name
	Pool
	Task
	Subprocess
 Start    Intermediate    End	Event



Pool represents an Organization. It is inside a Pool that processes will be modeled. Pools can represent not only the Organization that executes the process but also another Organization, along with data and artifacts that are exchanged during the execution processes.

Tasks performed during process execution are modeled with the use of Activity element. An Activity may represent, for example, the execution of tests on a software product to verify possible errors. The linking of several Activity elements in an execution flow inside a Pool that represents an Organization describes a process to be performed.

In some situations, the number of tasks in sequence is high, which affects the process understanding. For that reason, the Subprocess element groups some Activity elements on a lower level of detail, and then omits some process activities, which improves the process readability. The omitted tasks and the relationship between them are visible in a low level of detail.

An Event element represents an event that occurs during process execution and affects the workflow in any way. Events are classified as initial, which is the one who initiates a process, final, which is the one that finishes a process, and intermediary, which occurs between an initial and a final event, but does not finish a process execution. In order to control a process execution flow, one must use the Gateway element. This element diverges and converges execution flow. Tasks may need or produce data as a result of their execution. Contracts, codes, or any other artifact type, either textual or not, that are relevant to the process are represented by BPMN with Data Object element.

In order to link all elements explained until now, three elements are used: Sequence Flow, Message Flow and Association. A Sequence Flow element is used to connect Activity, Subprocess, Event and Gateway elements, and set the order that each task will be executed, that is, the software process execution flow. The Message Flow element is used when it is needed to represent the flow of messages between two

participants (Pool). Finally, an Association element connects an artifact and its respective producer or consumer element.

Figure 2.1 shows an example of a process modeled using BPMN notation.

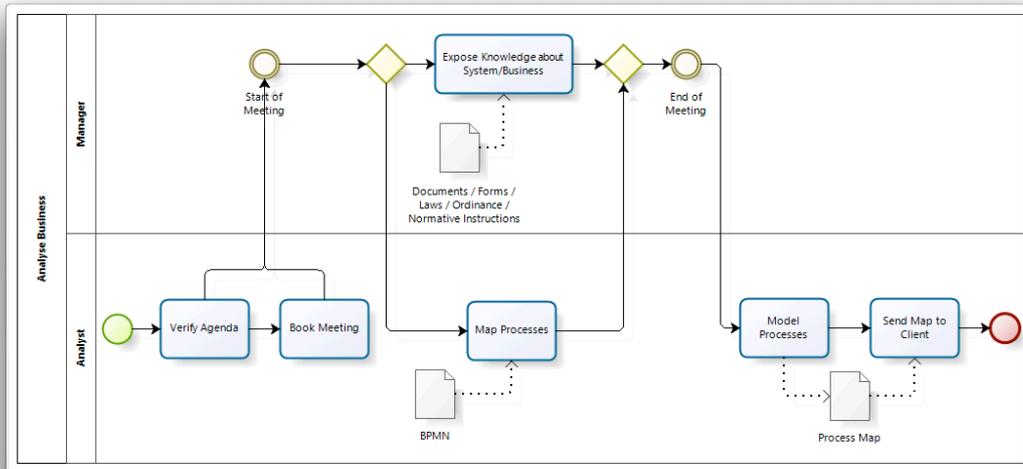


Figure 2.1 - A process modeled using BPMN notation.

## 2.4 Conclusion

Software engineering field is aimed at helping software developers to better develop their systems. Therefore, research and study is extensively performed in order to find ways and opportunities to improve software development in any way. This Dissertation proposes modifications that relates to software processes when developing computer software. Specifically, modifications deals with gathering data from software process specifications and manipulating their activities and artifacts. Therefore, this Dissertation has dedicated a chapter to explanations of concepts that surround processes such as their history, importance, formal definition and notations. Next chapter will continue investigation of proposed solutions to similar problems found on the literature with the description of an informal review. Right after that, this Dissertation exposes, describes and discusses the main study of this Master’s Degree, on Chapter 4.

## 3 Related Work

*On this chapter, it is presented an informal literature review of main research projects that were found during the elaboration of this Dissertation. They all comprise the related work of this Dissertation. Their concepts contributed to develop insights for this research. Seven research projects and Mylyn were investigated and they are described in the following sections.*

### 3.1 Introduction

A user, when executing activities on a computer, needs some related artifacts. For example, a researcher who is writing a paper about a specific topic needs a set of files, which are the articles he has read and he needs to read, a word processor to write the article, image files to add to the paper, sites with research related content. Similarly, a software engineer that deals with software process execution requires a list on a file with the main activities to be executed, requirement documents with essential requirements to be followed, use cases to be written and any other document related to the development phase the software is in. All of these documents are accessed by the software engineer from time to time in order develop the computer system.

As it was explained, the set of artifacts needed to the execution of an activity plus the artifacts produced on the execution of that activity forms an activity context. For activities that belong to a large and complex software process, with a great amount of specified artifacts, the activity context identification may be slow, tedious and, as a consequence, time-consuming. Researches on a way to better create task contexts are important and have been done.

In order to find relevant studies that dealt with problems similar to the ones discussed in this Dissertation, an informal literature review has been performed. Therefore, this Chapter is responsible for displaying the main projects found on the review and discussing their main concepts and limitations. These works were found after talks with close software engineers and after searches for projects that deals with similar concepts. This Chapter is divided in nine sections, besides this one. Each section describes a project studied. Section 3.2 is related to Presto and Placeless projects and Section 3.3 characterizes Task Tracer project. Section 3.4 details UMEA project. Section 3.5 explains the concept of Process-centered Software Engineer

Environment (PSEE), which is an important research field for software process related projects. Section 3.6 illustrates WebAPSEE, a project that uses concepts from PSEE. Section 3.7 analyzes TABA station project and Section 3.8 justifies the main reason for not using a Traceability Matrix to solve the problems of this Master's Degree research. Finally, Section 3.9 depicts Mylyn, the project on which this research was based. Section 3.10 concludes this Chapter.

## **3.2 Presto and Placeless Projects**

The concept of dealing with artifacts in a way to associate them with a context and facilitate their recovery to the execution of a task is used on Presto and Placeless from Xerox PARC (DOURISH *et al.*, 1999). Researchers from Xerox Palo Alto Research Center (PARC) started the actual ways of managing and organizing documents. According to the research done, there are three types of data associated to a document: its content, its properties and its localization on the system. A document's properties are data inherent to it and are barely explicitly described. For example, a given document is used by software engineers in a project. Data about which groups of professionals use that document or data about the project to which that document belongs is not properly described in a computer readable way. Researchers then concluded that currently the location of the document is mostly emphasized for its management, organization and recovery, rather than its characteristics, represented by its properties. By emphasizing a document's properties, contexts can be created, which are useful to the execution of tasks. The recovery of suitable documents for a specific project may be free of effort, as well as the gathering of documents produced by a group of software engineers on a given day. In projects developed by Xerox PARC, properties of documents are defined manually by the user in a key-value fashion through a tag mechanism as in "project=placeless". After setting the most relevant and important properties on all documents, documents and files related to Placeless project can be recovered by the use of a smart search on the file system and then be used on the execution of an activity. This approach creates a context for an activity by adding properties to the documents. Figure 3.1 shows Presto's interface. Although this way of dealing with documents is useful, it is necessary that all documents be manually classified. Moreover, at each new category creation, the entire set of documents that belong to that category may be updated.

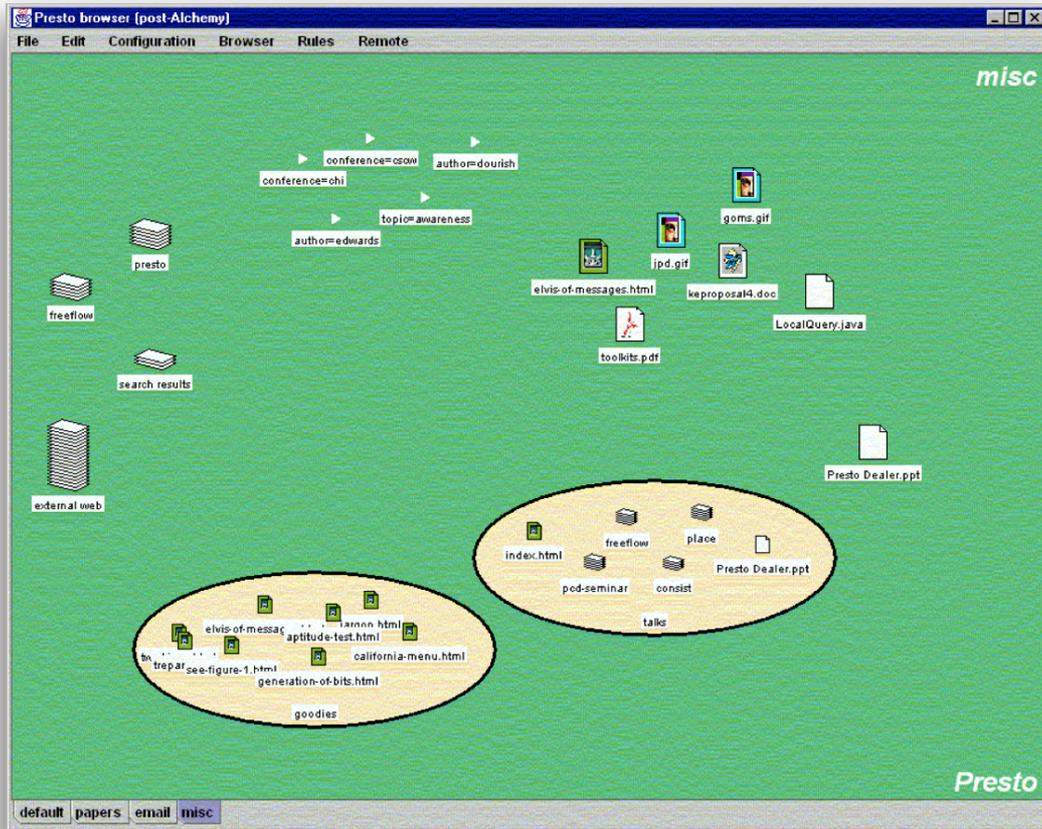


Figure 3.1 – Presto’s interface. Image extracted from (DOURISH *et al.*, 1999).

### 3.3 Task Tracer

Another related work is Task Tracer (DRAGUNOV *et al.*, 2005) developed in Oregon University. Researchers created a computer system that automatically builds the task context of the task being executed based on data collected from past and executing tasks. They named that task context as task profile. Task profile creation, or task context creation, also helps user on what is referred to as recovery after interruption. Thus, the less effort is required in the creation and recreation of contexts, which means more focus on the effective work of the task. Task Tracer monitors activities related to Microsoft Office, Visual Studio and Internet Explorer, it stores relevant data in a database and use them to infer a task context, or task profile. Figure 3.2 shows TaskExplorer’s interface, which is used to select tasks when dealing with TaskTracer. Although the majority of computer professional activities use tools from Microsoft Office package and Internet Explorer browser, activities related to software engineering are not like this. The integration between the development environment and the activities of a software engineer usually happens on other applications such as

Eclipse IDE and diagram modelers and deals with several types of artifacts, such as documents, spreadsheets, diagrams and images. Task Tracer does not support this type of activity, which makes it inadequate for that type of professional.

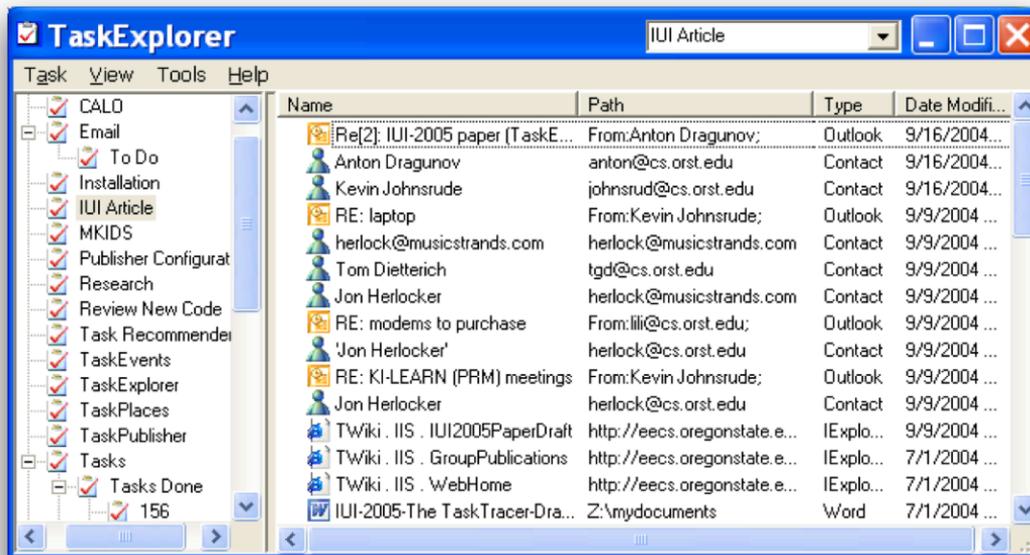


Figure 3.2 - Task Tracer's interface. Image extracted from (DRAGUNOV *et al.*, 2005).

### 3.4 UMEA

In a broader sense, UMEA (KAPTELININ, 2003) work is highlighted. Computer science department of University of Umeå, in Sweden, developed UMEA. The system focus on monitoring user activities on computer desktop and, by doing it, it aims to organize a task context for the task being executed. UMEA names that context as project space. In order to work, UMEA creates a virtual work environment to monitor activities. This work environment integrates typical professional daily services such as document access, folders, URLs, calendars, contacts. They all are available to the user from the virtual work environment. UMEA also solves context change problem because it stores a user interaction history on a database. By storing those data, UMEA is able to recreate a task context at any given time. UMEA's interface is shown in Figure 3.3. UMEA's limitation is the lack of integration with new technologies. If the user, during his workday, needs to use a tool that UMEA does not support, then task context will not be created.



Figure 3.3 - UMEA's interface. Image extracted from (KATPELININ, 2003).

### 3.5 PSEE

Two decades ago, researchers that understood the importance of the execution of a software process started to notice that there was no suitable computer aid to support software engineers during software process execution. Then several of them focused their efforts in the definition of a mechanism that would enable software process to be managed systematically. The mechanism would be able to help software engineers during some work related to software processes. The same researchers then created an environment able not only to model software processes, but also to execute them and facilitate analysis. The environment was named Process-Centered Software Engineering Environment (PSEE) (FUGGETTA & GHEZZI, 1994). PSEEs consist of software development environments that allow the modeling and execution of software processes in a certain degree of automation.

If software processes are expressed in a formal notation, PSEEs can be used to support a variety of activities such as process analysis, simulation and enactment. PSEEs, like computer systems, allow software engineers to follow activity deadlines, artifact creation and consumption, and by doing so, it allows them to coordinate the activities of software development groups (REIS & REIS, 2007).

Several works (AMBRIOLA *et al.*, 1997, ARBAOUI *et al.*, 2002, FUGGETTA, 1996, MATINNEJAD & RAMSIN, 2012) have studied the architecture and functionality of the so-called PSEEs. They have also compared existing frameworks in order to find trends and/or just to present a review of them to the public.

Moreover, a deep investigation of the history and the future of Process Centered Environments can be found in (GRUHN, 2002).

PSEEs represent a direction of investigation towards the understanding of how software engineers manipulate software process artifacts and how the mechanism can be improved. Thus, a PSEE that deals with software process modeling and execution as well as activity and artifact creation and manipulation was found and it was investigated in order to provide insights for the solution of the problems of this Master's Degree Dissertation.

### **3.6 WebAPSEE**

On 2005, a partnership between FINEP (Financiadora de Estudos e Projetos) and SERPRO (Serviço Federal de Processamento de Dados) from Belém, Pará, started a project that resulted in the creation of a PSEE named WebAPSEE (REIS, 2003). Its development was coordinated by Software Engineering Lab from the University of Pará (LABES-UFGPA) with the participation of SERPRO-Belém, Eletronorte (Centrais Elétricas do Norte do Brasil S/A) and Universität Stuttgart, in Germany. On October 2<sup>nd</sup>, 2006, WebAPSEE version 1.0 was publicly introduced to the Free Software Community.

According to (REIS & REIS, 2007), WebAPSEE environment is based on three underlying assumptions:

- Software process execution and automation – The automated aid for software process execution facilitates the work performed during the execution of a software process and makes software process models a reality in software organizations. Thus, an aid in software process introduction to organization is offered.
- Flexibility – Software models describe software processes in a generic way and, for that reason, they are static. They do not update as the software process execution goes on. However, flexible representations of the software process are necessary to keep activity consistency. Hence, it is necessary to provide a software process flexibility level suitable for the dynamic characteristic of a software process execution.

- Registered information richness (metrics and decisions) – Not only a higher number of tasks are automated, but also a higher volume of data is registered, which facilitates decision making during a software process execution.

For these reasons, WebAPSEE environment has as its objective to provide automation and flexibility simultaneously during software process management. Therefore, WebAPSEE allows visual modeling and execution of software processes. Furthermore, other functionalities are supported such as process reuse aid, artifact version control (SALES *et al.*, 2008), besides typical project management field functionalities like process visualization as a Gantt chart, critical path generation, exhibition of the analytic structure of the project and management report generation (FRANÇA *et al.*, 2009).

Two unique features of WebAPSEE are software process execution flexibility, which allows a user to define, change and delete software process elements (artifacts, activities, etc) at runtime, and the technologies used in its conception, which are open-source, starting from Java language, passing through frameworks to develop components, and also the database management system and external integrated tools (COSTA *et al.*, 2007).

The progress of a software process execution is accomplished through notifications between participants, which tell them what activities are being executed or finished. In order to keep consistency of an executed software process between participants, WebAPSEE provides two interaction interfaces. The first of them is Manager Console, present in Figure 3.4-a. In this interface, managers are able to visualize the entire software process. Software process activities are displayed in ellipses and different colors are used for different activity states. An activity may be “ready”, “cancelled” or “failed”. Figure 3.4-b shows the second interaction interface used by WebAPSEE, which is named Task Agenda. There are two types of that interface: Web and Desktop ones. Regardless of its type, by making use of that interface, software engineers are able to set activity states that are under their responsibility, check activity completeness in a list view and manage artifacts of activities.

WebAPSEE automates and thus facilitates much of the work of the management of software processes. However, it should be noted that the association between an artifact that has been already imported to the system’s repository and activities that use that artifact during their execution is manually done by software engineers, even though most of this information is presented in the software process that underlines the development of the software. The manually allocation of artifacts to the suitable contexts may not solve the artifact search and context change problems

because much work and time will still be spent on support work and not on the execution of the activity.

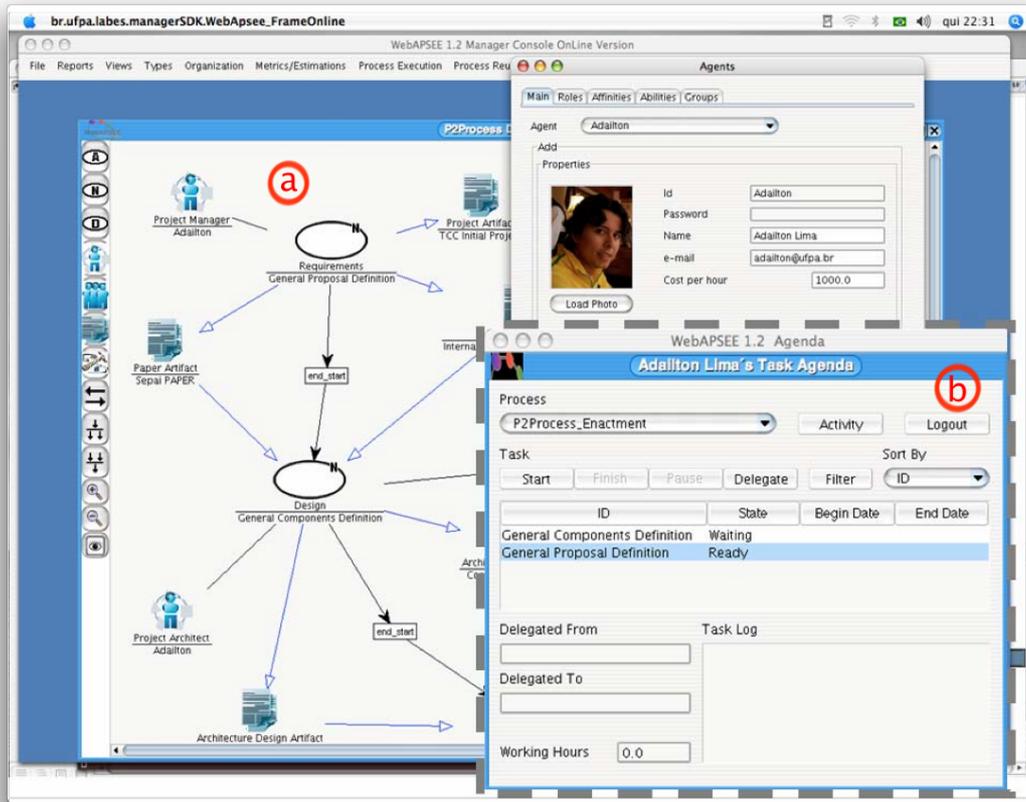


Figure 3.4 - Main Interface of Manager Console and Task Agenda. Image extracted from (REIS & REIS, 2007).

### 3.7 TABA Station

TABA station is a meta-environment, from which it is possible to define software processes and then, based on a particular software process, generate a software development environment fully adapted to specific project particularities (ROCHA et al., 1990). TABA was developed in Computer and System Engineering Program of COPPE/UFRJ and it distinguishes itself from other works not only by providing aid to software engineers when performing software process activities, but also by providing a way of executing those software processes in a customized and adapted way (VILLELA et al., 2001).

TABA's main objectives are to provide help to project management activities, to improve software product quality and to increase productivity, from the automated creation and availability of a suitable software development environment in order to allow software engineers to control the project and measure the evolution of activities

based on data collected as development takes place. The integration with tools is another feature. TABA offers a tool integration infrastructure that aids users during the execution of software processes (TRAVASSOS, 1994). TABA Station also features a repository that stores software project data collected during its lifecycles in order to allow analysis to be made, such as an evaluation of a software process (GOMES *et al.*, 2001).

Figure 3.5 illustrates AdaptPro tool (BERGER, 2003). It is this tool, which is a component of TABA Station, that an Organization may use to instantiate a specific software development environment to aid a software process execution. In short, software engineers may characterize and plan the software project that will underline the project execution based on an organizational default process and, finally, instantiate a software development environment to aid planned software process execution.

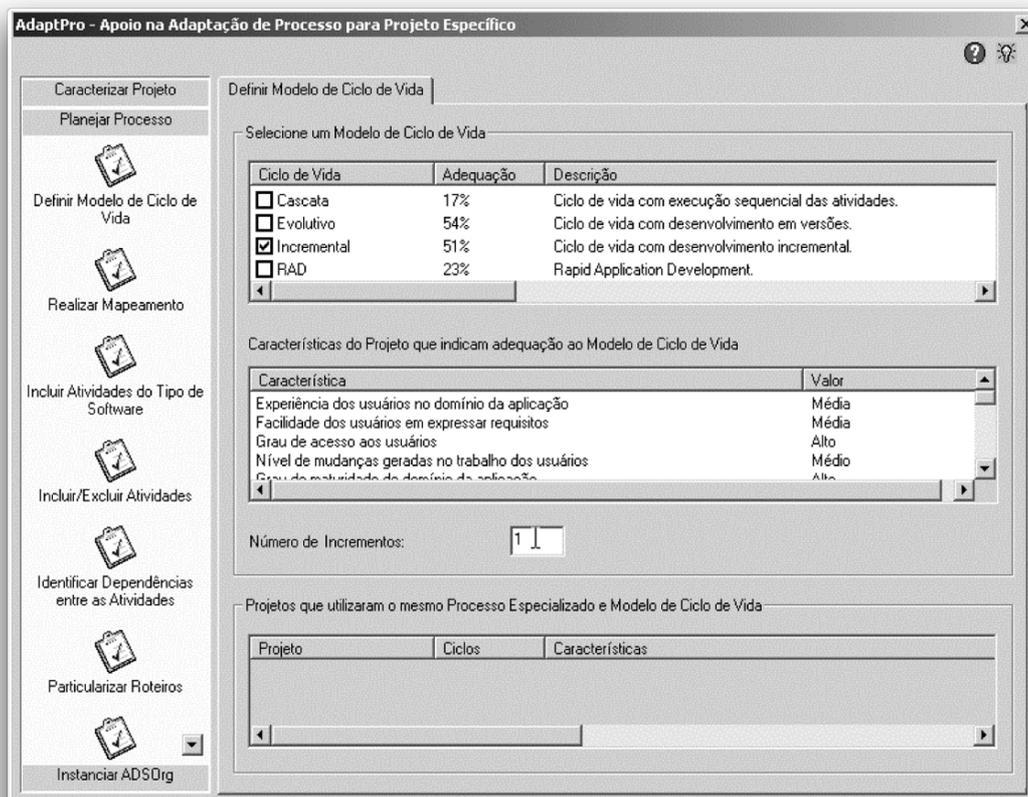


Figure 3.5 - AdaptPro - a software process adaptation aid tool. Image extracted from (ROCHA *et al.*, 2005).

However, modifications that may happen on software process specification will not really affect the execution of the process until a new software development

environment is created to that new software process. Thus, when software process receives modifications, its impacts may be costly.

### 3.8 Software Traceability

A useful approach to manage an activity context, that is, the relationship between one activity and its related artifacts, is the use of a traceability matrix. Indeed, artifact traceability during software development has been under study. In general, the objective of traceability is to increase software quality by providing a way to analyze change, maintenance and evolution effects that may happen during a software lifecycle. Furthermore, traceability also plays an important role in identifying and comparing new and already known software requirements, artifacts reuse (when new artifacts are similar or equal to others that already exist), testing and inspection of the entire software being developed. Artifact traceability, and as a consequence, traceability matrices, provide clear communication between users and developers, which improves the produced documentation and software acceptance (SPANOUKIS & ZISMAN, 2005). Figure 3.6 shows an example of a traceability matrix.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1			REQ1	REQ1	REQ1	REQ1	REQ1	REQ1	REQ1	REQ1	REQ1	REQ1	REQ1	REQ1	REQ1	REQ1
2	Requirement Identifiers	Reqs Tested	UC	UC	UC	UC	UC	UC	UC	UC	UC	UC	UC	TECH	TECH	TECH
3			1.1	1.2	1.3	2.1	2.2	2.3.1	2.3.2	2.3.3	2.4	3.1	3.2	1.1	1.2	1.3
4	Test Cases	321	3	2	3	1	1	1	1	1	1	2	3	1	1	1
5	Tested implicitly	77														
6	1.1.1	1	x													
7	1.1.2	2		x	x											
8	1.1.3	2	x											x		
9	1.1.4	1			x											
10	1.1.5	2	x												x	
11	1.1.6	1		x												
12	1.1.7	1			x											
13	1.2.1	2				x		x								
14	1.2.2	2					x		x							
15	1.2.3	2								x	x					
16	1.3.1	1										x				
17	1.3.2	1										x				
18	1.3.3	1											x			
19	1.3.4	1												x		
20	1.3.5	1												x		
21	2.1.1	1														x
22	2.1.2	1														x

Figure 3.6 - A traceability matrix being used to trace software requirements.

The best way to improve software artifact traceability is the creation of a matrix that represents the association between each activity and artifacts (SUNDARAN *et*

*al.*,2010). Nevertheless, traceability matrix is not usually used with efficiency. The difficulty of an efficient use arises from the fact that the creation of traceability relations is not generally an automated process. Besides, it is costly, in terms of efficiency, because it takes a considerable amount of time to be performed, and it can easily fail because analysts perform it with minimal computer aid. In most of the cases, a spreadsheet is used or it is manually done. In addition to it, the possible automation of this procedure requires a considerable processing capacity because it takes into consideration a scenario with a lot of artifacts and activities presented in a software process. Moreover, difficulties get worse when software process flexibility takes place, because new artifacts may be created in new contexts or activities can be re-executed. In this case, the entire traceability matrix needs to be reprocessed (SPANOUDAKIS & ZISMAN, 2005).

## **3.9 Mylyn**

### **3.9.1 Overview**

Researches done in the last years by scientists from the University of British Columbia, Canada, explored the way programmers have been dealing with implementation activities when using Eclipse IDE, specifically code edition tasks (KERSTEN & MURPHY, 2005, KERSTEN & MURPHY, 2006). The results show that the majority of code modifications performed on software projects affects more than one file. That is, when a programmer is warned that an edition will be made on a specific code element (renaming a class, method, variable, etc), he carefully updates, if needed, all of other elements that deals with the given edited code element, which may result in further modifications. This behavior usually is the result of a natural interdependency contained in a software code, when classes are interconnected either due to modeling problems or due to programming language characteristics. The fix of the consequences of a code edition can be a complex task because it is not easy to find the elements that were affected by the edition. Thankfully, IDEs available in the market facilitates this job by pointing out elements that were affected by a given code edition, or even by updating them automatically, which is the case of the Eclipse IDE.

That research raises a discussion about two main important points: code edition and the search for files related to the code edition. These activities are usual in a workday of programmers. The first of the two points is related to the code edition itself. When the task of modifying a specific part of the code is informed to a programmer, the first step taken is to identify where the code edition will take place. As it was shown by

the research done by the Canadian university, chances are low that a programmer will modify only one class, or file. He must first identify all of the files that will directly be affected by the edition. Usually, these files are spread within the project structure in several different locations. This results in a search for those files and, as a consequence, an additional time spent.

The second important observed point is related to the search of the files indirectly affected by the first class edited. Commonly, the edition of a part of the code has consequences in other parts and that characteristic may be treated. If the used IDE does not provide support for the type of modification being performed, programmers once more need to spend an additional time and effort to search for classes affected by the edition. That constant search for related files and the time and effort spent on that task raise a question: is the representation of all of the project files, which is actually in a tree structure, suitable and sufficient? For projects with a few files, it may be easy to search for files by browsing that structure. In the most common cases, where there is a lot of project files, that ease is not verified. Classes affected by an edition made by a programmer may be spread across several other folders, which may be spread across several other packages. In short, the search for files related to a task to be performed is not always easy and, once these files are found, it is not guaranteed that it will be simple to perform the proposed task, because it will have consequences, such as more editions to other files to be made.

That scenario considers the edition of one task. The most common situation is that programmers need to perform several tasks at the same time, which results in stopping the execution of a task to perform another, either because the second one has a higher priority or because the first one does not have all of its required resources available at that moment. Changing tasks results in a context change, which is the change of files relevant to that task execution. One may note that programmers are constantly engaged to the search and maintenance of the current task context. Therefore, two critical factors negatively affect the productivity of a programmer when considering the sheer number of files spread on the software project and the difficulty in the search for those files to build a context: high amount of information and task context maintenance (MURPHY, 2009, KERSTEN & MURPHY, 2006).

For those reasons, the creation of a mechanism to deal with information is important. Then, researchers from the University of British Columbia developed a mechanism that captures, models and persists relevant elements and relations for the execution of a task and named that mechanism as Mylyn (KERSTEN, 1999).

Introduced in 2005, Mylyn is an application for programmers that write Java code and use Eclipse platform. Later, developers of Mylyn started a company and

initiated the development of Tasktop (TASKTOP, 2014), a commercial version of Mylyn, which was introduced in 2008, and currently exists.

Mylyn's objective is to solve the two problems discussed, allow programmers to spend more time working on tasks assigned to them and spend less time searching for classes associated with that task. This is achieved by facilitating the search for classes of that task, which represents a small subset of the classes of the project.

In order to let the search of a task context easier and highlight its classes to programmers, Mylyn has a degree of interest (DOI) function implemented. That function is responsible for monitoring the work performed during code implementation and maintenance and calculate the importance of each class in relation to the task being performed. For example, when a programmer edits a given method during a task execution and then needs to access another class, the DOI function considers both classes as relevant to that task execution. That relevance is reflected in a model maintained by the DOI function. Based on that data, Mylyn is able to filter out data shown to the programmer, presented in the structured views of Eclipse IDE, and display only classes that are important to the execution of the current task. Mylyn has been highly accepted by programmers on their academic and professional work, and currently it is used by thousands of programmers daily (KERSTEN & MURPHY, 2006).

### **3.9.2 Characteristics**

Mylyn aids programmers on implementation and maintenance of software code when performed on Eclipse IDE. Based on the programmer's interaction with the code, Mylyn's DOI function creates task contexts, in other words, it notes what are the classes and parts of the codes relevant to the task currently being executed. Once this is finished, Mylyn manages views included in Eclipse and filters not relevant classes, displaying only classes of the task context to programmers. Interface becomes clean, objective and focused on the task executed. This increases the productivity of programmers, once they do not need to spend additional time and effort on the search for classes of a task context, either during a task execution or during a task change.

For these concepts to function, Mylyn has four main components that work together. They are: Interface, DOI function, Working mechanism and Context and Task Import and Export mechanism. Each one of the components mentioned has a different function on Mylyn workings. Figure 3.7 illustrates each one of these components and the relationship with other Mylyn's components.

Interface component is responsible for displaying tasks and classes. It is the work area of a programmer. It is where class filtering takes place according to the task context and programmers' interactions. DOI function is the component that scores and

decides what classes belong to a given task context, so visual filtering can be made. Saving Mechanism persists to disk data related to task contexts. For that reason, programmers are able to change tasks or close the IDE window without the need to rebuild a task context again. That mechanism is also responsible for saving interactions that were performed during the work of a programmer. At last, Context and Task Import and Export Mechanism allows the export of data saved by Saving Mechanism and thus, these data can be used in another instance of the IDE. It is also that mechanism that performs the import of data in another instance of the IDE. Each one of these components will be described in detail in the following sections.

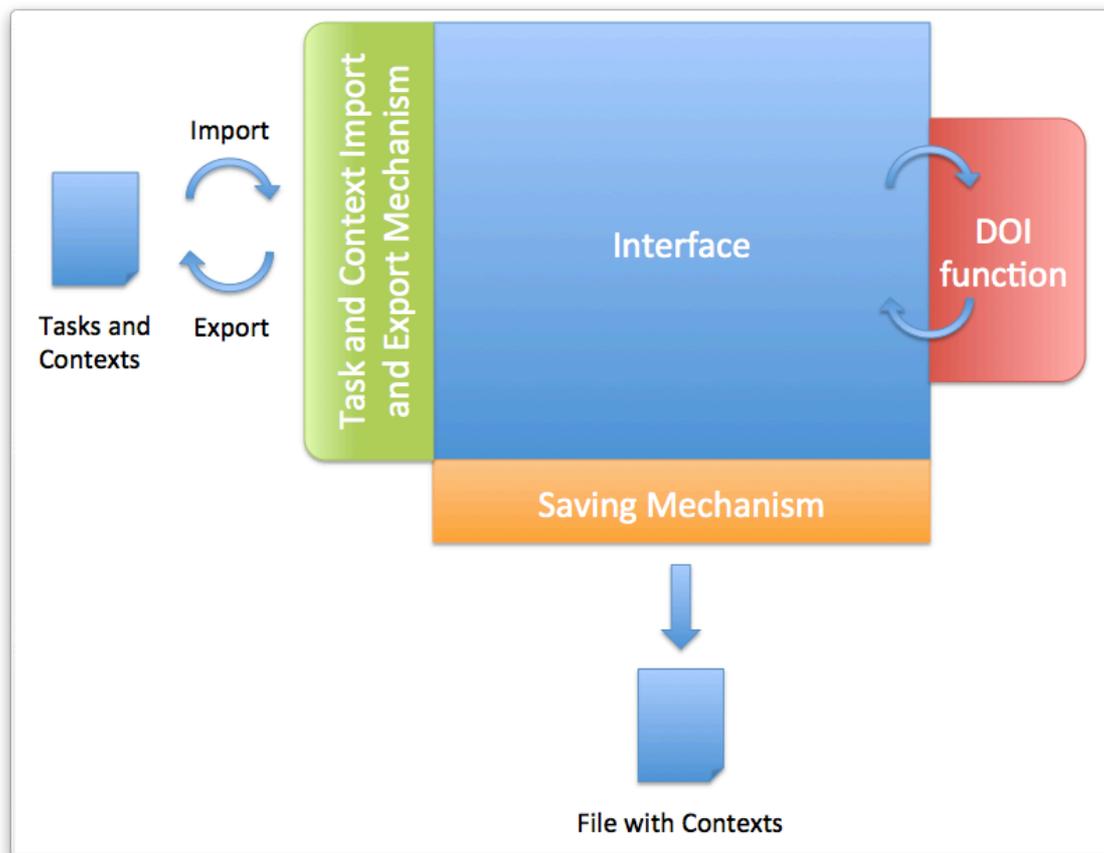


Figure 3.7 - Mylyn's components and their relationship.

### 3.9.2.1 Interface

When using Mylyn during the development of a system, a programmer increases its productivity once the main, most important and necessary classes to perform a given task are highlighted and accessible in an easier way on a task-based interface. The way data are displayed is one important characteristic of Mylyn. Eclipse IDE has plugins that makes work environment more suitable to Java developers. These

plugins belong to a tool called Eclipse Java Development Tool (JDT). That tool is the one responsible for coloring some reserved Java words during code implementation, to run Java Virtual Machine (JVM) on a code execution and, most importantly, to manage user's interface. JDT manages views that help programmers in code implementation. These views are Package Explorer, Type Hierarchy View, Java Outline View, and others. However, these views are not configurable to display only what is important to the task being performed. Thus, it is said that the interface is not task-based. Mylyn's views are: Mylyn Package Explorer (Figure 3.8-a), Mylyn Problems List (Figure 3.8-b), Mylyn Outline (Figure 3.8-c) and Mylyn Tasks (Figure 3.8-d).

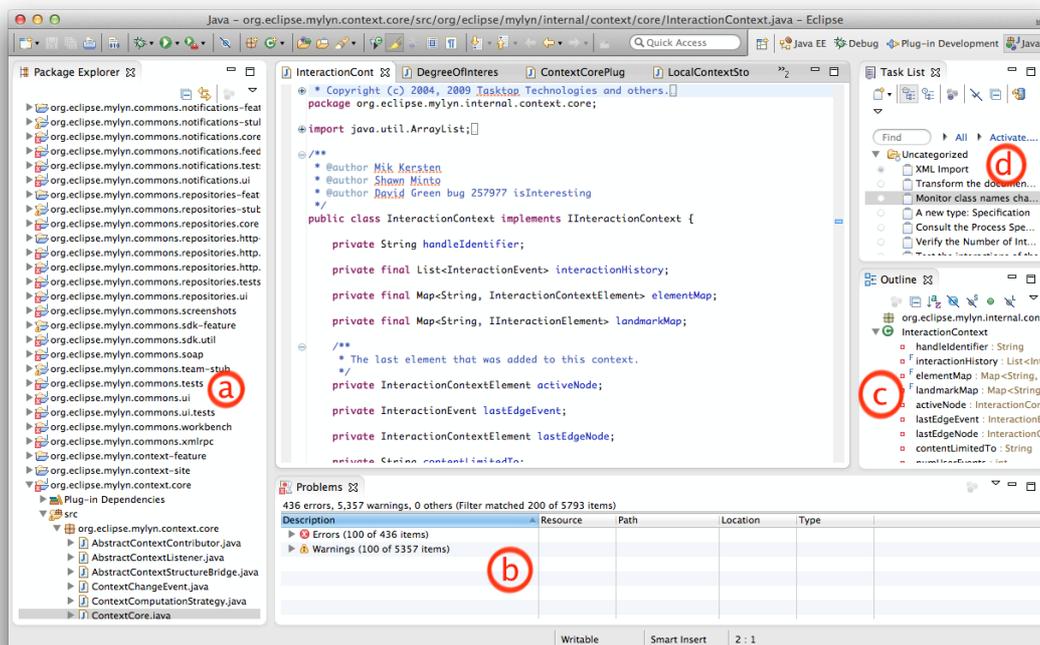


Figure 3.8 - Mylyn's interface.

The first of these views, Mylyn Package Explorer, shows the project structure with the filtering of irrelevant artifacts to the current task. On the hierarchical structure shown on Mylyn Package Explorer, besides the names of the elements that contain other elements (a package, for example), there is a number that indicates how many classes are being displayed. Mylyn shows on Mylyn Problem List the list of most relevant problems to the executing task. So, programmers may focus on the main problems that affect their actual work, and also efficiently recover them if needed, when compared to the way it was done in the past, when all problems were displayed and there was no classification. Mylyn also has Mylyn Outline view. That view is useful to large classes with a lot of methods and variables. With Mylyn filtering, only code elements relative and relevant to the task being executed are displayed on this view,

which allows the programmer to better locate some particular code elements to access by searching for their names. Mylyn also introduces a new view for Eclipse IDE: Mylyn Tasks. In this view, all tasks are listed and displayed. It is also possible to create and delete a task, to set its beginning and to inform that it is finished.

### **3.9.2.2 DOI**

Mylyn's objective is to display only relevant classes to the current task in execution and, as a consequence, lower time and effort spent in the search for relevant code elements. This has a positive impact on programmers' productivity because their efforts will be focused to the task to be performed. Thus, a classification of project files must be done. Mylyn needs to know what are the files that are mostly being used, and then highlight them from the remaining files.

The way file classification is done deals with the monitoring of programmers' interactions and the use of a DOI function applied to elements of the project. For example, when a programmer is executing a given task, and he selects or edits a class, Mylyn's DOI function recognizes the affected class and increases the interest value associated with that element, which means that this element is more important than others to the current task. After classifying all project elements, DOI function can apply a filter, and based on the interest value of each element, it is possible to choose some elements. Thus, Mylyn is able to show only the most important project files to that programmer.

Mylyn's DOI function maintains two important structures so that an interest value may be calculated for Java classes. The first structure is an interaction event list that represents the interaction history that a programmer had with the code. At each class selection or edition, for example, it is added an entry to that list with data about the interaction. This allows DOI function to calculate the interest value based on the history of interaction events. Besides, Mylyn maintains in memory a list of objects, which references project classes. An interest value is associated to each object presented in the list. That interest value is a floating point number that represents the actual interest for that class. When a programmer selects or edits a class, Mylyn adds entries to the interactions history list and calculates the interest value of that class to the current task, which usually results in the increase of that class' interest value. As the time passes by, if that class is not selected or edited, its interest value is gradually lowered. Therefore, at any given time, a class' interest value reflects that class' importance to the current task.

Mylyn's DOI function's classification mechanism, by default, classifies files with negative interest value as not interesting and omits them from the programmer's view.

If a programmer selects a file, that action contributes with 1 point to the increase of that class' interest value, by default. If the programmer is editing that class, each keystroke, that is each character being typed during the edition, corresponds to an increment of 0.7 points in the interest value associated to that class. Each selection and edition event performed by the programmer on a project file affects the interest value of other classes. Thus, the interest values of other classes are decreased by 0.017 points by default. For that reason, if a class is not used during the execution of a particular task, DOI function classifies it as not interesting and omits it from the programmer's view. The set of all interactions a programmer may perform as well as their contributions to the interest value of a class are described later in this section.

Algorithm 3.1 represents concepts of interest value calculation done by DOI function that Mylyn uses to classify project classes. The objective of the algorithm is to know all types of interaction events ever performed with a given class and add up all positive scores that these interactions contribute to that file. After that, scores related to other interaction events are subtracted from the partial interest value. The final value is then returned. On Algorithm 3.1, "getValue()" is the method called to start the calculation of the interest value of a given class. The algorithm is divided in two parts: additions and subtractions. "getEncodedValue()" method is responsible for calculating the additions. Thus, it adds to the variable "value" the scores given by each of the five possible interaction events. These interaction events are explained in Section 3.9.2.2.1. "getDecayValue()" method is responsible for calculating the decay of interest value of the given class. That method has two values to work with. The first one is "eventCountOnCreation". This value is an ordinal number associated with the interaction event that added that class in the current task context. The second value is "userEventCount". This value is an ordinal number associated with the last performed interaction event. By calculating the difference between those numbers, the method is able to determine how many events took place since the addition of the class in the task context. The decay value is calculated based on this value. A multiplication of this value to a constant of decay is done and the decay calculation is finished. Finally, "getValue()" calculates the difference between the addition of points and the decay value to find out the final interest value.

Mylyn's DOI function's interest value model is aware of the case in which there is a misinterpretation of a code element's interest value. If a programmer considers that a given class is not relevant to the current task's execution and if the class is in that task's context, or even if the programmer knows that a particular class is important to the current task, but it is not currently being displayed, he is able to manually set that class' interest value.

Figure 3.9 shows a flow of one example of the use of Mylyn. Mylyn's DOI function usage may start with the creation of either tasks or artifacts. In this example, a task is firstly created. Mylyn provides a task creation wizard for programmers. They must indicate that a new task is about to be created. Once a new task appears on Mylyn's Tasks list, programmers are able to erase the "new task" default name and register a new name for the task. If needed, programmers may also set task parameters such as the time that task may finish and its priority. It is important to notice that the context associated with the new task is empty. It happens not only because there is any class created, but because no context are initially created for a new task.

Mylyn's artifacts are elements that programmers interact with, which are Java classes. The creation of artifacts starts with the use of Eclipse Java class creation wizard. After programmers create a Java project, a package and a class, Mylyn's DOI function recognizes the existence of the class in its model and associates to it the number zero as interest value. It needs to be mentioned that Mylyn's DOI function will only associate an interest value to a just created class if any task is active and the interest value will be associated to the new class in relation to the active task. If no task is being executed, then the new artifact will not belong to any context and will not receive any interest value. If a programmer performs a selection to the class he just created, assuming that this class belongs to a task context, then the interest value of that artifact will receive the number of selection interaction events times the points selection interaction events contribute to artifact's interest value. In this example, this number is  $1 \times 1 = 1$ . As said, other interaction events negatively affect a particular interest value when they contribute to the decay value. In this example, it is assumed that a programmer performed ten selections on other artifacts created later. Thus, the interest value associated with the first artifact (or class) has to be updated. As explained, Mylyn's DOI function gets the difference between the ordinal numbers that represent the event on the creation of the artifact and the last event performed by the programmer and then multiplies it to a constant of decay. The decay final value is then subtracted from the actual interest value associated to that artifact and then this value is finally updated. In numbers, the final interest value is  $(1 \times 1) - (10 \times 0.017) = 0.83$  points of interest.

After some interactions with other Java classes, that interest value will become negative, which is interpreted as if this class is no longer interesting to the current task and it will be deleted from that task context and also omitted from the programmer's view.

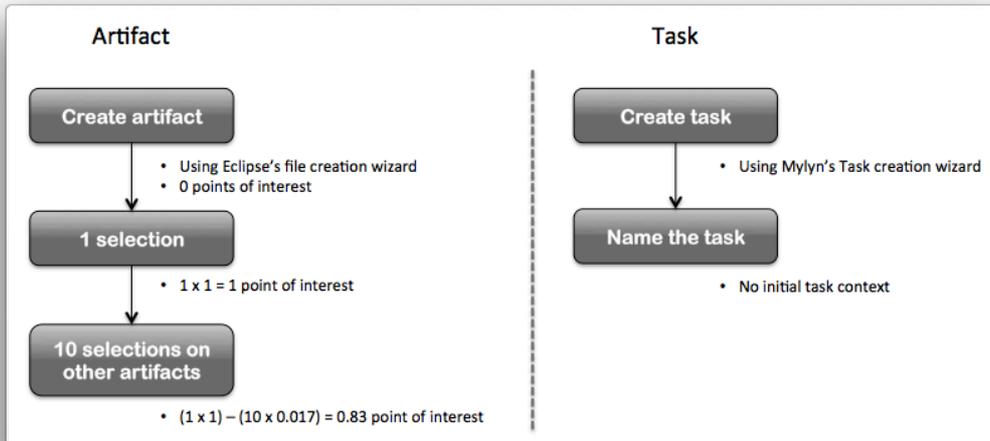


Figure 3.9 - An example of Mylyn's DOI function usage. A task interaction process example is three steps on the left and a task interaction process example in two steps on the right.

```

01 public float getValue() {
02     float value = getEncodedValue();
03     value -= getDecayValue();
04     return value;
05 }
06
07 public float getEncodedValue() {
08     float value = 0;
09     value += selections * 1f;
10     value += edits * 0.7f;
11     value += commands * 1f;
12     value += manipulationBias;
13
14     value += predictedBias;
15     value += propagatedBias;
16
17     return value;
18 }
19
20 public float getDecayValue() {
21     return (context.getUserEventCount() - eventCountOnCreation) * 0.017f;
22 }

```

Algorithm 3.1 - Algorithm of Mylyn's Degree of Interest (DOI) function.

### 3.9.2.2.1 Interactions with Task Context

Mylyn's DOI function monitors the interactions of a programmer during the implementation and maintenance of the code of a system. By doing that, DOI function

is able to capture relevant data about programmer's actions and then create a model with project classes and associated values that represent the interest for that file in relation to the current task. Based on it, Mylyn automatically identifies a task context and manages the interface to facilitate the activities of a programmer when implementing a system.

Regardless of what action a programmer performs, Mylyn's DOI function registers on its model six pieces of information, described in Table 3.1. Every interaction event performed by programmers happens at a given time and has a target element. DOI function captures those data and organizes them for easy maintenance of interest values and, as a consequence, of task contexts. The time when an interaction event took place is registered. The type of interaction, according to Table 3.2, is also captured.

Each interaction event has an Origin, that is, the tool used to cause the interaction event. The Content Type, that stores data about the element that received the interaction, is also registered, alongside a reference to the Target element. Finally, a field is reserved to the registration of data related to state change that took place during the interaction event, named as Delta.

Table 3.1 - Data about one interaction event captured by Mylyn

<b>Name</b>	<b>Description</b>
<b>Time</b>	The time the event took place.
<b>Type</b>	The type of the interaction event.
<b>Origin</b>	A reference to the tool that caused the interaction event.
<b>Content Type</b>	A reference, with the description, to the element that received the interaction.
<b>Target</b>	A reference to the element that received the interaction event.
<b>Delta</b>	State change that happened with the interaction event.

Some interaction events represent a consequence of a direct action of a programmer that deals with the code. For example, if a programmer selects and opens a class to visualize the code it contains, a Selection interaction event was performed directly by the programmer and Mylyn's DOI function register the suitable data. Nevertheless, there are other types of interaction events that may occur during the implementation of a system: the indirect interaction events. These events happen without a programmer's interference and DOI function also registers them. An example of an indirect interaction event is when a programmer decides to rename a class that is open. In order to avoid syntax errors on the code that is being implemented, Java plugins existent on Eclipse IDE update references to the class that was just renamed.

By doing that, some interaction events, or some edits, take place without the direct intervention of the programmer. In this example, the interaction event described is the Propagation interaction event. Mylyn's DOI function captures both direct and indirect interaction events and takes both into consideration on the creation of a task context.

Researchers of the University of British Columbia, when developing Mylyn, identified five types of interaction events that a programmer may perform on the code. Mylyn's DOI function monitors and captures them. These interaction events are described in Table 3.2. Direct events are Selection, Edition and Command. Selection interaction event consists of using mouse or keyboard to make code-editing window, after opened, the working window of the programmer, in other words, make it the main window. Edition interaction event corresponds to the change of the code of a class or method, either by adding new pieces of information or by taking them. The interaction event named as Command represents actions that a programmer might apply on classes according to the use of the Eclipse IDE. Some Command interaction event examples are saving or compiling a code. There are two indirect interaction events studied: Propagation and Prediction. Propagation interaction event occurs when any interaction event affects other project elements that are somehow related to the target element of the first interaction event. For instance, a method renaming may cause editions on another classes that have references to that specific method. Prediction interaction event is a consequence of the registration of historical data done by DOI function during the execution of tasks. Mylyn's DOI function, when checking that history list, is able to predict what files, classes, methods or variables may be relevant to the current task's execution.

Table 3.2 - Types of interaction captured by Mylyn.

Type	Interaction Event	Description
Direct	Selection	Select the code with the mouse or the keyboard.
	Edition	Textual code editions.
	Command	Interaction events such as save, compile, etc.
Indirect	Propagation	Interaction event that propagates to other related elements.
	Prediction	Ability to predict what elements will be useful.

### 3.9.2.3 Saving Mechanism

Mylyn's Saving Mechanism is different from Eclipse IDE's Saving Mechanism. Mylyn's Saving Mechanism, which is the subject of this section, is responsible to persist to disk data about tasks and contexts from time to time. By doing that, a

programmer that is dealing with the code is able to perform a task change, and a context change, with success. When he returns to the interrupted task, the set of classes that he was working with will be available the same way it was before the task change. It is important to note that the Saving Mechanism persists data to disk about the tasks, their contexts and the interactions that are performed by the programmer whenever one of the tasks is activated.

In addition to it, it is important to mention that the functioning of Saving Mechanism is indifferent to the number of projects or classes in a workspace. A same directory, in each workspace, is used to persist all data to disk. In other words, each workspace has a specific directory to which the Mylyn's Saving Mechanism saves all interesting data. The directory `[WORKSPACE]/.METADATA/.MYLYN/tasks` stores an XML file with a list of all tasks (and additional data about them, such as name and status of completeness) that are existent on Mylyn to that workspace. The directory `[WORKSPACE]/.MYLYN/contexts` stores one or several XML files with data about the context of each task along with data about classes, methods and variables contained at each task context. Moreover, the interest value of each code element and data about the last interactions performed are stored on this XML file.

#### **3.9.2.4 Task and Context Import and Export Mechanism**

If a programmer wants to or needs to export all data about task contexts and interest values either for a backup or to take it to another workspace, he needs to use Context and Task Import and Export Mechanism. As the name implies, this mechanism import and export data from contexts and tasks. When exporting, it creates a zip file with all files that Saving Mechanism uses to persist data to disk and it makes it available in the directory pointed by the programmer that is exporting the data. If the operation requested is to import data, the mechanism process data included in the export zip and updates tasks and contexts with a new set of data. After importing, the programmer may continue his job in another workspace. If the new workspace do not have one of more classes that are existent in a particular task context, DOI function, which is responsible to score classes, methods and variables, continues to work normally and also decreases the interest value of those not-encountered classes according to the occurrence of other interaction events.

#### **3.9.2.5 Drawbacks**

Mylyn helps to increase a programmer's productivity when it facilitates the recovery of some project elements that are important to a codification task. This lowers the time and effort spent by a programmer in support tasks that are not the main task.

However, when developing a software product, the software undergoes a path with several steps, which starts from its conception, passes through the coding phase and finishes on its delivery, maintenance and possible retirement of it. These steps are described by software lifecycle. Mylyn aids programmers in one of these steps: the implementation of the code. That step is an important one on the development of a software product, but it is not the only one. The other steps have their own relevance, difficulties and particularities that make them require attention to the way they are performed.

Besides, in a software development project, several documents are produced. They contribute to the systems' quality, validity, and verification, among other functionalities. Code documents, that are software classes, are important, essential and do contribute to the creation of the software. Mylyn aids programmers on creation and maintenance of code documents, but Mylyn is aimed at the implementation step only. A desirable scenario is aiding the creation and maintenance of all of the project's documents, trying to guarantee a high quality level. It may be importante to note that Tasktop, the commercial version of Mylyn, does manage other types of project documents that may be present on other steps of the software development process other than implementation. However, it still lacks a direct integration with the software process and does not base its workings on that process for task context definition, as it will be explained in the following paragraphs.

Mylyn creates task contexts automatically based on interaction events performed by programmers with the code. When selecting a task, Mylyn associates some data that is considered important, according to DOI function, to that task. However, in order to task displaying and data association to tasks work well, it is necessary that this task belong to the field of codification. Tasks that don't relate to code implementation do not need to be registered on Mylyn. In other words, Mylyn aids tasks related to coding software. Other several tasks that are performed before, or in parallel to, the implementation are not being aided.

Not only it lacks the management of tasks existent in other steps of the development of the system, but it also lacks a software process onto base the definition of tasks. What happens is that a programmer manually does the definition of tasks through the IDE. There is no justification about the origin of the task. If tasks were based on a software process, tasks could be derived from a software process. If so, tasks would have a reason to exist and the software process that creates them would justify their existence. Besides, tasks would not need to be manually created, as it is actually done. One would just need to check the software process and display the proposed tasks. This could be done by syncing Mylyn to a repository of tasks based on

a software process. However, software process artifacts and their relationship with tasks may not be present on the repository.

Finally, Mylyn does not take into consideration the concept of software development process. For that reason, it is aimed at software implementation step only. The consideration of a software process development results in a better base for software creation activities and, as a consequence, for coding activities.

### **3.10 Conclusion**

The entire set of research projects described in this Chapter in any way deals with problematic situations similar to the one studied by this Master's Degree research. Either by aiding users on executing tasks and dealing with artifacts or by aiding software engineers during software process execution, each project represents a significant contribution to several different areas of software engineering field. The main problem that could be observed after observing each project studied is the lack of automation when dealing with a sheer number of artifacts. Moreover, although some projects are specifically aimed to the software engineer field, and related to software process concept, most of them are not directed to this field and have their contributions focused to general task management field. As a consequence, any of these projects considered a software process, or any other process, as a base for executing tasks. In the next chapter, this Dissertation exposes and details the main research made.

## 4 MylynSDP

*On this chapter, MylynSDP is introduced and detailed. Hence, Mylyn's components are pictured and explained, as well as their relationships. A special attention is brought to MylynSDP's DOI function, as it is the mechanism that solves the problems identified on this Master's Degree research project. MylynSDP and its DOI function has code snippets illustrated and their working explained with examples.*

### 4.1 Introduction

Mylyn's DOI function, although limited, proves to be a starting point for the development of an expansion of its concepts. The way it deals with both artifact search (Java classes, in the case) and context change problems are significantly important for the solution proposed by the research of this Dissertation. As a consequence, Mylyn had its code gathered, studied and modified, so its benefits to programmer could be expanded to software engineers from all phases of a software development process. The final implementation, which features a new DOI function, was named MylynSDP.

This chapter is organized as follows. Section 4.2 presents an overview explaining a bit more about the situation that the concepts discussed in this Dissertation are in. Next, Section 4.3 formally defines the main concepts and nomenclature used when referring to this research. Section 4.4 describes characteristics of MylynSDP, including its components and the new DOI function. Several code snippets of main classed important to the comprehension of this project are illustrated on figures. Most of them were modified from the original version. Section 4.5 concludes this Chapter.

### 4.2 Overview

For some years, researchers from the field of software engineering have been studying the relation between the quality of computer software and the software process that underlies its creation. The idea is that a software process with high quality, when executed, increases the chances of the development of a high quality software

product. As a consequence, efforts focused on definition and on improvement of software processes have been initiated.

A software development process is the set of policies, practices, definitions, activities and artifacts that together rule the creation of a system. Generally, a software process is a textual or graphic document with the description of the steps to be followed so that the development of the software will be carried out as desired. In that document, there are elements that define what are the software development activities and, moreover, there is the representation of the order of those elements. Besides, as each activity may consume or produce artifacts, other textual or graphical elements represent those artifacts. The relation between artifacts and the activities must be specified as well. That is, there is a representation of what artifacts are consumed and produced by each one of the activities either by textual or graphical description.

However, as the development of a system may be complex, the creation and definition of its software development process may also be. A software process that has dozens of activities, represented in the form of text or even graphical, and has other dozens of artifacts is difficult to be understood. Whenever a software engineer accesses the document with the software process specification, he needs additional time and effort to understand what are the next activities to be executed and to understand what artifacts he needs to access to perform a given task.

Besides, the descriptive and theoretic nature of a software process specification document creates a discrepancy between its modeling and its execution. In other words, what is modeled not necessarily represents what is actually executed, because there are differences between the theory and the practice. A software process specification works as a model for the development of a software and a software process execution represents an instance of that model, with particularities that happens only during the execution.

Figure 4.1 shows an extract of an example of a software process model. There is an activity named "Develop Test Case" on the model. One may note that "Develop Test Case" activity needs two artifacts, which are called "Use Case" and "Test Case", and produces the artifact named "Test Case". Although that specification is clear and objective, its execution may not be. Generally, a system has several use case and test case documents. However, software process specifications use one symbol to represent an undetermined number of documents. During the execution of "Develop Test Case" activity, a software engineer may have to search for use case and test case documents among several others project documents. Another unwanted situation is that not all use case and test case documents are useful to the execution of a particular task. Rarely, a software process specification document indicates what particular

documents are necessary to the execution of a task. It only indicates the type of the document. This requires the software engineer to search, among use case and test case documents, the ones that will aid him to execute the proposed task.

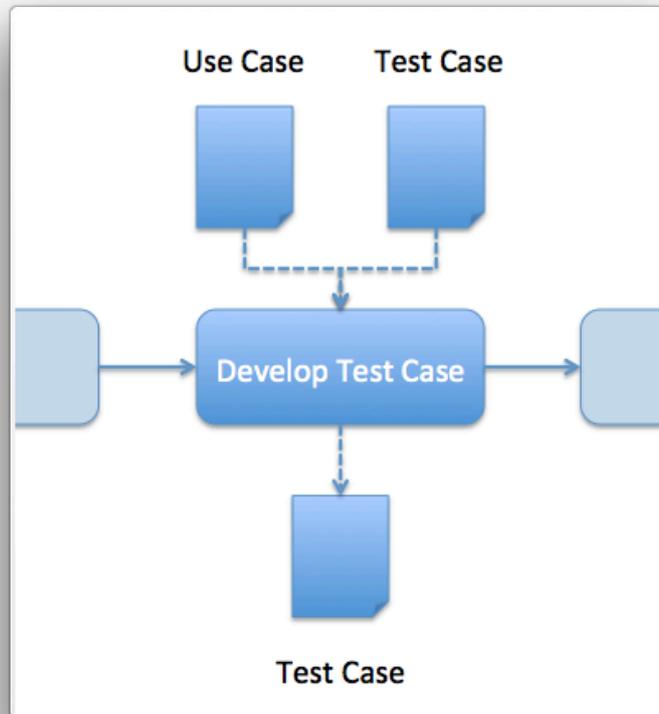


Figure 4.1 - An extract of an example of a software process. The same artifact symbol is used to specify several documents from the execution of a software process.

### 4.3 Concepts

Under the concepts of this Master's Degree's Dissertation, an activity is recognized as the representation on a software process specification of a finite job performed within a given time. On the other hand, a task is a finite job performed during the execution of a software process by a software engineer within a given time in order to achieve an objective. An activity, from the software process specification, represents a task that a software engineer performs during the execution of a software process. The relation between an activity and a task is not a one-to-one relation. An activity can be instantiated more than once in several different tasks. Each one of these tasks has a logical reference to the activity that based its creation. Furthermore, a task, from the execution of a software process, has one source activity only. For example, the activity

A can be the source of tasks A' and A'', but A' have only activity A as its source. Task A' cannot be created from activities A and B.

Some software development process notations, such as SPEM, have activity and task concepts in a different way. These notations considers tasks as decompositions of activities. This approach differs from the one used on this Dissertation which considers tasks as instances of activities.

The name "artifact" is used to define the documents, or files of any kind, that are used, either consumed or produced, during the execution of an activity or task. On a software process specification document, an artifact is a theoretical concept, it represents one or more documents and it is related to software process activities. It can be described either in text or graphically. During the execution of a software process, an artifact is a document that exists. It may be digital and can be manipulated by a software engineer when creating, editing or deleting the file. Similarly to activities and tasks, a software process specification artifact may be instantiated into several software process execution artifacts.

Having it in mind, a task context is defined as the set of all artifacts used during the execution of a task, either by consuming or producing that artifact. This is similar to what happens to an activity context. Both contexts may diverge due to the natural difference between theory and practice. This divergence can be verified in a case where A' is from type A that uses an artifact T1' from type T1, and it now also uses another artifact T1'' from the same type T1. Alternatively, task A' may now use an artifact T2' from type T2, which is a situation that was not initially modeled on the software process specification.

As a consequence of this Master's Degree's study and for the validation of the study, an extended DOI function has been developed with the implementation of the concept task context and software process in order to provide better ways to search software process execution artifacts to the software engineer. This allows the increase of his productivity by lowering the time and effort dedicated to search of artifacts related to the execution of a software process. Initially, a software engineer selects a task from a set of tasks and artifacts relevant to that task, which are in a greater set of artifacts, are highlighted. The highlight is made with the omission of non-relevant artifacts. It is always possible for the software engineer to find the other available artifacts so he can use them, which means that a task context do not faithfully reproduce an activity context modeled on a software process specification.

Two important concepts are defined: a degree of interest function and contexts based on the process. Studies performed during the Master's Degree created a DOI function to filter and highlight artifacts on a software process execution. This function

takes into consideration particularities inherent to all steps of a software process execution. It works on existing artifacts and calculates a value that is associated to the artifact and that represents its interest to the task being executed. The DOI function and its algorithm are presented in Section 4.4.4.

An innovation that the study brings to the software process execution field in software engineering is the context being based on the software process. The entire initial configuration of the available data to the software engineer is based on the software process. Thus, he is aided on the definition and selection of tasks, on the filtering and highlight of the artifacts associated to these tasks, and also on the calculation of the interest of these artifacts in relation to the tasks, once all of these data are initially gathered from the underlying software process.

## **4.4 Characteristics**

The study performed on this Master's Degree aids software engineers during the execution of the software process with a degree of interest function that classifies artifacts based on their interactions with artifacts and based on the software process specification. It results on the highlight of the most relevant artifacts in relation to the task being executed.

This facilitates artifact visualization, because it reduces non-relevant data being displayed to software engineers. As a consequence, they are able to focus on what is more important to the work being executed.

The implementation developed on this Master's Degree study is based on Mylyn, which is explained on Section 3.9. Indeed, Mylyn adopts similar concepts to solve information overflow and context creation problems. However, its functioning is aimed at implementation only and there is no initial base for contexts, which results in no task contexts initially created.

For the development of the implementation of this Master's Degree study, Mylyn's code has been accessed, some useful parts of that code were modified, some contributions and functionalities were added and the code was recompiled. The final implementation was named MylynSDP (PORTUGAL & OLIVEIRA, 2013, PORTUGAL & OLIVEIRA, 2014).

Figure 4.2 shows components of MylynSDP and their relationship. There are five components: Interface, Software Process Specification Import Mechanism, Restore Mechanism, Saving Mechanism and Degree of Interest Function. Interface is where tasks and artifacts are viewed and where the work is done. Software Process Specification Import Mechanism, as the name implies, imports the software process

specification, while recognizing activities, artifacts and their relationships. Restore Mechanism manages the relationship between tasks and their types (the activities from the software process specification), as well as the relationship between artifacts and their types. The Saving Mechanism persists to disk data about context, interest values and software engineer's interactions with the artifacts. Finally, DOI function is responsible to classify artifacts according to that artifact's interest in relation to the task being executed, as well as define initial contexts based on the imported software process specification. All of the five components were briefly described here are detailed in the next sections.

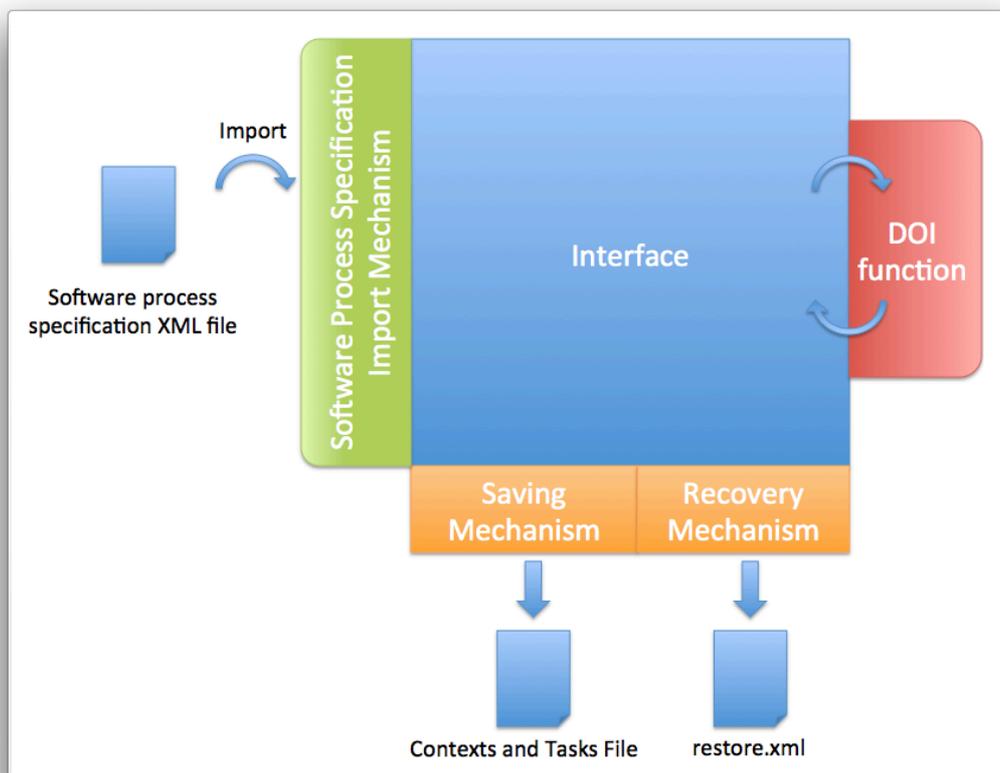


Figure 4.2 - MylynSDP's components and their relationship.

#### 4.4.1 Interface

The interface is where tasks and artifacts are listed and that software engineers perform tasks. Figure 4.3 shows MylynSDP's interface. On the right column of Figure 4.3-a, one may find the task list. Besides each task, there is a button so that software engineers may signal whether a task is being executed. On that column, there is a button for the creation of a new task. During the creation, a wizard helps software engineers to set parameters such as task name and type (Figure 4.4).

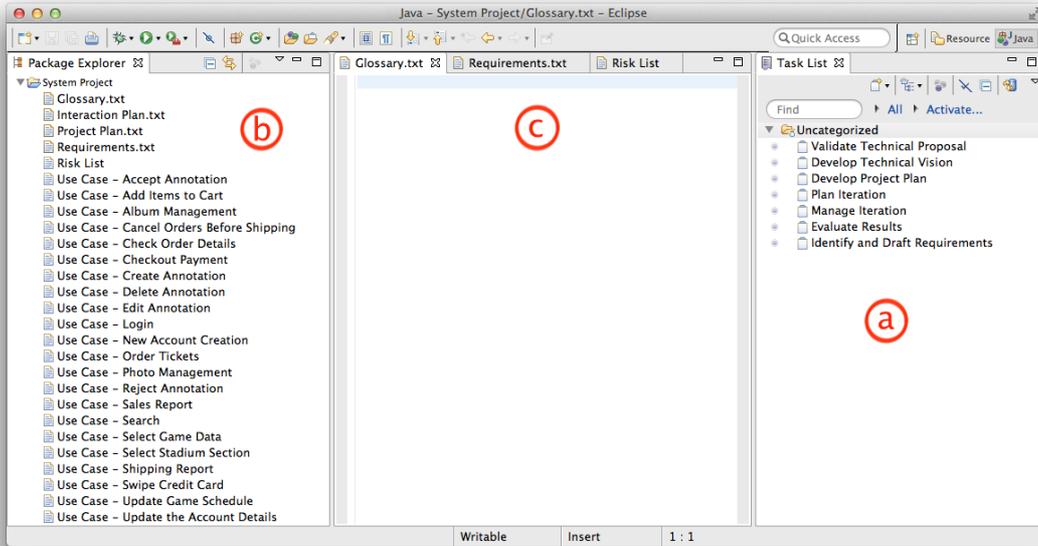


Figure 4.3 - MylynSDP's interface.

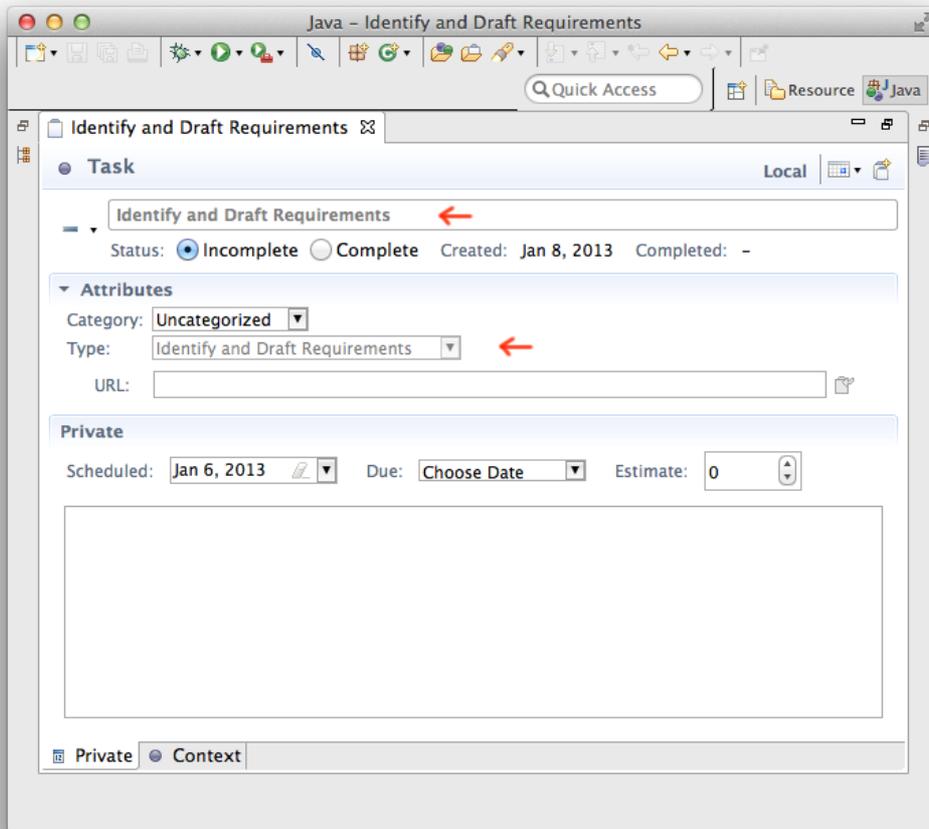


Figure 4.4 - New task creation wizard.

The left column, Figure 4.3-b, shows the artifact list. Initially, no task is marked as active, or having its execution started. Therefore, the artifact list shows all task of the project without any filtering. From the moment that a software engineer activates a task, that is, marks its execution as started, artifacts that do not belong to that task's context, according to the software process specification, are omitted from the software engineer's view. The creation of an artifact is similar to the creation of a common file on the Eclipse IDE. However, as the type of the artifact must be captured, a wizard was developed for that purpose. During the creation of a new artifact, the software engineer sets the name, local and type of the artifact. The wizard is illustrated in Figure 4.5. MylynSDP has been designed to work with one artifact per file. Although it is possible to have several content types in a single artifact (such as an use case artifact with the description of several use cases in it) it is not desirable to have a situation like that. MylynSDP does not consider sections of artifacts because of the sheer number of types of artifacts that it may encounter.

The central area of the interface (Figure 4.3-c) is where artifacts are opened, edited and tasks are executed. If artifacts are opened and the software engineer decides to change the current task, and as a consequence change the context, all of the saved artifacts are closed to make room for the artifacts of the new task.

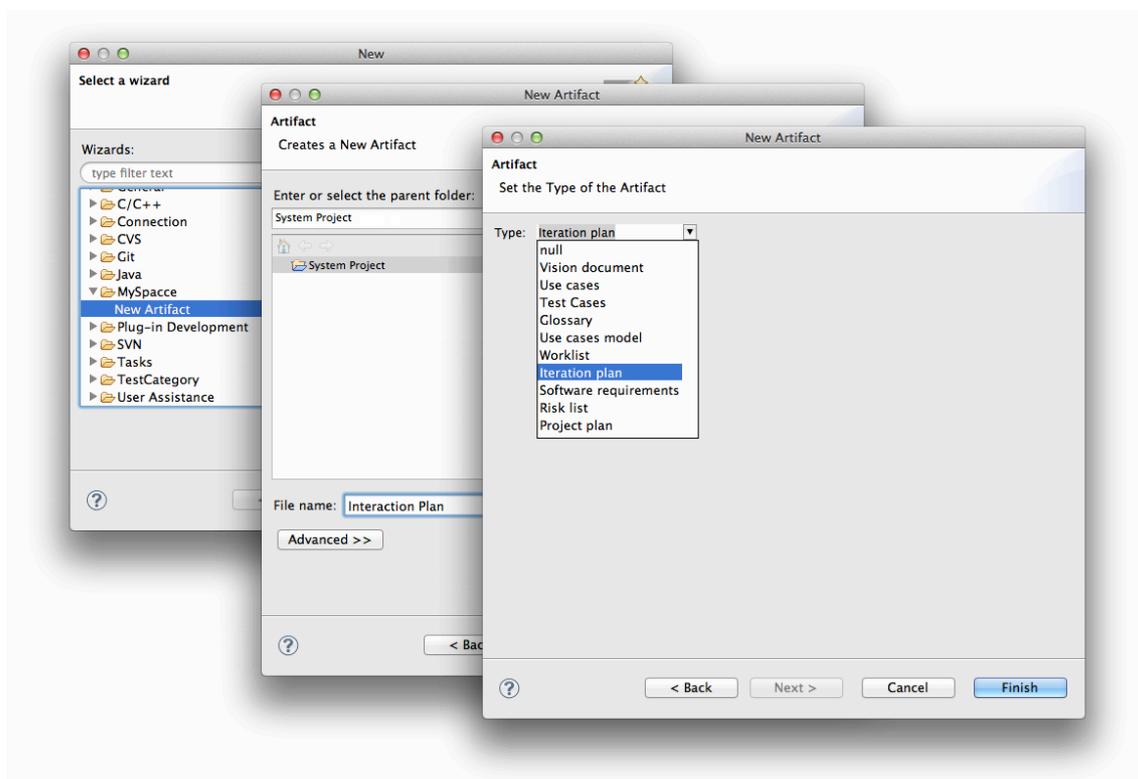


Figure 4.5 - New artifact creation wizard.

#### 4.4.2 Software Process Specification Import Mechanism

All of MylynSDP's working is based on the specification of the software process, starting on the definition of a type for each task and artifact, to the definition of initial task contexts at the moment of task creation. Therefore, all of the facilities provided by MylynSDP starts by importing a software process specification to the plugin.

However, a software process specification document may exist in several formats, such as textual or graphical, or both. For that reason, it was defined that the software process must be imported to MylynSDP in an XML format. Then, XML rules were created for the definition and creation of the software process specification document. These rules are specific for that purpose and are easy to learn and understand. Conversion between the original software process specification document and the XML format is left for the software engineer. A software process specification XML file supported by MylynSDP is illustrated on Figure 4.6. Some XML tags had their content omitted for the sake of simplicity.

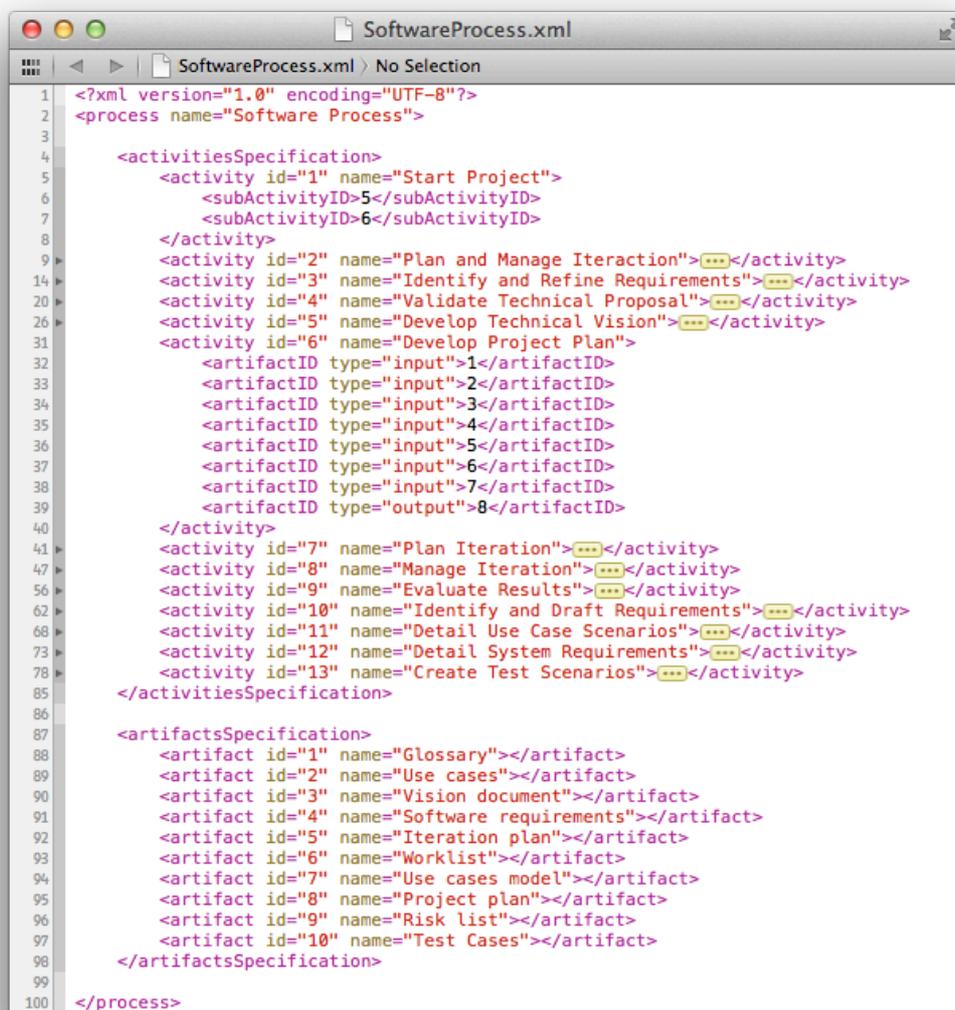
The XML document starts with a `<process>` tag that has the name of the software process as a parameter. Inside that tag, the XML document is divided in two groups by `<activitiesSpecification>` tag and `<artifactsSpecification>` tag. The first one of these tags groups all information about the activities of the software process. `<activity>` tag is used for this purpose. That tag has `"id"` and `"name"` as parameters, which are responsible to set a unique id and a name for the activity, respectively.

Activities may have a relation to artifacts, consuming or producing them, during their execution. Moreover, activities may contain other activities. To represent both cases, `<artifactID>` and `<subactivityID>` are nested on `<activity>` tag. `<artifactID>` tag indicates that a given artifact is used during the execution of the activity that contains it. This tag has `"type"` parameter to indicate if the artifact is consumed (`type = "input"`) or produced (`type = "output"`). On the contents of `<artifactID>` tag is the unique ID of an artifact. `<subactivityID>` tag represents an activity that is inside another one. It does not have any parameters and its contents shows the unique ID of the contained task.

The second part of the software process specification XML document is defined by `<artifactsSpecification>` tag. It is on this part that information about artifacts is described. Each artifact is represented by `<artifact>` tag nested inside `<artifactsSpecification>`. `<artifact>` tag has the parameter `"id"` to store a unique id and the parameter `"name"` that is filled with the name of the artifact.

Software Process Specification Import Mechanism is a modification of the Mylyn's Context and Task Import and Export Mechanism, explained in Section 3.9.2.4.

Although the purpose of both mechanisms is to import an XML, so the plugin may work, the way it is done changes dramatically. MylynSDP's import mechanism accesses the XML to gather information about activities, artifacts and associated contexts, if any. Once this is done, the mechanism allocates space on the workspace of Eclipse IDE to persist the software process specification, so it can be accessed later. After that, the new import mechanism creates a small folder structure in a zip file that contains the software process specification so the import operation may normally continue and Mylyn's import mechanism can create the required workspace structures to keep on with the work.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <process name="Software Process">
3
4     <activitiesSpecification>
5         <activity id="1" name="Start Project">
6             <subActivityID>5</subActivityID>
7             <subActivityID>6</subActivityID>
8         </activity>
9         <activity id="2" name="Plan and Manage Interaction">...</activity>
14        <activity id="3" name="Identify and Refine Requirements">...</activity>
20        <activity id="4" name="Validate Technical Proposal">...</activity>
26        <activity id="5" name="Develop Technical Vision">...</activity>
31        <activity id="6" name="Develop Project Plan">
32            <artifactID type="input">1</artifactID>
33            <artifactID type="input">2</artifactID>
34            <artifactID type="input">3</artifactID>
35            <artifactID type="input">4</artifactID>
36            <artifactID type="input">5</artifactID>
37            <artifactID type="input">6</artifactID>
38            <artifactID type="input">7</artifactID>
39            <artifactID type="output">8</artifactID>
40        </activity>
41        <activity id="7" name="Plan Iteration">...</activity>
47        <activity id="8" name="Manage Iteration">...</activity>
56        <activity id="9" name="Evaluate Results">...</activity>
62        <activity id="10" name="Identify and Draft Requirements">...</activity>
68        <activity id="11" name="Detail Use Case Scenarios">...</activity>
73        <activity id="12" name="Detail System Requirements">...</activity>
78        <activity id="13" name="Create Test Scenarios">...</activity>
85    </activitiesSpecification>
86
87    <artifactsSpecification>
88        <artifact id="1" name="Glossary"></artifact>
89        <artifact id="2" name="Use cases"></artifact>
90        <artifact id="3" name="Vision document"></artifact>
91        <artifact id="4" name="Software requirements"></artifact>
92        <artifact id="5" name="Iteration plan"></artifact>
93        <artifact id="6" name="Worklist"></artifact>
94        <artifact id="7" name="Use cases model"></artifact>
95        <artifact id="8" name="Project plan"></artifact>
96        <artifact id="9" name="Risk list"></artifact>
97        <artifact id="10" name="Test Cases"></artifact>
98    </artifactsSpecification>
99
100 </process>
```

Figure 4.6 - An example of an importable software process specification XML document.

### 4.4.3 Recovery Mechanism

Recovery Mechanism is a brand new functionality, when compared to the original Mylyn. The purpose of this mechanism is to provide a way to browse a software process specification in order to gather some information, keep track of what tasks and artifacts were created and allow a logical relationship between task or artifact and its type (activity or artifact included in the software process specification, respectively).

In order to understand how it keeps track of created tasks, one must know that Mylyn uses a unique internal identification named “*handle*” for tasks. It is through a “*handle*” that MylynSDP is able to locate a task’s context, to know if a given task is active or to apply a command on it, such as activating it. For that reason, it is necessary to persist that unique id along with data about the creation of a new task. Thus, in a future moment, such as to verify if a given task is active, MylynSDP’s code may properly make reference to a particular task. The same applies to artifacts. Mylyn uses a unique compound identifier to make reference to a given artifact. It is formed by the concatenation of the address (folder hierarchy) from the root to the folder that contains the artifact and the name of the artifact. It is possible to use this address as an id because, on Eclipse IDE, two files in the same folder are not allowed to have the same name. Therefore, Recovery Mechanism persists the address of the artifact as well as its name, and also other relevant data such as the type of the artifact at the moment of its creation.

All information described is persisted to disk in a XML file called “*restore.xml*”. This file is created by Software Process Specification Import Mechanism in the current Eclipse IDE’s workspace<sup>1</sup>. Its complete address on the file system depend on the workspace being used and is *[workspace]/.metadata/.mylyn/.restore.xml*. There is only one “*restore.xml*” file for each workspace, regardless of the numbers of projects on this workspace.

Figure 4.7 shows an example of a “*restore.xml*” document. It can be noted from the start that this document is similar to the imported software process specification document. Indeed, the software process specification is integrally copied, by the suitable import mechanism, to “*restore.xml*” file. However, after *<activitiesSpecification>* tag and *<artifactsSpecification>* tag, which store information

---

<sup>1</sup> Workspace is a folder created and used by Eclipse IDE in order to store project’s documents and configuration files. Mylyn and MylynSDP accesses files contained on that folder to get information about artifacts and to store permanent data about software process execution.

about activities and artifacts from the software process specification, respectively, there are two other tags: *<tasksExecution>* and *<artifactsExecution>* tags. The first one contains information about created tasks. It is common that there are more tasks than the ones specified on the software process specification because an activity may be re-executed in two more different tasks. At the moment of the creation of a task on the Interface, the software engineer sets the name and the type of the task. By confirming the tasks's creation, Recovery Mechanism immediately creates a *<task>* tag nested inside *<tasksExecution>* with the parameters "name", which contains the task name, and "type", which contains the task type. This Mechanism also creates a parameter called "handle" and fills it with the unique id Mylyn just used to set the new task. This process is repeated to each new task created.

A similar situation happens with the creation of artifacts. When using the specific wizard to create artifacts, a software engineer sets a directory, a name and a type for the artifact that is being created. After a confirmation, Recovery Mechanism creates an *<artifact>* tag nested inside *<artifactsExecution>* tag on "restore.xml" document with the parameters "url" and "type". The first one of the parameters is filled with the concatenation of the address and the name of the artifacts. The second parameter is filled with the type of the artifact.

The class that contains the code for the Recovery Mechanism is "MylynSDPRestoreXML" and it has several methods that provide different functionalities. These methods are used to interact with "restore.xml" file. Most of them are used by DOI function when registering a new task, artifact or when building up a task context based on data from software process specification. Among the methods included in the class, it can be cited the ones from Table 4.1.

Table 4.1 – Main methods of MylynSDPRestoreXML class.

Method Name	Method Description
<i>createRestoreXml</i>	This method creates, on a specific folder, "restore.xml" file, which is used to store the software process specification as well as data from software process execution such as which artifacts were either used or produced on each activity.
<i>saveTask</i>	This method saves a new task to "restore.xml" file, along its name, "handle" and after linking it to its type (software process activity).

<i>saveArtifact</i>	This method saves a new artifact to “ <i>restore.xml</i> ” file, after getting its unique identifier, which is its location address in the project hierarchy, and after linking it to its type (software process artifact)
<i>getTaskType</i>	This method accesses “ <i>restore.xml</i> ” file and returns the type of a task based on the “ <i>handle</i> ” provided. It is useful when linking tasks and activities.
<i>getArtifactType</i>	This method accesses “ <i>restore.xml</i> ” file and returns the type of a task based on the “ <i>url</i> ” and artifact type provided. It is useful when linking execution and specification artifacts.
<i>getTaskTypes</i>	This method accesses “ <i>restore.xml</i> ” file and returns a list of all existing task types alongside their names. It is useful to discover what tasks have (or should have) a particular artifact in their context.
<i>getArtifactTypes</i>	This method accesses “ <i>restore.xml</i> ” file and returns a list of all existing artifact types alongside their names. It is useful to discover what artifacts belong (or should belong) to a particular task context.
<i>getArtifactsIDsForActivitySpecification</i>	This method returns the unique identifier of artifacts that are associated to an activity context. It is particularly useful to create task contexts when a Specification interaction event is performed because it helps on the linkage between specification activity contexts and execution task contexts.
<i>deleteTask</i>	This methods accesses “ <i>restore.xml</i> ” file and delete an entry for a given task. It is useful when a software engineer deletes a task.
<i>deleteArtifact</i>	This method accesses “ <i>restore.xml</i> ” file and delete an entry for a given artifact. It is useful when a software engineer deletes an artifact.

The Algorithm 4.1 shows a snippet of the code of Recovery Mechanism. The method shown is “saveTask()” method, which is used when a software engineer creates a new task, with the aid of a suitable wizard. This method accesses the “restore.xml” file, looks for the suitable location among XML tasks and registers a new XML tag that represents the created task. The parameters of the new XML tag are filled with information offered by the software engineer about that new task.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <process name="Software Process">
3
4   <activitiesSpecification>
5     <activity id="1" name="Start Project">
6       <subActivityID>5</subActivityID>
7       <subActivityID>6</subActivityID>
8     </activity>
9     <activity id="2" name="Plan and Manage Interaction">...</activity>
14    <activity id="3" name="Identify and Refine Requirements">...</activity>
20    <activity id="4" name="Validate Technical Proposal">...</activity>
26    <activity id="5" name="Develop Technical Vision">...</activity>
31    <activity id="6" name="Develop Project Plan">
32      <artifactID type="input">1</artifactID>
33      <artifactID type="input">2</artifactID>
34      <artifactID type="input">3</artifactID>
35      <artifactID type="input">4</artifactID>
36      <artifactID type="input">5</artifactID>
37      <artifactID type="input">6</artifactID>
38      <artifactID type="input">7</artifactID>
39      <artifactID type="output">8</artifactID>
40    </activity>
41    <activity id="7" name="Plan Iteration">...</activity>
47    <activity id="8" name="Manage Iteration">...</activity>
56    <activity id="9" name="Evaluate Results">...</activity>
62    <activity id="10" name="Identify and Draft Requirements">...</activity>
68    <activity id="11" name="Detail Use Case Scenarios">...</activity>
73    <activity id="12" name="Detail System Requirements">...</activity>
78    <activity id="13" name="Create Test Scenarios">...</activity>
85  </activitiesSpecification>
86  <artifactsSpecification>
87    <artifact id="1" name="Glossary" />
88    <artifact id="2" name="Use cases" />
89    <artifact id="3" name="Vision document" />
90    <artifact id="4" name="Software requirements" />
91    <artifact id="5" name="Iteration plan" />
92    <artifact id="6" name="Worklist" />
93    <artifact id="7" name="Use cases model" />
94    <artifact id="8" name="Project plan" />
95    <artifact id="9" name="Risk list" />
96    <artifact id="10" name="Test Cases" />
97  </artifactsSpecification>
98
99  <tasksExecution>
100    <task handle="local-1" name="Validate Technical Proposal" type="Validate Technical Proposal" />
101    <task handle="local-2" name="Develop Technical Vision" type="Develop Technical Vision" />
102    <task handle="local-3" name="Develop Project Plan" type="Develop Project Plan" />
103    <task handle="local-4" name="Plan Iteration" type="Plan Iteration" />
104    <task handle="local-5" name="Manage Iteration" type="Manage Iteration" />
105    <task handle="local-6" name="Evaluate Results" type="Evaluate Results" />
106    <task handle="local-7" name="Identify and Draft Requirements" type="Identify and Draft Requirements" />
107  </tasksExecution>
108  <artifactsExecution>
109    <artifact url="/System Project/Glossary.txt" type="Glossary" />
110    <artifact url="/System Project/Vision Document.txt" type="Vision document" />
111    <artifact url="/System Project/Requirements.txt" type="Software requirements" />
112    <artifact url="/System Project/Interaction Plan.txt" type="Iteration plan" />
113    <artifact url="/System Project/Worklist" type="Worklist" />
114    <artifact url="/System Project/Project Plan.txt" type="Project plan" />
115    <artifact url="/System Project/Risk List" type="Risk list" />
116    <artifact url="/System Project/Use Case - Order Tickets" type="Use cases" />
117    <artifact url="/System Project/Use Case - Select Game Data" type="Use cases" />
118    <artifact url="/System Project/Use Case - Select Stadium Section" type="Use cases" />
119    <artifact url="/System Project/Use Case - Swipe Credit Card" type="Use cases" />
120  </artifactsExecution>
121
122 </process>

```

Figure 4.7 - An example of a "restore.xml" document.

```

29 public class MylynSDPRestoreXML {
30
31     private static File restoreXml;
32
33     public static void saveTask(String handle, String name, String type) {
34
35         Document doc = readXML();
36         Element root = doc.getRootElement();
37         Element tasksExecution = root.getChild("tasksExecution");
38
39         Element newTask = new Element("task");
40         Attribute _handle = new Attribute("handle", handle);
41         Attribute _name = new Attribute("name", name);
42         Attribute _type = new Attribute("type", type);
43
44         newTask.setAttribute(_handle);
45         newTask.setAttribute(_name);
46         newTask.setAttribute(_type);
47
48         tasksExecution.addContent(newTask);
49
50         writeXML(doc);
51     }
52
53     public static void saveArtifact(String url, String type) {}
54
55     public static void updateTaskName(String handle, String newName) {}
56
57     public static void updateTaskType(String handle, String newType) {}
58
59     public static String getTaskType(String handle) {}
60
61     public static String getArtifactType(String artifactUrl) {}
62
63     public static HashMap<String, String> getTaskTypes() {}
64
65     public static HashMap<String, String> getArtifactTypes() {}
66
67     public static List getArtifactsIDsForTaskSpecification(String taskType) {}
68
69     public static List<String> getURLsForArtifactExecution(String artifactType) {}
70
71     public static List<String> getURLsForArtifactExecution() {}
72
73     public static Document readXML() {}
74
75     public static void writeXML(Document doc) {}
76
77     public static void createRestoreXml(Document doc) {}
78
79     public static void deleteTask(String handle) {}
80
81     public static void deleteArtifact(String url) {}
82
83 }

```

Algorithm 4.1 - Recovery Mechanism code snippet.

#### 4.4.4 DOI function

The objective of this Master's Degree study is to aid software engineers during the execution of software processes with the creation of task contexts to facilitate the visualization of a list of artifacts available to be accessed and by omitting artifacts that

do not belong to the current task context. However, in order to make it, it is essential the use of a classification mechanism according to a predefined criteria. The referenced mechanism is a degree of interest (DOI) function. DOI function classifies artifacts as belonging or not to the task context being executed. This mechanism does that based on two criteria: the interaction events performed by a software engineer with the artifacts and the software process specification document.

DOI function is responsible to gather information of each software engineer's interaction with the artifacts. It is also responsible to access the software process specification to define initial task context to the new tasks being created.

The DOI function modeled and implemented within this Master's Degree work is an extension of the DOI function used by Mylyn. For that reason, the main functionalities (mapped interactions, interaction scores) that the Mylyn's DOI function had are included in the MylynSDP's DOI function. New functionalities were added so that the new DOI function was able to adapt itself to the new reality of software process, such as the ability to access the software process specification and a new type of interaction event.

Table 4.2 shows interaction types that a software engineer may have with artifacts. Besides the name of each interaction event, there is a short description illustrating when each interaction event happens. There are no significant modifications for the five first interaction types. However, it can be noted that a new type of interaction is introduced: the Specification interaction event. This new type is useful for two new functionalities that were introduced in the new DOI function, which is explained below.

Table 4.2 - MylynSDP's types of interaction event.

<b>Interaction Events</b>	<b>Description</b>
Selection	Perform the selection of an artifact with the use of a mouse or keyboard.
Edition	Editions and modifications in the context of artifacts.
Command	Interaction events such as saving.
Propagation	Interaction type that propagates to other related artifacts. Applicable to Java class artifacts only.
Prediction	Event used to predict possible useful artifacts for the current executing task.
Specification	Interaction event performed on artifacts when the initial context of a task is being created based on the imported software process specification.

DOI function considers that artifacts with a positive interest value are interesting, that these artifacts belong to the current task context and that they should be displayed in the software engineer's view. As expected, negative interest values or values equal to zero are not considered interesting to the current task being executed and then they are omitted, and as a consequence, they are excluded from the context. It should be noted that there is not a superior limit one artifact's interest value.

As it was said before, the initial definition of a task context is done based on the software process specification imported when MylynSDP started to be used. Therefore, when a software engineer creates a new task, the DOI function accesses the software process specification, through the Recovery Mechanism, and gathers a list of execution artifacts that initially belongs to the task context of that task. This list is obtained following concepts illustrated on Figure 4.8: once DOI function knows the unique identifier of executing tasks (in the case, the task being created), it is possible to access the "*restore.xml*" document (that keeps every relationship between types and their instances) and: (1) obtain an activity's definition, included on the software process specification, from which the task has been instantiated. By having data about the activity, DOI function; (2) accesses the software process specification to know which specification artifacts are related to this activity. That list of artifacts represents the context of the activity. Based on that specification artifact list, DOI function accesses "*restore.xml*" document again to; (3) find what execution artifacts are instances of those specification artifacts. That new list of execution artifacts forms the task context being searched.

A similar procedure happens on the creation of the artifacts. When a given artifact is created, its type is set, and the DOI function is able to access "*restore.xml*" document, through Recover Mechanism, and obtain a list of artifacts whose contexts should have the given task.

Consequently, if a new task is being created, some artifacts need to have their interest value increased, so they may be added to the new task context. If the software engineer is creating an artifact, that artifact needs to belong to some contexts. In order to introduce some artifacts to some contexts, their interest values in relation to that context need to be increased. However, as no interaction event has ever been performed on that artifact, this would not be possible. Therefore, a new interaction type was introduced: the Specification interaction event. The new interaction event is an indirect event performed by DOI function to increase the interest value of some artifacts. For example, when the creation of a task or artifact is being carried on, DOI function performs a Specification interaction event on the artifacts that must have their interest value increased.

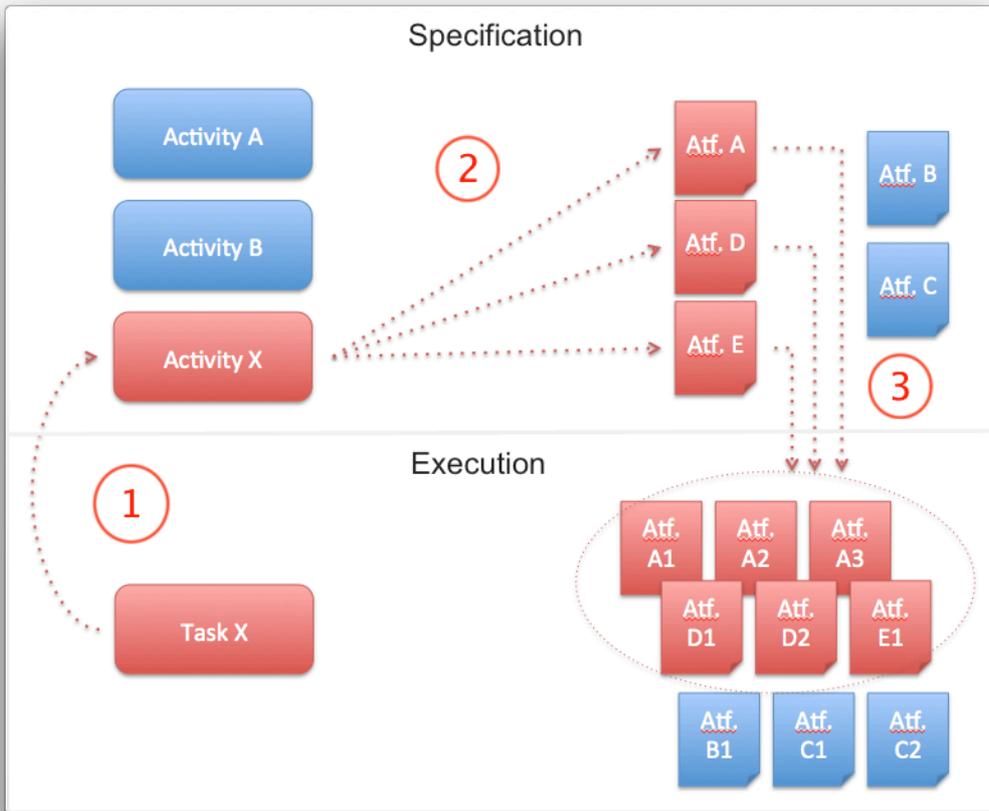


Figure 4.8 - Explanation of the concepts used to create an initial task context.

Specification interaction type needs to increase an artifact's interest value more than other interaction types score contributions in order to guarantee that this artifact will be added to the suitable context and it will remain there for some time even if it is not frequently interacted at first. Therefore, it is a special interaction event that causes a high increase on the interest value of an artifact. Table 4.3 shows the scores that are added to interest values of each interaction event that a software engineer may perform on a software process artifact.

Table 4.3 - Interaction event types and their scores.

Interaction Event Type	Score
Selection	1 point
Edition	0.7 points
Command	1 point
Propagation	1 point
Prediction	1 point
Specification	5 points

Figure 4.9 shows the class diagram of a small part of MylynSDP's entire implementation. Variables and methods were omitted for the sake of visualization simplicity.

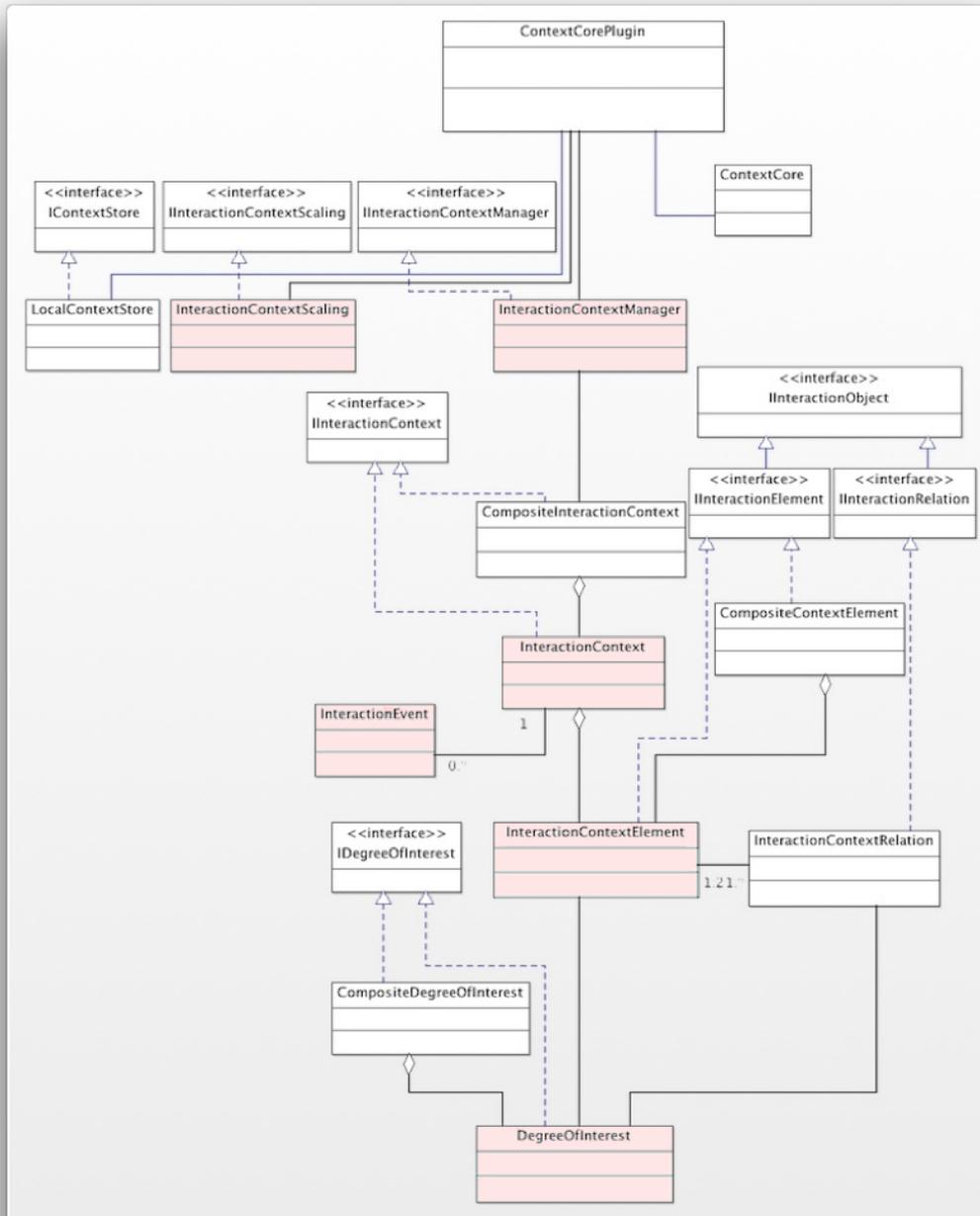


Figure 4.9 - Class diagram of a part of MylynSDP. Some classes are highlighted.

Although there are several classes on the diagram, some of them are highlighted either because they had their implementation modified or because they are important to the understanding of the working logic of DOI function. “ContextCorePlugin” class is responsible to manage everything related to task

contexts, user interactions and interest points. It has a relationship with “*InteractionContextScaling*” class and another relationship with “*InteractionContextManager*” class. The latter class has a relationship with an object of the “*CompositeInteractionContext*” class, which is comprised of objects of “*InteractionContext*” class. “*InteractionContext*” class has a relationship with objects from “*InteractionEvent*” class and it is also comprised of objects from “*InteractionContextElement*” class, which finally has a relationship with an object from “*DegreeOfInterest*” class.

Part of the code of “*InteractionContextScaling*” class is illustrated in Algorithm 4.2. This class contains constants to the calculation of the score given by each interaction event. It can be noted that the variable “*DEFAULT\_EVENT*”, which is 1, is the score constant of an event. However, in the case of an edition interaction event, the score constant is considered 0.7. This value is expressed by the variable “*DEFAULT\_EVENT\_EDIT*”. Besides, the limit value between an interesting artifact and a non-interesting value is defined by “*DEFAULT\_INTERESTING*” variable. Interest values greater than zero are considered interesting and should belong to the task context. It is known that, when software engineers do not interact with a given artifact, its interest value decreases according to the rate of other interaction events are being performed. The decay constant is defined by “*DEFAULT\_DECAY*” variable and has the value 0.4. “*DEFAULT\_DECAY*” constant was modified from 0.017 to 0.4 in order to better filter out results when few interactions are performed during a task execution. Thus, artifacts are filtered more quickly.

```
25 public class InteractionContextScaling implements IInteractionContextScaling {
26
27     private static final float DEFAULT_INTERESTING = 0f;
28
29     private static final float DEFAULT_EVENT = 1f;
30
31     private static final float DEFAULT_EVENT_EDIT = .7f;
32
33     private static final float DEFAULT_DECAY = .4f;
34
35     private final Map<InteractionEvent.Kind, Float> interactionScalingFactors =
36         new HashMap<InteractionEvent.Kind, Float>();
37
38     public float get(InteractionEvent.Kind kind) {
39         if (interactionScalingFactors.containsKey(kind)) {
40             return interactionScalingFactors.get(kind);
41         } else {
42             return DEFAULT_EVENT;
43         }
44     }
```

Algorithm 4.2 - InteractionContextScaling class' code snippet.

The main class responsible to manage the entire set of information about task contexts, artifacts, interaction events and interest values and calculation is “*InteractionContextManager*”. This class has important methods such as “*activateContext()*”, “*addInteractionEvent()*” and “*deleteContext()*”, used to manage contexts. “*InteractionContextManager*” class also maintains a variable to store the current task context being executed. The variable is “*activeContext*”. Algorithm 4.3 is a snippet of the implementation of this class. Methods had their codes omitted in order to display as many methods as possible on algorithm image. “*InteractionContextManager*” full implementation has more than 30 methods and more then 1500 lines of code.

```

50 public class InteractionContextManager implements IInteractionContextManager {
51
52     public static final String SOURCE_ID_DECAY = "org.eclipse.myllyn.core.model.interest.decay";
53
54     public static final String CONTEXT_FILE_EXTENSION = ".xml.zip";
55
56     public static final String CONTAINMENT_PROPAGATION_ID = "org.eclipse.myllyn.core.model.edges.containment";
57
58     public static final String CONTEXT_HISTORY_FILE_NAME = "activity";
59
60     private final CompositeInteractionContext activeContext = new CompositeInteractionContext(
61         ContextCore.getCommonContextScaling());
62
63     private final LocalContextStore contextStore;
64
65     public InteractionContextManager(LocalContextStore contextStore) {}
66
67     public void activateContext(String handleIdentifier) {}
68
69     public void addAttentionEvents(Map<String, List<InteractionEvent>> attention, InteractionContext temp) {}
70
71     public void addErrorPredictedInterest(String handle, String kind, boolean notify) {}
72
73     public void addGlobalContext(IInteractionContext context) {}
74
75     private IInteractionElement addInteractionEvent(IInteractionContext interactionContext, InteractionEvent event) {}
76
77     public List<InteractionEvent> collapseEventsByHour(List<InteractionEvent> eventsToCollapse) {}
78
79     public void deactivateAllContexts() {}
80
81     public void saveContext(IInteractionContext context) {}
82
83     public InputStream getAdditionalContextData(IInteractionContext context, String contributorIdentifier) {}
84
85     public void deactivateContext(String handleIdentifier) {}
86
87     public void deleteElements(Collection<IInteractionElement> elements) {}
88
89     public void deleteElements(Collection<IInteractionElement> elements, IInteractionContext context) {}
90
91     public void deleteElements(Collection<IInteractionElement> elements, boolean isExplicitManipulation) {}
92
93     public void deleteElements(Collection<IInteractionElement> elements, IInteractionContext context,
94
95     public void deleteContext(final String handleIdentifier) {}
96
97     private float ensureIsInteresting(IInteractionContext interactionContext, String contentType, String handle,
98
99     private void eraseContext(String handleIdentifier) {}
100
101     public IInteractionContext getActiveContext() {}
102
103     public Collection<InteractionContext> getActiveContexts() {}
104
105     public IInteractionElement getActiveElement() {}

```

Algorithm 4.3 - InteractionContextManager class' code snippet.

“*InteractionContext*” class represents a task context. It is that class that stores what artifacts belongs to a given context. The variable that holds the representation of

artifacts is “*elementMap*”. Moreover, “*InteractionContext*” class has a variable called “*numUserEvents*”. This variable is used as a counter to the number of interactions that happens in a given context. Calculation of an artifact’s interest value uses that variable to estimate the decay of this value. A code snippet from “*InteractionContext*” class is shown in Algorithm 4.4. Two other important variables found on this class are “*interactionHistory*” and “*contextScaling*”. The former stores interactions that took place when that context was active while the latter is a reference to the class that holds default values for the calculation of artifact interest. Although a new type of interaction must be recorded on “*interactionHistory*” and modifications were made to “*InteractionContextScaling*” class, no major modifications were made to “*InteractionContext*” class because the new interaction event was designed to work just as existing interaction events do and because “*contextScaling*” variable is passed to objects of “*InteractionContextElement*” classes, so they can calculate their interest value.

```
32 public class InteractionContext implements IInteractionContext {
33
34     private String handleIdentifier;
35
36     private final List<InteractionEvent> interactionHistory;
37
38     private final Map<String, InteractionContextElement> elementMap;
39
40     private int numUserEvents;
41
42     private final IInteractionContextScaling contextScaling;
43
44     public InteractionContext(String id, IInteractionContextScaling scaling) {
45         this.handleIdentifier = id;
46         this.contextScaling = scaling;
47         this.interactionHistory = new ArrayList<InteractionEvent>();
48         this.elementMap = new HashMap<String, InteractionContextElement>();
49
50         for (InteractionEvent event : interactionHistory) {
51             parseInteractionEvent(event);
52         }
53     }
```

Algorithm 4.4 - InteractionContext class' code snippet.

On Algorithm 4.5, it is illustrated the definition of the variables of “*InteractionEvent*” class. This class is used to represent an interaction event that software engineers are able to perform on artifacts during software process execution. It is important to note a variable named “*interestContribution*”. It is responsible to store the contribution value a given interaction event will give to the affected artifact’s interest

value. The effective contribution, that is, the increase in points that will happen on the artifact, is the result of the multiplication of “*interestContribution*” by the value of the score constant related to that interaction event. “*interestContribution*” variable is useful when interaction events are combined, for saving purposes for example. Thus, their values are also added up, which reflects on the value of this variable. It can be noted, on this class’ code snippet, the creation and initial definition of Specification interaction event alongside the definition of other interaction events.

```

19 public class InteractionEvent {
20
21     public enum Kind {
22         SELECTION, EDIT, COMMAND, PREDICTION, PROPAGATION, SPECIFICATION;
23
24     @Override
25     public String toString() {
26         switch (this) {
27             case SELECTION:
28                 return "selection"; //$NON-NLS-1$
29             case EDIT:
30                 return "edit"; //$NON-NLS-1$
31             case COMMAND:
32                 return "command"; //$NON-NLS-1$
33             case PREDICTION:
34                 return "prediction"; //$NON-NLS-1$
35             case PROPAGATION:
36                 return "propagation"; //$NON-NLS-1$
37             case SPECIFICATION:
38                 return "specification"; //$NON-NLS-1$
39             default:
40                 return "null"; //$NON-NLS-1$
41         }
42     }
43 }
44
45 private final Kind kind;
46 private final Date date;
47 private final Date endDate;
48 private final String originId;
49 private final String structureKind;
50 private final String structureHandle;
51 private final String navigation;
52 private final String delta;
53 private final float interestContribution;
54
55 public InteractionEvent(Kind kind, String structureKind, String handle, String originId) {
56     this(kind, structureKind, handle, originId, 1f);
57 }
58
59 public InteractionEvent(Kind kind, String structureKind, String handle, String originId, String navigatedRelation) {
60     this(kind, structureKind, handle, originId, navigatedRelation, "null", 1f); //$NON-NLS-1$
61 }
62
63 }
64
65
66
67
68
69

```

Algorithm 4.5 - InteractionEvent class' code snippet.

An element of a context, that is, an artifact that belongs to a task context, is represented by “*InteractionContextElement*” class. It is important to note that, if a software process execution artifact belongs to two different task contexts, two different objects from “*InteractionContextElement*” are created and associated to different task contexts. It allows each object of “*InteractionContextElement*” to have its own specific interest value in relation to a task context. An artifact’s interest value is represented by the variable “*interest*”, which is an object from “*DegreeOfInterest*” class. A code snippet

from “*InteractionContextElement*” class, as well as the definition of its variables, is shown in Algorithm 4.6. The two most important methods of this class are shown in Algorithm 4.6: “*getInterest()*” and “*getContext()*” methods. As expected, the former method returns the interest value associated with the artifact in relation to a given task context and the latter method returns an object from “*InteractionContext*” class that represents the task context that this artifact belongs to. Although there have not been final modifications on this class, “*getInterest()*” and “*getContext()*” methods were edited during implementation of MylynSDP in order to preview the results of modifications on other parts of the code.

```

25 public class InteractionContextElement implements IInteractionElement {
26
27     private String handle;
28
29     private String kind;
30
31     private final DegreeOfInterest interest;
32
33     private final InteractionContext context;
34
35     public InteractionContextElement(String kind, String elementHandle, InteractionContext context) {}
38
39     public InteractionContextElement(String kind, String elementHandle, InteractionContext context,
49
50     public String getHandleIdentifier() {}
53
54     public void setHandleIdentifier(String handle) {}
57
58     public IDegreeOfInterest getInterest() {
59         return interest;
60     }
61
62     public InteractionContext getContext() {
63         return context;
64     }
65
66     public String getContentType() {}
69
70     public void setKind(String kind) {}

```

Algorithm 4.6 - InteractionContextElement class' code snippet.

The class that contains the code of DOI function is “*DegreeOfInterest*”. As it was explained, in order to allow DOI function to classify artifacts, the working of other classes, and thus other components, is essential. Algorithm 4.7 shows the algorithm that governs the operation of DOI function. By examining Algorithm 4.7, it can be noted that the algorithm is divided in three functional parts. The first one is the information storage done by variables. This information is the number of the times that each interaction event type was performed on a given artifact. Variables “*edits*”, “*selections*”, “*commands*” store it. Variables “*predictedBias*”, “*propagatedBias*” are used to increase an interest value if either that artifact suffers an indirect interaction event. “*manipulationBias*” variable is used when a not interesting artifact that once belonged to a particular task context is set to be part of that context again.

```

25 public class DegreeOfInterest implements IDegreeOfInterest {
26
27     private final List<InteractionEvent> events = new ArrayList<InteractionEvent>();
28     protected IInteractionContextScaling contextScaling;
29     private float edits = 0;
30     private float selections = 0;
31     private float commands = 0;
32     private float predictedBias = 0;
33     private float propagatedBias = 0;
34     private float manipulationBias = 0;
35     private float specificationBias = 0;
36     private final float userEventCount = 0;
37     private final InteractionContext context;
38     private final int eventCountOnCreation;
39
40     private void updateEventState(InteractionEvent event) {
41         switch (event.getKind()) {
42             case EDIT:
43                 edits += event.getInterestContribution();
44                 break;
45             case SELECTION:
46                 selections += event.getInterestContribution();
47                 break;
48             case COMMAND:
49                 commands += event.getInterestContribution();
50                 break;
51             case PREDICTION:
52                 predictedBias += event.getInterestContribution();
53                 break;
54             case PROPAGATION:
55                 propagatedBias += event.getInterestContribution();
56                 break;
57             case MANIPULATION:
58                 manipulationBias += event.getInterestContribution();
59                 break;
60             case SPECIFICATION:
61                 specificationBias += event.getInterestContribution();
62                 break;
63         }
64     }
65
66     public float getValue() {
67         float value = getEncodedValue();
68         value += predictedBias;
69         value += propagatedBias;
70         return value;
71     }
72
73     public float getEncodedValue() {
74         float value = 0;
75         value += selections * contextScaling.get(InteractionEvent.Kind.SELECTION);
76         value += edits * contextScaling.get(InteractionEvent.Kind.EDIT);
77         value += commands * contextScaling.get(InteractionEvent.Kind.COMMAND);
78         value += manipulationBias;
79         value += specificationBias * 5;
80         value -= getDecayValue();
81         return value;
82     }
83
84     public float getDecayValue() {
85         if (context != null) {
86             if ((context.getUserEventCount() - eventCountOnCreation) <
87                 initialEventThresold) {
88                 return (context.getUserEventCount() - eventCountOnCreation) *
89                     contextScaling.getDecay();
90             } else {
91                 return (context.getUserEventCount() - eventCountOnCreation) *
92                     contextScaling.getDecay() * 0.5f;
93             }
94         } else {
95             return 0;
96         }
97     }

```

Algorithm 4.7 - DegreeOfInterest class' code snippet.

The new “*specificationBias*” variable exists to cover the case when an artifact belongs to a newly created task context having a software process specification as a base. Every variable described here starts with the value zero. It is noted the presence of the variable “*contextScaling*”. That variable is the one that allows DOI function to obtain the score constants related to each of the interaction events. The variable “*eventCountOnCreation*” stores the ordinal value of the interaction event that happened when the artifact was added to that context. This value is used to calculate the decay of the artifact’s interest value. An object from “*InteractionContext*” value that represents the context manages the decay value.

The second part of DOI function’s algorithm is more functional. It is responsible to update the number of selections, editions or any other interaction types that are being performed on that artifact. The main method of this part is “*updateEventState()*”, which will be explained in detail later. The third part of the code is the calculation of that artifacts’ interest value itself. This part includes three methods: “*getValue()*”, “*getEncodedValue()*” and “*getDecayValue()*”. Together, these methods are able to return an interest value to the current artifact.

The concept behind of the algorithm of DOI function is as follows. First, each interaction event is multiplied by its score constant. After that, each final result is added up to define a partial interest value. However, as an artifact’s interest value decreases as the software engineer performs other interaction events, the algorithm calculates a decay value to be subtracted from the partial interaction value. After the subtraction, the artifact has a final interest value defined.

The work of the DOI function starts with the interaction events performed by the software engineer. At each interaction event performed, “*updateEventState()*” function of the artifact on which the interaction event was performed is called. This function receives the object of the “*InteractionEvent*” class that represents the interaction event, checks the type of interaction event and updates the corresponding variable. “*getInterestContribution()*” method returns that interest event’s contribution value as it was explained.

Artifacts’ interest values are updated at each interaction event performed by the software engineer, or from time to time, if he does not interact with the artifacts. In order to access an artifact’s interest value, “*getValue()*” method must be used. As it can be seen in Algorithm 4.7, the variable “value” is filled with the final value of “*getEncodedValue()*” method. The first instruction of the second method is to define the value zero for “value” variable. Just after this, multiplication between the number of selections that artifact has suffered and the selection interaction event score constant is made. The referenced constant is a number obtained through “*get()*” method of

“*contextScaling*” variable, which is an object instantiated from “*InteractionContextScaling*” class. That is the class that contains all constants. The same multiplication is performed for edition and selection interaction event types. Next, value of “*manipulationBias*” variable is added to “*value*” variable. This is particular useful when that artifact is about to be added to that task context other than the first time. Once it is done, the multiplication between the variable “*specificationBias*” and its score constant, which is 5, is added to the partial interest value. “*specificationBias*” variable does not have a value other than zero or 1. Initially, its value is zero, and it is changed to 1 only if a Specification interest event has happened. A Specification interest event type is performed on one or more specific artifacts right after the creation of a new task or the creation of a new artifact when Recovery Mechanism, after accessing the software process specification, discovers that some given artifacts should belong to some given task contexts. The occurrence of this interaction event indicates that the artifacts whose interest values are being calculated must belong to the task context of the task being executed, as it was defined in the software process specification. Therefore, if the value of “*specificationBias*” variable is 1, the interest value will be largely increased. Currently, its value is increased by roughly 5 points. This value is arbitrary and it was reached from observations and tests applied to the DOI function. After the increase or not of this value, the artifact’s interest value is decreased.

In order to calculate the decay value of a given artifact’s interest value, “*getDecayValue()*” method is used. This method needs to know how many interaction events happened since the creation of that artifact. Thus, it calculates the difference between the ordinal number of the last happened interaction event (stored in “*context*” variable) and the ordinal number of the interaction event that happened at the moment of the creation of the artifact (stored in “*eventCountOnCreation*” variable). The resulting value is then multiplied by the decay constant contained in “*contextScaling*” variable. However, in order to address the issue of low interactions on some task contexts, “*getDecayValue()*” method was modified from the original method. Right before calculating decay value, a subtraction between the ordinal number of the last interaction event and the ordinal number of the interaction event that added that artifact to the current task context is made. It is useful to discover how many interactions events were performed until the current moment. Depending on the result of that subtraction, “*getDecayValue()*” method may have its final decay value reduced to its half or not. If reduced, it means that the method’s power will be a bit weaker, which is suitable for low interacted contexts. After that, “*getDecayValue()*” methods returns a partial value. “*getEncodedValue()*” method is then able to subtract from “*value*” the

interest's decay value. Once it is done, "value" is returned to "getValue()" method, so it can be added to the values of the variables "predictedBias" and "propagatedBias". This happens when the artifact is the subject of a propagation indirect interaction event or when it has been predicted that it will be used in another context. Finally, "value", which stores the interest value of that artifact, is returned to the suitable parts, and DOI function algorithm is over until the next software engineer's interaction, which restarts all calculations all over again.

Figure 4.10 shows the execution flow of the creation and use of tasks and artifacts on MylynSDP.

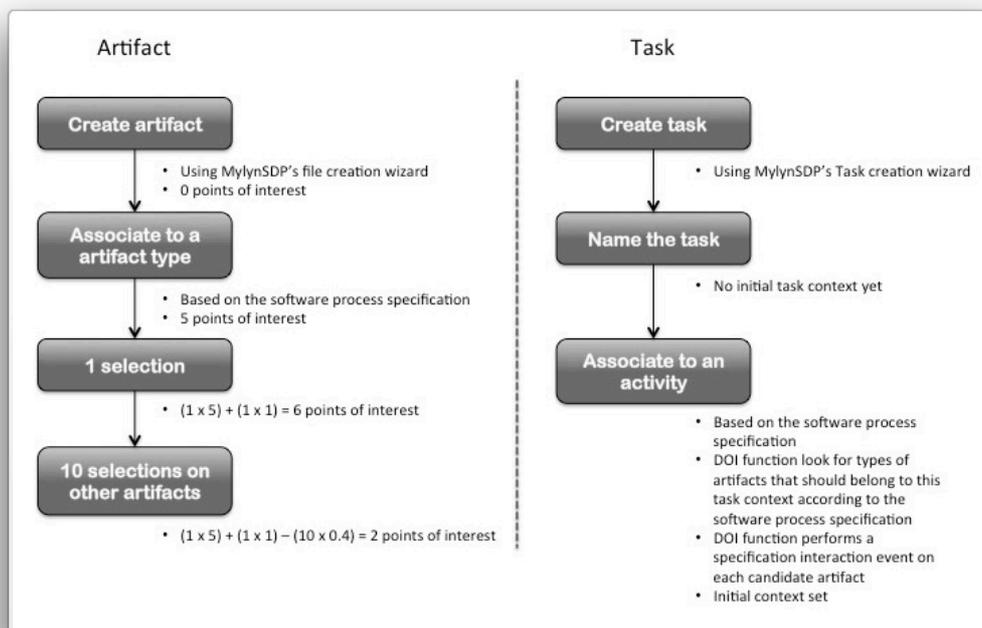


Figure 4.10 - An example of Mylyn's DOI function usage. A task interaction process example in four steps on the left and a task interaction process example in three steps on the right.

MylynSDP's usage may start with the creation of either a task or an artifact. When creating a task, software engineers must use MylynSDP's Task Creation wizard that was specifically designed to it. An entry will be listed in MylynSDP's Task view but no context is created until that moment. Next, software engineers may give a name to the new task or set any other parameters such as the expiration date or the priority of the task. Right after that, and most importantly, software engineers should set the task type, which is the software process activity on which that task is based. There is a list of task types, based on the software process imported before any task or artifact creation. Once a task type is set, it cannot be changed later. Following the setting of a

task type, DOI function performs some actions. First, it checks the task type set. Second, it accesses the software process specification and looks for the activity that represents the new task. Third, DOI function seeks what are the artifact types that relates to that activity, which is that activity context. Finally, DOI function scans the list of already created artifacts and look for software execution artifacts with that type. Once those artifacts are found, MylynSDP's DOI function performs a specification interaction event on them, so their interest value can be increased, which makes these artifacts belong to the new task's context and creates an initial context for that task.

When creating an artifact, software engineers may use MylynSDP's file creation wizard. At the second screen of the wizard, software engineers set the software process artifact on which the current artifact is based. Once it is finished, an entry is created on MylynSDP's Artifact view for the new artifact. In addition to it, MylynSDP's DOI function may begin to act. If no task was active at the creation moment, then DOI function does nothing. However, if any task was active, it means that the creation of the artifacts is relevant for that task's execution. For that reason, DOI function needs to increase the interest value of the new artifact in relation to the current task. DOI function does this by performing a specification interaction event. It should be noted that DOI function does that even if the new artifact does not belong to that task context according to the software process specification. If there are no tasks created, then DOI function cannot work. Once the new artifact received a specification interaction event, its interest value goes to 5 points. When supposing that a software engineer performed one selection to the just created artifact, its interest value is increased by the number of interactions performed times the points that this interaction contributes to the interest value. Up to the moment, that artifact's interest value is  $(1 \times 5) + (1 \times 1) = 6$ . It is then assumed that the software engineer performed ten other interaction events on other artifacts that were created earlier. Every one of these interactions contributes to a decrease in the interest value of the artifact used in this example. DOI function calculates the decay value as the multiplication of a constant of decay by a value that represents the number of interactions performed since the creation of that artifact, excluding the creation interaction event. The final interest value is then the subtraction of the current interest value and the decay value. When put in numbers, calculations happens as follows  $(1 \times 5) + (1 \times 1) - (10 \times 0.4) = 2$  points of interest.

After a certain number of interactions performed to another artifacts, MylynSDP's DOI function notes that the interest value of the artifact used in this example is negative and decides to exclude it from the current task context regardless if it should belong to it according to the software process specification. For that reason,

differences in the execution and the specification of a software process may be verified.

Indeed, all of the algorithms described in this section are important to the full understanding of DOI function and MylynSDP workings. However, not all of them had modifications on their code for the final version of MylynSDP. The ones who were modified had their modifications explained on their respective descriptions. In any case, the Table 4.4 below summarizes what classes have been edited for MylynSDP implementation and, for the classes that were not edited, it shows the reason they were included in the Dissertation.

Table 4.4 - Summary of the description of the six algorithms characterized on this section.

<b>Classes</b>	<b>Modifications</b>
InteractionContextScaling	Modifications made on constants such as <i>DEFAULT_EVENT_EDIT</i> and <i>DEFAULT_DECAY</i> .
InteractionContextManager	No modifications – This class was described due to its importance for the system, which can be noticed by checking its name and size.
InteractionContext	No modifications – This class was described for understanding purposes.
InteractionEvent	Modifications on the registration of Specification interaction event. It is decided on this class when this event occurs and how it is saved and treated.
InteractionContextElement	No modifications – This class was described for understanding purposes. It is important to know how elements relate to contexts and how they store their interest value.
DegreeOfInterest	Modifications made on this class includes: <ul style="list-style-type: none"> <li>• Changes to the way of calculating interest values</li> <li>• Consideration of changes for interaction event constants</li> <li>• Consideration of Specification interaction event.</li> </ul>

#### 4.4.5 Saving Mechanism

Mylyn and MylynSDP aid software engineers to execute their jobs. While Mylyn is focused on the implementation of a system, MylynSDP, described in this Dissertation, aims to work on all steps of a software process. The way software engineers are aided is with the filtering of some artifacts that are non-relevant to the task being executed, that is, artifacts that do not belong to that task's context. The task change is a procedure that forces software engineers to release some artifacts, which will be reused later. The release of artifacts also happens when software engineers have to leave their workspace and close the IDE, the Eclipse IDE in the case. In both cases, the presence of a mechanism that is able to persist to disk data about the usage of task contexts is important.

Saving Mechanism is the component responsible to persist to disk information about the existence of tasks, information about their contexts, as well as information about every interaction event that happens during the execution of work on the IDE. It is from that information persistence that a task context can be recreated after a task change, or at the moment a software engineer reopens the Eclipse IDE. Mylyn, the framework on which MylynSDP was based, keeps a hidden folder structure on the workspace in use. It is on that structure that Saving Mechanism works. Figure 4.11 shows this folder structure.

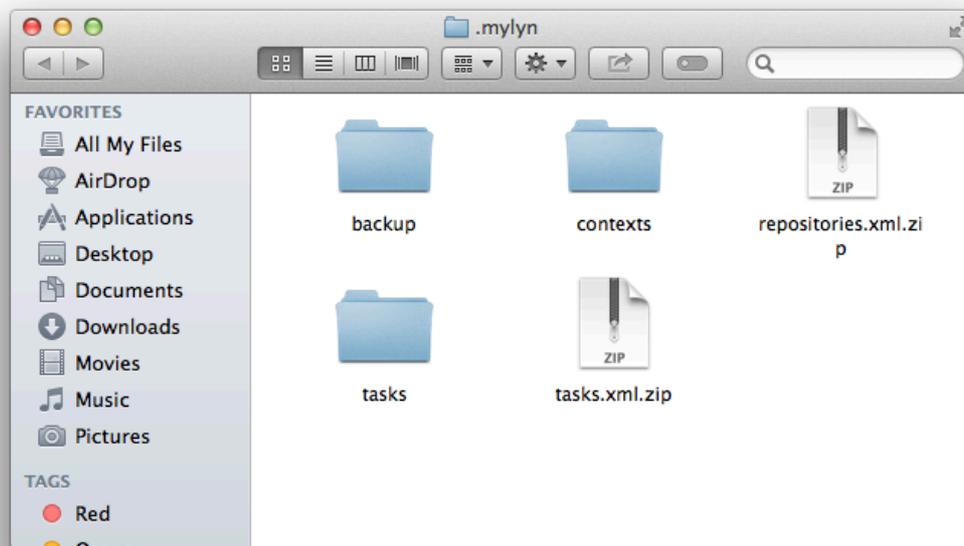


Figure 4.11 - Folder structure managed.

It can be noted the existence of three folders and two files. “*repository.xml.zip*” file keeps an .xml file with data about repositories that are being used. Although Mylyn supports the use of remote repositories, MylynSDP is aimed at the local repository, that is, the workspace used by Eclipse IDE. “*tasks.xml.zip*” file contains an XML file with the main information about the task list and its tasks. Information about created tasks is stored on the parameters of a `<Task>` tag nested under a `<TaskList>` tag on this file. It is also stored on this document whether a task is active or not, the type of the repository where that task is stored, the task’s creation date, finish date, estimated time, name and priority. All of these parameters can be set at the moment of the creation of a task. It is also persisted an identifier for that task. “*tasks*” folder contains as much XML files as there are tasks created by a software engineer. Mylyn uses this space to store the name of the files that were opened at the moment of a task context was finished, either because of a task change or because of Eclipse shut down.

“*contexts*” folder also has one XML file to each task created by a software engineer. It is on this folder that some information about task contexts is persisted. Opposed to general intuition, neither Mylyn nor MylynSDP persist what artifacts belongs to a given context. Instead, their suitable Saving Mechanism stores data about what interaction events were performed while that task was active. By doing that, every interest value of every artifact of a task context are recalculated at each task change or at each start of Eclipse IDE. However, it would be computationally expensive if information about every single interaction event that happened in a task context were stored. For that reason, Mylyn and MylynSDP group interaction events that have the same type and the same target element. This grouping creates a compound interaction event whose score contribution in an interest value is the addition the contribution value of each grouped interaction event. Information saved on XML files existent on “*context*” folder are stored on the parameters of `<interactionEvent>` tag and are related to the instant when the interaction event happened, the instant when the interaction event finished (for the case of grouped interaction events), target artifact, interaction event type, interest value’s contribution, ordinal number that represents the order of this interaction event among all events ever performed and the number of grouped events, if applicable.

It should be noted that all of this information, displayed on the folder structure described, are imported and exported by Mylyn’s Context and Task Import and Export Import Mechanism. Moreover, MylynSDP’s Software Process Specification Import Mechanism converts the imported XML document to a Mylyn readable format, as it was explained on Section 3.9.2.4, and saves it on that folder structure.

Major modifications were not made on the code of this mechanism. The main contribution was the adaptations that had to be performed so a new interaction event was considered valid by this mechanism. Therefore, the new interaction event type was registered on “*InteractionEvent*” class and a call for the method “*processInteractionEvent()*” is performed right after the creation of a new task or a new artifact. Saving Mechanism treats the new Specification interaction event as a normal event. It also stores it and gets it when necessary, the same way it is done with other interaction events. Algorithm 4.8 shows the code snippet of the registration of an interaction event of the type Specification.

```
56
57 ITask activeTask = TasksUiPlugin.getTaskActivityManager().getActiveTask();
58
59 if(activeTask != null) {
60
61     ContextCorePlugin.getContextManager().processInteractionEvent(
62         new InteractionEvent(InteractionEvent.Kind.SPECIFICATION, "resource",
63             fullPath, "org.eclipse.jdt.ui.PackageExplorer", 1f)
64     );
65
66 }
67
68
```

Algorithm 4.8 - Code snippet with the registration of a Specification interaction event.

## 4.5 Conclusion

On this chapter, MylynSDP was introduced. MylynSDP is an extension of Mylyn Eclipse plugin developed to solve artifact search and context change problems in all phases of a software process execution and thus help software engineers to be more productive. MylynSDP features a new Degree of Interest (DOI) function that monitors interaction events performed by software engineers with artifacts and that infers a particular artifact interest in relation to an activity being executed. As explained on this chapter, MylynSDP’s DOI function work is based on the software process specification imported at the beginning of the execution.

Some meaningful modifications to Mylyn and its DOI function’s original code were implemented. Four main differences can be spotted. Firstly, MylynSDP has the ability to import software process specifications and to process what activities and artifacts are included on a specification as well as their relationship. This information constitutes an initial activity context and it is used on creation of task contexts.

Secondly, two wizards were implemented to assist software engineers when creating tasks and artifacts for software process execution. Both wizards are necessary to link execution elements (tasks and artifacts) to specification elements that represent their types (activities and artifacts). Thirdly, the Specification Interaction event was introduced in order to help DOI function create initial contexts for new tasks or new artifacts. Fourthly, Mylyn's DOI function calculations were reviewed and some values were changed. This was done to cover new realities where artifacts are less or more interacted.

## 5 Validation Study

*On this chapter, a validation study is presented and explained in details. This validation study was conducted following formal guidelines from experimental studies but cannot be considered one due to the lack of comparison of results to a treatment. This validation study is based on TAM questionnaire and its results are also described here.*

### 5.1 Introduction

This Master's Degree Dissertation studies a way to help software engineers better find interesting artifacts related to the task being performed during the execution of a software process. When a sheer number of artifacts are available to be used on a software process task, software engineers need to spend additional time and effort in the search for the suitable artifacts rather than spend them on the work proposed by the task. As a consequence, their productivity may decrease. DOI function is a mechanism that helps software engineers to automatically locate suitable artifacts related to the task being executed by classifying them according to their interest to the current task. An extended version of Mylyn's DOI function was implemented featuring two aspects: the ability to deal with any software process artifact, from conception to delivery and maintenance phases, and the ability to access the underlying software process to create initial task contexts, which makes it a process-based DOI function. The final implementation was named MylynSDP.

As explained in the previous section, MylynSDP is an Eclipse plugin that contains the DOI function subject of this Master's Degree study. The underlying assumption that motivated its study is that the search for artifacts during a software process execution may negatively affect the productivity of software engineers. Thus, after implementing the DOI function with the new concepts, an experimentation study should be conducted to assess the assumptions predefined when the study has been initiated. However, due to time difficulties and shortage of participants, it was decided to conduct a formal validation study rather than an experimental one.

For that reason, a validation study was carried out from October 2013 to November 2013 in order to validate the concepts introduced by this Master's Degree study. The validation study consisted of three steps. The first one was a training phase

on which participants were taught concepts needed to understand the validation study such as the DOI function mechanism, the project they were about to deal with and the software process they would be executing. The second step asked participants to interact with MylynSDP and its DOI function when executing a software process by performing and solving five guided exercises, in which they had to access multiple artifacts. On the second step, participants were asked to answer a questionnaire about their experience when dealing with MylynSDP's DOI function. Moreover, the time each participant took to execute each task was recorded for analysis purposes. In addition to time, comments were noted in order to help to draw conclusion about each participant's behavior when executing the proposed exercises and dealing with artifacts. The validation study has been tested twice before its execution, and improvements were made on the text of the exercises to avoid misunderstandings and on the implementation of DOI to better filter results. The validation study is described in detail in the next paragraphs.

This chapter is divided in nine more sections. Section 5.2 explains the software process used on the validation study and its origin. Section 5.3 describes participants that took part in this validation study, as well as their knowledge degree of Eclipse's working and level of expertise with software processes. Section 5.4 is about the training phase participants went through so they could better understand what was the study about and what were some particularities of the chosen software process, such as artifact naming convention. Section 5.5 is related to the explanation of validation study exercises that participants had to solve. Each exercise is described in detail along with the goal of proposing that exercise. Section 5.6 describes Technology Acceptance Model (TAM) questionnaire and the statements participants had to judge. Section 5.7 is concerned with analysis done to validation study results. Answers to TAM questionnaire and the time measured during execution of exercises are displayed and discussed. Section 5.8 presents threats that could affect the validation of the study. Section 5.9 concludes this chapter.

## **5.2 Software Process**

In the middle of 2013, a real software process, with a great number of artifacts, was searched in order to conduct the validation study. The chosen software process comes from SIGA/EPCT project (SIGA EPCT, 2014). SIGA/EPCT project is managed by federal academic institutions in Brazil and it aims at developing an integrated system to manage academic institutional routine processes. The final objective is to manage all academic data of federal institutions in Brazil that includes information

about student and professor registrations, institutes, departments, teaching rooms, classes, teaching materials, attendance ratings, researches being conducted and research papers. The project is endorsed and sponsored by Brazilian Government.

The system being developed is also named SIGA/EPCT. Brazilian federal academic institutions are currently developing it for their own use, which makes them their own client. In other words, they are the ones that are developing the systems and, at the same time, they are the ones that will use it. The system is being developed using open-source technology and it works online, which means that a user needs a browser and an Internet connection to access it. One of the available functionalities of the current version of the system is the ability that students and federal academic workers have to register for an ID that proves that they study or work on that institution. The system is online in full-time.

Access for the software process that is guiding the development of SIGA/EPCT was granted for the purpose of this Master's Degree research. An authorization is attached at the end of this Dissertation on APPENDIX A. As the development of the system was being carried out by the time the software process was accessed, it is possible that the software process taken for this research differs from the software process that is being used right now on the project because modifications were made to the software process after the access made to gather it. Additionally, several artifacts were copied, as they were currently, to be used on the validation study of this Master's Degree. Copied artifacts include Use Case documents, Requirement documents, Business Rule documents, Test Cases documents, Glossary, Interface Project, Database Models and SQL scripts.

The software process specification accessed from SIGA/EPCT has 10 activities and 14 types of artifacts. Each of the activities included in the software process is executed for a single use case and, when finished, the process is restarted for another use case. Participants had a copy of the software process printed in paper and they had another copy of the software process in a digital file. During the validation study, participants had to deal with more than 350 available artifacts, which is the real number of artifacts on the chosen project. Artifacts were not modified and they had their name and content preserved. The software process specification of SIGA/EPCT project is attached on APPENDIX A at the end of this Master's Degree Dissertation.

### **5.3 Participants**

Invitations were sent to students with software engineering background and seven of them were able to participate in the validation study of this Master's Degree.

Therefore, an appointment has been scheduled with each one of them so they could operate a computer with Eclipse's plugin MylynSDP and the new DOI function installed. During the months of October 2013 and November 2013, each of the seven participants attended to the invitation. The validation study has been conducted with six Doctorate students and one Master's Degree student of software engineer.

All participants were students with the Software Engineering group at COPPE Department at the Federal University of Rio de Janeiro (UFRJ) in Brazil at the time of the execution of the validation study. Their experience with software development process ranges from theoretical classes to actually working as a group in industry. Three of the participants have already worked with software development process in a real-world environment in industry either by modeling or executing them. Other three participants stated that their experience with software processes are academic-related and was carried out on a course. One participant described its experience with software process as limited to a subject-related course in the past.

In addition, participants were asked about their experience with Eclipse IDE. Five of them answered that they have used Eclipse either on industry-related projects or on their own projects. Two participants did not have practical experience with Eclipse though. One of them answered that their experience is limited to studying it on class or on textbooks. Another participant declared that their experience with Eclipse was none.

## **5.4 Training**

After choosing the software process and selecting participants, it was time to start the validation study training. It was carried out individually, like the validation study itself, at Federal University of Rio de Janeiro (UFRJ) with each of the seven participants. Some documents were prepared in order to help the researcher to gather data about the study. The documents were Time Record Form, Validation Study's Exercises document, printed Software Process Specification, Characterization Questionnaire, Consent Form and Final Questionnaire. Each of these documents is attached on APPENDIX B at the end of this Master's Degree Dissertation.

Initially, participants were asked to read and sign the Consent Form. This document represents an authorization provided by participants to use the data collected for academic purposes only and this document assures that the study is confidential, though no personal information will be shared. Once participants agreed in taking part of the validation study, they were given the Characterization Questionnaire. The objective of this document is to learn about participants' experience with Eclipse

IDE and software development processes as well as their level of education. Next, participants went under a training, which will be detailed in the next paragraph, to learn about the concepts they were about to deal with. At the end of the training, participants had access to the software process, both on paper and digitally, and then the validation study effectively started. While reading the exercises proposed on the Validation Study's Exercises document and performing tasks on the MylynSDP on a computer, the researcher was able to register task execution time on the Time Record Form as well as some comments. Finally, when the execution of the tasks has finished, participants were asked to answer a final questionnaire comprised of twelve questions. The final questionnaire is detailed in Section 5.6. The entire validation study took approximately one hour for each participant.

The training step took place moments before participants were able to perform tasks. A short presentation was created to explain what a software process is, its importance, the problem that motivated this study and the solution provided. The presentation also explained how to operate MylynSDP, that is, how to start and finish an activity, where tasks and activities are and what one should do if an artifact is not found on a task context. In addition to it, the training phase explains SIGA/EPCT project, its objective and the naming convention used for artifacts. Participants were free to interrupt the explanation being given to ask questions whenever they wanted. Furthermore, at the end of the presentation, participants were asked if they had any question before starting the validation study.

## **5.5 Validation Study Exercises**

After the training step, the Validation Study's Exercises document was handed to participants. It was a 5-page document containing guided exercises that participants were asked to solve. Exercises are described in detail later in this section, but they were not complex, because the way participants get access to artifacts is what being studied, not the complexity of tasks. Exercises required participants to know what software engineering documents are and demand participants to either create, complete or edit a short paragraph of the description of components. Additionally, participants were presented with a computer running Eclipse with MylynSDP and its DOI function. The underlying software development process had already been imported on MylynSDP and five tasks had been created representing the five exercises participants had to solve. Participants were asked to solve one exercise at a time and in the proposed order. Moreover, participants were free to ask questions during the

execution of exercises. The study was conducted on an iMac running Mac OS 10.8.5 (Mountain Lion) and Eclipse 4.3.1 (Kepler).

Each of the five exercises in the Validation Study Exercises document started with a brief text that contextualizes participants on the software project being executed. It explains the software process activity that was about to be executed as well as the use case that relates to it. However, the explanation text did not give any clue about the documents that would be necessary to complete the exercise, neither their location. As previously said, the time participants took to solve each exercise was recorded. Time measurement do not consider the time participants took to read the exercise, but only the time needed to solve the exercises, which starts from the moment participants activate a task to the moment they consider it done. After that, they were asked to not interact anymore with MylynSDP, because most interactions could be misunderstood as valid DOI function's interaction events on artifacts, which would be false.

Table 5.1 shows a brief description about each proposed exercise and its objective. Exercise #1 explains that participants were assigned to two use cases and that the execution of the software process activities relates to these use cases. Next, exercise #1 asks participants to access one of the use cases' specification and create a brief description about it. Before the start of the exercise, MylynSDP's Artifact view shows every one of the more than 350 artifacts of the project. By the time participants activate the proposed task, MylynSDP's DOI function accesses the software process specification and filters out more than half of artifacts displayed. From that time, as long as participants interacts with artifacts, either selecting them when browsing or opening them to check them, DOI function updates that artifact's interest value and filters out not interesting ones. The objective of this exercise was to observe participants actions when facing a low filtered task context because it was the first time that task was being executed.

Exercise #2 demands participants to correct a use case description and to access both a requirements document and a business rule document in order to complete the same use case description. However, it is then assumed that few interaction events have been previously made to some artifacts, in addition to the interactions made by DOI function when accessing the software process specification. The objective of this exercise is to compare observations made when executing exercises #1 and #2, which deals with low and normal filtering.

Table 5.1 - Exercises of validation study.

Exercises	Objectives
1	Participants need to write down a use case description based on two other artifacts. This exercise is aimed at monitoring participants' behavior when facing low filtering scenario due to the lack of interactions with artifacts.
2	Participants are required to edit a use case description according to two other artifacts. However, this exercise simulates a case in which software engineers are not new to the executing tasks because some interactions have been performed on the task context.
3	Participants review a use case description, which has acronyms written on it. Participants then seek for Glossary artifact, which do not belong to this task context based on the software process. This exercise is aimed at observing participants' reactions and what they do when a needed artifact do not belong to the executing task context.
4	Participants write down a brief description for a test case after accessing other test case descriptions and after checking a note left on a use case description. This exercise was created to simulate a task change when interrupted by exercise #5.
5	Participants are asked to create an SQL table code based on a database model and on other existing SQL codes. This exercise was designed to interrupt exercise #4's execution and simulate a task change, so reaction of participants can be observed.

Exercise #3 shows participants a note left by another software engineer in which it is written a message that says that the description of a third use case, not initially assigned to the current participant, was completed on the previous day, but it was not revised. Thus, the second software process specification's activity was not fully performed. Participants then are asked to revise the use case description pointed in the message and to search for misspelled words, grammar mistakes and, most importantly, the use of acronyms. It was said that acronyms were not allowed, so that participants would have to access the Glossary document in order to write down the full word an acronym means. However, Glossary is not defined on that activity context according to the software process specification. As a result, it was filtered out and did not appear to participants in the suitable view. The objective of this exercise was to observe what participants would do when facing a situation in which a needed artifact did not belong to the current task context. Although there is a button to show all the other artifacts that are not in a particular task context, there are several ways an artifact can be found when it is filtered out, which includes opening another task in whose context there is the desired artifact, make a search on the IDE for the missing artifact or even perform a search on the file system for the artifact.

Exercises #4 and #5 simulates a context change. The former exercise asks participants to briefly describe a single test case based on the tests that are specified

to be made. However, when participants started to write down some words of the proposed description, exercise #5, a high priority task, interrupted them demanding that they switched their attention to the new task. Exercise #5's explanation text says that software engineers of the organization needs participants to complete an SQL document based on a database model document accessed when performing the last activity of the software process specification. After finishing exercise #5, participants were allowed to return to exercise #4 and finish it also. As explained, the objective of these exercises was to simulate a context change with two exercises and observe what are the implications of that action on the software engineer work.

After completing the five proposed exercises, participants were asked to answer a twelve-item questionnaire with statements about their experience when executing the software process using MylynSDP and its DOI function. The explanation of the questionnaire is found on the next section.

## **5.6 Technology Acceptance Model**

Technology Acceptance Model (TAM) (DAVIS, 1986, DAVIS, 1989) is a model used to study to what extent a person seems to accept or reject a given technology. The model is an adaptation of the Theory of Reasoned Action (TRA) (AJZEN & FISHBEIN, 1980), which is a widely studied model from social psychology that investigates people behavior. According to TRA, a person's intention is determined by the person's attitude and subjective norm concerning the behavior in question. TRA is very general and it was designed to explain virtually any human behavior. TAM, on the other hand, is less general than TRA because it was designed to be applied only to computer usage behavior. TAM specifies two variables that may influence a system usage: perceived usefulness and perceived ease of use.

Perceived usefulness, as the word useful implies, is defined as the degree to which a person believes that using a particular system will help them perform their job better. Perceived ease of use refers to the degree to which a person believes that using a particular system would be free of effort. One person may find a system useful for his daily life work, but he also may find it hard to use, which is evidenced by perceived ease of use. In contrast, a person may consider a system easy to use, but useless for his duty.

Technology Acceptance Model has been used in several papers such as (LAURIDSEN, 2011, SANTO, 2012, VAZ *et al.*, 2012. An extensive research about TAM is found on (BAGOZZI, 2007, DAVIS, 1993, DAVIS *et al.*, 1989). TAM takes form in a questionnaire with twelve statements and seven possible answers. Statements

were divided in two groups. The first group of six statements relates to perceived usefulness and the second groups of the remaining six statements relates to perceived ease of use. Each of TAM's statements is aimed at measuring a specific metric on participant's answers.

Table 5.2 shows each statement used on this Master's Degree's validation study and the metric that was being observed. The set of seven possible answers contains the 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". Once the questionnaire was completed, participants were thanked and the validation study was over.

Table 5.2 - Statements of the validation study alongside the metrics observed.

Number	Statement	Metric
S1	Using DOI function on my job allows me to perform tasks quickly	Work more quickly
S2	Using DOI function improves my job performance	Job performance
S3	Using DOI function on my job increases my productivity	Increase productivity
S4	Using DOI function enhances my effectiveness on the job	Effectiveness
S5	Using DOI function makes it easier to do my job	Make job easier
S6	I consider DOI function useful in my job	Useful
S7	Learn to use DOI function was easy for me.	Easy to learn
S8	I consider it easy to get DOI function to do what I want it to do	Controllable
S9	My interaction with DOI function was clear and understandable	Clear and understandable
S10	I consider DOI function flexible to interact with	Flexible
S11	I consider it easy to become skillful at using DOI function	Skillful
S12	I consider DOI function easy to use	Easy to use

## 5.7 Analysis

Answers to TAM questionnaire and exercises' execution times were then saved and processed in order to be later analyzed, which helps to draw conclusions. Researcher's comments were also saved because they may help to explain a particular low or high result. The validation study was not a formal experiment study mostly due to time problems. However, it followed a certain degree of formalism in order to be better carried on. As previously mentioned, statements contained on TAM questionnaire were divided into two groups: those related to the perceived usefulness and those related to the perceived ease of use. Thus, the validation study has two well-defined objectives, which are described in Table 5.3 and Table 5.4

Table 5.3 - First goal of validation study.

<b>Analyze</b>	DOI function
<b>With the purpose of</b>	Characterize
<b>With respect to</b>	Usefulness
<b>In the context of</b>	Academic management system development
<b>Under the perspective of</b>	Software engineers executing a software development process

Table 5.4 - Second goal of validation study.

<b>Analyze</b>	DOI function
<b>With the purpose of</b>	Characterize
<b>With respect to</b>	Ease of use
<b>In the context of</b>	Academic management system development
<b>Under the perspective of</b>	Software engineers executing a software development process

Each of the twelve statements had its seven possible answers calculated in percentage. The percentage of each answer was then transported to a spreadsheet and a graphic was generated. This was done to better visualize which features were considered more important and which were not by participants. Figure 5.4 shows the graph and the percentage of each answer given in the questionnaire. It can be noted that in all statements “I completely agree” answer has been given in more than 50% of the times, except on statement #5. This will be commented in the following paragraph. The good overall result about MylynSDP and its DOI function usefulness and ease of use shows that they are likely to be accepted by software engineers during the execution of a software process. However, some points worth mentioning.

Statement #5 deals with the ability of the technology to make participants’ job easier. This was the only statement that was not rated more than 50% on “I completely agree” answer. Nevertheless, the other answers were divided into “I slightly agree” and “I partially agree”, which are positive answers and do not negatively affects the final general opinion about MylynSDP’s DOI function capacity of making jobs easier.

In statement #6, which deals with usefulness of the technology being observed, one of the participants did not considered DOI function to be useful in his job. The same participant wrote a comment at the end of the questionnaire in which he explains that he had been working with software development and he did not see DOI function useful in this field. As explained, MylynSDP’s DOI function was designed to be useful in every phase of the software process, since its conception to its delivery and maintenance, including its development, phase in which the system is really implemented. As MylynSDP’s DOI function was based on another DOI function specifically aimed at the implementation phase, a generalization had to be done. The

generalization of some concepts naturally makes them less suitable for a particular field in favor of dealing with more cases. Perhaps, this is why this participant understood Mylyn's DOI function as not so useful on software development field.

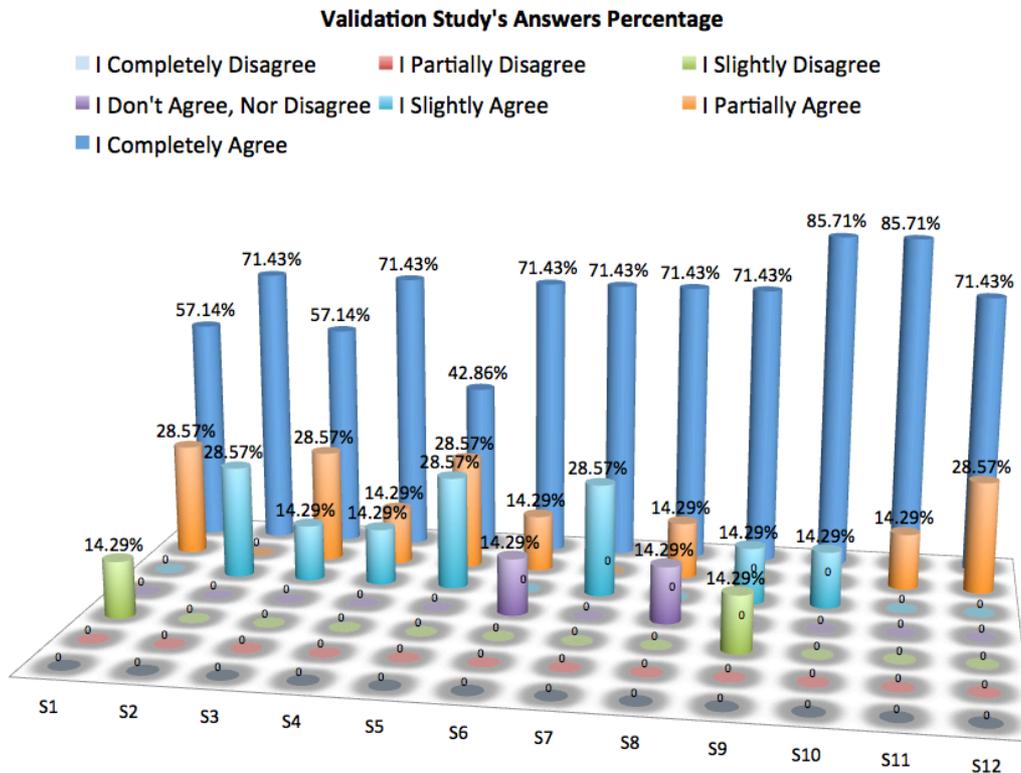


Figure 5.1 - Percentage of answers given by participants on the questionnaire.

Another important comment can be made on the answers given to statement #8. Answers given to this statement measure the opinion of participants about whether it is easy to get DOI function to do what is intended or not, which is called controllability. One of the participants did not consider controllability as one feature included in DOI function. It should be noted that the participant who evaluated such score had some difficulties with Mac operating system's interface when dealing with scrolling, minimizing and closing documents. Although these difficulties may have slightly affected that participant's opinion, the overall score for Mylyn's DOI function's controllability was good enough.

Statements #1 and #9 had both the "I slightly do not agree" answer, which is considered a low score and indicates that they must be investigated. Statement #1 deals with how quick a job would be performed when working with the technology being observed, which was the DOI function. One participant of the validation study thus did not consider the use of DOI function in software engineering field as a

mechanism that would help software engineers to work more quickly. Unfortunately, the same participant did not make a comment about it either written or verbally. For that reason, no explanation can be given to that low score. However, the scores given by other participants shows that the majority of answers considers DOI function a mechanism that helped them work quicker. Thus, the low score received by statement #1 can be considered an outlier.

Similar to statement #1, statement #9 also received a low score. One of the participants of the validation study slightly disagreed that DOI function is clear and understandable by software engineers when working with them. Although there were no explanations at the end of the questionnaire, that participant left a comment. The participant suggests that DOI function's filtering should be improved by the use of keywords. The participant explains that keywords can be used either to look for particular words in the name of the artifacts and their contents or to mark artifacts, similarly to the use of tags, according to whatever software engineers want. However, the use of keywords may not be the most suitable approach due to the fact that most used keywords may not belong to the underlying software process. Perhaps, using words from the already created Glossary could achieve better results.

Statement #3 deals with productivity, which is, in general terms, the amount of work produced in a period of time. Answers given to statement #3 help to picture how accurate MylynSDP's DOI function and concepts involved in its working aid software engineers to get more productive. It can be seen that more than half of participants perceived that DOI function may help them produce more when executing software processes. Although this is a good final score for statement #3, it was expected higher score on this metric as MylynSDP is mostly aimed at increasing productivity. Participants did not left any comment about it.

Finally, one last point should be mentioned and it is related to statement #2. This statement measures participant's opinion about the improvement on job performance that the use of DOI function may cause. Two participants answered that they slightly agree to that statement and others completely agreed. The overall score for that statement is good and the use of DOI function in all phases of the software process execution tends to result on a performance enhancement. Although a good final score was observed on statement #2, no comments were left to explain the reasons for not a higher score.

The time each participant took to solve each one of the five exercises was measured and is displayed on Table 5.5. Times are in mm:ss.cc format, where mm stands for two-digit minutes, ss stands for two-digit seconds and cc stands for centiseconds, which is the hundredth of a second. Note that the fourth column of times

shows the execution time for exercises #4 and #5 together because they simulated a context change and then exercise #5 happened during the execution of exercise #4. Two points can be highlighted concerning the times.

Table 5.5 - Execution times for each of the participants on each of the exercises.

<b>Total Time</b>	<b>E1</b>	<b>E2</b>	<b>E3</b>	<b>E4 &amp; E5</b>
<b>P1</b>	19:13.21	13:09.88	06:42.01	17:19.32
<b>P2</b>	12:48.14	06:57.90	04:09.82	11:20.61
<b>P3</b>	06:40.14	05:11.61	05:03.62	10:13.14
<b>P4</b>	12:06.63	06:56.34	05:36.09	18:26.97
<b>P5</b>	11:15.77	08:28.03	04:22.90	21:00.98
<b>P6</b>	09:41.56	04:28.41	03:04.93	05:45.61
<b>P7</b>	14:20.39	12:49.01	07:40.77	14:45.78

The first point relates to the comparison between exercises #1 and #2. Exercise #1 asks participants to write a short description for a use case document. To do that, participants should access a business rule, included on a business rule document, and a requirement, included on a requirements document. Exercise #2, although different, has same similarities. Participants had to edit another use case description and for that, they should check a business rule, on a suitable business rule document and the glossary document. The main difference between both exercises that that exercise #1 have not received any interaction other than the initial interaction events performed by DOI function when initiating the activity. For that reason, several artifacts were still visible by the time the participants started the exercised. Exercise #2 simulates a case in which some interaction events were performed on some artifacts. In other words, it acts as if the user is not totally new to the exercise. As a consequence, it can be noted that all measured times were reduced, especially for participants that did not executed the first exercise quickly. Skilled participants, the ones that solved the first exercise in less than ten minutes, also reduced their times, but it was not observed a significant reduction in one participant's execution time of exercise #2.

The second point to be commented compares the times of the exercises #1 and #4 & #5. Participants P1, P2, P6 and P7 were able to reduce their time or perform at a similar time from exercise #1 to the last one. This worth highlighting because its a comparison between one exercise with low filtering, which is exercise #1, and two exercises with good filtering, which are exercises #4 & #5. No comments made by the author of this Dissertation explain the reasons why other participants had their time increased when they executed the two final exercises. Thus, it can be concluded that the amount of work was the main cause as expected.

## 5.8 Threats to Validity

Although there are not formal ways to conduct a validation study, some experimental guidelines were followed in order to offer a baseline to the execution of this Master's Degree's validation study. There are some threats that may influence positively or negatively the outcomes and observations made on this validation study. It is believed that none of these threats had a significant impact on the results described earlier though. Threats to validity are listed in the following paragraphs.

The validation study was carried on with the presence of seven participants. Although all participants had a high expertise in the software engineering field as well as software process modeling and execution, it is known that seven is a low number of participants to draw meaningful conclusions about the pros and counters of the use of DOI function on a software process execution. It would be necessary more participants and opinions in order to better find out trends in the execution that either facilitates or make software engineer's activities harder. However, the results of this validation study executed with seven participants can be used as a starting point for more investigation.

During the conception of the validation study, it was decided to use a real software process, from a real software project, in order to better observe how DOI function would be affected by particularities featured on real software processes. Participants were first introduced to that software process during the training made right before the execution of the validation study. Therefore, none of the participants knew the software process before and as a consequence any of them were experienced with that particular software process. The fact that it was the first time that participants interacted with the software process used in the validation study has its consequences. Participants took some time to understand what software process activity was about to be executed. Thus, results of the validation study may be slightly affected because each participant was totally new to the project.

Every software project has its own particularities. One of the particularities of the software project used in the validation study was the naming convention used to name artifacts. Most participants complained about the name of the artifacts because they were long and confusing. Moreover, participants noted that few artifacts' name did not follow the naming convention. Although this was noted before the execution of the validation study, nothing could be done to mitigate this threat. The reason is that artifacts used in the validation study execution were the same artifacts used in the software process real execution, the execution performed by the owner of the software

process. Thus, it was a particularity that came along with the software process used in the validation.

The validation study was executed using an iMac. Six of the seven participants were not familiar with Mac operating system and they had minor problems when performing some tasks during the execution of the validation study such as scrolling with the mouse and minimizing a window with the contents of an artifact. One participant had problems with writing on the keyboard because it was set to international English rather than Brazilian Portuguese, which has some implications on the position of keys such as the tilde (~). Participants were free to ask any questions they wanted during the execution of the validation study, such as “how do I scroll?”, and thus these problems were solved with quick instructions. It is believed that these difficulties have not affected the overall performance of participants in the validation study.

## **5.9 Conclusion**

A validation study has been conducted in order to assess concepts that were discussed in this Dissertation. Therefore, participants were invited to take part in the validation study, which consists of executing some software process activities and answering a questionnaire about their experience.

As it can be drawn from the results, participants were positive about MylynSDP's importance for software engineering field as well as its new DOI function. Most of them well understood the concepts that underlie this research project. Moreover, when observing execution times for each of exercises on the validation study, it can be noted that as much filtered a context is, as much higher are the benefits of DOI function to the software engineer.

This chapter is concerned with explanations to the validation study since its conception to its execution and analysis of results. Every aspect of the validation study, such as its participants, exercises, phases and results, are detailed with enough information to its comprehension.

## 6 Conclusion

*On this chapter, conclusions are drawn. In addition, this chapter explains some of the limitations MylynSDP's DOI function has. Finally, some room for future work is presented and the concepts involved in their development are discussed.*

### 6.1 Conclusions

Software processes are used when developing computer software in order to guide the work of software engineers towards the production of a system with quality. During the execution of software processes, activities are performed. By performing activities, artifacts are either consumed or produced. Depending on the size and complexity of the software being developed, its underlying software process may also be big and complex. Moreover, the number of artifacts used in the execution of the software process can be excessive high. To perform an activity under a condition such as the one described, software engineers may search for suitable artifacts that will be used during the activity's execution. The search for the set of suitable artifacts that relates to that activity execution is performed among several other artifacts that are not interesting at the moment. For that reason, that search can be tiring, confusing, error-prone and time-consuming.

In addition to this, at any given moment, an activity execution may be interrupted either by the presence of another activity with higher priority or by parallel execution of activities. As a consequence, the current activity context, which comprises the artifacts that are being used during the execution of that activity, must be either closed or left aside to give room for the new activity context. A new search for context is then made. When the interrupting task is finished, the software engineer then may resume the interrupted activity execution. If necessary, a search for its context is then performed one more time. This is known as context change problem and it negatively affects the software engineer's performance on the task as well as its productivity once he must spend additional time and effort on the search for suitable artifacts rather than on the work of the activity.

This Master's Degree work proposed a new way of executing software processes with the use of a Degree of Interest (DOI) function in order to solve the artifact search and context change problems. A DOI function scores element according

to predefined algorithms and was used to infer artifact's interest value in relation to the software process activity that was being performed. Mylyn, an Eclipse plugin, helps programmers when executing coding tasks, using an implementation of a DOI function. Mylyn's DOI function scores Java classes based on the frequency of use and filters out unused classes. However, Mylyn's DOI function is aimed only at the implementation phase of computer software. Moreover, tasks contexts are created manually.

For that reason, Mylyn's DOI function was extended in order to help software engineers during every phase of the execution of a software process, since conception to delivery, maintenance and retirement. Additionally, the new DOI function is able to consider the existence of an underlying software process that guides the development of the software and, as a consequence, activity contexts can be initially created automatically. This is feasible because the relation between activities and artifacts are already defined on the software process specification. The final implementation of DOI function also works as an Eclipse plugin and as named MylynSDP.

As a way to summarize and compare MylynSDP's features with each one of the related work researched on this Master's Degree, Table 6.1 condenses each related work project's drawback and compares it with MylynSDP.

Table 6.1 - MylynSDP's features and related work's drawbacks put in a nutshell for comparison purposes.

	<b>Related Work drawback</b>	<b>MylynSDP</b>
<b>Presto and Placeless Projects</b>	Manual classification of documents in order to put them in a context; Update of the entire set of documents that belongs to a new value of property.	Artifacts are automatically added to a task context based on their relation with software process specification artifacts. Artifacts that belong to a new task context are automatically added to it.
<b>TaskTracer</b>	Limited to Microsoft Office package and Internet Explorer browser.	Eclipse's file editors belong to the reality of the work of a software engineer. Plus, virtually any file type can be manipulated by referring it as an external file.
<b>UMEA</b>	Lack of integration with new technologies.	MylynSDP was created on top of Eclipse IDE, which is extremely extensible and may have new

		plugins to implement new functionalities.
<b>WebAPSEE</b>	Manual allocation of artifacts to activities.	Artifacts are automatically associated with tasks based on the software process specification.
<b>TABA Station</b>	Software development environment needs to be recreated to reflect modifications on the software process specification.	Modifications to the software process specification require software engineers to reupload it to MylynSDP. This is not a longstanding job.
<b>Software Traceability</b>	Considerable amount of time due to lack of automatization; Minimal computer aid; Considerable processing capacity; Traceability matrix needs to be reprocessed to reflect new artifacts or activities	Contexts are created in real time; Computer-aided execution; Relatively low processing capacity; No additional processing required when adding new artifacts or tasks.
<b>Mylyn</b>	Aimed only at implementation phase of software process; Aimed only at code documents only; Aimed only at coding tasks only; Do not consider the existence of a software process; Do not base its working in a software process.	MylynSDP is aimed at all phases of a software development process; MylynSDP is aimed at virtually all documents types, including diagrams, images, text documents and spreadsheets; MylynSDP is aimed to support any task type; MylynSDP takes into consideration the existence of a software process specification to its DOI calculations; MylynSDP base its workings on the underlying software process specification when, for example, associates artifacts and task contexts.

In order to observe the implications of the concepts proposed in this Master's Degree Dissertation, a validation study has been conducted. Seven participants interacted with MylynSDP's DOI function using a real software process and real artifacts. Participants were Master's Degree and Doctoral Degree students and they have high level of expertise and experience with software process execution. After solving some guided exercises, participants were asked to answer a twelve-statement questionnaire. The questionnaire is used to measure how useful a technology is and how easy it is to deal with it. All but one of the statements had more than 50% of high evaluation, which means that MylynSDP, its DOI function and the surrounding concepts received good results and good acceptance.

Furthermore, some point should be noted. The first point is related to the frequency artifacts are interacted with in the new reality. It was noted that some artifacts are less interacted with during other phases of the software process execution. For instance, when writing a use case document on a given day, software engineers will not open and close the same document as much times as it would be done with Java classes. For that reason, DOI function was adjusted to better cover both cases, the ones in which artifacts are greatly interacted and the ones in which artifacts are not so interacted.

Moreover, when the activity being executed do not require a lot of interactions to be performed on its artifacts, not interesting artifacts seemed to take longer to be omitted. Therefore adjustments made on DOI function changed the decay value in order to let it omit some not interesting artifacts quicker.

Although software processes represent a guide to the order activities should be performed, it was noted that it was not relevant to the aid DOI function was designed to provide. In other words, the order of the software process activities were not important when executing the software process with the aid of MylynSDP's DOI function. Thus, there is not any mechanism that prevents software engineers to execute one activity before another one.

As this DOI function is aimed at helping one software engineer at a time, the role element of some software processes notations, such as BPMN, was not featured in MylynSDP. It could be particularly useful in order to filter activities and tasks for software engineer that uses MylynSDP. However, roles were not necessary to the first version of DOI function for software engineering field. An implementation of roles for MylynSDP is considered for the future though.

## 6.2 Limitations

As previously explained, MylynSDP's DOI function was based on Mylyn's DOI function. For that reason, Mylyn plugin's code was downloaded and accessed in order to understand how Mylyn's DOI function works and filters classes. This was an awful task because there were few documentation files available that explain how Mylyn and its DOI function works. Neither on Mylyn's official website there were useful documents that explains in detail how Mylyn components works or relates.

Furthermore, Mylyn's code is divided into more than 200 Java projects and contains far more Java packages. Several of the hundreds Java classes were accessed in order to understand how Mylyn works, and how its DOI function detects an user interaction, calculates artifacts' interest value and manages contexts saving. The sheer number of Java classes was a problem that Mylyn itself was not helpful at solving because, as it was a discovery task, new classes were accessed all the time. That is different of when a programmer performs a coding task and looks for some related classes. Several questions about the code were raised and the most pertinent ones were submitted to Mylyn's developer e-mail list in which creators and other Mylyn developers help was really appreciated.

MylynSDP works with artifacts that can be opened using Eclipse's default file editor. However, this limits the set of type of files that can be used by software engineers. For that reason, external artifacts may be imported during artifact creation on the suitable wizards designed for that purpose. Thus, Microsoft Office Word and Excel documents as well as databases schemas, class diagrams and interface sketches can be externally opened when an artifact icon is double clicked on the artifact view. However, when this is the way artifacts are opened, MylynSDP is not able to process editing and command interaction events because the operating system is the one charged of the management of the artifact.

Lastly, when importing a software process specification to MylynSDP, a suitable mechanism checks what are the artifact and activity types and saves it in a document so DOI function can understand what activities and artifacts may be created. However, it is extremely important that names of activities and artifacts in the software process specification should be properly written. For example, if an activity uses an artifact named "Class Diagram" and another activity uses the same artifact, but its name was misspelled as "Classes Diagram", MylynSDP will treat them as two different types of artifact. Thus, names of activities and artifacts should be properly written and follow a convention.

### 6.3 Future Work

The concepts that surround MylynSDP and its DOI function are expandable and they can be used in several ways to help software engineers. Numerous ideas were raised but due to time constraints they could not be implemented and tested. Some other ideas were created as improvements of the current concepts already in use by MylynSDP. All future work will be mentioned in this section.

MylynSDP maintains a log file with interactions that were performed by software engineers on artifacts. This log is extremely important for future analysis but it is not easy to understand. A better human readable log file is handy to check, for example, is the execution of the software process followed as expected or if there were major issues that should be fixed. That log file is able to tell if some activities had more artifacts used than the specified in the software process or if some activities used less artifacts than it needed. A future work can also be a mechanism that suggests modification to the software process specification based on the way it was executed.

During the validation study, it was observed that some activities throughout the software development do not require intense interaction events. For example, a software engineer who is reviewing a use case document of the specification of computer software may not switch documents often. For that reason, the time an artifact is being used should also be taken into consideration by DOI function in the calculation of the interest value of an artifact. Currently, just clicks and keystrokes are being used as interaction events. Thus, artifacts opened for a longer time would be considered more important than artifacts that remain closed for most of the time.

DOI function's scoring algorithm, as mentioned, process interaction events with documents such as selections and edits. However, other input methods should also be considered like image editions. Software engineers often deal with diagram creation or any other image manipulation. Image documents, and other similar types of artifacts, are not fully supported by MylynSDP.

Software engineers deals with several types of artifacts that range from documents, to tables, code and diagrams. In order to make DOI function able to deal with all types of artifacts, their contents were not taken into consideration. A useful approach considers sections of written documents as different parts of the calculation of interest value. As a consequence, DOI would be able to calculate which section is interesting for that activity execution, rather than the entire document. That means that the granularity of filtering should be enhanced.

The final suggestion of work that can be done in the future deals with groups of software engineers. Some studies research collaboration on the execution of software

process (GRAMBOW *et al.*, 2011, GRAMBOW *et al.*, 2013). DOI function is a great feature to help group of software engineers better execute their activities. Thus, a useful concept is that DOI functions communicate with each other, maybe with a shared repository of data, and change information to better calculate artifact's interest value and create contexts more effectively. Software engineers that are new to the project would benefit the most from this feature because they would not have to browse over low filtered artifacts as contexts from other software engineers would help his DOI function to better filter the context of his current activity.

# BIBLIOGRAPHIC REFERENCES

- AALST, W. M. P. van der, HOFSTEDE, A. H. M. ter, WESKE, M., 2003, "Business Process Management: A Survey". In: **Proceedings of the 2003 International Conference on Business Process Management**, pp. 1-12, Eindhoven, The Netherlands, June 2003.
- AALST, W. M. P. van der, 2007, "Trends in Business Process Analysis: From Verification to Process Mining". In: **Proceedings of the 9<sup>th</sup> International Conference on Enterprise Information Systems (ICEIS 9)**, pp. 12-16, Funchal, Madeira, Portugal, June 2007.
- AALST, W. M. P. van der, 1999, "Formalization and Verification of Event-driven Process Chains". **Information and Software Technology**, vol. 41, no. 10, pp. 639-650, July 1999.
- AJZEN, I., FISHBEIN, M., 1980, **Understanding Attitudes and Predicting Social Behavior**. Englewood Cliffs, NJ. Prentice-Hall.
- AMBRIOLA, V., CONRADI, R., FUGGETTA, A., 1997, "Assessing Process-Centered Software Engineering Environments". **ACM Transactions on Software Engineering and Methodology**, vol. 6, no. 3, pp. 283-328, July 1997.
- ANDERSON, K., M., SHERBA, S. A., LEPTHIEN, W. V., 2002, "Towards Large-Scale Information Integration". In: **Proceedings of the 24<sup>th</sup> International Conference on Software Engineering**, pp. 524-534, Orlando, FL, USA, May 2002.
- ANNOSI, M. C., PASCALE, A., GROSS, D. *et al.*, 2008, "Analyzing Software Process Alignment with Organizational Business Strategies using an Agent- and Goal-oriented Analysis Technique – an Experience Report". In: **Proceedings of the 3<sup>rd</sup> i\* International Workshop**, pp. 9-12, Recife, Brazil, February 2008.
- ANTONIOL, G., CANFORA, G., CASAZZA, G. *et al.*, 2002, "Recovering Traceability Links between Code and Documentation". **IEEE Transactions on Software Engineering**, vol. 28, no. 10, pp. 970-983, October 2002.
- ARBAOUI, S., DERNIAME, J. C., OQUENDO, F. *et al.*, 2002, "A Comparative Review of Process-Centered Software Engineering Environments". **Annals of Software Engineering**, vol. 14, no. 1-4, pp. 311-340, December 2002.

- ASHRAFI, N., 2003, "The impact of software process improvement on quality: In theory and practice". **Information and Management**, vol. 40, no. 7, pp. 677-690, August 2003.
- ASUNCION, H. U., ASUNCION, A. U., TAYLOR, R. N., 2010, "Software Traceability with Topic Modeling". In: **Proceedings of the 32<sup>nd</sup> ACM/IEEE International Conference on Software Engineering**, pp. 95-104, Cape Town, South Africa, May 2010.
- BAGOZZI, R. P., 2007, "The Legacy of the Technology Acceptance Model and a Proposal for a Paradigm Shift". **Journal of the Association for Information Systems**, vol. 8, no. 4, pp. 244-254, April 2007.
- BARJIS, J., 2008, "The importance of business process modeling in software systems design". **Science of Computer Programming**, vol. 71, no. 1, pp. 73-87, March 2008.
- BERGER, P. M., 2003, **Instanciação de Processos de Software em Ambientes Configurados na Estação TABA**. M.Sc. dissertation, COPPE/UFRJ, Rio de Janeiro, RJ, Brazil.
- CAMPOS, A., OLIVEIRA, T., 2012. "Modeling Work Processes and Software Development: Notation and Tool". In: **Proceedings of the 13<sup>th</sup> International Conference on Enterprise Information Systems**, pp. 337-343, Beijing, China, June 2011.
- COSTA, A. SALES, E., REIS, C. A. L. *et al.*, 2007, "Apoio a Reutilização de Processos de Software através de Templates e Versões". In: **VI Simpósio Brasileiro de Qualidade de Software**, pp. 47-61, Porto de Galinhas, Pernambuco, Brazil, June 2007.
- DAVIS, F. D., 1993, "User acceptance of information technology: system characteristics, user perceptions and behavioral impacts". **International Journal of Man-Machine Studies**, vol. 38, no. 3, pp. 475-487, March 1993.
- DAVIS, F. D., BAGOZZI, R. P., WARSHAW, P. R., 1989, "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models". **Management Science**, vol. 35, no. 8, pp. 982-1003, August 1989.
- DAVIS, F. D., 1989, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology". **MIS Quarterly: Management Information Systems**, vol. 13, no. 3, pp. 319-340, September 1989.

- DAVIS, F. D., 1986, **A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results**. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.
- DOURISH, P., EDWARDS, W. K., LAMARCA, A. *et al.*, 1999, "Using Properties for Uniform Interaction in the Presto Document System". In: **Proceedings of the 12<sup>th</sup> annual ACM Symposium on User Interface Software and Technology**, pp. 55-64, Ashville, NC, USA, November 1999.
- DRAGUNOV, A. N., DIETTERICH, T. G., JOHNSRUDE, K. *et al.*, 2005, "TaskTracer: A Desktop Environment to Support Multi-tasking Knowledge Workers". In: **Proceedings of the 10<sup>th</sup> International Conference on Intelligent User Interfaces**, pp. 75-82, Sand Diego, CA, USA, January 2005.
- FRANÇA, B. B. N., SALES, E. O., REIS, C. A. L. *et al.*, 2009, "Utilização do Ambiente WebAPSEE na implantação do nível G do MPS.BR no CTIC-UFPA". In: **VIII Simpósio Brasileiro de Qualidade de Software**, pp. 310-317, Ouro Preto, MG, Brazil, June 2009.
- FUGGETTA, A., 2000, "Software Process: A Roadmap". In: **Proceedings of the Conference on The Future of Software Engineering**, pp. 25-34, Limerick, Ireland, June 2000.
- FUGGETTA, A., 1996, "Functionality and architecture of PSEEs". **Information and Software Technology**, vol. 38, no. 4, pp. 289-293, April 1996.
- FUGGETTA, A., GHEZZI, C., 1994, "State of the Art and Open Issues in Process-Centered Software Engineering Environments". **Journal of Systems Software**, vol. 26, no. 1, pp. 53-60, July 1994.
- GOMES, A., MAFRA, S., OLIVEIRA, K. *et al.*, 2001, "Avaliação de Processo de Software na Estação Taba". In: **XV Simpósio Brasileiro de Engenharia de Software**, pp. 344-349, Rio de Janeiro, RJ, Brazil, October 2001.
- GOTH, G., 2009, "The Task-Based Interface: Not Your Father's Desktop". **IEEE Software**, vol. 26, no. 6, pp. 88-91, November 2009.
- GRAMBOW, G., OBERHAUSER, R., REICHERT, M., 2011, "Towards Automatic Process-Aware Coordination in Collaborative Software Engineering". In: **Proceedings of the 6<sup>th</sup> International Conference on Software and Database Technologies**, pp. 5-14, Seville, Spain, July 2011.
- GRAMBOW, G., OBERHAUSER, R., REICHERT, M., 2013, "Enabling Automatic Process-aware Collaboration Support in Software Engineering Projects". In:

**Software and Data Technologies**, vol. 303, **Communications in Computer and Information Science**, Springer Berlin Heidelberg, pp. 73-88, 2013.

GREEN, G. C., Hevner, A. R., COLLINS R. W., 2005, "The impacts of quality and productivity perceptions on the use of software process innovations". **Information and Software Technology**, vol. 47, no. 8, pp. 543-553, June 2005.

GRUHN, V., 2002, "Process-Centered Software Engineering Environments: A Brief History and Future Challenges". **Annals of Software Engineering**, vol. 14, no. 1-4, pp. 363-382, December 2002.

IBM, 2012, Rational Unified Process (RUP). Available in <<http://www-01.ibm.com/software/awdtools/rup/>>. Accessed on June 9, 2014.

KAPTELININ, V., 2003, "UMEA: Translating Interaction Histories into Project Contexts". In: **Proceedings of the Conference on Human Factors in Computing Systems**, pp. 353-360, Ft. Lauderdale, FL, USA, April 2003.

KERSTEN, M., MURPHY, G. C., 2005, "Mylar: a degree-of-interest model for IDEs". In: **Proceedings of the 4<sup>th</sup> International Conference on Aspect-oriented Software Engineering**, pp. 159-168, Chicago, IL, USA, March 2005.

KERSTEN, M., MURPHY, G. C., 2006. "Using Task Context to Improve Programmer Productivity". In: **Proceedings of the 14<sup>th</sup> ACM SIGSOFT International Symposium on Foundations of Software Engineering**, pp. 1-11, Portland, OR, USA, November 2006.

KERSTEN, M., 1999, **Focusing knowledge work with task context**. Ph.D thesis, University of British Columbia, Vancouver, B. C., Canada.

LAURIDSEN, B., 2011, "Understanding the Influence of the Technology Acceptance Model for Online Adult Education". In: **Proceedings of the 16<sup>th</sup> TCC Worldwide Online Conference**, pp. 1-16, Honolulu, HA, USA, April, 2011.

MARTIN, J., 1991, **Rapid Application Development**. Indianapolis, IN, USA, Macmillan Publishing Co., Inc., 1991.

MARCINIAK, J. J., 2002, **Encyclopedia of Software Engineering**. 2<sup>nd</sup> ed. John Wiley and Sons, New York, January 2002.

MATINNEJAD, R., RAMSIN, R., 2012, "An Analytical Review of Process-Centered Software Engineering Environments". In: **IEEE 19<sup>th</sup> International Conference and Workshops on Engineering of Computer-Based Systems**, pp. 64-73, Novi Sad, Serbia, April 2012.

- MURPHY, G., 2009, "Attacking Information Overload in Software Development". In: **IEEE Symposium on Visual Languages and Human-Centric Computing**, pp. 4, Corvallis, OR, USA, September 2009.
- OMG, 2008, Software & System Process Engineering Metamodel (SPEM). Available in <http://www.omg.org/specs/SPEM/2.0/PDF>. Accessed on June 9, 2014.
- OMG, 2011, Business Process Model and Notation (BPMN). Available in <http://www.omg.org/specs/BPMN/2.0/PDF>. Accessed on June 9, 2014.
- OSTERWEIL, L. J., BROWN, J. R., STUCKI, L. G., 1978, "ASSET: A Lifecycle Verification and Visibility System".
- PARASURAMAN, R., RILEY, V., 1997, "Humans and automation: Use, misuse, disuse, abuse". **Human Factors**, vol. 39, no. 2, pp. 230-253, June 1997.
- PEFFERS, K., TUUNANEM, T., ROTHERNBERGER, M. A. *et al.*, 2007, "A design science research methodology for information systems research". **Journal of Management Information Systems**, vol. 24, no. 3, pp. 45-77, December 2007.
- PILLAT, R. M., OLIVEIRA, T. C., FONSECA, F. L., 2012, "Introducing Software Process tailoring to BPMN: BPMNt". In: **International Conference on Software and System Process**, pp. 58-62, Zurich, Switzerland.
- PORTUGAL, I. S., OLIVEIRA, T. C., 2013, "Introducing Software Process Specification to Task Context". In: **Proceedings of the 25<sup>th</sup> International Conference on Software Engineering and Knowledge Engineering**, pp. 22-25, Hyatt Harborside at Longan Int'l Airport, Boston, USA, June 2013.
- PORTUGAL, I. S., OLIVEIRA, T. C., 2014, "Using Task Contexts to Improve Software Process Execution". In: **Proceedings of the XVII Ibero-American Conference on Software Engineering**, pp. 109-122, Pucón, Chile, April 2014.
- PRESSMAN, R. S., 2010, **Software Engineering: A Practitioner's Approach**, 7<sup>th</sup> ed. New York, NY, USA, McGraw-Hill.
- REIS, C. A. L., REIS, R. Q., 2007, "Laboratório de Engenharia de Software e Inteligência Artificial: Construção do ambiente WebAPSEE". In: **ProQuality (UFLA)**, pp. 43-48, Lavras, MG, Brazil, January 2007.
- REIS, C. A. L., 2003, **Uma Abordagem Flexível para Execução de Processos de Software Evolutivos**. Ph.D. thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brazil.

- ROCHA, A. R., MONTONI, M., SANTOS, G. *et al.*, 2005, “Estação TABA: Uma Infra-estrutura para Implantação do Modelo de Referência para Melhoria de Processo de Software”. In: **IV Simpósio Brasileiro de Qualidade de Software**, pp. 49-60, Porto Alegre, RS, Brazil, June 2005.
- ROCHA, A. R. C., SOUZA, J. M., AGULAR, T. C., 1990, “TABA: A Heuristic Workstation for Software Development”. In: **Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering**, pp. 126-129, Tel-Aviv, Israel, May 1990.
- SALES, E., REIS, C. L., REIS, R. Q., 2008, “Apoio a Gerência de Configuração de Artefatos de Software integrado a Execução de Processos de Software”. In: **XXII Simpósio Brasileiro de Engenharia de Software**, pp. 156-171, Campinas, SP, Brazil, May 2008.
- SANTO, R. E., 2012, **Serviços de Apoio ao Planejamento, Execução e Empacotamento de Revisões Sistemáticas da Literatura**. M.Sc. dissertation, COPPE/UFRJ, Rio de Janeiro, RJ, Brazil.
- SCRUM.org, 2011, Scrum Guide. Available in <<http://www.scrum.org/scrum-guide>>. Accessed on June 9, 2014.
- SIGA EPCT, 2014, Sistema Integrado de Gestão Acadêmica da Educação Profissional e Tecnológica. Available in <<http://www.sigaepct.net>>. Accessed in June 9, 2014.
- SPANOUKAKIS, G., ZISMAN, A., 2005, “Software Traceability: A Roadmap”. In: **Handbook of Software Engineering and Knowledge Engineering**, vol. 3, World Scientific Pub. Co., pp. 395-428, 2005.
- SUNDARAM, S. K., HAYES, J. H., DEKHTYAR, A. *et al.*, 2010, “Assessing traceability of software engineering artifacts”. **Requirements Engineering**, vol. 15, no. 3, pp. 313-335, September 2010.
- TASKTOP, 2014, Tasktop. Available in <<http://tasktop.com>>. Accessed on June 9, 2014.
- VAZ, V. T., CONTE, T. U., TRAVASSOS, G. H., 2012, “Empirical Assessment of WDP Tool: A Tool to Support Web Usability Inspections”. In: **Proceedings of the 38th Latin America Conference on Informatics**, pp. 1-9, Medellin, Antioquia, Colombia, October 2012.
- TRAVASSOS, G. H., 1994, **O Modelo de Integração de Ferramentas da Estação Taba**, Ph.D. thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brazil.

VILLELA, K., SANTOS, G., GALLOTA, C. *et al.*, 2001, “Estendendo a Estação TABA para a criação de Ambientes de Desenvolvimento de Software Orientados a Organização”. In: **XV Simpósio Brasileiro de Engenharia de Software**, pp. 344-349, Rio de Janeiro, RJ, Brazil.

# APPENDIX A – SIGA EPCT SOFTWARE PROCESS AUTHORIZATION OF UTILIZATION AND SPECIFICATION

*This appendix shows SIGA EPCT software development process authorization of utilization and the specification used on the validation study of this Dissertation.*

## A.1. SIGA EPCT Software Process Authorization of Utilization

	<b>CONIF</b> CONSELHO NACIONAL DAS INSTITUIÇÕES DA REDE FEDERAL DE EDUCAÇÃO PROFISSIONAL, CIENTÍFICA E TECNOLÓGICA	
--	--	---

OFÍCIO Nº. 09/2013/SIGA-EPCT-FORTI

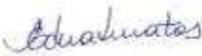
Salvador, 18 de outubro de 2013.

**DECLARAÇÃO**

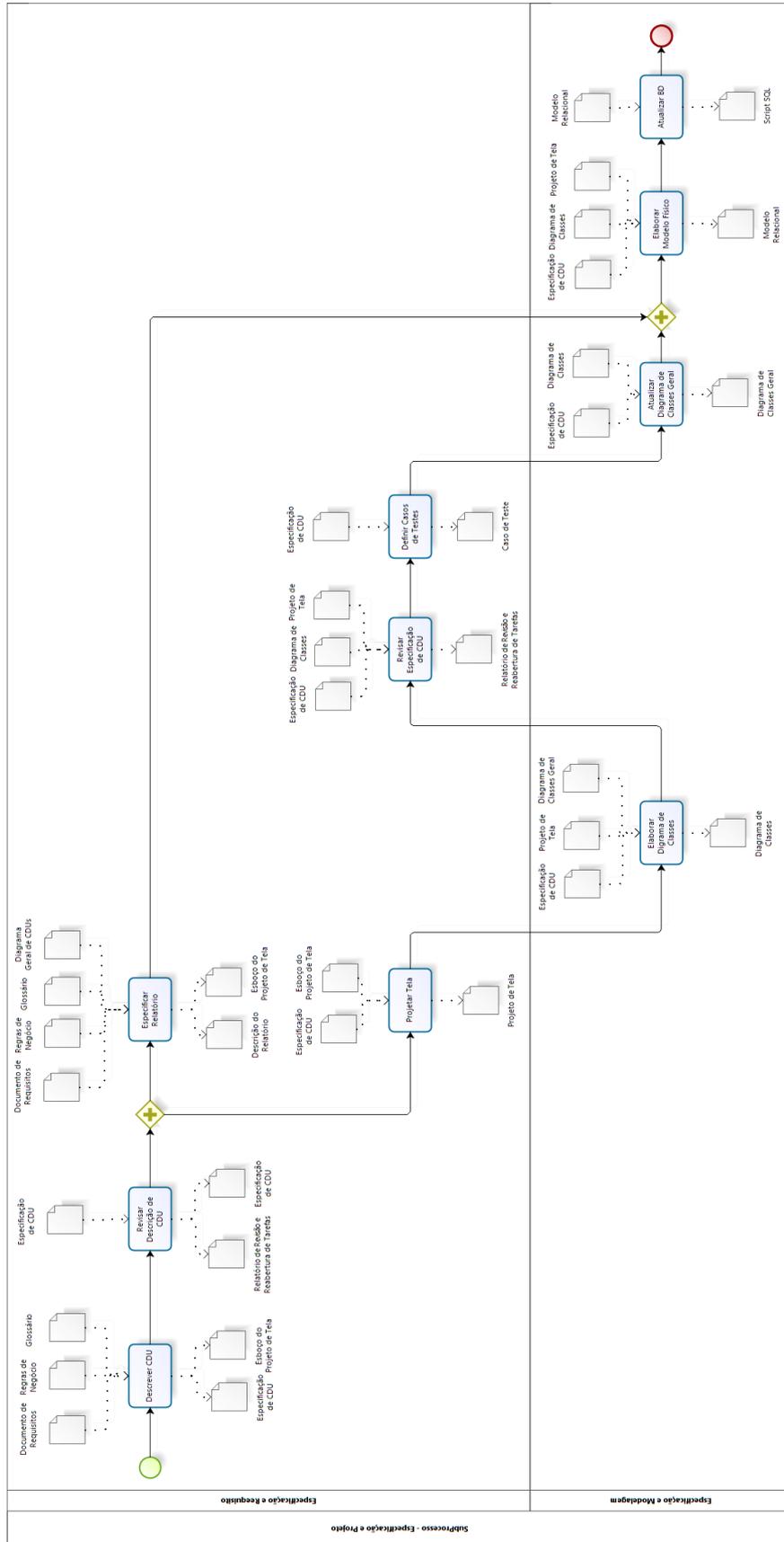
Declaramos para os devidos fins, que cederemos aos pesquisadores do grupo de pesquisa Prisma da COPPE/UFRJ, o acesso aos dados do projeto SIGA/EPCT, que compreende os dados do ambiente de gerenciamento de tarefas, REDMINE, e do sistema de controle de versão, SVN) para serem utilizados nas pesquisas conduzidas pelo grupo que está sob a orientação do Professor Doutor Toacy Cavalcante de Oliveira.

Esta autorização está condicionada ao cumprimento dos pesquisadores do grupo de pesquisa disponibilizar os seus resultados para o FORTI, incluindo o nome do projeto do SIGA-EPCT nas publicações científicas decorrentes dos projetos de pesquisa conduzidos.

Atenciosamente,

  
Edna da Silva Matos  
Coordenadora do GT-SIGA-FORTI

## A.2. SIGA EPCT Software Process Specification



# APPENDIX B – VALIDATION STUDY DOCUMENTS

*This appendix presents documents used on validation study. These documents were used to gather information about participants and their behavior during validation study and help draw conclusions. They were manipulated before, during or after the exercises of the validation study.*

## B.1 Consent Form

# Formulário de Consentimento

### Estudo

Este estudo visa caracterizar a utilização de uma função de grau de interesse (DOI) para a recuperação de artefatos relevantes a uma atividade de um processo de desenvolvimento de software.

### Idade

Eu declaro ter mais de 18 anos de idade e concordar em participar de um estudo conduzido por Ivens da Silva Portugal na Universidade Federal do Rio de Janeiro.

### Procedimento

Este estudo acontece em uma única sessão, dividido em duas etapas. A primeira etapa consiste em realizar quatro atividades propostas descritas em um processo de desenvolvimento de software. A segunda etapa consiste em responder um questionário estruturado de avaliação de tecnologia, composto por doze perguntas. Eu entendo que, uma vez que o estudo tenha terminado, os trabalhos que desenvolvi serão analisados visando entender a eficiência dos procedimentos e das técnicas propostas.

### Confidencialidade

Toda informação coletada neste estudo é confidencial e meu nome não será divulgado. Da mesma forma, me comprometo a não comunicar meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

## **Benefícios e Liberdade de Desistência**

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e apresentado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada à minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e ferramentas para a Engenharia de Software.

## **Pesquisador Responsável**

Ivens da Silva Portugal

Programa de Engenharia de Sistemas e Computação (PESC) – COPPE/UFRJ

## **Professor Responsável**

Prof. Toacy Cavalcante de Oliveira

Programa de Engenharia de Sistemas e Computação (PESC) – COPPE/UFRJ

Nome (em letra de forma): \_\_\_\_\_

Assinatura: \_\_\_\_\_ Data: \_\_\_\_\_

## **B.2 Characterization Questionnaire**

# **Questionário de Caracterização**

Código do Participante:

Data:

### **1) Formação Acadêmica**

( ) Doutorado

( ) Doutorando

( ) Mestrado

( ) Mestrando

( ) Graduação

( ) Graduando

Ano de Ingresso/período: \_\_\_\_\_/\_\_\_\_

Ano de Conclusão (ou previsão de conclusão)/período: \_\_\_\_\_/\_\_\_\_

## 2) Formação Geral

### 2.1) Qual é sua experiência com o ambiente de desenvolvimento Eclipse?

- Nenhuma.
- Já estudei em aula ou livro.
- Pratiquei em projetos em sala de aula.
- Usei em projetos pessoais.
- Usei em projetos na indústria.

### 2.2) Qual é a sua experiência com Processos de Desenvolvimento de Software?

- Não conheço Processo de Desenvolvimento de Software.
- Já li material sobre Processo de Desenvolvimento de Software.
- Já participei de um curso sobre Processo de Desenvolvimento de Software.
- Tenho lidado com Processo de Desenvolvimento de Software como parte de uma equipe, relacionada a um curso.
- Tenho lidado com Processo de Desenvolvimento de Software como parte de uma equipe, na indústria.

2.3) Por favor, explique sua resposta. Inclua o número de semestres ou número de anos de experiência relevante em que tem lidado com Processo de Desenvolvimento de Software. (Ex.: “Eu trabalhei por 2 anos como modelador de Processo de Desenvolvimento de Software na indústria”)

---

---

---

---

---

Obrigado pela sua colaboração!

## B.3 Time Record Form

# Formulário de Marcação de Tempo

Código do Participante:

## 1) Exercício 1

### 1.1) Momento de Acesso aos Artefatos

SIGA-EDU-CDU-MATRI-033 – **Vincular Aluno a Classe:**

- \_\_\_\_\_

SIGAEPCT – **REQ Matrícula** – Redmine dos Projetos do SIGA-EPCT:

- \_\_\_\_\_

SIGAEPCT – **RGN Matrícula** – Redmine dos Projetos do SIGA-EPCT:

- \_\_\_\_\_

### 1.2) Término de Exercício

- \_\_\_\_\_

### 1.3) Comentários:

---

---

---

---

## 2) Exercício 2

### 2.1) Momento de Acesso aos Artefatos

SIGA-EDU-CDU-INFRA-003 – **Manter Ambiente de Aprendizagem:**

- \_\_\_\_\_

SIGAEPCT - **Glossario SIGA** - Redmine dos Projetos do SIGA-EPCT:

- \_\_\_\_\_

SIGAEPCT – **RGN Infraestrutura** – Redmine dos Projetos do SIGA-EPCT:

- \_\_\_\_\_

## 2.2) Término do Exercício

- \_\_\_\_\_

## 2.3) Comentários:

---

---

---

---

## 3) Exercício 3

### 3.1) Momento de Acesso aos Artefatos

SIGA-EDU-**CDU-PELET**-061 – **Fechar Período Letivo:**

- \_\_\_\_\_

SIGAEPCT - **Glossário** SIGA - Redmine dos Projetos do SIGA-EPCT:

- \_\_\_\_\_

### 3.2) Término do Exercício

- \_\_\_\_\_

### 3.3) Comentários:

---

---

---

---

#### 4) Exercício 4

##### 4.1) Momento de Acesso aos Artefatos

**Plano\_Teste\_SIGA\_EDU\_CDU\_MATRI-033:**

- \_\_\_\_\_

**SIGA-EDU-CDU-MATRI-033 – Vincular Aluno a Classe:**

- \_\_\_\_\_

##### 4.2) Término do Exercício

- \_\_\_\_\_

##### 4.3) Comentários:

---

---

---

---

#### 5) Exercício 5

##### 5.1) Início do Exercício

- \_\_\_\_\_

##### 5.2) Momento de Acesso aos Artefatos

**SIGA-EDU-Tuoppi-82-patch038:**

- \_\_\_\_\_

**SIGA-EDU-DB-INFRA-003-ManterAmbienteDeAprendizagem:**

- \_\_\_\_\_

## 5.2) Término do Exercício

- \_\_\_\_\_

## 5.3) Comentários:

---

---

---

---

## B.4 Validation Study Exercises

# Exercícios do Estudo

## Exercício 1

A Organização onde você atua desenvolve Sistemas com alta qualidade em um curto prazo de tempo. Parte desse bom desempenho se deve ao fato de todas as atividades do desenvolvimento de um sistema estarem bem descritas, documentadas e representadas em um processo de desenvolvimento de software. Outras razões para tal bom desempenho são a qualificação dos colaboradores e a excelência das ferramentas utilizadas para a execução do processo.

Recentemente, a Organização foi contratada com o objetivo de realizar o desenvolvimento de um Sistema de Integrado de Gestão Acadêmica. A equipe de Engenheiros de Software da Organização onde você atua modelou um processo de desenvolvimento de software para auxiliar durante o desenvolvimento do Sistema. O documento está disponível para consulta.

Atualmente, este processo de software está sendo executado para que o projeto do Sistema de Gerência Acadêmica seja especificado. O processo é executado de forma que todas as atividades são realizadas

por Caso de Uso (CDU). Foram designados dois Casos de Uso para seu trabalho. São eles:

- “Vincular Aluno a Classe” (módulo Matrícula) e
- “Manter Ambiente de Aprendizagem” (módulo Infraestrutura).

No momento, a especificação que está sendo construída é a do Caso de Uso:

- “Vincular Aluno a Classe”.

Este Caso de Uso não possui uma descrição na sua especificação. Nesse sentido, este exercício pede que você escreva uma breve descrição do que se trata o Caso de Uso no espaço indicado no documento.

## Exercício 2

Parabéns!

A descrição do Caso de Uso “Vincular Aluno a Classe” está escrita e futuros membros da equipe poderão rapidamente entender do que se trata esse Caso de Uso. Agora é hora de escrever um resumo para o outro Caso de Uso que foi atribuído à você:

- “Manter Ambiente de Aprendizagem”.

Parte da descrição desse Caso de Uso, que pertence ao módulo Infraestrutura, já foi escrita por outro membro da equipe de Engenheiros de Software. Entretanto, a descrição contém várias siglas de termos utilizados corriqueiramente durante o projeto. Apesar das siglas estarem documentadas em um Glossário, sua utilização não é permitida, pois afeta negativamente o entendimento do Caso de Uso.

Além disso, foi identificado que, ao final da descrição desse Caso de Uso, você deve inserir uma nota explicativa relativa à regra de negócio 1.006 daquele módulo.

Diante do cenário apresentado, este exercício propõe que você corrija as siglas encontradas na descrição do Caso de Uso, substituindo-as pelos respectivos significados e, ainda, adicione uma frase explicativa referente à regra de negócio mencionada.

## Exercício 3

Muito bem.

Mais uma vez, sua contribuição ajudou que os outros membros da equipe de Engenheiros de Software da Organização entendam rapidamente o objetivo de determinado Caso de Uso.

Antes de iniciar a próxima atividade, a saber “Revisar Descrição de CDU”, para os casos de uso que lhe foram atribuídos, nota-se um aviso que deixaram para você em sua mesa.

Olá.

Eu sou o responsável pela especificação do Caso de Uso “Fechar Período Letivo”, do módulo Período Letivo.

Ontem, consegui terminar de escrever a descrição para esse caso de uso. Gostaria de pedir que você revisasse essa descrição afim de que as chances de erros sejam diminuídas.

Muito obrigado.

Como foi visto, um outro membro da equipe pede que você revise a descrição que ele fez para o caso de uso Fechar Período Letivo. Tendo apresentado isto, este exercício demanda que você acesse a tarefa de revisão de descrição do caso de uso mencionado e corrija qualquer erro (digitação, concordância, siglas) que for encontrado na descrição do Caso de Uso mencionado.

## Exercício 4

Ótimo.

O tempo passou e o desenvolvimento do Sistema Acadêmico está correndo muito bem. Algumas tarefas foram realizadas por outros participantes sobre os casos de uso aos quais você estava associado. Dessa forma, a próxima tarefa a ser realizada é “Definir Casos de Teste”.

Para realizar essa atividade, você deve apenas escrever uma breve descrição de um caso de teste para o caso de uso

- “Vincular Aluno a Classe”.

Uma nota importante é que esse Caso de Uso está associado ao número 33 (vide nome do artefato no Sistema).

Outra nota importante é que a numeração do caso de teste é T62 e já está escrita no documento de Caso de Testes.

Por fim, a última nota importante é que o caso de teste é relativo à observação presente na especificação do Caso de Uso mencionado (na descrição).

Então, este exercício pede que você escreva a descrição de mais um teste para o caso de uso “Vincular Aluno a Classe”, descrevendo-o com base na observação presente na descrição da especificação do caso de uso.

## Exercício 5

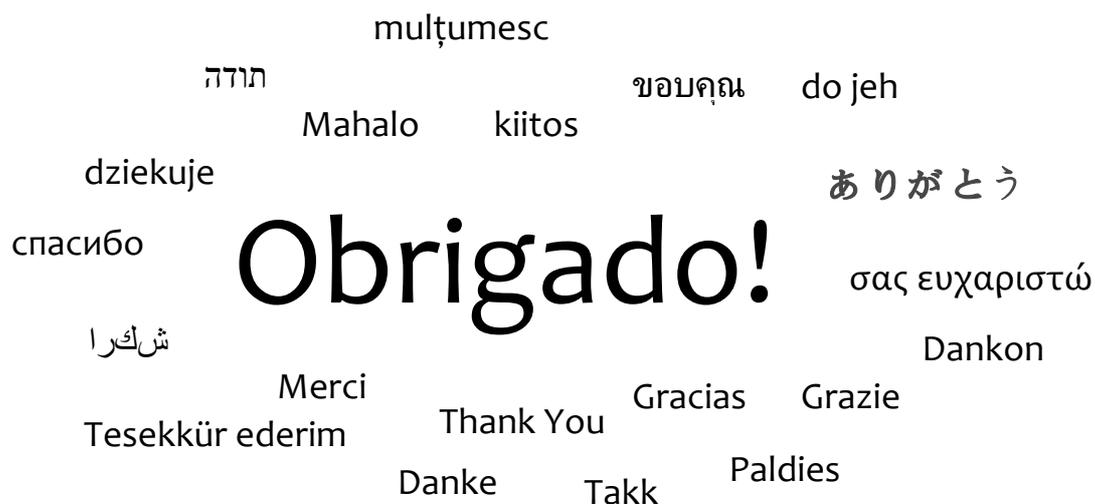
O desenvolvimento do caso de uso

- “Manter Ambiente de Aprendizagem”

está quase terminando. Atualmente, a atividade em execução é a última “Atualizar BD”.

Entretanto, a equipe de Engenheiros de Software da Organização apontou uma irregularidade no documento com o script SQL para o banco de dados. Essa irregularidade se refere à criação dos atributos da tabela: “unidade\_organizacional”. Seus atributos estão presentes no documento de modelo relacional adequado, o qual você deverá consultar.

Este exercício pede que você escreva o código de criação dos campos da tabela explicitada acima no artefato “patch38”. Uma comparação com o código de outras tabelas pode ser feita para que se entenda a sintaxe da linguagem.



Sua participação foi de grande importância.

## B.5 Final Questionnaire

# Questionário de Avaliação de Tecnologia

## Direções

Este questionário possui 12 perguntas.

Ao final do questionário, há um espaço para comentários.

Cada pergunta possui uma afirmação em seu enunciado, seguido de um espaço para a resposta.

Você deve marcar a resposta que melhor se adequa à sua opinião sobre a afirmação.

Apenas uma resposta deve ser marcada.

As perguntas podem ser respondidas em qualquer ordem.

O espaço para comentários no final do questionário deve ser utilizado para eventuais críticas, sugestões e dúvidas que porventura surjam. Dessa forma, a sua participação auxilia ainda mais o desenvolvimento do Estudo.

Código do Participante:

Data:

## Questionário

1) A utilização da Função DOI no meu trabalho me permite realizar tarefas mais rápido.

Discordo      Discordo      Discordo      Não      Concordo      Concordo      Concordo  
Completamente      Parcialmente      Razoavelmente      concordo,      Razoavelmente      Parcialmente      Plenamente  
nem discordo

2) A utilização da Função DOI melhora meu desempenho no trabalho.

Discordo      Discordo      Discordo      Não      Concordo      Concordo      Concordo  
Completamente      Parcialmente      Razoavelmente      concordo,      Razoavelmente      Parcialmente      Plenamente  
nem discordo

3) A utilização da Função DOI no meu trabalho aumenta minha produtividade.

Discordo      Discordo      Discordo      Não      Concordo      Concordo      Concordo  
Completamente      Parcialmente      Razoavelmente      concordo,      Razoavelmente      Parcialmente      Plenamente  
nem discordo

4) A utilização da Função DOI aumenta minha eficácia no trabalho.

Discordo      Discordo      Discordo      Não      Concordo      Concordo      Concordo  
Completamente      Parcialmente      Razoavelmente      concordo,      Razoavelmente      Parcialmente      Plenamente  
nem discordo

5) A utilização da Função DOI facilita a realização do meu trabalho.

Discordo      Discordo      Discordo      Não      Concordo      Concordo      Concordo  
Completamente      Parcialmente      Razoavelmente      concordo,      Razoavelmente      Parcialmente      Plenamente  
nem discordo

6) Eu considero a Função DOI útil no meu trabalho.

Discordo      Discordo      Discordo      Não      Concordo      Concordo      Concordo  
Completamente      Parcialmente      Razoavelmente      concordo,      Razoavelmente      Parcialmente      Plenamente  
nem discordo

7) Aprender a utilizar a Função DOI foi fácil para mim.

Discordo      Discordo      Discordo      Não      Concordo      Concordo      Concordo  
Completamente      Parcialmente      Razoavelmente      concordo,      Razoavelmente      Parcialmente      Plenamente  
nem discordo

8) Considero fácil fazer a Função DOI realizar o que eu objetivo.

Discordo      Discordo      Discordo      Não      Concordo      Concordo      Concordo  
Completamente      Parcialmente      Razoavelmente      concordo,      Razoavelmente      Parcialmente      Plenamente  
nem discordo

9) Minha interação com a Função DOI foi clara e compreensível.

Discordo      Discordo      Discordo      Não      Concordo      Concordo      Concordo  
Completamente      Parcialmente      Razoavelmente      concordo,      Razoavelmente      Parcialmente      Plenamente  
nem discordo

10) Eu considero a Função DOI flexível para interagir.

Discordo Completamente	Discordo Parcialmente	Discordo Razoavelmente	Não concordo, nem discordo	Concordo Razoavelmente	Concordo Parcialmente	Concordo Plenamente
---------------------------	--------------------------	---------------------------	----------------------------------	---------------------------	--------------------------	------------------------

11) Considero fácil me tornar habilidoso na utilização da Função DOI.

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Discordo Completamente	Discordo Parcialmente	Discordo Razoavelmente	Não concordo, nem discordo	Concordo Razoavelmente	Concordo Parcialmente	Concordo Plenamente

12) Eu considero a Função DOI fácil de ser utilizada.

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Discordo Completamente	Discordo Parcialmente	Discordo Razoavelmente	Não concordo, nem discordo	Concordo Razoavelmente	Concordo Parcialmente	Concordo Plenamente

## Comentários

Obrigado por separar um tempo para participar deste estudo. A sua experiência, o seu ponto de vista e suas observações são de grande importância para o desenvolvimento desse trabalho. Por isso, gostaria de saber sua opinião sobre o conceito da Função DOI aplicada na realidade de um Processo de Software. Há alguma crítica? Ou uma sugestão? E uma ideia de melhoria?

Este espaço é de livre escrita. Agradeço pelos comentários deixados.

---



---



---



---



---

---

---

---

---

---

---

---

---

---

---