Knowledge-based interaction in software development

Dimitris Panagiotou, Fotis Paraskevopoulos and Gregoris Mentzas
Information Management Unit, Institute of Communication and Computer Systems, National Technical University of Athens, 9 Iroon Politechniou street, Athens, Greece

Abstract. Modern software development is highly knowledge intensive; it requires that software developers create and share new knowledge during their daily work. However, current software development environments are “syntactic”, i.e. they do not facilitate understanding the semantics of software artefacts and hence cannot fully support the knowledge-driven activities of developers. In this paper we present KnowBench, a knowledge workbench environment which focuses on the software development domain and strives to address these problems. KnowBench aims at providing an intelligent, semantic user interface environment for software developers. Such a tool aims to ease their daily work and facilitate the articulation and visualization of software artefacts, while also support concept-based source code documentation and related problem solving. Building a knowledge base with software artefacts by using the KnowBench system can then be exploited by semantic search engines or P2P metadata infrastructures in order to foster the dissemination of software development knowledge and facilitate cooperation among software developers.

Keywords: Software development, knowledge workbench, semantic annotation, semantic wiki

1. Introduction

Modern software development consists of typical knowledge intensive tasks, in the sense that it requires that software developers create and share new knowledge during their daily work. Although most software developers use modern state of the art tools, they still struggle with the use of technologies that are “syntactic”, i.e. they use tools that do not facilitate the understanding of the concepts of the software artefacts they are managing (e.g. source code).

Flexible ways of solving problems are necessary when a developer is frustrated investigating source code that he has never seen before (i.e. when extending a third party’s software system) and is not capable of understanding its rationale. Additionally, there are many situations that find a developer seeking source code that is already developed by others. He might not be aware of its existence, or even if he is, he is not able to find it effectively [2,13].

Recent trends in data integration and combination have led to the Semantic Web vision (http://www.hpl.hp.com/semweb/sw-vision.htm). The Semantic Web strives to provide integration among information scattered across different locations in a uniform fashion. It defines a “common language” between these information resources.

Additionally, there is a growing interest in the last few years on exploiting Semantic Web technologies in software engineering. Since 2005 the Semantic Web Enabled Software Engineering (SWESE) conference takes place every year with promising results. The Semantic Web Best Practice and Deployment Working Group (SWBPD) in W3C included a Software Engineering Task Force (SETF) to investigate potential benefits of applying semantics in software engineering processes. As noted by SETF [12], advantages of applying Semantic Web technologies to software engineering include reusability and extensibility of data models, improvements in data quality, enhanced discovery, and automated execution of workflows.

In this paper we propose a new system – code-named KnowBench for knowledge workbench – which is the result of exploiting Semantic Web technologies in the
context of a well known software development IDE such as Eclipse (http://www.eclipse.org/), in order to provide the basis for better collaboration in the software development domain and strive against the aforementioned problems. KnowBench aims at providing software developers with an intelligent, semantic user interface environment which aims to ease their daily work and facilitate source code documentation and problem solving.

The rest of the paper is organized as follows. In the next section, we describe what motivated us in developing the KnowBench system. Afterwards, we describe its functionality. Thereafter, we give the results of its preliminary evaluation and we discuss the benefits of using the system. Finally, we conclude the paper indicating our thoughts for future work.

2. Motivation

Modern IDEs such as Eclipse provide means for documenting and annotating source code, e.g., JavaDoc and Java annotations. However, this kind of documentation is unstructured and the means to retrieve it are limited.

In most cases, software developers when in need of specific functionalities find themselves seeking in forums, web sites or web search engines using keyword search for an API capable of delivering these functionalities. Additionally, software developers often use dedicated forums and mailing lists to report on bugs and issues they have with specific source code fragments and have to wait for answers from experts. This switching of environments while developing code is obstructive and requires a great amount of effort in order to maintain scattered resources of software development related knowledge [7].

We argue that a more formal description of the aforementioned knowledge could be beneficiary for the software developers and the means to articulate and visualize this knowledge should be unobtrusive i.e. in the IDE itself in order to provide unified access to it. On top of that, knowledge about source code or software components should be described in a common way in order to avoid ambiguities when this knowledge is retrieved and used by others.

Documentation of software, bug fixing and software related knowledge transfer between employees of a large software house can be facilitated by the use of ontologies to sufficiently capture and describe software artefacts in a coherent way. Furthermore, ontologies are ideal for interlinking software artefacts and the resulting semantic relationships can be exploited in order to maximize productivity when seeking related knowledge of a specific software artefact.

Towards this end we have been motivated to design and implement an environment – namely KnowBench – that offers the opportunity to articulate and visualize software artefacts driven by an ontology backbone. The created artefacts formulating an organization’s software development knowledge base can be exploited by semantic search engines and P2P metadata infrastructures to enable knowledge sharing and reuse in order to boost the collaboration among the employees and speedup time-consuming procedures when facing problems with code or searching for APIs or components to be reused.

3. KnowBench

The KnowBench is designed and implemented in order to assist software development work inside the Eclipse IDE exploiting the power of semantic-based technologies (e.g., manual/semi-automatic semantic annotation, software development semantic wiki, knowledge base graph browser).

KnowBench exposes functionality that can be used for articulating and visualizing formal descriptions of software development related knowledge in a flexible and lightweight manner. This knowledge can be then retrieved and used in a productive manner by integrating the KnowBench with a semantic search engine and a P2P metadata infrastructure such as GridVine [11]. This integration will assist the collaboration and the formation of teams of software developers who can benefit from each others’ knowledge about specific problems or the way to use specific source code while developing software systems. We have already integrated and tested KnowBench with a semantic search engine and GridVine in the context of a European project (name not provided to guarantee anonymity). However, the description of these systems is out of the scope of this paper. A demo of the (name not provided to guarantee anonymity) system is available online – link to be provided if the paper is accepted.

3.1. Ontologies

We have deployed a set of software development ontologies to the KnowBench system in order to describe and capture knowledge related to software artefacts. The system architecture allows for the extension
or even the use of different software development ontologies. The set of the deployed ontologies constitutes of three separate ontologies.

The artefact ontology describes different types of knowledge artefacts such as the structure of the project source code, reusable components, software documentation, knowledge already existing in some tools, etc. These knowledge artefacts typically contain knowledge that is rich in both structural and semantic information. Providing a uniform ontological representation for various knowledge artefacts enables us to utilize semantic information conveyed by these artefacts and to establish their traceability links at the semantic level.

An artefact is an object that conveys or holds usable representations of different types of software knowledge. It is modelled through the top level class Artefact. There are some common properties for all types of artefacts such as:

- `hasID` – which identifies an artefact in a unique manner. Recommended best practice is to identify the artefacts by means of a string conforming to a formal identification system;
- `hasTitle` – which defines a title of an artefact (e.g. class name or document title);
- `hasDescription` – which describes what an artefact is about;
- `createdDate` – which models date of creation of an artefact;
- `usefulness` – which models how frequently an artefact is used as a useful solution.

The problem/solution ontology models the problems occurring during the software development process as well as how these problems can be resolved. This ontology is essential to the KnowBench system, as source code can be documented when a certain problem is met and the respective solution to it is described.

A problem is an obstacle which makes it difficult to achieve a desired goal, objective or purpose. It is modeled through the class Problem and its properties:

- `hasID` – a problem is identified by its identifier;
- `hasTitle` – a summary of a problem;
- `hasDescription` – a textual description of a problem;
- `hasSeverity` – it models how severe the problem is;
- `hasSolution` – every problem asks for a solution. This is the inverse property for the property isRelatedToProblem defined for the class Solution;
- `hasSimilarProblem` – it models similar problems.

A solution is a statement that solves a problem or explains how to solve the problem. It is represented through the Solution class and its properties:

- `hasID` – an unambiguous reference to a solution;
- `isRelatedToProblem` – a solution is defined as the means to solve a problem;
- `hasCreationDate` – date of creation of a solution;
- `hasDescription` – a textual description of a solution;
- `hasPrerequisite` – a solution requires a level of expertise in order to be used;
- `hasComment` – a solution can be commented by the people that used it;
- `suggestsArtefact` – a solution recommends using an artefact in order to resolve a problem the solution is defined for;
- `isAbout` – a solution can be annotated with the domain entities (see below);
- `isRelatedToSEEntity` – a solution can be annotated with the general knowledge about software engineering domain (see below).

The annotation ontology describes general software development terminology as well as domain specific knowledge. This ontology provides a unified vocabulary that ensures unambiguous communication within a heterogeneous community. This vocabulary can be used for the annotation of the artefacts. We distinguish two different types of annotations: (a) domain annotation – software providers in different domains should classify their software using a common vocabulary for each domain. Common vocabulary is important because it makes both the users and providers to talk to each other in the same language and (b) software engineering annotation – general knowledge about the software domain including software development methodologies, design patterns, programming languages, etc.

By supporting different types of annotation it is possible to consider information about several different aspects of the artefacts. We note here that the vocabulary will be used for the annotation of the artefacts, e.g. code, knowledge documents, etc. as well as expertise of the users and solutions of problems. These annotations will be used to establish the links between these different artefacts.

The main concepts of these ontologies are depicted in Fig. 1.

### 3.2. Software development semantic wiki

KnowBench utilizes the DevWiki system which is the successor of SoWiSE [reference not provided to
In order to assist software developers in the articulation and navigation of software development-related knowledge, DevWiki uses a lightweight and flexible editor with auto-completion and popup support. Browsing through knowledge is done like surfing through a conventional wiki using the semantic links between different knowledge artefacts. This browser is available inside the Eclipse IDE so that the software developer does not have to switch to another external browser.

DevWiki provides assistance to software developers in accomplishing their tasks in a more flexible fashion and shortens their total effort in terms of time. In order to achieve this goal, we approach software development documentation and related problem solving by combining lightweight yet powerful wiki technologies with Semantic Web standards.

An overview of the provided functionality is given below:

- Validation of the user input when incorrect values for selected concepts, properties, related instances or property values are introduced. The semantic wiki informs the user about the wrong input value and assists him/her in the way to track down the problem and resolve the issue.

- Auto-complete facilities provide assistance to the user for selecting the right concept, properties and related instances of an object property as a value.

- Intuitiveness – the software developer is able to understand what is required as input using the auto-complete facilities of the Semantic Wiki.

- Syntax colouring – the software developer is assisted with syntax colouring to easily determine what kind of information is recognized by the wiki engine in order to avoid mistyping.

- Multipage editor consisting of the semantic wiki editor and the semantic wiki browser which facilitates navigation of the knowledge base in an HTML-fashion inside the Eclipse environment.

- Navigation through the semantic wiki is enabled via following an instance’s semantic links to other related instances.

- On the fly synchronization with the knowledge base. DevWiki keeps no files anywhere and all information is persisted directly to the knowledge base.

Figure 2 depicts the user interface of DevWiki.
Fig. 2. DevWiki browser and editor.

Fig. 3. Knowledge base graph browser.
3.3. Knowledge base graph browser

The KnowBench facilitates knowledge base browsing using graphs. The KB graph browser (see Fig. 3) provides several advanced features such as zooming in/out, animation, filtering, auto-layout, expansion, collapse of nodes, etc.

An overview of the provided functionality is given below:

- Visualization of concepts, instances and their relationships using graph nodes (different kind of nodes for concepts or instances) and arrows respectively.
- Zooming in/out the whole graph.
- Rotation of the graph by 90, 180, 270 degrees.
- Large/small icons of the graph nodes.
- Smooth floating animation of the graph while dragging around a node to focus on.
- Auto-layout of the graph when it is moved by dragging a node.
- Filtering of all instance or concept nodes.
- Expansion/collapse of concept nodes will show/hide instances that do not have outgoing semantic links to other instances (object property instantiations).
- Legend of the graph nodes (i.e. concepts and instances).
- Concept clusters indicate how much “loaded” is a concept in terms of its instances and in comparison with the overall KB load – this is indicated with a 0 to 10 scale.
- tooltips on every node indicating its name and type (concept or instance).

3.4. Manual semantic annotation of source code

An important aspect of the KnowBench is the ability to annotate semantically source code. We have extended the standard Eclipse JDT editor to add this possibility. The software developer is able to annotate source code with semantic annotation tags that are available or define new tags and extend the used annotation ontology.

The manual semantic annotation of source code provides granularity regarding the respective source code fragment to be annotated (see Fig. 4 for an example). This granularity level is restricted by the underlying Eclipse platform itself as the IJavaElement interface is exploited to map between source code fragments and metadata. This limits the selected source code fragments to be annotated in the nearest Java elements that surround the source code fragment at hand (e.g. package/class/method/variable etc.). Additionally, by using IJavaElement the semantic annotation mechanism is capable of managing annotations of several objects, such as Java packages, classes, methods, etc.

When a manual semantic annotation is added the following metadata are added to the knowledge base automatically:

- **Code entity instance** – the type of the code entity (i.e. the corresponding subclass of the Code entity concept in the ontology) depends on the auto-detected Java element that surrounds the selected source code fragment. “Is about” or “Is related to SE entity” and “Is contained in” object properties are instantiated with respective values. “Is about” refers to a domain annotation while “Is related to SE entity” refers to a software engineering annotation. “Is contained in” refers to the relevant source code file that contains the code entity.

- **Source file instance** – the source file that contains the code entity. “Contains” and “Located in” object properties are instantiated with corresponding values. The “Contains” property is the inverse property of “Is contained in” and indicates that a source file contains code entities. When multiple annotations are added in other code fragments then it is apparent that the source file instance will have as many instances of the “Contains” property as the annotated code entities of the file. “Located in” indicates the physical file resource of the source file.

- **File instance** – the physical file instance that a source file is located.

- **Annotation class** – in the case that the software developer chooses to extend the annotation ontology and add a new concept then a new class is added in the ontology as well as the corresponding “SubclassOf” relation at the desired level of the ontology hierarchy tree.

It is obvious that annotations in the same source code file will not reproduce all the above mentioned metadata if they already exist. For example, if the software developer adds an annotation to a source file, the first time will result in the creation of all the previous metadata and the second time only instances of the Code entity concept and “Is about” or “Is related to SE entity” properties will be created. Additionally, it is possible to annotate the same code entity with multiple annotations – in this case only “Is about” and “Is related to SE entity” properties are instantiated.
3.5. Semi-automatic semantic annotation of source code

This kind of annotation is supervised by the user and results in new concepts and sometimes in their classification inside the annotation ontology hierarchy tree. These new or existing concepts of the annotation ontology are then used to annotate semantically the source code corpora. This procedure assists and shortens the time consuming task of manually annotating source code.

The need for assisting the software developer in annotating semantically source code (either one single file of code or a batch of files residing in a directory) is evident. The notion of semi-automatic comprises a user-controlled process with the system acting as a mediator (proposing semantic annotation concepts/taxonomies of concepts) and the user having the final decision of acceptance or rejection or revision and acceptance of the system’s propositions. Thus, the time consuming task of manually annotating source code can be significantly decreased.

In order to achieve this goal in KnowBench we had to elaborate on ontology learning [9] and information extraction techniques [5]. Ontology learning is needed in order to extract ontology concepts from source code corpora that can be used for annotating it. On the other hand, information extraction is needed in order to instantiate the annotation ontology with individuals (like in manual semantic annotation), either of newly proposed and created or existing ontology concepts. A framework exploiting both technologies is Text2Onto [4] which we adopted in order to implement the desired functionality.

The architecture of Text2Onto is based on the Probabilistic Ontology Model (POM) which stores the results of the different ontology learning algorithms. A POM as used by Text2Onto is a collection of instantiated modelling primitives which are independent of a concrete ontology representation language.

KnowBench takes advantage of two modelling primitives, i.e. concepts and concept inheritance. In the respective modelling primitives KnowBench employs different Text2Onto algorithms in order to calculate the probability for an instantiated modelling primitive. In particular the following measures are calculated:

- Concepts are identified using the following algorithms: Relative Term Frequency (RTF), Term Frequency Inverted Document Frequency (TFIDF), En-
Fig. 5. Semi-automatic semantic annotation editor.

Fig. 6. Semantic annotation concepts.

tropy and the C-value/NC-value method in [6]. For each term, the values of these measures are normalized into the interval $[0..1]$, after taking the average probability of the algorithms output combination which is used as the corresponding probability in the POM.

Regarding concept inheritance, KnowBench combines Text2Onto algorithms such as matching Hearst patterns [8] in the corpus and applying linguistic heuristics mentioned in [14].

The KnowBench semi-automatic semantic annotation editor is depicted in Fig. 5. After processing a source code corpus, the developer can supervise and change the proposed annotations and their position in the concept hierarchy as per his/her needs using this editor. He/she is able to filter the results dynamically by dragging the threshold slider, which means that if a given proposed annotation concept has a lower confidence value than the selected threshold then this is not included in the list, which is refreshed dynamically. The confidence percentage is the calculated probability of a concept to be precise and it reflects the POM probability of a term. Additionally, the developer can rename the proposed concepts and change their position in the concept hierarchy (by choosing their parent concept in the annotation ontology tree). Changing the position of a concept is automatically reflected on the annotation type (Domain or Software Engineering).

4. Usage scenario

Dave is writing an XML editor as a plug-in for Eclipse. He uses the semi-automatic semantic annotation mechanism to annotate the source code. He reviews the suggestions provided by the system (see Fig. 6) and submits them to the system.

He also describes some guidelines in the wiki about “validating and character encoding XML code” as in Fig. 7.

John is a new Eclipse developer and intends to write an XML editor. He is connected to the P2P network and performs a structured query to find knowledge or source code that is related to “XML editor” as in Fig. 8.

He retrieves the results as in Fig. 9 and now he is able to reuse existing source code and take into consideration the guidelines documented by Dave.

5. Evaluation

We have conducted evaluation of KnowBench in small teams of software developers in a total of 9
persons in the following organizations: (1) a multinational Brussels-based company specializing in the field of Information and Communication Technology (ICT) services (Intrasoft International S.A. – 2 developers), (2) a leading Hungarian association dealing with open source software at corporate level (Linux Industrial Association – 2 developers), (3) an Italian company which operates in the Information Technology market, focusing on business applications (TXT e-Solutions – 2 developers) and (4) the corporate research laboratory of
the Thales group – a global electronics company serving Aerospace, Defence, and Information Technology markets worldwide (Thales Research & Technology – 3 developers).

5.1. Method

Scenario-based evaluation is drawn from the well-known technique of scenario-based design [3,10]. Briefly, a scenario describes sequences of actions taken by a user with a specific goal in mind. Scenarios are paper-based descriptions of key processes. Comments on these by users tell designers:

– what users want from a system;
– how they intend to use it;
– how the system could be structured to facilitate that use.

It establishes a structured dialogue with representatives of user groups, to enable the developers to use this data as a source for formative design. According to this method, end-users will write scenarios, based on how they would use the system according to a specific user role, and for a specific task. For the application of this method in the evaluation of the KnowBench system we adopted a slightly different configuration to capture more aspects of the system in a more structured way. Our approach also involved end users but instead of writing scenarios they had to perform specific tasks using the system and then to document their experience using predefined questionnaires (see Table 1 for an example). The process took place using an online collaborative platform (CENTRA – http://www.korimvos.gr/en/products/centra.htm).

5.2. Execution

The execution of the scenario-based evaluation involved a considerable effort in preparing the evaluation material, installing the KnowBench system, coaching the end-users, and organizing and performing the sessions with the end users.

The adopted process had four distinct steps: (a) Plan: identification of user classes of the KnowBench system for the user sites (i.e. developer, knowledge provider, manager, etc.) and selection and recruitment of user participants according to the user classes, (b) Conduct evaluation session: briefing and training (if necessary) required to undertake the task (i.e. in the aims/functions of the system) and feedback elicitation using questionnaires, (c) Analyze and document results: consolidation and analysis of scenario-based evaluation results and (d) Feedback results to designers: report for the KnowBench designers who in turn report back to evaluators with adjustments of the system.

5.3. Example of a scenario-based questionnaire

The purpose of this questionnaire (see Table 1) was to understand how developers perceived several tasks while using DevWiki and how smooth is the interaction between them and the KnowBench system.

5.4. Results

Overall, the KnowBench system has been well appreciated by the software developers. It is a simple, user-friendly and intuitive system aiming to improve their productivity. In that respect, the provided functionalities seem to integrate well with the Eclipse IDE without encumbering the developer with time-consuming functions. In fact, these functionalities demonstrate a good value proposition for the developers’ work.

The initial effort needed to start working with the system is minimal; nevertheless, it seems that some of its functions convey deeper logic which is not immediately capturable by the end users. The graphical user interface could be improved further in order to reveal this logic, while some help functions could also be of great value on this. Users were also satisfied by the level of “contextualization” offered by the system (meaning helping the developer know where he/she is at every moment). Finally, the performance of the KnowBench
Table 1
Scenario-based questionnaire example

<table>
<thead>
<tr>
<th>Question</th>
<th>Percentage of affirmative answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can the developer easily understand how he/she can create an annotation?</td>
<td>100% (9/9)</td>
</tr>
<tr>
<td>Does the KnowBench system provide the developer with the friendly and easy-to use way for creating annotations?</td>
<td>89% (8/9)</td>
</tr>
<tr>
<td>Can the developer easily find a term for annotation?</td>
<td>78% (7/9)</td>
</tr>
<tr>
<td>Can the developer easily understand what to do in order to create a new wiki page?</td>
<td>100% (9/9)</td>
</tr>
<tr>
<td>Is the wiki syntax clear and easy to follow?</td>
<td>78% (7/9)</td>
</tr>
<tr>
<td>Can the developer easily locate a property he/she wants to instantiate using the wiki support?</td>
<td>100% (9/9)</td>
</tr>
<tr>
<td>Can the developer easily locate an individual he/she wants to associate with the currently-created wiki page?</td>
<td>78% (7/9)</td>
</tr>
<tr>
<td>Can the developer easily understand what is required as input data?</td>
<td>78% (7/9)</td>
</tr>
<tr>
<td>Does the DevWiki system provides the developer with the friendly and easy-to use pop-up menus for creating wiki pages?</td>
<td>89% (8/9)</td>
</tr>
<tr>
<td>Can the developer easily understand what to do in order to create a new wiki page?</td>
<td>100% (9/9)</td>
</tr>
<tr>
<td>Is the wiki syntax clear and easy to follow?</td>
<td>78% (7/9)</td>
</tr>
<tr>
<td>Can the developer easily locate a property he/she wants to instantiate using the wiki support?</td>
<td>100% (9/9)</td>
</tr>
<tr>
<td>Can the developer easily locate an individual he/she wants to associate with the currently-created wiki page?</td>
<td>78% (7/9)</td>
</tr>
<tr>
<td>Can the developer easily understand what is required as input data?</td>
<td>78% (7/9)</td>
</tr>
<tr>
<td>Does the Graphical User Interface (GUI) behave as the developers expect?</td>
<td>78% (7/9)</td>
</tr>
<tr>
<td>Can the developer easily understand the messages displayed by the KnowBench system (e.g. error messages)?</td>
<td>55% (5/9)</td>
</tr>
<tr>
<td>Does the developer know where he is at all times, how he got there, and how to get back to the point from which he started?</td>
<td>78% (7/9)</td>
</tr>
<tr>
<td>Can the developer easily recover his/her error and retry a task?</td>
<td>55% (5/9)</td>
</tr>
<tr>
<td>Is the response time of the KnowBench system acceptable?</td>
<td>100% (9/9)</td>
</tr>
</tbody>
</table>

system i.e. response time left the developers with very good impressions.

More specifically, annotations are a powerful tool in managing knowledge and the KnowBench implementation seems to be very efficient. Developers were rather comfortable with the use of annotations and could use them in an intuitive manner i.e. name annotations freely. Additionally, semi-automatically creating annotations was even more appreciated and greatly assisted them in speeding up the error-prone and cumbersome annotation process. However, they did not seem to understand their real value and thus their deeper connection with the inner logic of the system.

The developers showed great comfort in using DevWiki. Instantiation takes place quickly and the types of information that are needed for the creation of a wiki page are clear, while interactive mechanisms i.e. pop-ups make the creation a very simple task.

6. Contribution

KnowBench provides an intelligent, semantic user interface environment for software developers. It offers an easy to use environment to facilitate knowledge articulation pertinent to software development as well as it provides means to annotate manually or semi-automatically this kind of knowledge in order to foster easier knowledge acquisition and sharing when the tool is integrated with a semantic search engine and a P2P metadata infrastructure. The semi-automatic annotation mechanism provides a way to classify source code in a quick manner, thus facilitating code reuse when a developer searches for code that is related to a software engineering or domain concept. For example, the developer might be interested in finding code that is written in Java for banking applications. Furthermore, KnowBench provides intuitive visualization and navigation through this knowledge exploiting the wiki paradigm and the graph metaphor. Finally, all functionality is integrated into the Eclipse environment which is convenient for the software developer as he/she does not have to switch environment in order to articulate or use knowledge.

7. Discussion

There are emerging efforts in research towards exploiting Semantic Web technologies in software engineering. For example, SemiIDE [2] is a framework which is based on meta-models in order to describe semantically software engineering processes. OSEE [13] is a tool which aids the construction of a knowledge base which captures software related artefacts. SeRiDA [1] combines object-oriented programming and relational databases, with semantic models.
With KnowBench integrated into an IDE, it is easier for software developers to create new knowledge. The advantage of recording the solution to difficult or frequently asked questions/problems is that developers are recording their explicit knowledge about particular problems, as well as tacit knowledge that they have internalized, while creating software development knowledge artefacts that can be organized, managed, and reused.

Wikis can significantly help developers fill this knowledge management need. KnowBench provides this kind of wiki-based support for knowledge provision. DevWiki offers innovative features especially in software development but is also a contemporary semantic wiki in the general sense since it provides many features common in most of the state-of-the-art semantic wikis [reference not provided to guarantee anonymity].

KnowBench is seamlessly integrated into the Eclipse environment and thus gives software developers the incentives to develop or explore knowledge without bothering to open an external tool or web browser to accomplish this task which has the advantage of leaving the user in his/her current work context and not distracting him/her from the rest of his/her tasks. For example, he/she can develop source code and at the same time document problems that he/she encounters or provide guidelines and workarounds for a specific problem in source code that is commonly known to his/her organization by simply choosing to open the DevWiki editor next to the source code editor.

8. Conclusion

In this paper we presented the KnowBench system, a knowledge workbench which is integrated in the Eclipse IDE based on Semantic Web technologies, in order to assist software developers in articulating and visualizing software development knowledge. The resulting knowledge base can be exploited by semantic search engines or P2P metadata infrastructures in order to foster better and more flexible collaboration among software developers scattered across the globe.

We aim at providing support to the software developer for documenting code, finding the best solution to a problem or the way to move forward in critical decisions in order to save time during the heavy task of software development. Thus, in KnowBench we provide an easy to use semantic wiki.

Since manual creating knowledge may be very time-consuming and error-prone activity, the KnowBench system takes advantage of ontology learning and information extraction techniques to propose annotations. Both manual and semi-automatic semantic annotation of source code is facilitated by automated generation of corresponding metadata that assist also in describing and mapping Eclipse resources to the knowledge base.

We are planning to enhance the semi-automatic semantic annotation mechanism in order to derive more precise annotations by introducing JAPE grammar rules [5] of the GATE system that will be customized for source code. Furthermore, we want to further extend the manual semantic annotation in order to add the capability of annotating arbitrary lines of code regardless of the surrounding Java elements. However, this is a hard to solve and non-trivial issue.

References
