Extending and Integrating Wikis to Improve Software Documentation

Position paper for the Wikis for Software Engineering workshop

Filipe Figueiredo Correia

ParadigmaXis — Arquitectura e Engenharia de Software, SA
Avenida da Boavista, 1043, 4100-129 Porto, Portugal

FEUP — Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal

filipe.correia@fe.up.pt

ABSTRACT

Wikis present advantages regarding usage simplicity, collaboration and instant availability of contents. In what specifically concerns software documentation, they allow addressing a number of difficult issues like consistency, documentation reuse, reorganization of contents and collaboration.

Several systems exist supporting different strands of Wiki-based software documentation. Namely, Galaxy Wiki, XS-Doc, Trac and FitNesse are described, highlighting for each one how they support software development activities and artifacts.

Despite the value of existing solutions, the author argues that a broader approach is needed, to better support most of the software development activities from within a Wiki environment. Some requirements to achieve that objective are identified and described. They focus on how Wikis can be extended, and how can they be integrated with other tools, like IDEs.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia; I.7.2 [**Document and Text Processing**]: Document Preparation

General Terms

Wiki-based software documentation

Keywords

Wikis, software engineering, software documentation

1. INTRODUCTION

Wikis are systems that allow the collaborative creation and edition of Web page content using a Web browser. They are thus bound to hypertext, and follow a text-oriented model, using titles, paragraphs, lists, tables, figures, etc. as building blocks for creating documents. They provide the user with a simple syntax, allowing any word to be easily turned into a hyperlink, hence supplying a flexible mechanism for organizing contents.

Wikis are particularly appropriate in collaboration scenarios and when generalized availability of the produced contents is desired. The simplicity of Wikis makes them easily accessibly to a wide range of possible users. Incidentally, these are particularly valuable features in the context of software documentation.

Section 2 of this paper will focus on software documentation, introducing it's need and presenting some of its issues. Section 3 will discuss some existing Wiki-based solutions and techniques for software documentation. It also puts in contrast IDEs and Wiki-based approaches to supporting software development activities. Section 4 concludes by presenting the requirements for a Wiki-based solution to software documentation.

2. SOFTWARE DOCUMENTATION

Software documentation appears from the need to manage software projects' knowledge. The production of software documentation is a way of making this knowledge explicit and available to others, at the present and future times. However, despite its advantages it is an activity surrounded by resistance to adoption. As software artifacts evolve (source code, models, etc.), the respective documentation must be kept in-sync with them, in order to maintain its value. Documentation maintenance tasks are frequently tiresome to perform: the relation between documentation and the artifacts it describes is not explicit, thus requiring the developer to track which documentation parts will be affected by a particular change to a software artifact. This leads documentation authors to delay writing documentation to the last possible moment, in order to reduce the effort of repeatedly tracking and updating the same pieces of knowledge. Hence, it is not surprising that documentation easily becomes outdated, reflecting a snapshot of the system at a given point in time, and therefore loosing much of its value.

Some techniques have been devised over the years to help maintaining software documentation's consistency, being Literate Programming [10] one of the most note worthy. It relies on the concept of *verisimilitude*, that is, writing and storing source code and documentation physically close to each other, so that consistency is easier to keep by visual inspection. Different kinds of artifacts may be integrated in the documentation following this approach.

The production of documentation is quite commonly a collaborative activity, as it includes different members of the development team, and may even include external stakeholders, such as end-users. Stakeholders' participation may provide precious feedback, and be used to continually improve documentation's quality. Feedback is, however, difficult to obtain if no bidirectional channel exists, through which documentation can quickly be made available to anyone, and through which the documentation authors can receive back comments. This puts documentation in the condition of being part of a conversation, between documentation authors and consumers.

It is also important to point out that, as Integrated Development Environments (IDEs) tend to support wider ranges of software development activities, documentation production may also be regarded as a task that would benefit from this closer integration with the entire development process. At the same time, some Web-based platforms have been emerging that also aim on supporting development process activities, although they are more focused on artifacts other than source code. Software documentation as been a natural fit for Wiki environments.

3. DOCUMENTING USING WIKIS

Software documentation may assume different forms. Unit-tests, code annotations, requirements documents, architecture documents and user manuals, among others, may all be seen as forms of documentation. Several, if not most, of these kinds of documentation may follow a text-oriented model, which makes the expressiveness of a Wiki good enough for a wide range of software documentation types. Structuring information only by the use of titles, paragraphs, lists, tables and figures allows for a lot of **flexibility**.

Also, in the context of software documentation, **author collaboration** and the **generalized and instant availability** of contents are two very useful qualities, which can easily be achieved by Wikis.

Galaxy Wiki [12], XSDoc [4], Trac [6] and FitNesse [1] are just a few examples of Wiki engines that address the creation and maintenance of software documentation, taking into account also needs like **consistency maintenance**, **contents integration**, **reuse** and **process integration**, as will be further explored in the following sections.

3.1 Extended Wikis

Despite the qualities of Wikis for generic knowledge management tasks, it is arguable that a text-oriented model is not enough for software documentation. Software documentation frequently combines different kinds of artifacts — textual descriptions, models, source code, etc — which must be treated differently. These artifacts can be made part of documentation by using references (i.e., without copy-and-pasting) and thus maintaining their semantics. However, maintaining the semantics of artifacts doesn't mean that their underlying model is *understood*, and taken advantage of, by the Wiki engine. In order to ease their presentation, external artifacts are frequently directly converted to a format that fits the common text-oriented model of Wikis, while preserving, but not fully using, their original semantics.

A more concrete example of this scenario is the inclusion of a UML artifact as part of Wiki-supported documentation. Suppose this artifact is stored as a semantically rich format, such as XMI[1]. In spite of that, the form of representation that is more easily presented by a Wiki, while maintaining the graphic appearance expected by a user, is a static image file. This approach leads to the semantic opacity of the artifacts, and consequently raises two issues:

Granularity. It limits the reuse and integration of the UML artifact, by handling it as a simple image file. The semantics within the UML artifact are not explored (e.g., considering a class diagram, references could be made to individual classes, or sets of classes, instead of to the UML artifact as a whole).

Editing contents. Showing an image representation of an artifact and its textual description next to each other allows to visually identify possible inconsistencies between them. However, the conversion (of a semantically rich artifact) to a static — non-editable — image file makes necessary the use of external tools to update the non-native Wiki content, hence requiring a constant switch between different environments.

Going beyond the text-oriented model of Wikis means taking into account the semantics of external artifacts, being thus capable of taking a greater advantage of them. Semantically richer models for software documentation can be devised by taking a broader view on the software development process and by considering how the produced artifacts relate to each other [2, 3]. The systems described in the following paragraphs use different strategies for extending Wikis to support particular software development activities.

Galaxy Wiki. Xiao et al. has achieved interesting results in integrating source code and textual descriptions on a Wiki, in a way they are both editable from the Wiki environment [12]. Wiki pages correspond to classes, and can include documentation as textual descriptions that are, this way, bound to a specific class. In fact, textual descriptions and source code for a given class are internally stored in the same file,

[1]The **X**ML **M**etadata **I**nterchange is a standard based on the Extensible Markup Language (XML) for the exchange of metadata whose metamodel conforms to the Meta-Object Facility (MOF).

as regular commented source code. Although the main purpose of this research was not specifically in easing consistency maintenance, it may bring some benefits in this regard, as documentation and source code are kept close to each another. Also, despite the fact it does not consider artifacts other than textual descriptions and source code, the concept could be easily expanded to include other kinds of information (like UML models).

XSDoc. XSDoc [4], on the other hand, uses both source code and models in the context of Wikis, although, unlike Galaxy Wiki, it doesn't allow editing the source code (nor models) directly from the Wiki environment. Another fundamental difference is that it uses a multiple-source approach, storing documentation and source code in different files, even though these contents may be presented close to each other on a Wiki page. This separation makes easier the combination of different kinds of content, as well as the use of different tools and environments. It makes this appropriate approach, not only to source code documentation, but also to other kinds of documentation, like global architecture documents and requirements documents, among others. Another advantage is that documentation doesn't have to follow the same organization as source code, allowing the reordering of contents in a most appropriate way for human comprehension.

Trac. An additional common benefit of multiple-source approaches is the ability to reuse documentation. Both the linking and in-lining of content can be seen as forms of reuse. Trac [6] (also a multi-source approach), makes the in-line reuse of documentation possible, even if through the use of an additional plugin [9]. Unlike XSDoc it does not directly include the use of UML models, but manages to provide an overall deeper integration between documentation and the software development process by allowing the combination of a broader range of different artifacts, like source code files, issue tracker items, project milestones, source code revisions and other Wiki pages, among others.

FitNesse. FitNesse [1] is also an interesting Wiki engine regarding software documentation. It is classified by its authors as a software development collaboration tool, a software testing tool, a Wiki and a Web server, all in one. FitNesse allows the creation of acceptance tests in a collaborative way — as Wiki pages —, including stakeholders in the process of defining inputs and executing the tests. Wiki pages may include both textual descriptions and tests, which can be ran from within the Wiki page context. Tests may be seen as software documentation themselves, or as one more kind of software development artifact that may be further contextualized by textual descriptions. This means FitNesse fills the need for one more type of software documentation and one more type of software artifact — automated tests.

Galaxi Wiki, XSDoc, Trac and FitNesse all provide integration with IDEs in some way [12, 4, 11, 5].

3.2 Software life-cycle support

The usefulness of documentation may reveal itself in all stages of the software development life-cycle. IDEs aim on supporting several of these stages, and are an obvious choice concerning appropriate documentation support for each of them. Wikis are also able to support parts of the development life-cycle, and can be further extended to cover a wider scope of it.

However, in spite of the advantages supplied by Wikis in terms of simplicity, instant availability of contents and promotion of collaboration of all actors, they still present some drawbacks if compared with IDEs. IDEs are still more responsive, provide better work efficiency for a lot of tasks, and currently possess a wider range of features supporting the development process. Hence, Wikis and IDEs complement each other, and their use is not mutually exclusive.

Using both types of systems independently leads to a constant switch between environments, so it is not trivial to achieve their concerted use in an efficient way. It is usually more effective to focus on using one of the environments, if it answers all of our needs. This may why some features exist on both IDEs and development-oriented-Wikis, even though they frequently better-fit one of these environments. As some development activities may be supported by both, the choice between them ultimately depends on the context of use, and target audiences. IDEs are frequently more attractive to software developers, while Wikis are more attractive to non-technical actors.

4. CONCLUSIONS

Wikis have already showed to be capable, and to bring benefits, when used to support different software artifacts and documentation types, but it's still not practical to support all of the documentation activities along the development process using the existing solutions. Documentation needs crosscut all software engineering activities and artifacts. It is the author's belief that solutions for software documentation should take a comprehensive approach to the software development life-cycle. In spite of having already showed advantages for documentation, development-oriented-Wikis would still greatly benefit from this approach. Such Wiki solutions would take the following requirements into consideration:

- Make possible the creation of a wide range of **software documentation types**;
- Follow a multiple-source approach, allowing the **reuse of documentation artifacts**, by linking and in-lining them;
- Allow composition and **reorganization of contents** (i.e., documentation and other artifacts), making their presentation follow a sequence appropriate for human understanding;
- Allow the combination of a wide range of external content types, easing the maintenance of **consistency** by using *verisimilitude*;
- Make use of the semantic of contents, allowing to **edit external contents** from within the Wiki environment;

- Properly **integrate with IDEs**, as both environments, Wikis and IDEs, provide different benefits, depending on the usage context;
- Provide a **feedback channel**, by which readers can supply comments, and this way contribute to documentation's continuous improvement.

The use of **multiple types of documents**, the **reuse** of document parts, and the ability to **organize contents as one pleases** are features already provided by Wikis, and that the author believes to be of particular importance regarding software documentation. In the context of using **multiple document types**, Wiki-page templates can be tailored specifically for typical software development documents. Concerning **reuse**, a multiple-source approach can be followed: artifacts can be stored in different places, even if they may be edited together in the same Wiki page. However, this means that when creating distinct artifacts in a single Wiki page, each of them will have to be assigned a unique identifier, such that they can be included elsewhere (and become editable there too). The **reorganization of contents** comes naturally on a Wiki environment, but can still be expanded to include a broader scope of content types.

The maintenance of **consistency** can be eased by using *verisimilitude* but that advantage is easily lost if the need for a constant switch between environments arises. This constant switch becomes necessary whenever more than one platform is in use (e.g., a Wiki and an IDE) and a given artifact is not available on both or, in spite of having that artifact available on both platforms, it can only be edited on one of them. As such, Wikis should handle a wider scope of artifacts, but they should also allow those artifacts to be edited from within the Wiki environment. This may not be an easy task, as Wikis are Web-based solutions, in which contents are added as plain text representations. In order to express diverse artifacts, they will have to be represented as text too, which might not be their natural representation format. An alternative, is to use Web-browser's capabilities to provide rich interfaces (with technologies such as Javascript, Flash, etc.), and make use of WYSIWYG Wiki editors[2]. Even if not easier to implement, this alternative could be simpler to use if a high number of different Wiki syntaxes are at stake, due to a high number of different artifact types being considered.

Concerning **integration with IDEs**, the presentation of Web pages within the development environment may not be enough, in spite of being a common solution. Web-browsers are somewhat limited concerning their integration with non-Web resources, and embedding them in IDEs will therefore limit the degree of possible integration between IDE artifacts and Wiki artifacts. Instead, whenever Wikis expose an appropriate API[3], IDEs can be made to access Wiki contents and make them available to developers without using a Web-browser, in a way they can be strongly integrated with other artifacts. As an example, suppose an IDE that presents Wiki contents by directly retrieving them from the Wiki-engine. The Wiki syntax is interpreted by the IDE itself, and references to source code blocks can be easily followed by the developer, highlighting the referenced source code in the IDE context (i.e., in a way that it is available for being edited, as necessary). This approach can be equally applied to artifacts other than source code.

[2]A **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et editor is one in which the content is presented with an appearance very close to the one it will have after the edition.

[3]An **A**pplication **P**rogramming **I**nterface is a specification of a service provided by a software library or system, conceived to provide support to requests made by other systems.

Wikis favor collaborative authoring of contents, but other forms of collaboration exist. Giving **feedback** is also a form of collaboration; while readers are frequently not capable, or willing, to collaboratively author a document, they may be glad to contribute be giving back feedback. Due to their flexibility, Wikis can also serve this second purpose, and frequently they do, with the manual creation of a *comments* section on the bottom of each Wiki page. There is, however, room for improvement in this solution, as the comments become part of the document itself (which may not be the commenter's or authors' intention) and they are associated to the document as a whole, and not to a specific part of it.

The *Django Book* [8] is a book about the Django Web framework under a Web-page format. It only has two authors, but nonetheless was built with the help of a community in a collaborative way. The platform that supported this process provides a comments mechanism [7] that was used by the community to provided feedback throughout its making. It allows comments with the granularity of textual elements (titles, paragraphs, list items, etc.), and there is a clear distinction between the document itself and the feedback that was given about it. Although this platform is a Web-based approach, it is not a Wiki, but the concept could be easily adapted to an existing Wiki solution.

The list of presented requirements focuses on those regarded by the author as the most relevant, from the standpoint of a software documentation approach, that takes into account a wide scope of software development activities and artifacts. The guidelines suggested along with each requirement establish a possible path to a future implementation, or to the improvement of existing platforms.

5. REFERENCES

[1] Fitnesse — http://fitnesse.org/ [accessed on 2008/08/05].

[2] A. Aguiar. *A minimalist approach to framework documentation.* PhD thesis, Faculdade de Engenharia da Universidade do Porto, Sept. 2003.

[3] 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.

[4] A. Aguiar, G. David, and M. Padilha. XSDoc: an extensible wiki-based infrastructure for framework documentation. In *Jornadas de Ingeniería del Software y Bases de Datos*, Alicante, Oct. 2003.

[5] Band XI. Fitnesse for eclipse plugin — http://www.bandxi.com/fitnesse/ [accessed on 2008/08/05].

[6] Edgewall Software. The Trac Project — Integrated SCM & Project Management —

http://trac.edgewall.org/ [accessed on 2008/07/29].

[7] A. Holovaty and J. Kaplan-Moss. About the comment system of django book — http://www.djangobook.com/about/comments/ [accessed on 2008/08/24].

[8] A. Holovaty and J. Kaplan-Moss. The django book — http://www.djangobook.com/ [accessed on 2008/08/24].

[9] N. Kantrowitz. TracHacks — IncludeMacro Plugin — http://trac-hacks.org/wiki/includemacro [accessed on 2008/07/29].

[10] D. E. Knuth. Literate programming. *Comput. J.*, 27:97–111, 1984.

[11] M. Merli. TracHacks — EclipseTrac Plugin — http://trac-hacks.org/wiki/eclipsetracplugin [accessed on 2008/07/29].

[12] W. Xiao, C. Chi, and M. Yang. On-line collaborative software development via wiki. In *Proceedings of the 2007 International Symposium on Wikis*, pages 177–183, Montreal, Quebec, Canada, 2007. ACM.