Incremental Knowledge Acquisition in Software Development Using a Weakly-Typed Wiki

Filipe F. Correia
Faculdade de Engenharia,
Universidade do Porto
Rua Dr. Roberto Frias, s/n
filipe.correia@fe.up.pt

Hugo S. Ferreira
INESC Porto, DEI
Faculdade de Engenharia,
Universidade do Porto
Rua Dr. Roberto Frias, s/n
hugo.sereno@fe.up.pt

Nuno Flores
Faculdade de Engenharia,
Universidade do Porto
Rua Dr. Roberto Frias, s/n
nuno.flores@fe.up.pt

Ademar Aguiar
INESC Porto, DEI
Faculdade de Engenharia,
Universidade do Porto
Rua Dr. Roberto Frias, s/n
ademar.aguiar@fe.up.pt

ABSTRACT

Software development is a knowledge-intensive activity and frequently implies a progressive crystallization of knowledge, towards programming language statements. Although wikis have proved very effective, for both collaborative authoring and knowledge management, it would be useful for knowledge acquisition to better support team awareness and the recognition of knowledge structures, their relations, and their incremental evolution. This paper presents Weaki, a wiki prototype especially designed to support incremental formalization of structured contents that uses weakly-typed pages and type evolution. Weaki was applied in academic settings, by students of Software Engineering Labs.

Keywords

Semantic Wikis, Knowledge Acquisition, Software Engineering

1. INTRODUCTION

Software development is a knowledge-intensive activity and frequently implies a progressive crystallization of knowledge with a very focused goal: to obtain programming language statements to be executed by a computer [3].

Since most of the development effort is spent on formalizing knowledge, i.e., on gathering unstructured information from written documents and verbal conversations, and converting them to concrete code and models, it is important to effectively support such formalization process.

Software knowledge is typically captured in separate interrelated artifacts of different types, such as text, source code, models, and images, among others, which are often produced cooperatively, by different kinds of people, and at different moments of the software life cycle [1]. To cope with this complexity, software engineers use techniques and tools, that support the processing of knowledge required to obtain and evolve a computer program [3].

Due to their simplicity and effectiveness, for both collaborative authoring and knowledge management, wikis are nowadays extremely popular and massively used by software developers. *Weaki* aims at augmenting these benefits by allowing the incremental formalization of structured contents and thus improving awareness with weakly-typed pages and type evolution.

2. WEAKI: A WEAKLY-TYPED WIKI

Software development often requires a lot of team work. Nevertheless, knowledge about existing collaboration rarely goes beyond the content of the authored artifacts themselves. At best, a member knows who edited a certain artifact for the last time, but not who is the most proficient member on that artifact, or who are the members for which that artifact is important to. This "meta-knowledge" is specially relevant for new members, or members delving into artifacts for the first time. Eliciting these implicit relations is important to raise team awareness.

Software documentation, a primary source of knowledge in a software project, is frequently supported by wikis. Semantic Wikis can provide structured information, thus laying the grounds to achieve higher awareness. However, this frequently implies a trade-off between the benefits of structured information and some of the design principles that make the essence of what wikis are [2].

2.1 Key Design Principles

Weaki allows structured contents in a way that is non-intrusive for authors, with the goal of leveraging awareness and knowledge acquisition. Structure provided by traditional wikis is not enough if computers are to automatically process this information, and approaches taken by semantic wikis frequently demand an additional effort from authors. *Weaki* strikes a balance between informal and formal information, by taking advantage of the following concepts, some of them inspired in the design of programming languages.

Typed pages. A type in *Weaki* is equivalent to a class in object-oriented programming. As in most semantic wikis, pages have a type, which is used to formalize its structure.

Inheritance. Similar to the concept by the same name in object-oriented programming, inheritance is a way of defining classes (i.e. types) that are specializations of other classes. In the context of *Weaki*, it allows reuse of structure and a way to express *is-a* relations.

Weak-typing. *Weaki* allows the type of a page to change. A parallel can be made to weakly-typed programming languages, which allow types to be coerced to other types, that is, to switch the type of a given instance without actually converting the underlying contents.

Dynamic typing. Structure and contents of a given wiki page are authored independently, having the page's type as the single point of contact between them. This binding, between structure and contents, is made dynamically at runtime taking advantage of the existing document-oriented structure of the wiki page.

Everything is a page. Authors may create and edit types through the same mechanism used to create and edit contents, as both types and contents are authored as regular wiki pages. Some programming languages have also been designed to provide a uniform view over types and instances, namely, some object-oriented languages which follow the principle of "everything is an object".

These principles extend those of traditional wikis [2], introducing structure while maintaining the same base philosophy. The following design principles of wikis are frequently affected when additional structure is introduced, but they are specially taking into account in *Weaki*.

Organic. Both structure and content are open to be edited and evolved.

Mundane. Authoring content and structure is based on mostly the same set of text conventions of a traditional wiki. This means authors won't be confronted with structure-specific conventions, unless they choose so.

Overt. Although more elaborate views are also provided, the default view of a wiki page directly reflects the input required to produce it.

Incremental. *Weaki* takes this principle to both types and the relation between them and their pages.

Universal. Typing pages and defining type's structure can be seen as forms of organizing contents. *Weaki* allows to create the structure of a type using the same mechanism used to create regular pages.

Tolerant. Even if not strictly complying to its type, a page can be fully rendered, and the parts that do comply can be interpreted.

2.2 Features

The most important features of Weaki are detailed below.

Structure Emergence. While editing pages, recurrent common structures will emerge. These structures can at any time be captured as reusable Types.

Homogeneity. Types are wiki pages. All known wiki metaphors are applicable.

Scaffolding. Whenever someone wants to create a new page of a particular Type, the wiki automatically fills it with an initial skeleton, derived from that Type's structure.

Structured Views. Structured viewing filters out content not compliant to the page's Type. This provides a consistent view of every page of the same Type.

Content Assist. The creation of new content is assisted with context-aware suggestions, while editing a typed page.

Global Time Labels. Labelling a moment in time. View the entire wiki state at that moment.

Type Awareness. Types evolve. Pages evolve. Their level of compliance varies. Awareness of this metric allows balancing evolution vs. type adequacy.

Team Awareness. The "neighbourhood" consists of wiki contributors (authors, editors, even readers) that "inhabits" the same pages as you. Being aware of who they are, nurtures constructive "conversations".

3. CONCLUSION

Two kinds of benefit are achieved from applying the above design principles and features.

3.1 Incremental formalization

To declare the type of a wiki page is to classify it. The author may do so on the moment the page is created and, if the type in question is a new one, she may even concretely define it's structure at that time. However, it is frequently unrealistic to commit to a structure in an early stage of content authoring, as the required knowledge may only be acquired while the page's contents are written. Even when the author is lead to define a structure before creating the contents, the evolution of the first is usually still driven by the evolution of the later.

Weaki takes this factor into account, by letting authors focus solely on the contents. This supports experimentation and creativity, as it frees authors from the extra overhead of creating the structure. An author may at any time settle for a given approach to how contents are organized, and may then formalize their structure. Furthermore, having the option of disregarding structure while there is no immediate value in adding it, lowers the barrier to contribute to the knowledge base, specially for non-technical authors. Any author may add the structure when it reveals itself useful. Hence, the creation of structure is bottom-up, it emerges from the contents, instead of being defined *a priori*; it may be said to be Emergent Structure.

3.2 Awareness

Weaki improves awareness on a two-level basis:

Awareness through structure. The system can infer new relationships between several kinds of contents and recommends a possible structure. When a user is adding new content, the wiki analyses both the context where that content is being inserted and the structure of the content itself. It then evaluates the compliance between the content and the available types for that context and suggests the most suitable ones. This recommendation of structure improves the authoring task and allows users not to excessively invest on complying to a structure *a priori*. The structure will, eventually, emerge from the contents themselves.

Awareness through elicitation of data. By presenting information derived from the relationships other users have with the content ("most frequent editor(s)", "owner", "most frequent reader(s)", ranking, feeds), the system supports an automated emergence of knowledge, providing new levels of understanding and taking into account the social nature of software development.

4. REFERENCES

[1] A. Aguiar and G. David. WikiWiki weaving heterogeneous software artifacts. In *Proceedings of the 2005 international symposium on Wikis*, pages 67–74, San Diego, California, 2005. ACM.

[2] W. Cunningham. Wiki design principles. http://c2.com/cgi/wiki?WikiDesignPrinciples, Mar. 2009.

[3] P. N. Robillard. The role of knowledge in software development. *Commun. ACM*, 42(1):87–92, 1999.