XSDoc: an Extensible Wiki-based Infrastructure for Framework Documentation

Ademar Aguiar[1,2], Gabriel David[1,2], and Manuel Padilha[1]

[1] Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
{aaguiar,gtd,edeias}@fe.up.pt
http://www.fe.up.pt
[2] INESC Porto
http://www.inescporto.pt

Abstract. Good quality documentation is crucial for the effective reuse of object-oriented frameworks, and must be adaptable to the needs of different audiences. To satisfy these needs, framework documentation combines several kinds of documents and contents, resulting in hard, costly and tiresome to produce, specially when not supported by appropriate tools and methods. This paper presents XSDoc[3], an extensible infrastructure based on a WikiWikiWeb engine that supports the creation, integration, publishing and presentation of framework documentation. XSDoc helps on creating and annotating framework documents, on integrating different kinds of contents (text, models and source code), and provides a simple and economic cooperative web-based documentation environment that can be used standalone in a web-browser, or in an integrated development environment. A small example shows how easy it is to use XSDoc for producing part of a document for JUnit testing framework.

1 Introduction

Object-oriented frameworks are a powerful technique for large-scale reuse that, through design and code reuse, help developers achieve higher productivity, shorter time to-market, and improved compatibility and consistency [1,2].

But to benefit from framework reuse advantages, one first has to invest time on *understanding* the framework, and on *learning* how to use it. As one of the most complex kinds of object-oriented products, frameworks can be particularly hard to understand by first-time users, specially if not accompanied with appropriate documentation [1,3].

1.1 The importance of framework documentation

Good quality documentation helps developers achieve the degree of understanding they need to customize a framework and evolve its internal design and implementation. Concerning the importance of documentation for framework reuse,

[3] XSDoc means "Extensible Software Documentation"

Booch stated in [4] that *"the most profoundly elegant framework will never be reused unless the cost of understanding it and then using its abstractions is lower than the programmer's perceived cost of writing them from scratch"*.

Although not always recognized, documentation plays a central role also in many framework development tasks. Most of the development effort is spent on formalizing information, i.e., on reading and understanding requirements, specifications and other informal documents, in order to produce formal documents, such as source code files and specifications.

Good quality documentation is therefore a crucial success factor for framework reuse, being also very important for framework development, maintenance and evolution.

1.2 Difficulties with framework documentation

To document frameworks is, at least, an order of magnitude more difficult than to document object-oriented applications or class libraries, because we must cover not only a single concrete product (an application) but, instead, a tool that is able to produce a family of many similar concrete products (a framework).

Different approaches for documenting frameworks have been suggested, and some have proven to be effective in reducing the learning curve of frameworks, namely the approaches based on cookbooks, patterns, and meta-patterns. However, despite the research done in the community, it is still hard, costly and tiresome to define and write good quality documentation for a framework.

To be complete and useful, the documentation of a framework must include a large diversity of richly cross-linked contents, it must be presented in different ways to meet the needs of different audiences. It must describe the application domain covered by the framework, its purpose, how-to-use it, how it works, and details about its internal design.

Good quality documentation for a framework is complex to produce because is must cope with; *different audiences*, such as framework selectors, application developers, framework developers and developers of other frameworks; *different styles of documents*, such as framework overviews, example applications, cookbooks and recipes, design patterns, use cases, contracts, design notebooks and reference manuals, in order to provide multiple views (static, dynamic, external, internal) at different levels of abstraction and detail (architecture, design, implementation); and notations to represent its contents, such as free text, structured text, source code, object models, images, and formal specifications. [5].

1.3 Research overview

The fundamental goal of the research presented in this paper is to provide typical development environments (integrated or not) with tools and features able to support documentation activities along the framework lifecycle. These tools should make documentation convenient and attractive for framework developers, who are the best source of knowledge about the design and implementation

of a framework. By providing development environments with documentation capabilities we aim to close the gap between the activities of development and documentation, and thus motivate and assist developers to document while they code and design.

XSDoc is an extensible infrastructure for framework documentation that was built to achieve such goals. XSDoc is easy to integrate in development environments, including modern open integrated development environments (IDEs). Although originally motivated to support a specific approach to framework documentation [6,7], XSDoc can be used with other approaches, due to its extensibility and integrability with other tools.

This paper is organized as follows. The next section briefly overviews the most relevant kinds of software documentation techniques. Then, the architecture and key features of XSDoc are presented. A small example with JUnit framework documentation shows how it can be used both autonomously and integrated in the Eclipse IDE [8]. The final section discusses the key advantages of XSDoc and the work planned for the near future.

2 Software Documentation Techniques

The many artifacts produced during development can be categorized in source code, models, and documents, all of which require continual review and modification throughout the lifecycle.

Typically, different kinds of contents are maintained independently using different environments and editors (text editors, source code editors, visual model editors, document editors). Such configuration often requires constant switching between working environments, thus resulting inappropriate for maintaining the semantic consistency between several documents, as it disturbs developers and causes significant process inefficiencies.

2.1 Literate programming

The literate programming approach and tools [9] are a possible solution to the problem of having source code and documentation always consolidated. The technique was invented by Donald Knuth and involves writing documentation and code in a *single source document* (*verisimilitude*) *psychologically organized for comprehension by humans rather than computers*. Literate files can be *tangled* to produce computer-understandable code, and *woven* to produce human readable and comprehensible typeset documents. The technique provides significant incentives for developers to document while they code, potentially resulting in programs of higher quality and maintainability.

Despite its advantages, the technique is not widely used. This is mainly due to the integration difficulties of literate programming tools in mainstream development environments: it requires the combined use of three languages, a programming language, a documentation language and an interconnection language; it introduces too much overhead for small programs; the format of literate files

is complex, what seriously compromises their on-screen readability and understanding during development; and finally, the organization of the source code seen by the compiler is different from the original, as written by the programmer, what often cause problems when using tools that manipulate source code files, such as debuggers, generators, reverse-engineering tools, and refactoring tools.

2.2 Single source methods

Alternative documentation techniques to literate programming can be classified into either *single source* or *multiple source* methods.

Single source methods integrate code and documentation together in a same file, so there are no consistency problems between code and documentation as there is no replication of contents in the whole documentation bundle. The Sun's Javadoc utility is an example of a single source method that *weaves* Java source files to produce interface documentation in HTML, using simple Java commenting conventions. Javadoc cannot be considered a literate programming tool because it lacks the support for psychological arrangement of documentation. Other examples of similar tools are Doxygen and Doc++.

2.3 Multiple source methods

Multiple source methods maintain documentation and code in separate files. Traditional documentation written externally to source code is the most common example of such technique. Due to the separation of code and documents, documenters usually copy source code into document editors, and as soon as the code changes, the two documents become inconsistent. More sophisticated systems simulate the verisimilitude characteristic of literate programming paradigm with the help of tools that automatically manage the strict relationships between code and documents.

The most typical problem with this kind of tools is not allowing the edition of code and documents outside their tools, a serious obstacle for easy integration in mainstream development environments. A good example of such a system is Elucidative Programming [10], which enables the linking between source code and documents using the insertion of special directives. Similarly to literate programming, elucidative programming is intended for internal documentation, and thus only supports very simple structured documents and textual contents.

3 The XSDoc documentation infrastructure

The XSDoc is an infrastructure based on XML and WikiWikiWeb [11] that covers all the typical functionalities of a content management system for framework documentation. XSDoc uses a multiple source method.

XSDoc was specifically built to support a new approach to framework documentation, which aims to be simple and economic to adopt. The approach

reuses existing framework documentation styles, techniques and tools and combines them in a way that follows the design principles of minimalist instruction theory [6,7].

Currently, XSDoc supports source code written in Java and C++ programming languages, models described in UML, and XML documents. The XSDoc infrastructure results from the reification of two previous prototypes. The FrameDocMS [12] was a first prototype developed with the goal of evaluating the appropriateness of using Wiki and XML technologies to implement a content management system for framework documentation. The key idea of *Wiki-based software documentation* behind FrameDocMS was then generalized to object-oriented software documentation and evaluated with the WID[4] prototype [13]. As a result, XSDoc combines the best features of both prototypes and has an evolved architecture, more flexible and easy to extend.

The XSDoc infrastructure is composed by one WikiWikiWeb engine (XSDocWiki), plugins for seamless integration in open IDEs (currently only exists one for the Eclipse IDE), and a set of document templates (framework book, cookbook, pattern, use case, etc) markup languages, and converters of contents to and from XML. The Fig. 1 illustrates these components as well as their interconnections.

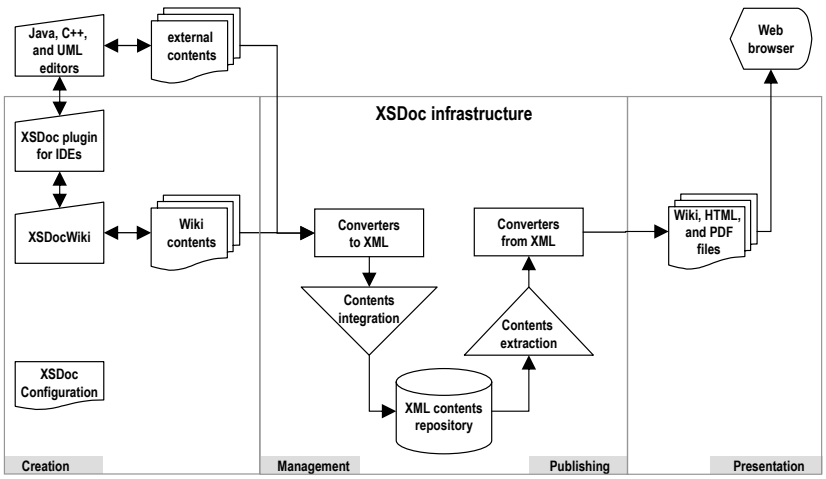

Fig. 1. XSDoc components and their interconnections.

[4] WID — Wiki-based Integrated Documentation

3.1 XSDocWiki

The main component of the XSDoc infrastructure is XSDocWiki, a WikiWikiWeb engine that extends a typical Wiki engine with several features useful for documenting object-oriented frameworks, and software in general.

The Wiki concept. A WikiWikiWeb, or simply a Wiki, is a very innovative and appealing collaboration tool [11]. It can be defined as a web platform for the collaborative edition of documents. Documents are composed by topics, where each topic corresponds to a web page. Using a simple web browser, everyone can edit any page of the wiki, by invoking the "Edit" option available on it. After saving, the new version of the page is uploaded to the web server and become immediately available.

There are several implementations of the Wiki concept, but all use very simple markup languages to support text formatting, and simple mechanisms based on WikiNames[5] for automatic linking of pages. Despite its simplicity, wiki names are very powerful because they support dynamic linking.

Extensions added to the original Wiki engine. The XSDocWiki was developed using the VeryQuickWiki engine [14] as a starting base, which was then extended with several features to make convenient the edition and visualization of typical software documentation contents, including the support for: linking and inlining source code fragments; linking and inlining UML diagrams; instantiation and validation of XML documents; accessing repositories of version control systems; adding new styles of documents using a plugin mechanism; and a few browsing controls to adapt presented contents to user needs. These extensions enhance the automatic linking mechanism originally restricted to Wiki pages to support also linking of source code, models and structured contents, using simple naming conventions (e.g. prefixes, suffixes, and patterns), which are very easy to learn and use.

To be flexible, XSDocWiki provides a plugin mechanism that supports the addition of new styles of documents (e.g. use-case, example, pattern, requirement). A XSDoc plugin typically includes: a document-template; a set of converters to map that style of document to and from XML, if necessary; a declaration of the elements to be parsed for automatic linking of WikiNames; and some lexical rules to use during the automatic linking phase. For example, for Java source files, it is declared that Javadoc comments may contain Wiki text, what enables the usage of wiki names to link to other source code contents, UML models or documents, structured or not.

So configured, the XSDocWiki promotes the collaboration of technical and non-technical people on an incremental edition and revision of framework documents, ensuring high availability of contents (always online), using simple fea-

[5] The word WikiName is a Wiki name because it JoinsCapitalizedWords, which is AnotherWikiName

tures, automated archiving, and only requiring a simple web browser, a tool currently very easy to integrate in a vast majority of development environments.

3.2 XML converters and presentation processors

As most of the documentation contents can be comfortably edited and linked using the Wiki, most of them will reside on Wiki pages stored in a file system, a version control system (currently, only CVS is supported), or a database.

However, source code programs and UML diagrams need special processing, because they must be converted from their original format to XML using XSL transformers, respectively using JavaML [15], Doxygen and SVG/XMI vocabularies.

At a later stage, the contents are filtered and formatted accordingly to be published and presented. Currently, XSDoc is able to output HTML files for online browsing, and PDF files for high-quality printing.

3.3 Integration Mechanisms

The components of XSDoc are closely integrated, both functionally and in terms of the information they exchange.

Multiple source approach. The integration of source code, UML models and structured documents is achieved through a multiple source approach, what means that source code and documentation reside in separate files. While this separation preserves source code files and UML files, it requires a way of managing the relationships between their contents.

These relations are supported in XSDoc by small extensions to the original hyperlinking and inlining mechanisms of the wiki engine, which enable writers to link and inline source code and models using simple textual references. These extensions are implemented using source code parsers, and XML technology. The information exchanged between the tools uses a textual format, both pure text files and XML files. A markup language is also used internally to normalize all the contents in a unique schema, when necessary.

Wiki-centric functional integration. The functional integration of the Wiki with the converters and processors is done within the Wiki and its specific extensions, using Java, servlets, Java Server Pages, and external programs, such as: a modified version of the jikes compiler to generate JavaML files, and the doxygen documentation generator for C++.

Integration in development environments. The integration of XSDoc in an industrial development environment is very easy to achieve in almost every situation, considering that XML is widely supported everywhere and the Wiki engine only needs a browser to run. We think that the combined use of XML and

Wiki makes this integration successful in almost every industrial development environment only with the low cost of small configurations in the environment.

Another goal of the infrastructure is its seamless integration in modern development environments, such as IDEs. The integration of the XSDoc infrastructure with IDEs is achieved through specific plugins. The plugin should enable the use of a web browser through which the XSDocWiki can be accessed, and should also provide a communication link between the IDE and the XSDocWiki, to enable their interoperation.

Side-by-side edition of all contents. Much tighter integration of the infrastructure in a development environment can be achieved with open IDEs, such as Borland's Together or IBM's Eclipse, which enable in a single environment the side-by-side edition and and synchronization of all kinds of contents: source code, UML models, and documents, thus eliminating the need to switch applications during development. In Fig. 2 is represented a snapshot of the Eclipse IDE with the XSDoc plugin.

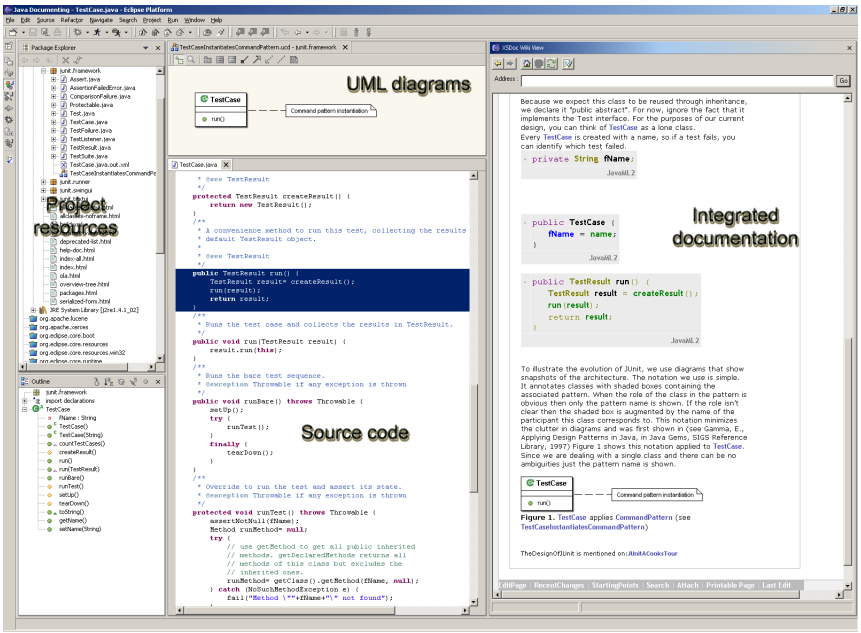

Fig. 2. A snapshot of the XSDoc plugin for the Eclipse IDE showing the integration of the development environment provided by Eclipse (on the left) with the documentation environment provided by XSDoc (on the right).

3.4 Extensibility Mechanisms

XSDoc was designed with a special concern on simplicity, flexibility and extensibility, so that it can be adapted to different project environments, ranging from literate programming environments to industrial integrated development environments.

As a result, in addition to the kinds of contents already supported and the template documents provided, XSDoc users are able to extend it with new templates, linking mechanisms, and formatting features. XSDoc administrators can also customize and develop plugins to support new programming languages or other kinds of contents.

4 Using the XSDoc infrastructure

The usage of XSDoc is very simple, and the basics can be learned very fast in few minutes by people already familiar with the use of a web browser.

As an example, it will be briefly presented how XSDoc can be used to write part of a document for the JUnit testing framework. Fig.3 shows part of a document that describes the instantiation of the `Command` pattern by the class `TestCase` of the JUnit framework. The text that is required to write is shown in Fig.3(top), and the resulting documentation, is represented in Fig.3(bottom). Any change on the code or models is automatically reflected in the documentation when the web page is refreshed by the browser, or when the involved contents are modified and saved.

4.1 Installation and configuration

Firstly, XSDoc must be installed and configured. In the current state of development, XSDoc is available as a Tomcat's web application archive (`xsdoc.war`) ready to be automatically deployed and installed in the application server.

After installation, it is required to create an area on the wiki to store the documents of the project, usually called a *wiki web*. The project's wiki web must then be configured to the specificities of the project at hands. The configuration includes the definition of: template documents, based on those already provided with XSDoc, or written from scratch; additional linking conventions and navigational properties, location of repositories; kind of integration with the development environment (standalone or a specific supported IDE); and user accounting. With XSDoc configured to the development environment, the developer can then use a web browser to access the XSDocWiki.

4.2 Contents creation

Documentation contents may be created *internally* with the XSDocWiki or *externally* with editors not included with XSDoc. Source code contents, and UML diagrams always require the use of external editors.

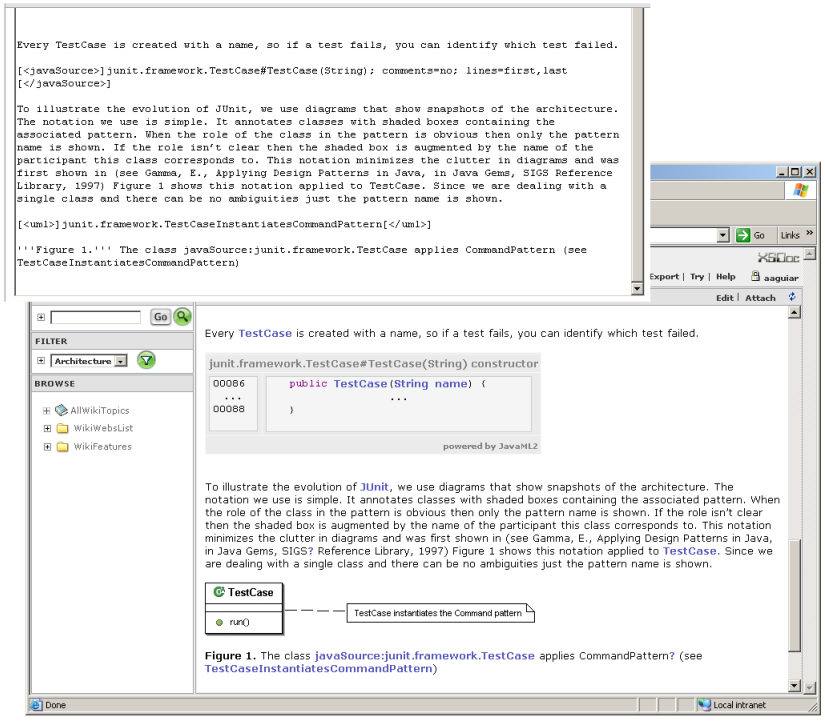

Fig. 3. Documenting the application of the Command pattern to the class TestCase: (top) text written (bottom) output obtained.

Using the XSDocWiki it is possible to create new documents, and to revise them using a web-based collaborative environment. The creation of new pages are triggered by following a non-existent topic, presented as a link marked with a question mark, as shown in Fig.3. Template documents can be associated with specific topic name patterns, and are instantiated at topic creation time. For example, the wiki name CommandPattern shown in Fig.3(top) is associated with the template DesignPattern. Depending on the level of integration with the development environment, the XSDocWiki may trigger the creation of source code and UML diagrams using external editors.

4.3 Contents integration

In addition to the hyperlinking mechanisms provided by HTML, XSDoc provides two dynamic mechanisms for the integration and synchronization of the possible kinds of document contents (source code, UML diagrams, XML files): inlining of contents, and automatic linking using Wiki names.

Inlining. The inlining of contents is defined with a reference to the specific contents, annotated with the tag predefined for its respective kind. Example: `[<javaSource>]junit.framework.TestCase[</javaSource>]`.

Whenever possible, these references use standard formats and rules, such as Javadoc references for Java source code. As an example, it is shown in Fig.3(top) a reference to a fragment of Java source code, which includes the first and last line of the method `setName(String)` from the `TestCase` class of the `junit.framework` package, from JUnit framework.

Linking. The definition of links to specific contents are realized with Wiki names and external references with predefined formats. Here is an example: `javaSource:junit.framework.TestCase`. In Fig.3(top) are shown other examples. `TestCase` and `CommandPattern` wiki names link to topics of the overall documentation. `CommandPattern` doesn't have its target defined yet, therefore the Wiki (see Fig.3(bottom)) presents it with a question mark.

In a similar way, XSDoc supports the inline and linking of C++ source code and UML models, using the tags `cppSource` and `uml`.

4.4 Contents publication and presentation

The contents are always available for online browsing through the XSDocWiki, but can also be exported to static HTML, for offline browsing, or to PDF files, for high-quality prints.

Source code is presented with syntax-highlighting, and smart hyperlinking to other source code files, formal documentation contained in Javadoc comments and Doxygen comments, and related documents.

5 Conclusions

XSDoc, an infrastructure combining a Wiki engine, document processing with XML technology, and easy integration in development environments, including open IDEs, shows that the production of framework documentation can be done easier with the help of appropriate tools.

From the work done, we conclude that the use of the XSDoc infrastructure can reduce the effort typically needed to document frameworks, specially when integrated in an open IDE, as it combines the simplicity, easiness and versatility of the collaborative document edition in the Wiki, with the powerful development features of IDEs, and the well-known qualities of XML technology in terms of information integration, processing and presentation.

The combination of both technologies result in a very attractive infrastructure, whose best qualities can be summarized as: easy to integrate in software development environments; easy to use by technical and not technical people; promotes the participation of all the team in the documentation process; improves team communication; provides an easy way to access, revise and evolve the documentation; and finally, enables a smooth integration of contents in a

controlled and structured way, while preserving the information in an universal format, the XML format.

In future work, the XSDoc tools will be improved with more browsing features to help adapt documentation contents to the user needs (zoom, exploration mode, error recovery, extensive search), new plugins for integration with other popular IDEs will be developed, and other popular Wiki engines will be supported. Another interesting feature to add to XSDoc it would be to support literate programming in XML, ie, to enable the definition of source code fragments in documents. In order to quantitatively and qualitatively evaluate the impact of these tools on the quality, understandability and usability of the resulting framework documentation, more user tests and experiments are required, namely in industrial settings.

References

1. Taligent Press. *Building Object-Oriented Frameworks*. Addison-Wesley, 1994.
2. Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson. *Building Application Frameworks — Object-Oriented Foundations of Framework Design*. John Wiley & Sons, 1999.
3. Greg Butler and Pierre Denommée. Documenting frameworks. In *Building Application Frameworks — Object-Oriented Foundations of Framework Design* [2], pages 495–504.
4. Grady Booch. Designing an application framework. *Dr. Dobb's Journal*, 19(2), February 1994.
5. Greg Butler, Rudolf K. Keller, and Hafedh Mili. A framework for framework documentation. http://www.cs.concordia.ca/faculty/gregb, 1998.
6. Ademar Aguiar. *A minimalist approach to framework documentation*. PhD thesis, 2003.
7. Ademar Aguiar and Gabriel David. A minimalist approach to framework documentation. In *Proceedings of the 13th Workshop for PhD Students in Object-Oriented Systems, ECOOP'2003 — European Conference on Object-Oriented Programming (to be published)*, 2003.
8. IBM. Eclipse, an open extensible integrated development environment.
9. Donald Ervin Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.
10. Kurt Nørmark. Requirements for an elucidative programming environment. In *Eight International Workshop on Program Comprehension*. IEEE, June 2000.
11. Ward Cunningham. The original wiki front page., 1999. Available from http://c2.com/cgi/wiki.
12. Ademar Aguiar and Gabriel David. FrameDocMS — um sistema de gestão de conteúdos para documentação de frameworks baseado em XML e WikiWikiWeb. In *Proceedings of XATA 2003, XML: Aplicações e Tecnologias Associadas (to be published)*, February 2003.
13. *e*deias. Wid — wiki-based integrated documentation. Available from http://www.fe.up.pt/~edeias/.
14. Gareth Cronin and Bill Barnett. Very quick wiki engine homepage. Available from http://veryquickwiki.sourceforge.net/.
15. Greg J. Badros. JavaML: a markup language for Java source code. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):159–177, 2000.