INTEGRATING SOFTWARE REPOSITORY MINING: A DECISION SUPPORT CENTERED APPROACH

Luiz Dourado Dias Junior[1] and Eloi Favero[2]

[1]Department of Electric Engineering, UFPA, Belem, PA
ldourado1980@globo.com
[2] Department of Electric Engineering, UFPA, Belem, PA
favero@ufpa.br

ABSTRACT

Mining software repositories (MSR) research had significantly contributed to software engineering. However, MSR results integration across repositories is a recent concern that is getting more attention from researchers each day. Some noticeable research in this sense is related to the approximation between MSR and semantic web, specially linked data approaches which makes it possible to integrate repositories and mined results. Manifested that way, we believe that current research is not fully addressing the practical integration of MSR results, specially, in software engineering due to not considering that these results needs to be integrated to the tools as assistance to activity performers, as a kind of decision making support. Based on this statement this research describes an approach, named Sambasore, which is concerned with MSR results inter-repository integration and also to decision making support processes, based on tool assistance modelling. To show its feasibility we describe the main concepts, some related works and also a proof of concept experiment applied to a software process modelling tool named Spider PM.

KEYWORDS

Network Protocols, Wireless Network, Mobile Network, Virus, Worms &Trojon

1. INTRODUCTION

Over the years, bug tracking, version control and project management systems (among others) have produced valuable information for the understanding of various aspects related to software engineering. Such information has been exploited by an area of research called mining software repositories (MSR), which aims to turn these repositories into active sources of knowledge to support decision making in software projects [1].

Past research have generated doubtless theoretical and practical contribution in several areas [2], [3], [4]: a) **Software Evolution**; b) **Project Management**: pattern recognition techniques applied for effort estimation activities, showing that smaller features tend to have low error between the estimated and held effort; c) **Prediction Failures**: evaluating the relationship between refactoring activities and failure occurrences, concluding that they are significant for reducing failures.

Although MSR has produced significant contributions to various software engineering activities, we understand that an effort to integrate these multiple contributions is at an initial stage. In [5], a framework has been proposed to facilitate the extraction and relationship among data in multiple repositories. Moreover, in [6] an ontology based integration mechanism was proposed, which

DOI : 10.5121/ijsea.2012.3605

focused on generating an integrated repository of facts about source code and also providing SPARQL endpoints to query submition.

The concern about MSR integration and the perception of its benefits became evident in recent research. Analysing [5] and [6], it's possible to notice that both claims for inter-repository integration, which is certainly relevant. However, we believe that MSR integration problem must be understood in a wider scope that considers supporting decision making inside case tools, with context based assistance using mined knowledge.

Understanding this integration comprehension as essential to the growth of practical MSR applicability, as well as for favoring its use as a mechanism to support decision making in software projects, this paper describes an approach (called Sambasore) that focuses on guiding the execution and integration of MSR experiments among themselves, as previous research, and also to case tools – which is our main contribution.

The remainder of this paper is organized as follows: Sambasore conceptual foundations (including software process technology and ontologies), some related work, the overview of Sambasore approach (emphasizing its main activities and integration components), a proof of concept (focusing on an specific integration scenario) by defining and describing the main steps to provide a set of assistances for Spider PM software process modeling tool and ultimately, a critical analysis of the results, limitations and potential.

2. BACKGROUND

The following subsections address the key conceptual elements that underlie the development of Sambasore approach, namely: a) data mining; b) software process technology; c) ontologies.

2.1. Data mining

Data mining is an activity that is part of a larger process of knowledge discovery (KDD), defined as the identification of new, valid and potentially useful patterns for a user or task, from large volumes of data [7].

Typically, KDD process involves activities such as selection, pre-processing, processing, mining, interpretation and evaluation of results, described below: a) **Selection**: defining a subset of variables or samples, from which knowledge discovery will be performed; b) **Pre-processing**: operations like cleaning, removing noise and outliers, defining strategies to address missing data, among others; c) **Mining**: selection and application of computational intelligence methods on data to obtain / extract patterns; d) **Interpretation and Evaluation**: includes the interpretation of obtained results and the possibility of returning to previous steps for refinement, disposal or evaluation of new patterns.

MSR is an applied KDD process. By proposing to facilitate the integration of MSR results, Sambasore is composed by typical data mining activities, focusing in some specific concerns related to integration, as we describe in section 3.

2.2. Ontologies

In computer science, ontologies are defined as a formal specification of a conceptualization [8]. Thus, ontologies have the status of representing conceptual models in a given domain, describing it declaratively and separating it from procedural aspects [8]. For this, elements such as concepts, relations, properties and data types are used.

Ontologies have been used to represent knowledge mined from software repositories and to integrate research results [6]. Sambasore uses ontologies on the same perspective, linking software process concepts and relationships to MSR mined knowledge and also to assistance definition. Adding this possibility makes it possible to define SPARQL queries using knowledge obtained from MSR, in order to provide assistance to software engineering actors.

2.3. Software Process Technology (SPT)

SPT proposes the development and adoption of process centered software engineering environments (PSEE), to automate the management of software processes [9]. Software process can be understood as a set of activities performed in software projects [9].

Processes are described abstractly, by models that include various types of information: who, when, where, how and why certain steps are taken. To facilitate this activity, several languages were created to support process modeling. In this context, it is worth noting OMG's effort to define a generic language for software development process modeling - called SPEM [10].

Besides the availability of a language for process modeling, some tools have been developed to support the modeling activity, among them Spider PM. This tool uses a specific modeling language (Spider PML), derived from some simplifications over SPEM [11].

Provided that assistance definition involves assisting software engineering actors when performing activities, we believe that SPT would (and specifically Spider PML) help us in specifying the context where assistance should be provided.

3. RELATED WORK

On the following subsections we briefly discuss recent work which is similar and relevant to our research.

2.1. On Mining Data across Software Repositories

With the increase in the size of the data maintained by software repositories, automated extraction of such data from individual repositories, as well as of linked information across repositories, has become a necessity [6].

A framework that uses web scraping to automatically mine repositories and link information across them has been proposed and implemented in two manners [6]: a) **first**: to automatically identify and collect security problem reports from project repositories that deploy the Bugzilla bug tracker; b) **second**: to collect security problem reports for projects that deploy the Launchpad bug tracker along with related vulnerability information from the National Vulnerability Database.

The main contribution of [6] in our research is to reinforce that linking information across repositories is a strong necessity. For one side, the referred work states a promising alternative to treat the problem, but this doesn’t reach MSR integration to case tools, like ours.

2.2. A Linked Data platform for mining software repositories

Mining software repositories involves the extraction of basic and value-added information, to support different stakeholders for various purposes [7]. To avoid unnecessary pre-processing and analysis steps, sharing and integration of both basic and value-added facts are needed [7].

To surpass the problem stated, SeCold was introduced as an open and collaborative platform for sharing software datasets. SeCold provides the first online software ecosystem Linked Data platform that supports data extraction and on-the-fly inter-dataset integration from major version control, issue tracking, and quality evaluation systems.

The approach is based on the same fundamental principle as Wikipedia [7]: researchers and tool developers share analysis results obtained from their tools by publishing them as part of the SeCold portal and therefore make them an integrated part of the global knowledge domain.
Although the research in [7] represents a great advance, we understand that their focus is infrastructure related. In our research we propose the same linked data/ontology based approach, but in addition to this we propose specific concepts to link MSR results to software process execution, transforming them into assistance to software engineering tools.

4. SAMBASORE APPROACH

Consider a typical software development project scenario, in which functional size estimation is needed to be done using function point analysis technique. Suppose further that the company in which the project is being developed has expertise in similar projects, and that this expertise is historically condensed in spreadsheets repository.

In this scenario, assume that an **independent estimator** will perform a size estimative **activity** for a system (**named X**) using a vision document as input (**named D**) and supported by a tool (**named T**). Considering that the company has a repository of previous estimates: how it could be used to support the estimator, providing him/her with estimates based on vision documents **similar to "D"**?

To address such scenarios Sambasore is proposed. In the illustrated scenario, an assistance project would be designed to **T** by a team composed by software/knowledge engineers and developers. The project would purpose to identify sources of knowledge (repositories that contain vision documents and estimate spreadsheets), apply data mining techniques in order to relate the artifacts to concepts in the ontology, and finally, build SPARQL queries whose results would be integrated to **T**.

Sambasore approach aims to act in this type of scenario, using ontologies (to define assistance SPARQL queries) and software process technology concepts (modeled as ontologies and used to define when and to whom SPARQL queries would be exhibited), in order to encourage the integration of MSR results and them to software engineering supporting tools. The following subsections describe the Sambasore approach focusing on its integration aspects, addressing its overview, planning and development activities.

4.1. Overview

To achieve the integration goal, we defined Sambasore comprised by: a) a **software process**; b) a **set of domain ontologies** – an ontology for multi-repository MSR integration, another to software process knowledge and one to define assistances (based on MSR results and software process knowledge); c) a **reference model** for developing knowledge extractors for software

repositories; d) a **reference architecture** for developing extractors; e) a **framework** for developing these extractors, and f) a **set of tools** to support the defined process, among which are included the **integration components**. Figure 1 illustrates how some of these elements are interrelated.

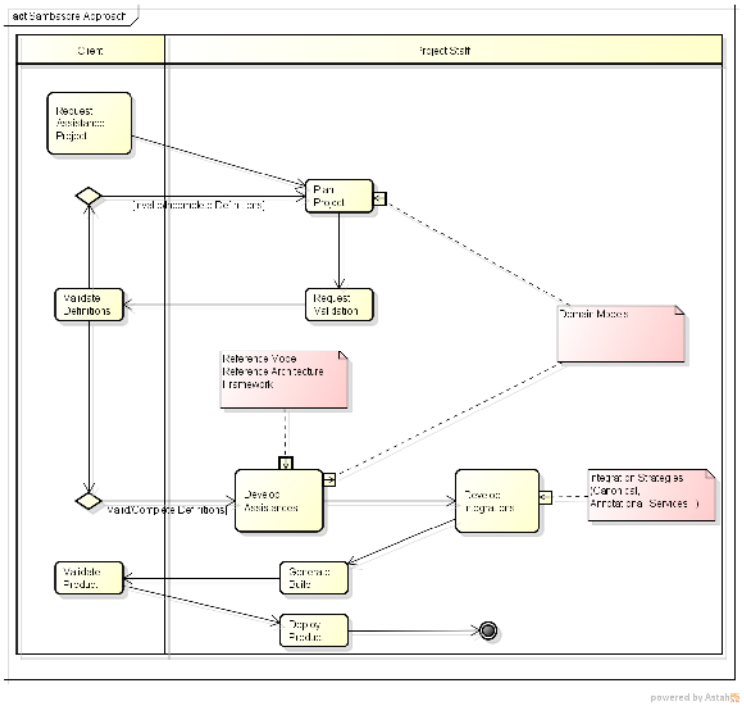

Figure 1. Sambasore approach overview.

In Figure 1, assistance and integration development activities are noteworthy. Assistance development activities addresses aspects related to inter-repository integration, and are influenced by the: model and reference architecture; Sambasore's framework; domain models provided (represented by Sambasore's ontologies). Model and reference architecture guides the development of miners/extractors and their integration through domain ontologies; Sambasore's framework guides the implementation of miners/extractors using Java language; domain ontologies provides MSR inter-results integration and assistance definition. The other activities focuses on the coupling of MSR results to case tools, using Sambasore tools and its integration components.

The remaining activities represent a simplified view of existing activities in standard software process. This is justified by the fact that integration is materialized by a maintenance effort to evolve the source code of to be assisted case tools. Understood this way, activities like definition validation, product build generation and deployment are expected. In the following sections, we describe planning and development activities, and also the integration components. For the scope of this paper we won't describe the ontologies used by Sambasore, but we will exemplify how they are used when describing the proof of concept.

4.2. Planning Phase

At this phase, the goal is to clearly define the project scope. In this context, the objective is to prioritize the assistance needs that must be met by the project and, in addition, to define

knowledge extracting strategies for the software repositories available. Figure 2 illustrates the activities of this phase, followed by detailed description:

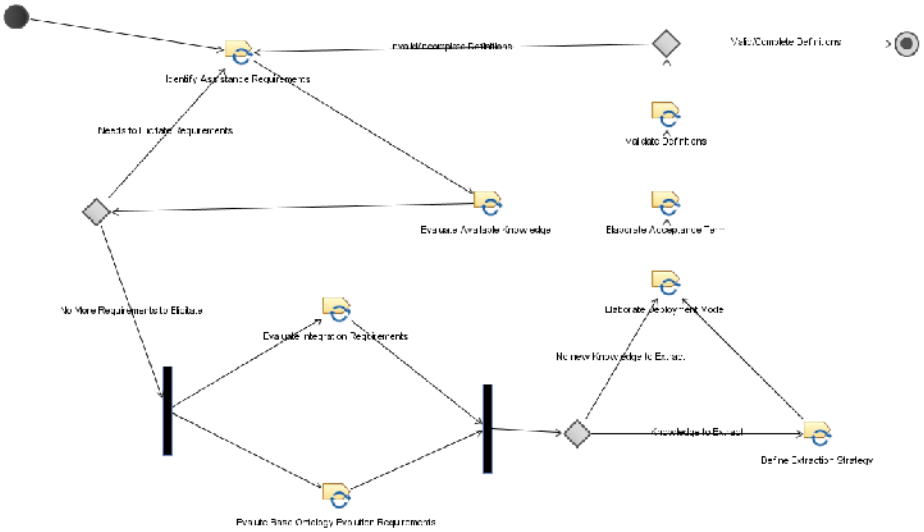

Figura 2.Planning phase.

- **A1 - Identify Assistance Requirements**: identifying assistance requirements and also other functional and non-functional ones (project scope);
- **A2 - Evaluate Available Knowledge**: identifying key concepts and relationships needed to meet assistance requirements and their availability on knowledge base;
- **A3 - Evaluate Integration Requirementes**: evaluating / identifying requirements for assistance integration to case tools (defined at project scope);
- **A4 - Evaluate Base Ontology Evolution Requirements**: identifying concepts and relations not covered by the knowledge base, in order to meet assistance requirements;
- **A5 - Definition Extraction Strategy:** defining required concepts and relations for each specified assistance, as well as data sources, strategies for selection, integration and pre-processing and also mining techniques;
- **A6 - Elaborate Deployment Model:** defining the project deployment model;
- **A7 - Elaborate Acceptance Term:** elaborating and getting an approved project acceptance term;
- **A8 - Validate Definitions**: interacting with project stakeholders for project definition validation and startup.

4.3. Development Phase

In the Development phase, the goal is to develop the set of specified assistances, the set of extractors that will extract the knowledge needed to assistance definition and finally to integrate such assistances to case tools, defined on project scope. To achieve these goals, the development phase is divided into iterations shown in Figure 3 and described below:

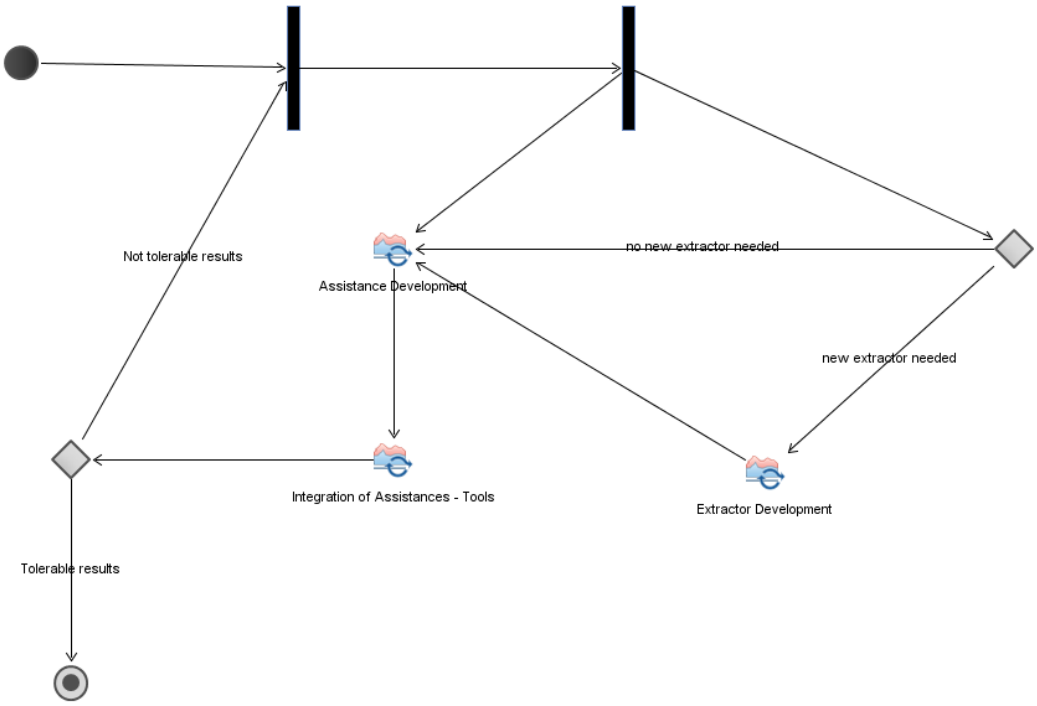

Figure 3.Development Phase

- **I1 – Assistance Development:** developing the specified assistances;
- **I2 - Integration of Assistance - Tool:** integrating the developed assistances to case tools;
- **I3 - Extractor Development:** developing knowledge extractors, if some assistance requires knowledge not provided by pre-existent facts on domain ontologies.

It is noteworthy that I3 activity is optional. At planning phase, specified assistances are verified to check if they require some knowledge not previously collected. If this is the case, a knowledge extraction strategy must have been defined. In the development phase, the planned extraction strategies must then be developed, which includes MSR conduction (if not previously done).

4.3.1 Assistance development iteration

The **Assistance development** iteration consists of the following set of activities:

- **A1 - Develop Information Resource To Assistance**: mapping data sources for each specified assistance;
- **A2 - Map Tool Integration Parameters:** defining parameters that need to be informed by the case tool (which will have assistances integrated to), so that the assistance engine can provide the content of knowledge sources;
- **A3 - Verify Queries and Tool Integration Parameters**: ensuring the quality of the source of knowledge defined, as well as the integration parameters.

4.3.2 Extractor development iteration

The **Extractor Development iteration** consists of the following set of tasks (T) and activities (A):

- **T1 - Evolve Base Ontology**: adding / modifying concepts, relationships and properties at integration ontologies for MSR results integration;
- **T2 - Develop Data Sources**: developing classes to recover data from defined daya sources, according to assistance requirements and knowledge extraction strategies;
- **T3 - Develop Selectors**: developing a set of classes for selection over previously collected data;
- **A1 - Develop Collectors**: developing classes to provide data transformation, selection, pre-processing and mining;
- **T4 - Develop Knowledge Extractor**: developing classes to concept, relationship and property extraction from data previously mined, as well as their integration into the knowledge base;
- **T5 - Configure Sambasore Project**: generating a project definition file containing properties to guide execution of developed extractors;
- **T6 - Execute Sambasore Project**: programs execution, integration ontology population and knowledge base evaluation (after populated by Sambasore project).

The tasks and activities of extractor development iteration are performed by knowledge engineers and developers. The former are responsible for T1, using the Protége editor [13]. The other tasks and activities are carried out by developers which should use Java IDE to support them (since T2, T3, A1, T4 requires extensions over Sambasore's framework - Java library). Later, developers should use Sambasore tools to execute the project - which will use the developed classes to effectively incorporate MSR results to the knowledge base, making them available to be consumed by previously developed assistances, which will be integrated to case tools using integration components.

4.3.3 Integration of Assistances Tool iteration

The **Integration of Assistance – Tools** iteration, as shown in Figure 6, consists of the following set of tasks (T) and activity (A):

- **T1 - Integrate Assistance to Tools**: creating or modifying the source code of case tools, as specified by assistance requirements and their integration parameters, so that the case tool can provide the integration parameter values and get assisted;
- **A1 - Build Generation:** generating a product build (modified tool) at verification environment;
- **T2 - Evaluate Results:** intends to realize some verification activities to check if the agreed the results were obtained.

The tasks and activities of this iteration are performed by knowledge engineers and developers. The former get involved in T2 together with developers, because the task aims to evaluate project results prior to client/stakeholders validation. The other tasks and activities are performed by developers, which should preferably use the canonical integration component (*file-based*), especially when the target tools have not been developed in Java. These components and integration scenarios are discussed in the following section.

4.4. Integration Components

MSR results integration to case tools is the final goal of Sambasore's approach. In this scenario, besides the activities that lead to this, we believe that some software components should allow, in theory, MSR results integration to any existent and to be developed case tool.

The integration solution core is based on two software components. The first (C1) is a web application, whereas the second (C2) is a file system monitor application. The first offers an integration service with a public method - designed to obtain a list of assistance with their corresponding content (text, URL or SPARQL query). The second component monitors a set of folder in the file system - when a file request (JSON formated and named with "req" extension) is added to the monitored folder, the component (C2) extracts the file information and turns it into a request to the web service (offered by C1). The web service response is saved to a new file (JSON formated and named with "res" extension) which is then read by the original application (case tool) and displayed as a set of resulting assistance. Figure 5 illustrates an example of an assistance request in JSON format - this structure can be used by any case tool that someone needs to provide assistance for.

```
{
"entityName":"Class",
"actionName":"RecoverSimilarClass",
"toolName":"Eclipse",
"contextItemFilePath":"/hellowordproject/src/HelloWord.java",
"outputFilePath":"/sambasore/communication/hellowordproject/timestamp.res",
"serviceUrl":"http://service.url",
"parameters":["className=HelloWord.java"]
}
```

Figure 5. Sample JSON format request.

JSON is a lightweight standard for information exchange between systems, considered readable for humans and easier for interpretation and generation by machines [14]. The standard was developed as a subset of the JavaScript language in 1999 and is language independent. [14] For these characteristics and the need to allow any tool integration to MSR results, we understood that this pattern could be easily adopted by any programming language that manipulates files.

It is important to describe some of the elements in Figure 5. The first is entityName, which is the type of entity manipulated at a particular case tool. The second is actionName, and corresponds to the action being performed over some entity. The third is toolName, and corresponds to the name of the case tool in which the entity is being manipulated. The fourth is contextItemFilePath which is the file path to entity manipulated content - such content is transferred to assistance engine whose extracts its content and inserts it into knowledge base. The fifth element is outputFilePath which is the file path where the assisted tool expects to get the response from assistance engine. The sixth element is serviceUrl which indicates the URL for web service component C1, and finally, parameters element represents a set of contextual parameter - in this case the name of the class we are interested to evaluate the similarity of.

To define integration solution we outlined three integration scenario bellow, which will be better discussed in subsequent paragraphs:

- **SCE1 –** Java based applications developed with aspect orientation support;
- **SCE2 –** Applications that support SOAP protocol;
- **SCE3** – Other Application that do not support SOAP.

For **SCE1** we developed an integration component **C3** which provides a set of class and annotation. The annotations are used to identify integration methods (at case tool source code), indicating that before or after method execution, a method will be called to pass integration parameter values to a façade that will be accessed by an aspect that, finally, will interact with **C1**. Such a scenario requires importing **C3** Java library at case tool source code project and in this case, accordingly to scenario description it will covers only tools that have been developed in Java language.

In scenario **SCE2** we considered applications (Java or not) having components that deals with SOAP protocol, or allow the use of this type of protocol. In this case, the case tool has the responsibility to prepare a SOAP request, submit it to the web service in **C1**, process and display the results. We believe that this scenario allows broader integration, due to the existence of tools developed in programming languages that supports SOAP (such as Java, PHP, Python and others).

Finally, **SCE3** fit applications that actually are not developed in Java and have no possibility to use SOAP protocol. In these cases, we chose an integration solution based on shared files in a file system. Files with specific naming (to indicate request and response) are saved by the tool (request), read and written (reply) by component **C2** and subsequently read by the tool. The basic principle of this mechanism is that, in theory, any tool (in software engineering context) can read and write files, and can thus be adapted to allow integration proposed in this paper. It is considered that this is the broader integration component.

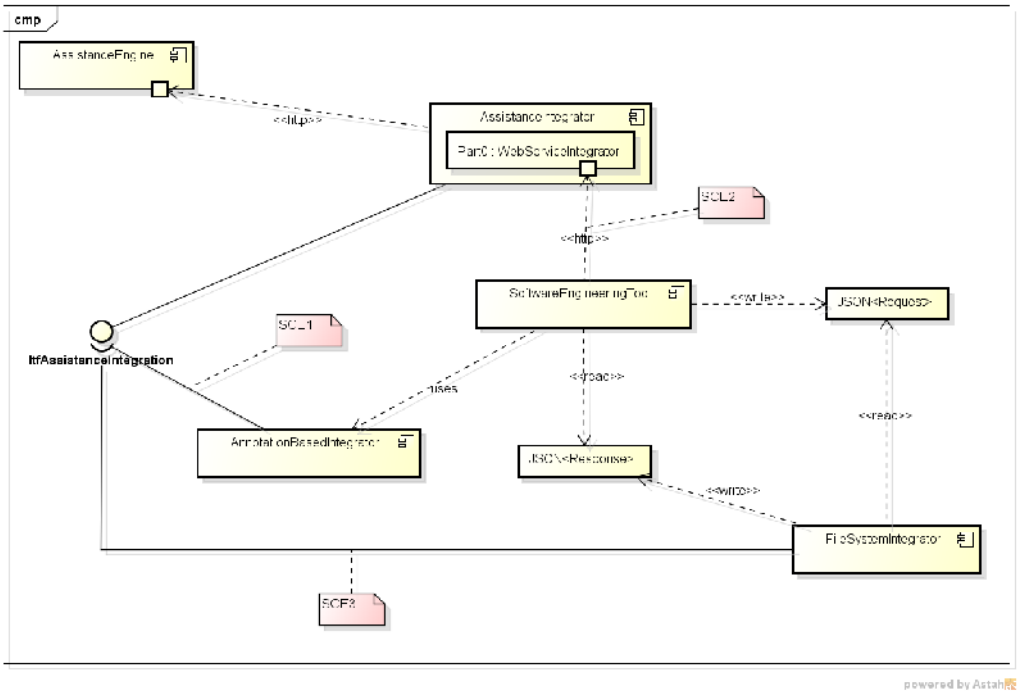

Figure 6. Architectural Overview of Integration Solution

Figure 6 shows an architectural view that contains **C1**, **C2** and **C3**, respectively called *WebServiceIntegrator*, *FileSystemIntegrator* and *AnnonationBasedIntegrator*. It can be noticed that, as illustrated previously by scenarios, the central component is **C1**. One can also note that **C1** interacts directly with assistance engine by HTTP requests, while others only reach the engine

through **C1** via *ItfAssistanceIntegration* interface. The integration scenarios described can also be identified in Figure 6 by a set of notes with specific corresponding labels.

5. SPIDER-PM ASSISTED

To illustrate the use of Sambasore, we applied the approach to provide some assistance to a software process modeling tool (Spider PM). This tool belongs to Spider project, institutionalized in 2009 at the Institute of Exact and Natural Sciences in Federal University of Para (UFPA - Brazil) and has the following main objectives, according to [15]:

- Identify free software tools that allows the creation of work products derived from the expected results described in the specific practices and goals in the areas of process model "MPS. BR "and CMMI;
- Specify and develop a suite of tools to provide a more integrated use in order to support the deployment of processes / areas of process models "MPS. BR "and CMMI.

Spider PM presents features related to software process elements breakdown as well as the relationships between these elements [16]. For this, it uses a language called Spider ML, which is a simplification of SPEM – a process modeling language defined by the OMG [16].

This tool was choose due to the following main reasons: a) **source code access**: by the partnership with Spider team we got access to Spider PM source code, what lot us develop some peace of code to evaluate the feasibility of the approach; b) **Java source code**: we have been working with Java for many years, so this factor would ease significantly the integration process, and consequently the proof of concept; c) **simplicity of Spider ML**: as we needed to define an ontology in software process domain (to allow assistances to be defined and integrated to software process), the simplicity eases our integration job.

5.1. Planning

As presented in section 4.2, the main objective of the planning phase is to identify and document the scope of an assistance project. Below, we describe the main results in each phase of activities cited:

- **A1 - Identify Assistance Requirements:** to perform proof of concept we proposed two assistances. The first (ASS1) intended to submit a process phase name and get phases that already composed process models together with phases with names similar to submitted one (SPARQL content). The second (ASS2) should present a set of results got from Google search on modeling phases in software processes (URL content);

- **A2 - Evaluate Knowledge Available:** as the software process domain ontology was not set, none of the concepts was available. Thus, it was necessary to build ontology altogether, based on SPIDER ML specification;

- **A3 - Evaluate Integration Requirements:** desired assistances relied only on phase names and the similarity between them. Thus, we realized that this would be applied to any of modeling entities used by Spider PM. Considering this, we defined two integration parameters - the first is the name of the entity being modeled by the process modeler, while the second is the type of entity (phase, activity, status, among others);

- **A4 - Evaluate Ontology Evolution Base Requirements:** besides SPIDER ML concepts and relationships, we noticed that to provide desired assistance we also needed some extra relationships and concepts - isStronglySimilarTo (**R1**), isFairlySimilarTo (**R2**), and isWeaklySimilarTo (**R3**) and a concept seeAlso (**C1**) (to indicate the co-occurrence strength of entities in software process models);

- **A5 - Extraction Strategy Definition:** we decided that **R1**, **R2** and **R3** would be obtained by JaroWinkler distance calculated between entity names. For this, each relation is associated with a range of possible JaroWinkler values (0 to 0.3 defines **isWeaklySimilarTo**, 0.4 to 0.7 defines **isFairlySimilarTo**, 0.7 to 1 defines **isStronglySimilarTo**). Moreover, for **seeAlso** we defined that the strength of association between entities would be addressed by calculating support and confidence as defined in [17];

- **A6 - Elaborate Deployment Model**: not performed;
- **A7 - Elaborate Acceptance Term:** not performed;
- **A8 - Validate Definitions:** not performed.

It's important to state some remarks about **A5**, **A6**, **A7** and **A8**. About A5, we must say that relationship definitions are hard assigned (some fuzzy strategy can be used), but, as long as the intention was just to validate the idea, we understood that it would suffice. A6, A7 and A8 were not performed because the goal not focused the end users, but the proof that the proposed approach would provide MSR results integration to a software engineering tool.

5.2. Development

The development phase is responsible for transforming assistance requirements in integrated software assistance applied to a tool. In the context of proof of concept, this step comprised iterations whose tasks involved SPARQL queries design for ASS1 assistance and source code evolvements in Spider PM for effective integration of ASS1. The iterations and its main activities are described in the following subsections.

5.2.1 Assistance Development iteration

5.2.1.1 Develop Information Resource to Assistance

This activity was carried out with the Sambasore Assistance Modeling tool, and the result was documented in artifact Map Resources for Assistance, and presented in Table 3 for review and discussion.

Table 3 - Map of resources for assistance.

Code	Type of Assistance	Source of Knowledge

ASS1	SPARQL	PREFIX //www.w3.org/1999/02/22-rdf-syntax-ns#
		PREFIX //www.semanticweb.org/ontologies/2011/10/software-assistance.owl#
		PREFIX //www.semanticweb.org/ontologies/2011/10/core-process-definition.owl#
		SELECT ?entityType2 ?name2
		WHERE {
		{
		?entity type #FOCUS_ENTITY_TYPE#.
		?entity name \#ENTITY_NAME#\.
		?entity hasSeeAlso ?seeAlso.
		?seeAlso seeAlso ?entity2.
		?entity2 name ?name2.
		?entity2 type ?entityType2.}
		UNION
		{?entity type #FOCUS_ENTITY_TYPE#.
		?entity name \#ENTITY_NAME#\.
		?entity isStronglySimilarTo ?entity2.
		?entity2 hasSeeAlso ?seeAlso.
		?seeAlso seeAlso ?entity3.
		?entity3 name ?name2.
		?entity3 type ?entityType2.
		}
		UNION
		{?entity type #FOCUS_ENTITY_TYPE#.
		?entity name \#ENTITY_NAME#\.
		?entity isFairlySimilarTo ?entity2.
		?entity2 hasSeeAlso ?seeAlso.
		?seeAlso seeAlso ?entity3.
		?entity3 name ?name2.
		?entity3 type ?entityType2.}
		}

ASS2	URL	http://www.google.com.br/ #hl=pt-BR&sclient=psy-ab &q=fases+do+processo+de+ software &oq=fases+do+processo+de+software &gs_l=hp.3..0j0i8i30j0i5i30.7970. 12300.1.12801.29.24.0.3.3.1.557. 9050.3-8j10j3.21.0...0.0. ReI6sQ99f2o&pbx=1 &bav=on.2,or.r_gc. r_pw.r_cp.r_qf.,cf.osb &fp=c96f3b0a734469e9&biw=1600&bih=796

From Table 3 it is possible to realize that the source of **ASS1** was a SPARQL query, while ASS2 source is a **URL**, or the path to an information resource that represents the knowledge to be provided.

It should be noted also that in Table 3 the query which is designed to ASS1 is generic about the entity type. This is possible due to the use of integration parameters #FOCUS_ENTITY_TYPE# and #ENTITY_NAME#.

5.2.1.2 Map Tool Integration Parameters

This activity was carried out based on the contents of the **Map Resources for Assistance**, produced in **Develop Information Resource to Assistance**. The result is shown in Table 4, and discussed below.

Table 4 - Specification of integration requirements to Tools.

Code	Description of assistance	Action	Implementation Units	Integration Parameters
ASS1	Present phases that have been modeled in conjunction with phases similar to the last one modeled	Name Change of Phase	spider.pm.gui.PhasePropertiesPanel	#FOCUS_ENTITY_TYPE# = entity type #ENTITY_NAME# = entity name

From Table 4 is possible to note that the implementation unit *PhasePropertiesPanel* needed to be evolved in Spider PM. This unit is responsible for controlling user interaction with the panel properties existing in a phase diagram, dealing among other events with name change, directly related to **ASS1**. The mapping on Table 4 is essential to support the coding phase.

5.2.2 Extractor development iteration

5.2.2.1 Data source development

This activity was developed in Eclipse IDE. We extended the *Sambasore framework* and developed the code for data source classes, which would allow the derivation of knowledge needed to provide **ASS1.**

We considered that providing **ASS1** would involve knowledge about process models and software entities that comprise these models. In this sense, we defined two data sources. The data sources classes developed are: a) **SpiderProcessModelInformationSource**: representing data about the relationship between two entities in a process model diagram; b) **ProcessModelInformationSource**: representing a process model produced in Spider PM (e.g., file name and "serialized" model).

It is noteworthy that in both data sources we extended *BaseInformationSource* from *SambaSore Miner*, which inherits from *BasicDbObject* database *MongoDB* - attributes were not defined because this superclass owns a HashMap to implement the behaviors needed - creating extensions was only done to conceptually separate data sources. The next subsection describes the implementation of data collectors that uses the classes described in this subsection to persist data from software repositories in MongoDB.

5.2.2.2 Data Collector Development

Considering the possibility of defining an assistant to any entity in a process model, we decided to develop two collectors: a) **SpiderProcessModelCollector**: responsible for persisting software

process model dataset; b) **SeeAlsoCollector**: responsible for the persistence of a dataset consisting of data defining the origin, the destination, the support and confidence of an association between two entities in a model. Figure 7 illustrates a stretch of implementing SpiderProcessModelCollector.

```
public class SpiderProcessModelCollector extends
        BaseSambaSoreDataSetCollector<ProcessModelInformationSource> {

    private String[] processModelPaths;
    private List<String> mappedFileTypes;


    public SpiderProcessModelCollector(
            Properties properties,
            ItfDataSetPreProcessor<ProcessModelInformationSource> preProcessor,
            ItfDataSetIntegrator<ProcessModelInformationSource> integrator,
            BaseDataSetTransformator<ProcessModelInformationSource> transformator) throws Exception {
        super(properties, preProcessor, integrator, transformator);

        this.processModelPaths = properties.getProperty(this.getClass().getSimpleName()+".processModelPaths") != null
                ? properties.getProperty(this.getClass().getSimpleName()+".processModelPaths").split(",") : null;

        String[] tmpMappedFileTypes = properties.getProperty(this.getClass().getSimpleName()+".extractable.suffix").split(",");
        this.mappedFileTypes = Arrays.asList(tmpMappedFileTypes);

    }

    @Override
    public List<ProcessModelInformationSource> extract() throws Exception {
        List<ProcessModelInformationSource> files = new ArrayList<ProcessModelInformationSource>(
                0);
        for (String processModelDir : this.processModelPaths) {
            subExtractDir(processModelDir, files);
        }
        return files;
    }

    private void subExtractDir(String dir,List<ProcessModelInformationSource> files) throws Exception {
        File inputFile = new File(dir);
        if (inputFile.isDirectory()) {
            for (File inputFile1 : inputFile.listFiles()){
                subExtractDir(inputFile1.getAbsolutePath(),files);
            }
        } else if (inputFile.isFile()){
            if (!inputFile.getAbsolutePath().contains("\\CVS\\")){
                if (getFileType(inputFile) != null){
                    ProcessModelInformationSource process = new ProcessModelInformationSource();
                    process.addFeature("processModel",new XStream().toXML(ModellingFileManager.loadModel(inputFile.getAbsolutePath())));
                    process.addFeature("processModelPath",inputFile.getAbsolutePath());
                    files.add(process);
                }
            }
        }
    }
```

Figure 7.Portions of the implementation process model collector.

With respect to Figure 7 is worth noting, first, the inheritance of the *BaseSambaSoreDataSetCollector* Sambasore class, because it implements a persistence mechanism based in *MongoDB* database (choose due to scalability). Moreover, it is important to cite the use of *generics* feature to typify *ProcessModelInformationSource* information entity which composes the dataset. Another important point is that as it was not necessary to perform classical mining activities such as integration, pre-processing and transformation. Finally, there is a method which overrides "*extract*" from *BaseSambaSoreDataSetCollector* and defines a logic which search files in a folder; check if the extensions are within a pre-defined set, and does the final *dataset* extraction.

5.2.3 Tool Assistance Integration Iteration

The integration iteration activities involve the integration itself, as well as build generation and, finally, an evaluation of it by the client. This section describes the main aspects of activities related to assistance integration to software engineering tools.

First, it is important to situate the exact usage scenario in which the **ASS1** assistance was proposed. Throughout the modeling of a software process typically phases are defined. Part of this process involves the definition of the phase name which in Spider PM occurs at the GUI shown in Figure 8.

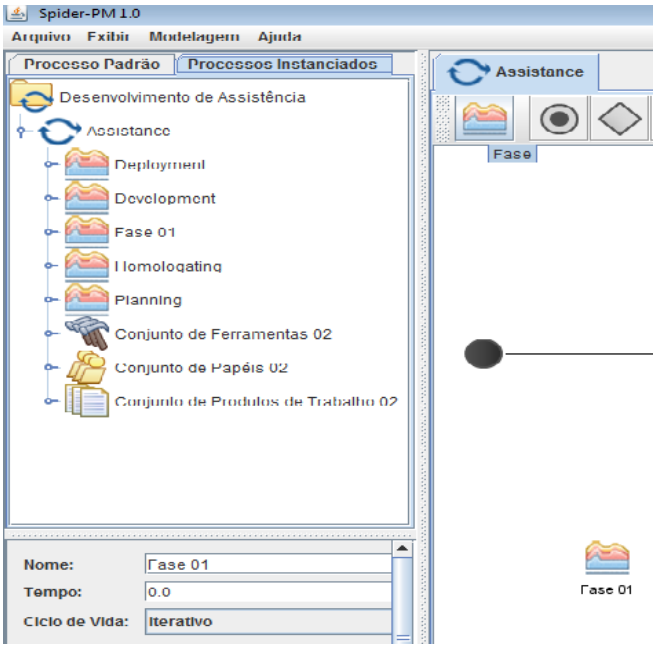

Figure 8.Creation / nomination phase in Spider PM.

Assistance **ASS1** was thought to be applied after the modification of the field "Nome" in Figure 8. Once modified, the file name and the model are subjected to *AnnotationBasedIntegrator* component (shown in Figure 6 of Section 4.4) that ultimately drives the web service integration. This service turns a set of extractors that acts over the submitted file, and include the knowledge in the knowledge base. Finally, the requested assistance contents are recovered and returned to the application (in this case Spider PM).

In this context, the first step of integration involved importing Spider PM Source code in Eclipse IDE. Later, *AnnotationBasedIntegrator* component was added to the project. After that, some modifications were performed on "spider.pm.gui.PhasePropertiesPanel" which is the controller class of GUI illustrated in Figure 8. Figure 9 illustrates the method which responds to the change in phase name (previously illustrated in Figure 8).

```
@AssistanceAfterAction(useGenericIntegrationFacade=true,
                       idFieldForActionId="updateComponentActionId",
                       configMethodForActionAssistance="configUpdateComponentAction")
private void nameTextFieldActionPerformed(ActionEvent evt) {
  if (!this.updating)
    updateComponent();
}
```

Figure 9.Method that accounts for the phase name change.

From Figure 9 it is possible to see some details regarding integration component used. First, it can be seen the use of *AssistanceAfterAction* annotation which indicates that an attempt to assistance recovery will be made after execution of this method, ie, after the name of the phase is modified.

Then we should mention the *useGenericIntegrationFacade* attribute indicating that the facade based component integration pattern is used. Later, there is the attribute *idFieldForActionId* which indicates the class attribute name of the assistance request identifier. Finally, *configMethodForActionAssistance* indicates which method is responsible for interacting with component façade and set values to integration parameters (Ex: the controller interacts with *GenericIntegrationFacade* to set *RoleUseName*). Figure 10 illustrates the implementation of the method *configUpdateComponentAction*.

```
//AssistanceIntegrator
public void configUpdateComponentAction(){
    String fileName = ModelPersistenceUtil.getPath(View.class.getName());
    if (fileName != null){
        String finalName = null;
        try {
            File file = new File(fileName);
            finalName = file.getAbsolutePath().replace(file.getName(), file.getName()+"-PM.model");
            ModellingFileManager.save(finalName);
        } catch (Exception e){
            e.printStackTrace();
        }

        if (finalName != null){
            this.updateComponentActionId = String.valueOf((new Date()).getTime());
            GenericIntegrationFacade.getInstance().setEntity(this.updateComponentActionId, "SpiderProcessModelData");
            GenericIntegrationFacade.getInstance().setActionName(this.updateComponentActionId, "change_entity");
            GenericIntegrationFacade.getInstance().setToolUseName(this.updateComponentActionId, "spider");
            GenericIntegrationFacade.getInstance().setTaskUseName(this.updateComponentActionId, "Process Modelling");
            GenericIntegrationFacade.getInstance().setRoleUseName(this.updateComponentActionId, "Process Modeller");
            GenericIntegrationFacade.getInstance().setFileName(this.updateComponentActionId, finalName);
            GenericIntegrationFacade.getInstance().setIntegratorServiceAddress(this.updateComponentActionId,
                        "http://localhost:8085/prjAutomaticSRKnowledgeCollectorWb/services/Service");
            GenericIntegrationFacade.getInstance().setViewerClass(this.updateComponentActionId, ChangeComponentAssistanceViewer.class);
            GenericIntegrationFacade.getInstance().setParametersForAssistanceParse(this.updateComponentActionId,
                    new String[]{
                    "#ENTITY_NAME#=\""+this.nameTextField.getText()+"\"",
                    "#FOCUS_ENTITY_TYPE#=cp:Phase"});
        }
    }

}
```

Figure 10.Configuration method for interacting with assistance façade.

In Figure 10 it is important to highlight the points 1 and 2 in red. The first point illustrates the software process model file name setting, defined when process modeler saves the model. Then in the second section, an interaction between *configUpdateComponentAction* method (implemented in action controller) and the integration façade of *AnnotationBasedIntegration* is highlighted.

In particular, the second section in Figure 10 illustrates basic integration parameters value setting. It is worth noting, in this context, the last line where an array of String is passed as parameter to *GenericIntegrationFacade* containing #ENTITY_NAME# and #FOCUS_ENTITY_TYPE# parameters. The first is one refers to the name of the entity (a phase in the example), while the second is the type of entity that is in focus. It is noteworthy that, at last, there is a reference to a concept of the domain ontology that was produced at the activity named as evolution of ontology base (not detailed in this paper). In the example, as envisioned that integration could be done with any entity modeled on Spider PM, it was considered valid to define a parameter that represents the type of entity modeled (thus allowing a single parameterized assistance to be defined). However, this decision creates a stronger coupling between Spider PM Integration implementation and the assistance engine, because the tool developer team needs to understand some specific ontologies details used by assistance engine. Another way to implement the same behavior could have been the setting of many assists (one for each entity type).

It's also important to notice the *viewerClass* integration parameter. This is responsible to point out a class that will get assistance results provided by *AnnotationBasedIntegration*, according to the designed user interaction pattern. A portion of code of this implementation is showed in Figure 11.

```
public class ChangeComponentAssistanceViewer extends AbstractAssistanceViewer {

    //AssistanceIntegrator
    @Override
    public void showAssistance() {
        if (this.assistances != null && this.assistances.length > 0){
            for (Assistance assistance : this.assistances){
                if (assistance != null){
                    if (Assistance.SPARQL.equals(assistance.getType()) &&
                            assistance.getJenaResults() != null
                            && !assistance.getJenaResults().trim().equals("")){
                            String[][] data = JenaResultSetUtil.fetchResultsAsArray(assistance.getJenaResults().
                                    replaceAll("http://www.semanticweb.org/ontologies/2011/10/Core-Process-definition.owl#", ""));
                            JTable table = new JTable(data,new String[]{"Entity Type","Entity Name"});
                            JScrollPane scroll = new JScrollPane(table);
                            JFrame dataFrame = new JFrame("People that modeled this entity also modeled:");
                            dataFrame.add(scroll);
                            Dimension preferredSize = new Dimension(200, 200);
                            dataFrame.setMaximumSize(preferredSize);
                            dataFrame.setSize(preferredSize);
                            dataFrame.setVisible(true);
                            break;
                    }
                }
            }
        }
    }

}
```

Figure 11.Assistance presentation implementation.

6. CONCLUSIONS

This paper presented the Sambasore approach which is based on ontologies and software process technologies, and aims to foment the integration of software repositories mining results and also between them and tools that support software engineering.

To illustrate the approach, a proof of concept was developed in partnership with Spider project. In this partnership, the Spider PM tool was experimentally modified to be endowed with two types of assistance: a) the first aimed to present the results of a Google search about modeling phases in software processes, and the other which is more contextual b) intended to show a list of phases (modelled before) that had already been used in conjunction with other similar to the one included in the model.

We believe that this proof of concept was satisfactory. Firstly because it made possible to evaluate most of Sambasore proposed activities, work products and tools. Secondly because it was possible to evaluate the most specific integration component, which implies that we could check integration concepts by using the most difficult to develop integration scenario, and finally, because the proof of concept supported the definition of a software process ontology, which will facilitate assistance integration into software engineering tools and processes.

This viability implies that any Java tool (that supports software engineering), developed with aspect orientation support, can be assisted using this kind of knowledge-based approach. Considering that the other integration scenarios are simpler to implement, we believe that this proof of concept helps in envision broader possibilities to MSR results integration into software engineering practice; it also can help to shift the current scenario where engineers are still taking much of the decisions based on experience to a new scenario, where decision making is based on collective and historical background.

Certainly, however, it is important to note some limitations and restrictions in this research. Although virtually any tool of this nature can be integrated, there are infrastructure requirements to let these tools actually be assisted by assistance engine. In particular, it is worth mentioning the availability of network access - considering that the integration component evaluated in proof of

concept interacts with a web service, there is a prerogative that the workstation should have intranet or internet access.

In a future work, part of a doctoral thesis study, we intend to evaluate some existent software engineering tools and the programming language used to develop them. If we can prove that all of them are made over languages that can read and write files in JSON format, we believe that this can strongly state the canonical integration component, and consequently to state that every software engineering tool can be assisted the way we described.

ACKNOWLEDGEMENTS

The authors would like to thank Spider project team by providing the context where the proof of concept should be performed and analysed.

REFERENCES

[1] Hassan, A.E., (2008) "The road ahead for Mining Software Repositories," Frontiers of Software Maintenance, pp.48-57.

[2] Holmes, R. & Begel, A., (2008) “Deep intellisense: a tool for rehydrating evaporated information”, In Proceedings of the 2008 international working conference on Mining software repositories (MSR '08). ACM, New York, NY, USA, 23-26.

[3] Layman, L. & Nagappan, N. & Guckenheimer, S. & Beehler, J. && Begel, A., (2008) “Mining software effort data: preliminary analysis of visual studio team system data”, In Proceedings of the 2008 international working conference on Mining software repositories (MSR '08). ACM, New York, NY, USA, 23-26.

[4] Ratzinger, J. & Sigmund, T. & Gall, C. H., (2008) “On the relation of refactorings and software defect prediction”, In Proceedings of the 2008 international working conference on Mining software repositories (MSR '08). ACM, New York, NY, USA, 35-38.

[5] Anbalagan, P. & Vouk, M., (2009) “On mining data across software repositories”, In Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR '09). IEEE Computer Society, Washington, DC, USA, 171-174.

[6] Keivanloo, I. & Forbes, C. & Hmood, A. & Erfani, M. & Neal, C. & Peristerakis, G. & Rilling, J., (2012) "A Linked Data platform for mining software repositories", Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on , vol., no., pp.32-35, 2-3 June 2012.

[7] Fayyad, U. & Piatetsky-Shapiro, G. & Smyth, P., (1996) “The KDD process for extracting useful knowledge from volumes of data”, Commun. ACM 39, 11 (November 1996), 27-34.

[8] Cimiano, P. (2010) “Ontology Learning and population from text”, Springer, 2010.

[9] Reis, C. L. (2003) “Uma Abordagem Flexível para Execução de Processos de Software Evolutivos”, Tese de Doutorado - PPGC - UFRGS, Março 2003.

[10] “Spem 2.0 Specification”, http://www.omg.org/spec/SPEM/2.0/.Last visited Aug, 2012.

[11] “Spider PM specification”, http://www.spider.ufpa.br/projetos/spider_pm/Spider-PM.pdf. Last visited Aug, 2012.

[12] “RUP 2002”, http://www.wthreex.com/rup/portugues/index.htm .Last visited Aug, 2012.

[13] “Protége Editor”, http://protege.stanford.edu/ .Last visited Aug, 2012.

[14] “Json Format”, http://www.json.org. Last visited Aug, 2012.

[15] “Spider Project”, http://www.spider.ufpa.br/index.php?id=sobre.Last visited Aug, 2012.

[16] “Spider PM specification”, http://www.spider.ufpa.br/projetos/spider_pm/Spider-PM.pdf. Last visited Aug, 2012.

[17] Agrawal, R. & Imieli, T. & Swami, A., (1993) “Mining association rules between sets of items in large databases”, In Proceedings of the 1993 ACM SIGMOD international conference on Management of data (SIGMOD '93), Peter Buneman and Sushil Jajodia (Eds.). ACM, New York, NY, USA, 207-216.