Exploiting Semantics in Collaborative Software Development Tasks

Dimitris PANAGIOTOU a,1, Gregoris MENTZAS a

a School of Electrical and Computer Engineering, National Technical University of 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 artifacts and hence cannot fully support the knowledge-driven activities of developers. In this paper we present SoWiSE, a Semantic Wiki environment which focuses on the software development domain and strives to address these problems. SoWiSE aims at providing software developers such a tool to ease their daily work and facilitate cooperation, context-aware navigation, ontology-based browsing and searching, concept-based source code documentation and related problem solving.

Keywords. Semantic Wikis, Software Development, Semantic Web, Collaborative Software Development

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 artifacts they are managing (e.g. code fragments). Moreover since software development is a highly collaborative task, developers are in need of simple and easy-to-use tools that also enable collaborative work.

Collaboration and 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. On the other hand, 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.

Most common practices used to resolve these problems include their discussion with other developers, finding relevant information on the Web, interacting with a dedicated forum, etc. But this kind of resolution is not unobtrusive and implies waste of...
valuable time. Sequentially, more adequate and unified means for working around these issues are required.

Wikis offer a fertile ground for investing on software development collaboration. These systems are gaining popularity and they provide flexible ways for articulating and sharing information among their users. But they suffer from information overload as pages written in a Wiki are unstructured and hard to retrieve at a later point of time.

On the other hand, recent trends in data integration and combination have led to the Semantic Web vision\(^2\). 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.

In this paper we propose a new system – code-named SoWiSE for Semantic Wiki for Software Development – which is the result of exploiting the linkage of the Wiki paradigm and Semantic Web technologies in order to provide means for better collaboration in the software development domain and strive against the aforementioned problems. SoWiSE aims at providing software developers such a tool to ease their daily work and facilitate cooperation, 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 SoWiSE system. Afterwards, we describe the SoWiSE system in detail and then give the results of its experimental evaluation. Thereafter, we refer to related work performed in the field of our interest and state our contribution to it. Finally, we conclude the paper indicating our thoughts for future work.

1. Motivation

Herbsleb and Grinter in [1] identify that under time pressure to build a software system, the developers proceeded with coding, and slowly the code diverged from design. However other people were still relying on that documentation to design their own components and test suites. Over time, and especially in integration, all these inconsistencies came to light. Testers pointed to documentation as demonstration of why code was failing certain tests, other developers pointed to documents that described behavior that they had assumed was still exhibited by the code. All of these inconsistencies had to be resolved as part of building a working product.

Additionally, Herbsleb et al. in [2] argue that collaborations over distance must contend with the loss of the rich, subtle interactions that co-located teams use to coordinate their work. Previous research has suggested that one consequence of this loss is that cross-site work will take longer than comparable single-site work. In contrast to the rich interaction of face-to-face work, there is very convincing evidence that the frequency of communication drops off sharply with physical separation among co-workers’ offices, and that the sphere of frequent communication is surprisingly small. Tom Allen [3], in a study of engineering organizations, reported that the frequency of communication among engineers decreased with distance. Further, he noted that when engineers’ offices were about 30 meters or more apart, the frequency of communication dropped to nearly the same low level as people with offices separated by many miles.

\(^2\) http://www.hpl.hp.com/semweb/sw-vision.htm
On the other hand these collaboration difficulties can be tackled using the Wiki paradigm. The success and great evolution of Wikis were based on their ease of use and significant flexibility they provided in supporting collaboration across a team of users or experts. Common usage of Wikis included project communication, intranets and documentation in enterprise settings. Nowadays, they are even more widely used by simple internet users who just need means to collaborate or exchange opinions with each other. A representative example of the “Wiki evolution” is Wikipedia.

Although Wikipedia’s success is unambiguous it still has some drawbacks. A large side effect of such a large Wiki is that the user can be frustrated when searching for information. Since the largest percentage of the information in a Wiki is textual, the only way to find information is through a keyword-based search. This side effect is the result of unstructured accumulated information, and hence, it poses problems for knowledge management and productivity. The only semantics of Wiki pages lies in the links between pages. Most Wiki engines generate navigational structures from these links: one can see all pages linking to the current one, and go to these related pages. But this navigation through related pages is quite limited, and does not address the need for more advanced information retrieval [4].

The strong presence and yet promising research area of the Semantic Web is exactly the driving wheel behind our motivation in exploiting and applying these innovative technologies in the software development domain in order to boost a rather absent Semantic Wiki playground. The development of the SoWiSE system aims at supporting software developers when they collaborate in order to achieve their tasks in a more productive way.

2. SoWiSE

As mentioned above, the purpose of our work is to provide assistance to software developers in accomplishing their tasks in a more flexible fashion and shorten 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.

We chose Wikis for their significant support in knowledge articulation by providing lightweight and flexible mechanisms as well as semantics in order to add structured formulation to the Wiki contents. This combination gives software developers the opportunity to capture their knowledge related with a software development artifact exploiting a structured formalism. This knowledge will be then reusable, more easily shared and retrieved in the future by the Semantic Wiki’s resilient user interface.

Finally, we decided to integrate SoWiSE seamlessly into the Eclipse IDE and deploy it as a plug-in for two reasons. The first reason is that we wanted to provide a proof of concept and chose Eclipse as a widely used and well known IDE which has also one of the broadest communities in the software development domain. Secondly, it was our intention not to change the working fashion of the software developer but instead to provide a tool environment that he is already familiar with and does not influence the way he is performing his tasks.

2.1. Software development ontologies

In order to ensure SoWiSE’s effectiveness we have used it with a set of software development ontologies that were developed in the context of TEAM [5], which also constitute the initial set of SoWiSE’s core ontologies. These ontologies can be extended by the SoWiSE system adding new concepts (classes). It is not intended to let the developer fully extend the ontology by defining relations and new properties as this would require basic knowledge and understanding of ontology engineering. However, the developer is free to extend the ontologies using any standard ontology editor such as Protégé [6].

These ontologies [7] combine content, interaction and context information to provide better support for knowledge sharing and reuse. They describe the structure and content of knowledge artifacts, their usage and organization and provide a basis for determining how artifacts are related. Different layers model different types of information: (a) the context layer describes people, various roles they assume within the organization, the projects they are involved in, teams performing the tasks, etc., (b) the interaction layer describes the structure of interaction (i.e. communication and collaboration) around bug reports, component reuse, discussion etc. and (c) the content layer determines the location, structure, content of software objects and related problem reports as well as the solution for problems.

The content ontologies used in SoWiSE are of crucial importance due to the fact that these are mainly the way to represent software development documents (e.g. a Java source code file). The content layer is further sub-divided into several ontologies, since they are sufficiently different domains, each meriting its own ontology for clarity. Each of these ontologies is relatively modular and is consequently usable for the most part independently of each other. The content layer consists of the following ontologies: (a) the software artifact ontology that describes different types of software artifacts such as the structure of the project source code, the components to be reused and software documentation, (b) the problem/solution ontology that describes the problems that may arise during coding (e.g. submitted bug and patch reports and their attributes) and how they are resolved, (c) the lifecycle ontology that describes decision made during programming or resolving problems and (d) the vocabulary that describes general software development terminology.

2.2. Architecture

Figure 1 depicts a simplified version of the Eclipse SDK architecture [8] and how the SoWiSE system is built on top of the EclipseWiki plug-in⁴ using the Eclipse Plug-in Development Environment (PDE) [8].

An Eclipse plug-in is the smallest atomic unit of the Eclipse Platform [8]. It is a common practice for software developers to build separate plug-ins and compose more complex tools out of them. All of the Eclipse Platform’s functionality is based on plug-ins apart from a small kernel known as the Platform Runtime. Each plug-in declares several named extension points and optionally extends one or more plug-ins using in

⁴ http://sourceforge.net/projects/eclipsewiki/
Based on this concept, we built the SoWiSE system exploiting well known and tested functionality of the Eclipse Platform such as SWT widgets, JFace, core runtime operations etc. On top of that, we needed some basic conventional Wiki functionality (non-semantic Wiki) in order to concentrate on building the semantic aspects of the Wiki and customize it for our specific domain: software development. EclipseWiki provides such basic functionality and it is available as an open-source Eclipse plug-in download.

The SoWiSE editor is developed as an Eclipse plug-in providing seamless integration with the IDE and is based on the Rich Client Platform (RCP) [8]. It takes advantage of out-of-the-box functionality of SWT widgets and JFace components for its GUI namely toolbars, buttons, user dialogs, text editors and browsers.

Additionally, it extends the basic functionality of EclipseWiki and provides more sophisticated means for creating and manipulating Wiki pages. During the effort of customizing EclipseWiki for our purposes, many pieces of code of the original EclipseWiki plug-in were modified. For instance, new color coding schemas for Java classes were defined and implemented by changing the existing code of EclipseWiki (Figure 2b). On the contrary, extending EclipseWiki required new Java classes to be implemented – the new action toolbar and Wiki page statistics which are realized as an individual Eclipse view is such an example (Figure 2c).

2.3. Features and usage

EclipseWiki already provided us among other features the following: (1) linking to local Wiki documents, Eclipse workspace resources, Java source code and urls, (2) definition of common prefixes to websites, Wikis and Eclipse resources using a local WikiSpace, (3) export of the user’s Wiki, (4) multipage editor with source, browser (TWiki4, SnipSnap7 and Ward's Wiki8 renderers) and syntax views, (5) headers and footers in the browser view and exported documents, (6) support for embedding other

---

5 SoWiSE can manipulate source code of whatever programming language, but it is currently optimized for use with Java.
6 http://twiki.org/
7 http://snipsnap.org/
8 http://c2.com/cgi/wiki
Wiki documents, eclipse resources and Java code in Wiki documents, (7) syntax coloring for links and (8) outline of document headings.

Some of the above features were customized and new ones were realized. The following describes customized and new supported features by SoWiSE as well as their usage.

2.3.1. Linking

Apart from EclipseWiki’s capabilities for linking, SoWiSE provides statistical information for each Wiki page displaying “existing links” and “unknown links”. Existing links represent how many Wiki pages were already created “automatically” while unknown links represent how many Wiki pages could be created automatically. Automatically, in this context means that SoWiSE recognizes words which match with a specific pattern and gives the possibility to the user of creating a new Wiki page (by clicking on a blue question mark, Figure 2b) and a link to it simultaneously.

2.3.2. Multipage editor

We have currently customized EclipseWiki’s Ward’s Wiki renderer tweaking mainly its appearance. This renderer is the default browser for SoWiSE. The browser’s action toolbar was redesigned providing a clearer and more elegant interaction point with the user. Different font styles were defined for rendering different parts of the Wiki page depending on the type of the rendered source code (e.g. import statements, comments, etc.). We are planning to add support for TWiki and SnipSnap syntaxes customizing the respective renderers of EclipseWiki.

The source editor of EclipseWiki is an extension of a standard Eclipse text editor which provides a complete set of common text editing functionalities. In this editor the user is able to compose a Wiki page usually by embedding a Java class file, documenting it using the Wiki’s syntax, attaching related resources and by tagging (see below) important terms that are classified in an ontology schema. In order to support tagging and attachments, customization of the original source editor of EclipseWiki was required.

On top of that, the customized editor is seamlessly integrated into the Eclipse IDE (being part of an Eclipse plug-in as mentioned above) and thus provides support for features such as: management of Wiki page files using standard Eclipse navigation views and menus (save, open etc.), advanced text find/replace mechanisms, cut/copy/paste facilities and so on. Finally, the multipage editor has a syntax view where the user can have an overview of the Wiki’s syntax rules.

2.3.3. Search

Search in SoWiSE is twofold. The user is able to search within a Wiki page using the typical search mechanism of Eclipse mentioned above, as well as to perform a search across the whole local WikiSpace. The search button in the action toolbar serves exactly this last option. It enables the user to perform an ontology-based query and find Wiki pages that are tagged with a specific term given by the user. The term is specified either as keyword(s) that the user remembers and is capable of filling out or by navigation in the ontology tree of the local WikiSpace.
An important aspect of the search mechanism is the use of inference. SoWiSE takes advantage of knowledge that is inferred by applying rules specified by the user. For example, if the user declares that an individual software developer is an expert in SWT, when the user actually searches for information related to SWT he is presented with all Wiki pages referring to SWT as well as related resources that link him to the expert.

2.3.4. Tagging

Tagged terms are easily identified in the source or browser views by a red underline symbolism. When the user hovers the mouse above the underlined term then a small popup dialog appears displaying the corresponding ontology concept and a small part of the ontology tree nearby that concept. Tagging is feasible only by highlighting a term in the source view of SoWiSE and pressing the “Tag code” button. The number of the tagged terms in a Wiki page is displayed in the Wiki page statistics part of the SoWiSE plug-in. Removal of a tag is accomplished by positioning the cursor over a tagged term and clicking on the “Remove tag” button or alternatively using SoWiSE’s context menu.

2.3.5. Attachments

The attachment mechanism of SoWiSE is used to relate a specific excerpt of a Wiki page with an information resource. The actual document is not attached persistently on the Wiki page but rather referenced by it. For example, this facilitates a developer
when looking at a piece of code in order to be able to find further related information and help about the usage or usefulness of that specific segment. Related resources could be of any electronic format: email messages, web resources, office documents, images, videos etc. At the same time, we are even exploring whether it is feasible to realize a model for associating “physical” resources such as a book with these Wiki page excerpts.

Performing attaching and detaching of resources using the corresponding buttons of the user interface follows the same approach of the tag/remove tag buttons. The number of attachments of the Wiki page is displayed in the statistics as well. Attached resources are identified in the Wiki page by clip symbols which are in fact active links to those resources. Similarly, as in the case of tagged terms, the excerpts that have one or more attached resources are surrounded by red parenthesis.

3. Experimental Evaluation

We have not performed a comprehensive assessment yet, but we have deployed SoWiSE in small teams of software developers consisting of four to six persons in the following organizations: (1) a Brussels based company specializing in the field of Information and Communication Technology (ICT) services (Intrasoft International S.A.), (2) a leading hungarian association dealing with open source software at corporate level (Linux Industrial Association) and (3) an italian company which operates in the Information Technology market, focusing on business applications (TXT e-Solutions). The feedback that we have received so far is positive and the fact that almost every software developer considers SoWiSE as an important tool in assisting him in his tasks is very promising.

More specifically, all of the users agree that SoWiSE’s GUI is easy to use and straightforward. They are satisfied by the fact that it is seamlessly integrated into the Eclipse IDE and they do not have to launch an external tool or change the way they work in their development environment. The convenient toolbar and the easiness of adding annotations to the source code are also well accepted by most of the users. Additionally, the way for finding relevant information or possible solutions to problems through the Wiki using the search facility of SoWiSE is more than welcome.

On the other hand, a significant percentage of the users would desire a more flexible way for tagging source code, namely semi-automatic generation of annotations in order to reduce the burden of manually created annotations. Thus, their role of being human annotators would change to supervisors saving valuable time. Another important issue according to the users is navigation. They consider that alternative means for navigation should be provided, including more intuitive representation of the Wiki’s contents such as faceted browsing, different views over the same Wiki page, etc.

4. Related Work

Besides general purpose Semantic Wikis, some domain specific implementations exist. In our context we have found out two implementations in a relative and broadest domain than software development: software engineering. Following is a brief description of these systems.
RISE – Reuse in Software Engineering [9]: The RISE project introduces the Riki Wiki combined with the Wikitology paradigm [10, 11]. With Wikitology, it is possible to derive semi-automatically an ontology needed for information retrieval from the pages in the Riki itself. Such a Wikitology automatically updates “itself”, i.e., it reflects the changes of contained knowledge, changed views of the users accounts, new projects, customers, rules, trends, and ideas inside the Wiki. By considering the Riki as the ontology, the ontology will be always collectively edited up-to-date and will automatically reflect all changes.

Hyena: Hyena [12] aims at providing a platform that makes RDF’s advantages available to software engineering without diverging too far from the concepts and tools that are familiar in that domain. It provides editing, GUI and publishing services for implementing editors of RDF-encoded custom models and it is implemented as an Eclipse plug-in. It includes some built-in ontologies for many basic software engineering tasks. These ontologies have themselves become part of the platform, because they provide services that will be useful for other ontologies.

5. Our Contribution

SoWiSE contributes in the Semantic Wiki sphere mainly at the software development domain rather than just providing another Semantic Wiki alternative. We feel that in this specific domain the software developer is confronting a lack of tools and options in supporting him to accomplish his task. On the other hand, software developers need not only tools, but in order for them to be helpful and successful these must be convenient and require little training effort. Hyena, for example, is too much complicated even for software developers providing a few sophisticated ontology editors and forcing the user to acquire the basic knowledge of an ontology engineer.

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. Of course, as described above this support should require minimum effort for learning SoWiSE which should also be integrated seamlessly into the software developer’s environment. For these reasons, we chose to develop SoWiSE as an Eclipse plug-in as a proof of concept and to integrate it seamlessly to the Eclipse IDE. On top of that, we designed and implemented a rather familiar and easy to use GUI in order to give the opportunity to software developers of just learning SoWiSE’s capabilities and not wasting time on learning its user interface.

Another important contribution of SoWiSE is the fact that it enables tagging of specific fragments of a Wiki page which is an immediate issue in current research. Most existing Semantic Wikis take the simple approach of having one Wiki page per concept. But this is not convenient for our specific domain, as in software development there is a clear discrimination between different fragments of a specific source code file which is documented as a Wiki page, for example. Thus, it is apparent that this simplification does not hold true in our case, having in mind that the software developer needs to tag different source code fragments in the same Wiki page. As Kiesel notes in [13] “This simplification makes it almost as easy to express semantic statements about a concept as it is to make a link to a Wiki page. However, this simplified model puts a restriction on the level of granularity of possible annotations: if
you can only refer to “the concept of the page”, how do you refer to a specific fragment of the page?”

6. Conclusion and Further Work

In this paper we presented SoWiSE, a Semantic Wiki for assisting software developers to cooperate and accomplish their tasks in a more flexible manner. The lightweight approach of the Wiki paradigm for documentation and problem solving is further facilitated by using Semantic Web technologies. We have evaluated the system in three large organizations and the resulting feedback is positive – a fact that motivates us to further extend the system.

We plan to extend the system by providing even more intuitive facilities for articulating, navigating and retrieving the Wiki's contents. A more formal and comprehensive assessment is planned and the results will have impact on the further development of SoWiSE.

Acknowledgements

This work was partly supported by the TEAM project, which is funded by the EU-IST program under grant FP6-35111. The authors are responsible for the content of this publication.

References