SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2003; **0**:1–15
Prepared using speauth.cls *[Version: 2002/09/23 v2.2]*

A universal documentation extension for arbitrary programming languages

Volker H. Simonis*

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, 72076 Tübingen, Germany

SUMMARY

Traditionally program documentation has never been treated as a first class citizen of computer programs and as such has not received wide support by language designers. Comment lines which are ignored by the compiler have been the broadest common denominator in virtually all programming languages. This paper proposes a universal documentation extension which may be applied non-intrusively to any arbitrary programming language. It may be used for automatic interface documentation generation as well as for linking external documentation with parts of the actual source code. The benefits of this new documentation scheme are: synchronized code and documentation, different levels of compiler support for program documentation and wider tool support due to the independence from the actual programming language and a standardized output format. A prototype implementation of the new approach is presented for the Java programming language and the DocBook system.

KEY WORDS: software documentation, programming languages, DocBook, XML, XSLT, Java

1. Introduction

From the very beginning programming languages knew the concept of comments. Because comment lines were completely ignored by the compiler they could contain arbitrary content. So it became good programming practice to use comments in order to document the most important and the most intricate parts of a program in prose. However, documenting a program in such a way has a number of serious drawbacks. First of all, the intended reader needs full access to the source code. Sometimes a subject may be most easily explained by a picture or a formula which is extremely hard to do by using merely ASCII characters. Finally, excessive documentation with comment lines can make the program code itself hard to read and edit.

These problems lead to the development of the concept of *Literate Programming* by D. Knuth [34] where the source code and the documentation are written into a single file using TeX [35]. This way

*Correspondence to: simonis@informatik.uni-tuebingen.de

Received May 21, 2003

Copyright © 2003 John Wiley & Sons, Ltd.

Draft
Don't redistribute or cite without explicit permission!

the full power of the TeX typesetting system can be used for the documentation. However, before compiling the program the source code has to be extracted from the documentation first.

In recent days, Java [22] introduced a new documentation system called JavaDoc. It is based on API documentation which is automatically generated by the compiler and which can be augmented by the programmer with the help of special comments which are inserted into the source code. However only high level, interface documentation can be achieved this way.

All these three mentioned approaches do not handle program documentation in its entirety. Therefore this paper proposes a new, universal and language independent documentation scheme which can be applied non-intrusively to any programming language.

The language extension is non-intrusive because it is completely transparent to any compiler which is unaware of the extension. Therefore, as a first step before compiler support will be available, the extension may also be handled by an external preprocessor.

The documentation scheme is language independent because it may be used with any programming language which offers simple comments. It is universal because it offers a uniform interface and output format no matter with which programming language it is actually used. Finally it is new in the sense that it combines well known and approved techniques in a new and innovative way.

The remainder of this paper is organized as follows. The next sections will provide some more in depth information about a wide range of software documentation styles and currently available program documentation systems. It also serves as a related work section. Thereafter, the new approach proposed by this paper will be introduced and a prototype implementation of the new ideas for the Java programming language will be presented. A section containing the conclusions and acknowledgements will finally end the paper.

2. Program documentation

The usefulness and necessity of a good software documentation is generally accepted by every programmer. However there is no such unity when it comes to the question what is a good documentation and there is even less agreement upon how to produce such a documentation. Nevertheless, the following features seem to be crucial for every documentation system:

1. Documentation and source code should always be consistent and synchronized.
2. The system should be easy to use in order to be accepted by the programmer. (I.e. programming should not be constrained and documenting should be as easy as just writing with a usual word processor.)
3. Different levels of documentation like interface or implementation documentation for different audience should be possible.
4. The documentation should be legible, appealing and equally well suited for various output formats like printed manuals, books or online browsing.
5. Interoperability, team and tool support are crucial today because projects tend to use more than one programming language, support more then one platform and are being worked one by many people simultaneously.

The following subsections will analyze currently available systems with respect to these criteria.

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

2.1. Literate programming systems

With an article published 1984 in the Computer Journal [34] Donald Knuth coined the notion of *Literate Programming*. Since those days for many people literate programming is irrevocably interweaved with Knuth's WEB [37] and TEX [35] systems.

Knuth justifies the term literate programming in [34] with his belief that "... the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature." To support this programming style, he introduced the WEB system which is in fact both a language and a suite of utilities. In WEB, the program source code and the documentation are written together into one source file, delimited by special control sequences. The program source can be split into parts which can be presented in arbitrary order. The `tangle` program extracts these code parts from the WEB file and assembles them in the right order into a valid source file. Another program called `weave` combines the documentation parts of the WEB files with pretty printed versions of the code parts into a file which thereupon can be processed by TEX.

This system has many advantages. First of all, it fulfills the "one source" property. Because source code and documentation reside in one file, they are always consistent with each other. Second, the programmer is free to present the code he writes in arbitrary order, thus simplifying it for a human reader to understand the program. This can be done by rearranging code parts, but also by using macros inside the code parts, which can be defined later on in the WEB file. This way a top-down development approach is supported, in which the structure of a program as a whole is presented in the beginning and then subsequently refined, as well as a bottom up design, in which a program is assembled out of low level code fragments defined before. `tangle` will always expand these macros at the right place when constructing the source file out of the WEB file.

Another feature of the WEB system is the automatic construction of exhaustive indexes and cross references by `weave`. Every code part is accompanied by references which link it to all other parts which reference or use it. Also, an index of keywords with respect to code parts is created and the source code is pretty printed for the documentation part. The best way to convince yourself of WEB's capabilities is to have a look at Knuth's TEX implementation [36]. It was entirely written in WEB and is undoubtfully a masterpiece of publishing and literate programming.

2.1.1. WEB and its descendants

Besides its many advantages, the WEB system also has a couple of drawbacks. Many of them apply only to the original WEB implementation of Knuth and have been corrected or worked around in numerous WEB clones implemented thereafter. In this section we will present some of them† and discuss their enhancements.

One of the biggest disadvantages of WEB was the fact that it was closely tied to TEX as typesetting system and to Pascal as implementation language. So one of the first flavors of WEB was CWEB [38] which extended WEB to C/C++ as implementation languages. It was implemented by Knuth himself together with Silvio Levy. CWEBx [40] is an alternative CWEB implementation with some extensions

†Only systems known to the authors will be mentioned here. A more complete overview may be found at the Comprehensive TEXArchive Network (CTAN) under http://www.ctan.org/tex-archive/web or at http://www.literateprogramming.org.

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

by Marc van Leeuwen. They both suffer from the same problems like WEB, as they are closely coupled to TEX and the C programming language.

To overcome these language dependencies, noweb [50] (which evolved from spiderWEB) and nuweb [10] have been developed by Norman Ramsey and Preston Briggs, respectively. They are both language independent concerning the programming language, whereas they still use LATEX for typesetting. Nuweb is a rather minimalistic but fast WEB approach with only four control sequences. Both noweb and nuweb offer no pretty printing by default, but noweb is based on a system of tools called filters which are connected through pipes. The current version comes with pretty printing filters for C and Java (see the actual documentation).

Another descendant of an early version of CWEB is FWEB [39]. FWEB initially was an abbreviation for "Fortran WEB", but meanwhile FWEB supports not only Fortran, but C, C++, Ratfor and TEX as well. These languages can be intermixed in one project, while FWEB still supports pretty printing for the different languages. On the other hand, FWEB is a rather complex piece of software with a 140 page user's manual.

Ross Williams' funnelWEB [67] is not only independent of the programming language, but of the typesetting language as well. It defines format macros, which can be bound to arbitrary typesetting commands (currently implemented for HTML and LATEX).

2.1.2. General drawbacks of WEB based literate programming tools

Though many of the initial problems of the WEB system have been solved in some of the clones, their sheer number indicates that none of them is perfect.

One of the most controversial topics in the field of literate programming is pretty printing where *pretty printing* stands for syntax highlighting[‡] and code layout and indentation. There are two questions here to consider: Is pretty printing desirable at all, and if yes, how should the pretty printed code look like? The answer is often a matter of personal taste, however there also exist some research results in this area like for example the work of Baecker and Marcus who dedicated a whole book [6] to the question of how to highlight, layout and typeset for maximum legibility and readability.

From a practical point of view it must be stated that doing pretty printing is possible for Pascal, although a look at the WEB sources will tell you that it is not an easy task. Doing it for C is even harder[§]. Taking into account the fact that `weave` usually processes only a small piece of code, which itself even does not have to be syntactically correct, it should be clear that pretty printing such code in a complex language like for example C++ will be impossible.

To overcome these problems, special tags have been introduced by the various systems to support the pretty printing routines. But this clutters the program code in the WEB file and even increases the problem of the documentation looking completely different than the source. This can be annoying in a develop/run/debug cycle. As a consequence, the use of pretty printing is discouraged. The only feasible solution could be just simple syntax highlighting instead of pretty printing, as it is done by many editors nowadays, which is a much easyer task than pretty printing.

[‡] *Syntax highlighting* denotes the process of graphically highlighting the tokens of a programming language.
[§] The biggest part of CWEB consists of the pretty printing module. Recognition of keywords, identifiers, comments, etc. is done by a hard coded shift/reduce bottom up parser.

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

Even without pretty printing and additional tags inserted into the program source, the fact that the source code usually appears rearranged in the WEB file with respect to the generated source file makes it very hard to extend or debug such a program. A few lines of code laying closely together in the source file may be split up to completely different places in the WEB file.

Once this could be called a feature, because it gave the programmer new means of structuring his program code for languages like Pascal which offered no module system or object hierarchy. As analyzed in [13] it could be used to achieve a certain amount of code and documentation reuse. However, the WEB macro system can also be misused by defining and using macros instead of defining and using functions in the underlying programming language.

Another problem common to WEB systems is their "one source" policy. While this may help to hold source code and documentation consistent, it breaks many other development tools like debuggers, revision control systems and make utilities. Moreover, it is nearly impossible for a programmer not familiar with a special WEB system to debug, maintain or extend code devolved with that WEB.

Even the possibility of giving away only the tangled output of a WEB is not attractive. First of all, it is usually unreadable for humans[¶], and second this would break the "one source" philosophy. It seems that most of the literate programming projects realized until now have been one man projects. There is only one paper from Ramsey and Marceau [49] which documents the use of literate programming tools in a team project. Additionally, some references can be found about the use of literate programming for educational purpose (see [12] and [56]).

The general impression confirms Van Wyk's observation in [77] "... that one must write one's own system before one can write a literate program, and that makes [him] wonder how widespread literate programming is or will ever become." The question he leaves to the reader is whether programmers are in general too individual to use somebody else's tools or if only individual programmers develop and use (their own) literate programming systems. The answer seems to lie somewhere in between. Programmers are usually very individual and conservative concerning their programming environment. There must be superior tools available to make them switch to a new environment.

2.2. Other software documentation systems

With the widespread use of Java a new program documentation system called JavaDoc was introduced. JavaDoc [22] comes with the Java development kit and is thus available for free to every Java programmer. The idea behind JavaDoc is quite different from that of WEB, though it is based on the "one source" paradigm as well. JavaDoc is a tool which extracts documentation from Java source files and produces formatted HTML output. Consequently, JavaDoc is tied to Java as programming and HTML as typesetting language[‖]. By default JavaDoc parses Java source files and generates a document which contains the signatures of all public and protected classes, interfaces, methods, and fields. This documentation can be further extended by specially formatted comments which may even contain HTML tags.

[¶]NuWEB is an exception here, since it forwards source code into the tangled output without changing its format.
[‖]Starting with Java 1.2, JavaDoc may be extended with so called "Doclets", which allow JavaDoc to produce output in different formats. Currently there are Doclets available for the MIF, RTF and LaTeX format (see [61]).

Copyright © 2003 John Wiley & Sons, Ltd.
Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

Because JavaDoc is available only for Java, Roland Wunderling and Malte Zöckler created DOC++ [76], a tool similar to JavaDoc but for C++ as programming language. Additionally to HTML, DOC++ can create LaTeX formatted documentation as well. Doxygen [24] by Dimitri van Heesch, which was initially inspired by DOC++, is currently the most ambitious tool of this type which can also produce output in RTF, PDF and Unix man-page format. Both DOC++ and Doxygen can create a variety of dependency-, call-, inclusion- and inheritance graphs, which may be included into the documentation. Notice that customized versions of tools like DOC++ and Doxygen may be used as preprocessors for the documentation extensions proposed in this paper (see section 3).

C# [11], Microsoft's answer to Java, comes with its own documentation system as well. In principle it works in the same way as JavaDoc. The only difference is the resulting output format which is XML. This is a big advantage compared to JavaDoc, because the output is not tied to a special typesetting language. Instead the produced XML format is specified in the Appendix E of the C# language definition [11]. Additional tools like NDoc [17] must be used to produce printable or displayable versions from the XML output of the C# documentation generator.

Synopsis [16] by Stephen Davies and Stefan Seefeld is another similar tool. Written mainly in Python [41] it supports an architecture of pluggable parsers and formatters for various source languages and output formats. Currently it supports Python, IDL and C++ as programming languages and among others HTML, DocBook and TexInfo as output formats. The interesting thing about Synopsis is the fact that it really parses the whole source code and builds an internal abstract syntax tree (AST) of the code. With the help of this AST exhaustive cross references can be build like for example linking every variable to the place where it was declared or to the place where its type is defined. Moreover, Synopsis can produce highlighted listings of the source files which are linked to the generated API documentation.

The new documentation tools presented so far are mainly useful for creating hierarchical, browsable HTML documentations of class libraries and APIs. They are intended for interface descriptions rather than the description of algorithms or implementation details. Although some of them support LaTeX, RTF or PDF output, they are not particularly well suited for generating printed documentation.

ProgDoc [57] is a program documentation tool which links program sources to the documentation by inserting special handles as comments into the source files. These handles can be referenced from the documentation. The actual source code is inserted into the documentation by a special preprocessor written in AWK [2]. The system is independent of the programming language and offers special syntax highlighters for many common languages like C/C++, Scheme, Java and XML. It uses LaTeX for typesetting and supports PDF, PS and HTML output.

Another approach which must be mentioned in this chapter is Martin Knasmüller's *Reverse Literate Programming* system [33]. In fact it is an editor which supports code folding [26, 33] and so called *active text elements* [45]. Active text elements may contain arbitrary documentation, but also figures, links or pop-up buttons. All the active text is ignored by the compiler, so no tangle step is needed before compilation. Reverse Literate programming has been implemented for the Oberon system [68].

The GRASP [26] system relies on source code diagramming and source code folding techniques in order to present a more comprehensible picture of the source code, however without special support for program documentation or literate programming. In GRASP, code folding may be done according to the programming language control structure boundaries as well as for arbitrary, user-selected code parts.

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

2.3. Software documentation in the age of IDEs

Nowadays, most software development is done with the help of sophisticated Integrated Development Environments (IDEs) like Microsoft Visual Studio [43], IBM Visual Age [27], Borland JBuilder [9], NetBeans [7] or Source Navigator [51] to name just a few of them. These development environments organize the programming tasks in so called projects, which contain all the source files, resources and libraries necessary to build such a project.

One of the main features of these IDEs is their ability to parse all the files which belong to a project and build a database out of that information. Because the files of the project can be usually modified only through the builtin editor, the IDEs can always keep track of changes in the source files and update the project database on the fly.

With the help of the project database, the IDEs can offer a lot of services like fast, qualified searching or dependency-, call-, and inheritance graphs. They allow fast browsing of methods and classes and direct access from variables, method calls or class instantiations to their definitions. Notice that all these features are available online during the work on a project, in contrast to the tools like JavaDoc or Doxygen mentioned in the previous section which provide this information only offline.

The new IDEs now deliver under such fancy names like *Code Completion* or *Code Insight* features like syntax directed programming [30] or template based programming which have been proposed already in the late seventies by [62, 44]. In the past, these systems could not succeed because of two main reasons: they where to restrictive in the burden they put on the programmer and the display technology and computing power have not been sufficient**. However, the enhancements in the area of user interfaces and the computational power available today allow even more: context sensitive prompting of the user with the names of available methods or with the formal arguments of a method, syntax highlighting and fast recompilation of affected source code parts.

All this reduces the benefits of a printed, highly linked and indexed documentation of a whole project. What is needed instead, additionally to the interface description provided by the IDE, is a description of the algorithms and of certain complex code parts. One step into this direction was Sametinger's DOgMA [53, 54] tool which is an IDE that also allows writing documentation. DOgMA, like modern IDEs today, maintains an internal database of the whole parsed project. It allows the programmer to reference arbitrary parts of the source code in the documentation while DOgMA automatically creates the relevant links between the source code parts and the documentation and keeps them up to date. These links allow a hypertext like navigation between source code and documentation.

While it seems that modern IDEs adopted a lot of DOgMA's browsing capabilities, they didn't adopted its literate programming features. However, systems like NetBeans [7], SourceNavigator [51] or VisualAge [60]) offer an API for accessing the internal program database. This at least would allow one to create extensions of these systems in order to support program documentation in a more comfortable way.

The most ambitious project in this context in the last few years was certainly the *Intentional Programming* project lead by Charles Simonyi [58, 59] at Microsoft. It revitalized the idea of structured programming and propagated the idea of programs being just instantiations of intentions.

**A good survey about the editor technology available at the beginning of the eighties can be found in [42].

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

The intentions could be written with a fully fledged WYSIWYG editor which allowed arbitrary content to be associated with the source code. Of course, this makes it easy to combine and maintain software together with the appropriate documentation. Some screen-shots of this impressive system can be found in chapter 11 of [15], which is dedicated solely to Intentional Programming. Unfortunately, this system was never made publicaly available.

2.4. Software documentation and XML

With the widespread use of XML [72] in the last few years it is not surprising that various XML formats have been proposed to break out of the "ASCII Straitjacket" [1] in which programming languages are caught until now. While earlier approaches to widen the character set out of which programs are composed like [1] failed mainly because of the lack of standards in this area, the standardization of UNICODE [63] and XML may change the situation now.

There exist two concurring approaches. While for example JavaML [4] tries to define an abstract syntax tree representation of the Java language in XML (which, by the way, is not dissimilar from the internal representation proposed by the early syntax directed editors) the CSF [55] approach tries to define an abstract XML format usable as intermediate representation by most of the current programming languages. Both have advantages as well as disadvantages. While the first one suffers from it's dependency on a certain programming language, the second one will always fail to represent every exotic feature of every given programming language. A third, minimalistic approach could completely ignore the syntax of the programming language and just store program lines and comments into as few as two different XML elements. Such an encoding has been proposed by E. Armstrong [3].

However, independent of the encoding's actual representation, once that such an encoding would be available, literate programming and program documentation systems could greatly benefit from it. They could reference distinct parts of a source file in a standard way or they could insert special attributes or elements into the XML document which could be ignored by other tools like compilers or build systems. Standard tools could be used to process, edit and display the source files, and internal as well as external links could be added to the source code.

Peter Pierrou presented in [48] an XML literate programming system. In fact it consists of an XML editor which allows one to store source code, documentation and links between them into an XML file. A tangle script is used to extract the source code out of the XML file. The system is very similar to the reverse literate programming tool proposed by Knasmüller, with the only difference that it is independent of the source language and stores its data in XML format. An earlier, but very similar effort described in [21] used SGML as markup language for storing documentation and source code.

Anthony Coates introduced xmLP [14], a literate programming system which uses some simple XML elements as markup. The idea is to use these elements together with other markup elements, for example those defined in XHTML [70], MathML [69] or DocBook [66]. XSLT [75] stylesheets are then used in order to produce the woven documentation and the tangled output files. A similar system has also been presented by Norman Walsh [65], the Author of DocBook. He introduces a few elements for source fragments which are located in their own namespace. Thus every XML vocabulary which allows the inclusion of new elements from a different namespace may be used to write the literate program. Finally XSLT stylesheets are used to weave and to tangle the literate program.

Copyright © 2003 John Wiley & Sons, Ltd.

Draft
Don't redistribute or cite without explicit permission!

Oleg Kiselyov suggested the representation of XML as an s-expression in Scheme called SXML [32]. SXML can be used to write literate XML programs. Different Scheme programs (also called stylesheets in this case) are available to convert from SXML to LaTeX, HTML or pure XML files.

Recently the Boost Initiative [8], an effort to provide free, peer-reviewed and portable C++ source libraries has started a new project called BoostDoc [23]. The goal of the project is to document all the Boost libraries in a consistent way and to keep the documentation synchronised with the constantly developing libraries. BoostDoc uses various tools like Doxygen [24] or Synopsis [16] to create an API documentation in XML format out of the library source files. This API documentation is later merged with the BoostBook documentation written by the programmer, where BoostBook is an extension of the DocBook [66] format specially tailored for C++ library documentation.

Some of the approaches presented in this section are quite new, but the acceptance of XML in the area of the source code representation of programming languages could give new impulses to the literate programming community. A good starting point for more information on literate programming and XML is the Web site of the OASIS consortium, which hosts a page specifically dedicated to this topic [46].

3. The new approach

After having discussed so many different approaches it is time now to present the documentation system advertised in the title of this paper. The new system is based on two properties fulfilled by virtually every programming language.

- Every programming language is based on a formal grammar and every compiler or interpreter internally builds a parse tree of a program when parsing it. Therefore it would be easy for each such tool to dump the parse tree in a XML format standardized on a per language basis.
- Every programming language offers line comments. Defining some of these comments to have a special semantic would enable the compiler to produce additional markup in the XML version of the parsed file. This comment format should also be standardized on a per language basis.

Once the two requirements postulated above are fulfilled it becomes easy to produce interface as well as implementation documentation from the resulting XML source code representation by using standard tools like XInclude [71] or XSLT [75] processors.

Taking into account the XML elements introduced by the programmer with the special comments presupposed before, it is possible to address arbitrary code parts and include them into the documentation. Given the standardized XML format, it becomes trivial to include source code into the documentation based on syntactic information (e.g. including a class or method definition by name). And finally an API documentation could be generated automatically by extracting the interface part together with possible documentation comments (eg. JavaDoc, C# or Doxygen style comments) from the XML representation.

The key point is in fact the per language standardization of the proposed special comment scheme and the XML representation of the source code because it will permit the development of documentation tools with respect to a standardized interface. How such a tool may look like will be demonstrated in section 4 while the following two subsections will discuss the special comment format and the representation of the sourc code in XML.

Copyright © 2003 John Wiley & Sons, Ltd.
Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

3.1. The comment format

For the semantics of the special line comments we propose the following simple extension to usual line comments:
line-comment-token ’<’|’>’|’<>’ *element-name* {*attribute*=*value*}*
where *line-comment-token* is the token which introduces a line comment in the specific programming language (e.g. ’`//`’ in C++/Java or ’`#`’ in AWK) and *element-name* denotes the name of the resulting XML element. If the character following the comment token is ’<’, the result will be an opening tag for the corresponding element, if the character is ’>’ the compiler will generate a closing tag for the corresponding element, and finally a ’<>’ after the comment token will introduce an empty element. All the additional text after *element-name* will be copied verbatim into the resulting element tag and should contain valid XML attributes in order to produce a well-formed XML document.

Notice that introducing opening and closing tags for an element has to be done in such a way that they do not intersect with the opening and closing tags produced by the compiler for certain programming language constructs. So for example placing a comment which will produce an opening tag just before a while loop and the comment for the closing tag inside that loop will in general produce a XML document which is not well formed because the introduced tags will overlap with the opening and closing tags of the while loop. Such errors however can be detected easily by the compiler.

The advantage of the fact that comments for opening and closing tags have to align with the structure of the program is that they can be used also as anchors for user defined code folding [26, 33]. For example jEdit [47], a cross platform, programmer’s text editor written by Slava Pestov uses ’`//{{{` *text*’ and ’`//}}}`’ line comments to specify the beginning and the end of a text fold. Unifying these notations would enable code folding for source code marked up with documentation comments as well as inclusion of arbitrary predefined code folds into the documentation. Notice that although the opening and closing comments may not overlap they may be nested.

3.2. The XML representation

Because most programming languages are defined by a grammar anyway, the simplest approach would be to define a XML DTD or a XML Schema [73] based on that grammar. There also exist already a number of XML mappings for various programming languages like for example JavaML [4] for Java or the generic approaches of Armstrong [3] and Sandø [55]. Like for the syntax of the documentation comment, the crucial point here is that is is highly desirable for the XML representation to be standardized together with the corresponding programming language in order to enable compatibility of code and interoperability of tools.

It has to be stressed however that we do not necessarily need a full compiler in order to create the XML representation. Tools like DOC++ [76] or Doxygen [24] which only partially parse the source file may be fully adequate. The advantage of using a full fledged compiler for this purpose would be the additional information which could be gathered like overload resolution or exact type information for every identifier in the source code. This information could be used for example for cross linking in the generated documentation.

Also, it is not strictly necessary to store the XML representation in files corresponding to the underlying source files. It can be useful storing this information in a database which may save time in the face of recompilation or it may simplify querying the information for big projects.

Copyright © 2003 John Wiley & Sons, Ltd.
Prepared using speauth.cls

Draft
Don’t redistribute or cite without explicit permission!

3.3. Advantages and drawbacks of the new approach

The proposed documentation system fulfills the first three properties postulated in section 2. The code and the documentation can be kept synchronized although they are mutually independent. Only the syntax of the new documentation comment has to be learned by the programmer. And finally, as stated before, the generated XML representation can be used to produce interface as well as implementation documentation.

Legibility, appealing look and eligibility for different output formats which was the fourth property from section 2, are mainly dependent on the typesetting system actually used. However, XML based documentation system are widely used and the prototype presented in section 4 which is based on DocBook [66] demonstrates the strength of this approach. Finally, interoperability and team and tool support is granted through the wide acceptance and support of XML and XML related technologies as industry standards.

One last benefit of the proposed documentation style is its applicability to multilingual documentation because once the relevant code parts have been identified and marked they can be included in the same way into arbitrary documents. With the Literate Programming approach described in section 2.1 several versions of the same documentation in different languages are not possible without duplication of the source code which is embedded inside the documentation. Also, even if possible, embedding all the documentation into the source code as for example with the JavaDoc style would become confusing already with the second language because the source code would contain more comments than actual program code. These arguments of course apply not only to multilingual documentation, but also for the case where different kinds of documentation (e.g. user documentation, developer documentation) have to be created for the same code.

There are two major drawbacks of the new documentation system. First of all, standardizing a computer language is a complicated and intricate task. Therefore adding the proposed extensions to the definition of already existing languages will be not easy. However there may be a good chance for the user community of each programming language to establish a De-facto standard for these extensions.

For some programming languages like C/C++ which use a preprocessor it may be difficult to reconstruct the source representation from the abstract syntax tree available to the compiler because the preprocessor step can potentially replace and change the source code. In particular the C/C++ preprocessor simply strips all the comments from the source code before feeding it to the compiler. Therefore tools like GCC-XML [31], an extension of the GNU C++ compiler [20] by Brad King which generates an XML description of a C++ program from GCC's internal representation, does not handle comments at all. However other tools like Synopsis [16] or techniques similar to the ones described in [5] may be used to overcome this problem.

4. A prototype implementation

This section will present a prototype implementation of the ideas presented in the last section. The prototype works for the Java programming language and uses DocBook for writing the documentation along with the DocBook XSL-FO stylesheets and a FO [74] processor to produce PDF documentation. Two pages of a resulting document are shown in Figure 6 and 7, respectively. Notice that the two pages

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

```
/**
 * A quick sort demonstration algorithm
 *
 * @author James Gosling
 * @author Kevin A. Smith
 * @version 1.3, 29 Feb 1996
 */
public class QSortAlgorithm {

  /** A generic version of C.A.R Hoare's Quick Sort algorithm.
   * It handles sorted arrays, and arrays with duplicate keys.
   *
   * If you think of a one dimensional array as going from
   * the lowest index on the left to the highest index on the right
   * then the parameters to this function are lowest index or
   * left and highest index or right.  The first time you call
   * this function it will be with the parameters 0, a.length - 1.
   *
   * @param a       an integer array
   * @param lo0     left boundary of array partition
   * @param hi0     right boundary of array partition
   * @return        returns nothing, just for demonstration purpose
   */
  //< Include ID="QSMethod" label='The whole "QuickSort" method.'
  public static void QuickSort(int a[], int lo0, int hi0) {
    int lo = lo0;
    int hi = hi0;
    int mid;

    if ( hi0 > lo0) {
      // Arbitrarily establishing partition element as the midpoint of
      // the array.
      //
      mid = a[ ( lo0 + hi0 ) / 2 ];

      //< Include ID="whileLoop" label="Loop through the array until indices cross"
      while( lo <= hi ) {
        // find the first element that is greater than or equal to
        // the partition element starting from the left Index.
        //
        while( ( lo < hi0 ) && ( a[lo] < mid ) ) {
          ++lo;
        }

        // find an element that is smaller than or equal to
        // the partition element starting from the right Index.
        //
        while( ( hi > lo0 ) && ( a[hi] > mid ) ) --hi;

```

Figure 1. The first part of the example program `QSortAlgorithm.java`.

were in A4 format initially and have been shrinked by a factor of 0.6 in order to fit the layout of this journal.

For the prototype the Java compiler which is available as a part of the Java Specification Request 14 [28] dedicated to adding Generics to the Java programming language has been used and extended. As XSLT processor version 6.5.2 of Michael Kay's Saxon [29] has been choosen. Furthermore, version 4.1.2 of the DocBook DTD and version 1.60.1 of the DocBook XSL-FO stylesheets [64] have been used and extended. As a last step the resulting XML document was run through the RenderX Formatting Object engine [52] to produce the final PDF version. The next three subsections will

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

```
        // if the indexes have not crossed, swap
        if( lo <= hi ) {
          swap(a, lo, hi);
          ++lo;
          --hi;
        }
      }
      //> Include

      //If the right index has not reached the left side of array
      //  must now sort the left partition.
      //
      if( lo0 < hi ) QuickSort( a, lo0, hi );

      // If the left index has not reached the right side of array
      // must now sort the right partition.
      //
      if( lo < hi0 ) QuickSort( a, lo, hi0 );

    }
  }
  //> Include

  public static void sort(int a[]) {
    QuickSort(a, 0, a.length - 1);
  }

  private static void swap(int a[], int i, int j) {
    int T;
    T = a[i];
    a[i] = a[j];
    a[j] = T;
  }

  private static void print(int a[]) {
    for(int i = 0; i < a.length; i++) {
      if (i > 0) System.out.print(", ");
      System.out.print(a[i]);
    }
    System.out.println();
  }

  public static void main(String argv[]) {
    int test[] = new int[] { 9, 5, 2, 6, 2, 7, 5, 1, 0, 4};
    print(test);
    sort(test);
    print(test);
  }
}
```

Figure 2. The second part of the example program `QSortAlgorithm.java`.

describe the extensions in more detail and explain how the mentioned systems work together to produce the final documentation.

In order to demonstrate the possibilities of the system a slightly modified version of a Quicksort class written by James Gosling and Kevin Smith which is presented verbatim in the Figures 1 and 2 will be used. Along with the implementation of the prototype it is available for download from http://www.progdoc.org/xprogdoc.

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

```
100 <!-- empty-line -->
101 <Include ID="whileLoop" label="Loop through the array until indices cross" line="38" colum="7">
102   <while-loop  line="39" colum="7">
103     <condition  line="39" colum="7">
104       <binary-expression operator="&lt;=" line="39" colum="17">
105         <var name="lo"  line="39" colum="14"/>
106         <var name="hi"  line="39" colum="20"/>
107       </binary-expression>
108     </condition>
109     <body single="false"  line="39" colum="7">
110       <comment value=" find the first element that is greater than or equal to " line="40" colum="9"/>
111       <comment value=" the partition element starting from the left Index." line="41" colum="9"/>
112       <comment value="" line="42" colum="9"/>
113       <while-loop  line="43" colum="9">
114         <condition  line="43" colum="9">
```

Figure 3. Some lines of the compiler generated file `QSortAlgorithm.xml`. These lines correspond to the Java source from line 35 to the first opening brace at line 41 in Figure 1.

4.1. Extending the Java compiler

The decision to use the JSR 14 prototype compiler was made because of two main reasons. First of all it offers the chance to immediately support the Generic Java constructs which will be added to the Java programming language in version 1.5 of the Java Development Kit. The second reason was the fact that the compiler is implemented in a clear and well structured way and contains a nice, easy to understand recursive descend parser. The compiler source code is available for free download at [28].

The compiler was extended to support the new command line option '`-x`', which instructs the compiler to dump the Java files given on the command line in XML format. An example of how this output looks like is presented in Figure 3. Notice the `Include` element at line 101 in Figure 3 which was introduced by the special comment at line 36 in `QSortAlgorithm.java`. The `label` attribute of this element is used in Listing 1 in Figure 6 in order to denote the content omitted from the listing. Also notice the fact that empty lines of the Java source file are represented by special XML comments like the one at line 100 in Figure 3. Though not strictly necessary, this information is preserved in order to simplify the production of the formatted Java output in a later step.

The mapping of the Java language to XML elements is straightforward. General language constructs are mapped to corresponding XML elements. Sometimes additional attributes are used to further describe the construct (e.g. the `operator` attribute for the `binary-expression` element (see Figure 3 line 104) or the `visibility` attribute for `method`, `var-def` and `class` elements). Additionally, every element has a `line` and a `column` attribute which denotes the exact position of the corresponding construct in the Java source file.

The first step in order to achieve these results was the introduction of two new tokens into the scanner part of the Java compiler. One token for line comments and one for empty lines. Notice that the scanner originally skipped all comments except the special JavaDoc comments. They where just stored in a symbol table along with the class definition or variable declaration they belong to and not reported directly to the parser.

The parser was changed to accept the new tokens. Therefore the production rule for *BlockStatement* [22, §14.2] was changed to accept line comments and empty lines alternatively to usual language

Copyright © 2003 John Wiley & Sons, Ltd.
Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

```
<!--
   ProgDocBook DTD version 0.1

   This DTD module is identified by the PUBLIC and SYSTEM identifiers:

   PUBLIC "-//OASIS//DTD DocBook XML V4.1.2 Extension ProgDoc V0.1//EN">
   SYSTEM "http://www.progdoc.org/xdoc/dtd/ProgDocBook.dtd"
-->

<!--========== Add 'Listing' and 'SourceBase' elements to the DTD ==========-->

<!ENTITY % local.formal.class "| Listing | SourceBase">


<!--========== Define the new 'Listing' and 'SourceBase' elements ==========-->

<!ELEMENT SourceBase EMPTY>
<!ATTLIST SourceBase xml:base CDATA #REQUIRED>

<!ELEMENT Listing EMPTY>
<!ATTLIST Listing recursive-include (true|false) "false"
                  java-doc (true|false) "false"
                  type (include|select|api) "include"
                  href CDATA #REQUIRED
                  anchor ID #REQUIRED
                  kind CDATA #IMPLIED
                  name CDATA #IMPLIED
>

<!--=== Allow SourceBase and Listing inside chapter or section elements  ===-->

<!ENTITY % local.divcomponent.mix "| SourceBase | Listing">

<!--========================= Import DocBook DTD =========================-->

<!ENTITY % DocBookDTD PUBLIC
          "-//OASIS//DTD DocBook XML V4.1.2//EN"
          "http://www.oasis-open.org/docbook/xml/4.0/docbookx.dtd">

%DocBookDTD;
```

Figure 4. The extension of the DocBook DTD.

statements. The productions for *ClassBodyDeclaration* [22, §8.1] and *InterfaceMemberDeclaration* [22, §9.1] were changed to additionally accept line comments and empty lines.

The parser builds an abstract syntax tree of the source code which is processed and augmented in turn by various transformers which perform task like resolving names, doing flow analysis, optimization and code generation. All these transformers where changed to simply ignore the subtrees representing line comments and empty lines. Finally, a new transformer was written which dumps the abstract syntax tree in XML format. This transformer will be prepented to the chain of transformers right after parsing has been finished if the compiler is given the new '`-x`' command line option.

4.2. Extending DocBook

The DocBook DTD was extended by two new elements as shown in Figure 4. The first one, `SourceBase`, has the single, required attribute `xml:base`. It can be used to specify a base path

Copyright © 2003 John Wiley & Sons, Ltd.
Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

under which source files considered for inclusion will be searched. `Listing`, the second element, can be used to include parts of a source file into the documentation. It has several attributes which will be described briefly now. The `href` attribute which is required denotes the file from which the code will be included. It will be interpreted relatively to the path which was set by the last `SourceBase` element if there was one at all. The `type` attribute which is also a required one is used to specify the kind of listing to produce. Setting the value of this attribute to `include` will tell the stylesheet by which the DocBook document will be processed to include all the code contained in the `Include` element with an ID attribute which is equal to the `anchor` attribute of the actual `Listing` element. Remember that such elements can be introduced by the programmer with the special documentation comments described in section 3.1 (e.g. line 24 and 36 in Figure 1). The first listing of the example document shown in Figure 6 was included by the following command:

```
<Listing href="QSortAlgorithm.java" type="include" anchor="QSMethod"
         recursive-include="false" java-doc="false"/>
```

Because the `recursive-include` attribute is set to `false` the nested `Include` element which spans the lines 36 to 57 in Figure 1 and 2 is not included into the documentation. Instead it is replaced by a link to the listing which contains these lines if the author decides to also include them, as it has been done in Listing 2 of Figure 6. Otherwise, a notice that the lines are not shown in the actual documentation will be printed.

Notice the use of the `label` attribute which can be declared in the documentation comment of the programming language (line 36 in Figure 1). It is used internally by the stylesheet during the transformation as a short description of nested code parts and does not have to be specified in the extended DocBook DTD.

`java-doc`, the last attribute in the example given above, instructs the stylesheet not to show JavaDoc comments which appear in the included source code. As a second example consider the following line which has been used to include Listing 4 into the document shown in Figure 7:

```
<Listing href="QSortAlgorithm.java" type="select" kind="class"
         name="QSortAlgorithm" anchor="classQSortAlgorithm"
         recursive-include="false" java-doc="true"/>
```

It sets the `type` attribute to `select` thus including not a range of code specified by the programmer, but a syntactic entity of the programming language which is specified by the additional attributes `kind` and `name`. Consequently, this example includes the source code of the whole `QSortAlgorithm` class. Notice that this time the JavaDoc comments which belong to the class are shown because the `JavaDoc` attribute is set to `true`. The `recursive-include` attribute is still set to `false` which prevents the inclusion of the `Quicksort` method because it is embedded into special documentation comments by the programmer. Instead it is replaced by a line with a link to the actual listing and the short description given with the `label` attribute in the source code (line 24 in Figure 1).

The last example shown below demonstrates how API documentation in JavaDoc format can be included into the documentation by setting the `type` attribute to `api`:

```
<Listing href="QSortAlgorithm.java" type="api" kind="method"
         name="QuickSort" anchor="QuickSortAPI"/>
```

Copyright © 2003 John Wiley & Sons, Ltd.

Draft
Don't redistribute or cite without explicit permission!

```
<xsl:template match="/">
  <!-- First of all we do the inclusion stage -->
  <xsl:variable name="doc1">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()|comment()" mode="IncludeMode"/>
    </xsl:copy>
  </xsl:variable>
  <!-- In a second step we pretty-print and link the included source code -->
  <xsl:variable name="doc2">
    <xsl:copy>
      <xsl:apply-templates select="$doc1" mode="PrettyPrint"/>
    </xsl:copy>
  </xsl:variable>
  <!-- Finally we hand over control to the original rule from 'docbook.xsl' -->
  <!-- which will be applied now to the new root element.                  -->
  <xsl:apply-templates select="$doc2" mode="originalRootRule"/>
</xsl:template>
```

Figure 5. The main transformation rule of the XSL-FO stylesheet in `ProgDocBookFO.xsl`.

The result of this example can be seen in Listing 5 in Figure 7. It contains the JavaDoc API documentation (lines 10 to 23 in Figure 1) for the `QuickSort` method in a nicely formatted way. Notice furthermore that the `anchor` attribute can additionally be used as a target for cross referencing, no matter of the value of the `type` attribute. In the PDF version of the example document, references like for example "see Listing 2" are true hyperlinks which can be navigated.

4.3. Extending the DocBook XSL-FO stylesheets

While the extension of the DocBook DTD required only a few lines of code, extending the DocBook XSL-FO stylesheets which produces formatting objects output from an input file which conforms to the newly defined DTD, proved much harder.

All the functionality of the new DocBook elements and attributes described in the previous section is effectively implemented in the extended stylesheet. It uses an XSLT 1.1 feature which treats result tree fragments as real node sets and different modes to implement a three step policy during the XSL transformation. As shown in Figure 5, the first step is used to include source code parts identified by corresponding `Listing` elements. In this step all the original DocBook elements are just copied recursively to a temporary tree and the `document()` function is used to replace the `Listing` elements with the actual source code from the source files in XML format.

The second step uses the newly constructed tree and transforms the XML version of the Java source code into valid DocBook elements. Like in the first step the original DocBook elements are just copied to the new tree. The second step is also used to establish the automatic links between nested code parts. It is the most elaborate step with more than 500 lines of code because a transformation rule for every single element which can appear in the XML version of the source file is needed.

In the third and last step, the root rule of the original XSL-FO stylesheet is called with the second intermediate tree as argument. At this stage the tree contains only valid DocBook elements an can be transformed into a formatting object file.

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

Software Documentation Test

Volker Simonis

This is just a test article in order to demonstrate the software documentation style proposed in the article "A Universal Documentation Extension for Arbitrary Programming Languages". It was written in XML using the DocBook DTD, transformed to Formatting Objects and finally translated into PDF by a FO engine. More information can be found in the enclosing article.

An implementation of the Quicksort algorithm

In this section we will present an implementation of the Quicksort algorithm in the Java programming language. Listing 1 gives an overview of the sort method.

Listing 1. QSortAlgorithm.java [Lines 25 to 70]

```
public static void QuickSort(int[] a, int lo0, int hi0) {
  int lo = lo0;
  int hi = hi0;
  int mid;

  if(hi0 > lo0) {
    // Arbitrarily establishing partition element as the midpoint of
    // the array.
    //
    mid = a[(lo0 + hi0) / 2];

    <Loop through the array until indices cross (see Listing 2)>

    //If the right index has not reached the left side of array
    //  must now sort the left partition.
    //
    if(lo0 < hi) QuickSort(a, lo0, hi);

    // If the left index has not reached the right side of array
    // must now sort the right partition.
    //
    if(lo < hi0) QuickSort(a, lo, hi0);

  }
}
```

Because of brevity, some details of the algorithm have been omitted in Listing 1. They will be presented in the next program listing:

Listing 2. QSortAlgorithm.java [Lines 37 to 56] (Referenced in Listing 1)

```
while(lo <= hi) {
  // find the first element that is greater than or equal to
  // the partition element starting from the left Index.
  //
  while((lo < hi0) && (a[lo] < mid)) {
    ++lo;
  }

  // find an element that is smaller than or equal to
  // the partition element starting from the right Index.
  //
  while((hi > lo0) && (a[hi] > mid)) --hi;

  // if the indexes have not crossed, swap
  if(lo <= hi) {
    swap(a, lo, hi);
    ++lo;
    --hi;
  }
}
```

The Quicksort class also contains a small test program to verify the algorithm:

Figure 6. The first page of the example document.

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

Listing 3. QSortAlgorithm.java [Lines 92 to 97]

```
public static void main(String[] argv) {
  int[] test = new int[] { 9, 5, 2, 6, 2, 7, 5, 1, 0, 4 };
  print(test);
  sort(test);
  print(test);
}
```

Listing 4 shows the whole source file one more time:

Listing 4. QSortAlgorithm.java [Lines 8 to 98]

```
/**
 * A quick sort demonstration algorithm
 *
 * @author James Gosling
 * @author Kevin A. Smith
 * @version 1.3, 29 Feb 1996
 */
public class QSortAlgorithm {

  <The whole "QuickSort" method. (see Listing 1)>

  public static void sort(int[] a) {
    QuickSort(a, 0, a.length - 1);
  }

  private static void swap(int[] a, int i, int j) {
    int T;
    T = a[i];
    a[i] = a[j];
    a[j] = T;
  }

  private static void print(int[] a) {
    for(int i = 0;i < a.length;i++) {
      if(i > 0) System.out.print(", ");
      System.out.print(a[i]);
    }
    System.out.println();
  }

  public static void main(String[] argv) {
    int[] test = new int[] { 9, 5, 2, 6, 2, 7, 5, 1, 0, 4 };
    print(test);
    sort(test);
    print(test);
  }
}
```

Listing 5 finally presents the JavaDoc documentation of the QuickSort method shown already in Listing 1.

Listing 5. Method QuickSort: A generic version of C.A.R Hoare's Quick Sort algorithm

A generic version of C.A.R Hoare's Quick Sort algorithm. It handles sorted arrays, and arrays with duplicate keys.

If you think of a one dimensional array as going from the lowest index on the left to the highest index on the right then the parameters to this function are lowest index or left and highest index or right. The first time you call this function it will be with the parameters 0, a.length - 1.

```
public static void
QuickSort(int[] a, int lo0, int hi0);
```

Parameters:	
`a`	an integer array
`lo0`	left boundary of array partition
`hi0`	right boundary of array partition
Return Value:	
	returns nothing, just for demonstration purpose

Figure 7. The second page of the example document.

Copyright © 2003 John Wiley & Sons, Ltd.
Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

Finally the XSL-FO stylesheet also contains some local customizations and some rules for a new DocBook element called `listing`. This element is effectively handled in the same way as the DocBook `example` element. It was only necessary to introduce it because listings have their own label, referencing style and numbering. Because the `listing` elements are created only in the second temporary tree during the transformation, they do not have to be declared in the extended DocBook DTD.

5. Conclusion

The new documentation style complies to the five demands postulated in section 2. It combines and uses well known and established techniques for documentation purpose in a new and effective way and proposes the standardization of the comment style together with the format of the XML representation as a an integral part of every programming language. Therewith, documenting becomes vendor and implementation independent in the same way as programming became vendor and implementation independent by the standardization of programming languages.

ACKNOWLEDGEMENTS

A lot of tools have been used to prepare this paper and to implement the prototype presented therein. I want to thank the authors and companies which made these tools available. They will be listed here in no particular order: Sun for the source code of their experimental Generic Java compiler, RenderX for their Formatting Objects engine, Michael Kay for Saxon, his free XSLT processor, Norman Walsh for the DocBook DTD and his DocBook XSL stylesheets and all the authors of the various documentation systems mentioned in this paper because they all inspired me to a greater or lesser extent. Finally I want to thank Rüdiger Loos and Roland Weiss for many discussions on the topic of program documentation and for revising this paper.

REFERENCES

1. P. W. Abrahams. *Typographical Extensions for Programming Languages: Breaking out of the ASCII Straitjacket.* ACM SIGPLAN Notices, Vol. 28, No. 2, Feb. 1993
2. A.W. Aho, B.W. Kernighan and P. J.Weinberger. *The AWK Programming Language.* Addison-Wesley, 1988
3. E. Armstrong. *Encoding Source in XML - A strategic Analysis.* http://www.treelight.com/software/encodingSource.html
4. G. J. Badros. *JavaML: A Markup Language for Java Source Code.* 9th Int. WWW-Conference, Amsterdam, May 2000
5. G. J. Badros and D. Notkin. *A Framework for Preprocessor-Aware C Source Code Analyses.* Software - Practice & Experience, Vol. 30, No. 8, July 2000
6. Ronald M. Baecker, Aaron Marcus. *Human Factors and Typography for More Readable Programs.* Addison-Wesley, 1990
7. T. Boudreau, J. Glick, S. Greene, V. Spurlin, J. Woehr. *NetBeans: The Definitive Guide.* O Reilly & Associates, 2002, http://www.netbeans.org/download/books/definitive-guide/
8. *The Boost Library.* http://www.boost.org
9. Borland Software Corporation. *Borland JBuilder.* http://www.borland.com/jbuilder
10. Preston Briggs. *nuWeb*, http://ctan.tug.org/tex-archive/web/nuweb
11. ECMA 334, ISO/IEC 23270 *C# Language Specification.* http://www.ecma-international.org/publications/standards/ecma-334.htm
12. Bart Childs *Literate Programming, A Practitioner's View* TUGboat, Volume 13, No. 2, 1992, http://www.literateprogramming.com/farticles.html
13. B. Childs and J. Sametinger. *Analysis of Literate Programs from the Viewpoint of Reuse.* Software - Concepts and Tools, Vol. 18, No. 2, 1997, http://www.literateprogramming.com/farticles.html

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!

14. A. B. Coates and Z. Rendon *xmLP - a Literate Programming Tool for XML & Text.* Extreme Markup Languages, Montreal, Quebec, Canada, August 2002, http://xmlp.sourceforge.net/2002/extreme/
15. K. Czarnecki and U. W. Eisenecker. *Generative Programming.* Addison-Wesley, 2000
16. Stephen Davies and Stefan Seefeld. *Synopsis.* http://synopsis.sourceforge.net
17. Jason Diamond. *NDoc.* http://ndoc.sourceforge.net/
18. by Nikos Drakos and Ross Moore. *Latex2HTML.* http://saftsack.fs.uni-bayreuth.de/˜latex2ht/ or: http://ctan.tug.org/ctan/tex-archive/support/latex2html
19. Free Software Foundation *The Fast Lexical Analyzer.* http://www.gnu.org/software/flex/
20. Free Software Foundation *The GNU Compiler Collection.* http://gcc.gnu.org
21. D.M. German, D.D. Cowan and A. Ryman. *SGML-Lite – An SGML-based Programming Environment for Literate Programming.* ISACC, Oct. 1996, http://www.oasis-open.org/cover/germanisacc96-ps.gz
22. J. Gosling, B. Joy and G. Steele *"Java Language Specification"* Addison-Wesley, 1996
23. Douglas Gregor. *The BoostBook Documentation Format*, http://www.cs.rpi.edu/˜gregod/boost/tools/boostbook/doc/html/
24. Dimitri van Heesch. *Doxygen.* http://www.doxygen.org
25. Carsten Heinz *"The Listings package"*, ftp://ftp.dante.de/tex-archive/help/Catalogue/entries/listings.html
26. T. D. Hendrix, J. H. Cross II, L. A. Barowski and K. S. Mathias. *Visual Support for Incremental Abstraction and Refinement in Ada95.* SIGAda Ada Letters, Vol. 18, No. 6, 1998
27. IBM Corporation. *Visual Age C++.* http://www-3.ibm.com/software/ad/vacpp
28. Java Community Process - Java Specification Request 14. *Adding Generics to the Java Programming Language.* http://jcp.org/aboutJava/communityprocess/review/jsr014/index.html http://developer.java.sun.com/developer/earlyAccess/adding˙generics
29. Michael Kay. *XSLT Programmer's Reference* Wrox Press Ltd., Birmingham, UK, 2002 http://saxon.sourceforge.net
30. A. A. Khwaja and J. E. Urban. *Syntax-Directed Editing Environments: Issues and Features.* ACM SIGAPP Symposium on Applied Computing, Indianapolis, Indiana, 1993
31. Brad King. *GCC-XML, the XML output extension to GCC!* http://www.gccxml.org/HTML/Index.html
32. O. Kiselyov. *SXML Specification.* ACM SIGPLAN Notices, Volume 37, Issue 6, June 2002 http://pobox.com/˜oleg/ftp/Scheme/xml.html
33. M. Knasmüller. *Reverse Literate Programming.* Proc. of the 5th Software Quality Conference, Dundee, July 1996
34. Donald E. Knuth *Literate Programming* The Computer Journal, Vol. 27, No. 2, 1984
35. Donald E. Knuth *The TEXbook* Addison-Wesley, Reading, Mass., 11. ed., 1991
36. Donald E. Knuth *TEX: The Program* Addison-Wesley, Reading, Mass., 4. ed., 1991
37. Donald E. Knuth *Literate Programming* CSLI Lecture Notes, no. 27, 1992 or Cambridge University Press
38. Donald. E. Knuth and Silvio Levy *The CWEB System of Structured Documentation* Addison-Wesley, Reading, Mass., 1993
39. John Krommes. *fWeb.* http://w3.pppl.gov/˜krommes/fweb.html
40. Marc van Leeuwen. *CWebx.* http://wallis.univ-poitiers.fr/˜maavl/CWEBx/
41. Mark Lutz. *Programming Python.* O Reilly & Associates, 2nd. ed., 2001
42. N. Meyrowitz and A. van Dam. *Interactive Editing Systems: Part I and II.* Computing Surveys, Vol. 14, No. 3, Sept. 1982
43. Microsoft Corporation. *Visual Studio.* http://msdn.microsoft.com/vstudio
44. J. Morris and M. Schwartz. *The Design of a Language- Directed Editor for Block-Structured Languages.* SIGLAN/SIGOA Symp. on text manipulation, Portland, 1981
45. H. Mössenböck and K. Koskimies. *Active Text for Structuring and Understanding Source Code.* Software - Practice & Experience, Vol. 27, No. 7, July 1996
46. The Oasis Consortium. *SGML/XML and Literate Programming.* http://www.oasis-open.org/cover/xmlLitProg.html
47. Slava Pestov, *jEdit - Open Source programmer's text editor.* http://www.jedit.org
48. P. Pierrou. *Literate Programming in XML.* Markup Technologies, Philadelphia, Pennsylvania, US, Dec. 1999, http://www.literateprogramming.com/farticles.html
49. N. Ramsey and C. Marceau *Literate Programming on a Team Project* Software - Practice & Experience, 21(7), Jul. 1991, http://www.literateprogramming.com/farticles.html
50. Norman Ramsey *Literate Programming Simplified* IEEE Software, Sep. 1994, p. 97 http://www.eecs.harvard.edu/˜nr/noweb/intro.html
51. Red Hat, Inc. *Source Navigator.* http://sourcenav.sourceforge.net
52. RenderX, Inc. *XEP Rendering Engine.* http://www.renderx.com/FO2PDF.html
53. J. Samtinger *DOgMA: A Tool for the Documentation & Maintenance of Software Systems.* Tech. Report, 1991, Inst. für Wirtschaftsinformatik, J. Kepler Univ., Linz, Austria
54. J. Samtinger and G. Pomberger *A Hypertext System for Literate C++ Programming.* JOOP, Vol. 4, No. 8, SIGS Publications, New York, 1992
55. S. E. Sandø, The Software Development Foundation *CSF Specification.* http://sds.sourceforge.net

Copyright © 2003 John Wiley & Sons, Ltd.

Prepared using speauth.cls

Draft

Don't redistribute or cite without explicit permission!

56. Stephan Shum and Curtis Cook *Using Literate Programming to Teach Good Programming Practices* 25th. SIGCSE Symp. on Computer Science Education, 1994, p. 66-70
57. Volker Simonis and Roland Weiss ProgDOC *- a New Program Documentation System* Andrei Ershov 5th. Intern. Conf. "Perspectives of System Informatics", July 2003, Novosibirsk, Russia, (to appear in LNCS) http://www.progdoc.org
58. C. Simonyi. *Intentional Programming - Innovation in the Legacy Age.* IFIP WG 2.1 meeting, June 4th, 1996
59. C. Simonyi. *The future is intentional.* IEEE Computer Magazine, Vol. 32, No. 5, May 1999
60. D. Soroker, M. Karasick, J. Barton and D. Streeter. *Extension Mechanisms in Montana.* Proc. of the 8th Israeli Conf. on Computer Based Systems and Software Engineering, 1997
61. Sun Microsystems, Inc. *The Doclets API.* http://java.sun.com/j2se/javadoc/
62. T. Teitelbaum and T. Reps. *The Cornell Program Synthesizer: A Syntax-Directed Programming Environment.* Communications of the ACM, Vol. 24, No. 9, Sept. 1981
63. The Unicode Consortium. *The Unicode Standard 3.0.* Addison-Wesley, Reading, Mass., 2000, http://www.unicode.org/
64. Norman Walsh *DocBook XSL Stylesheets.* http://docbook.sourceforge.net/projects/xsl
65. Norman Walsh *Literate Programming in XML.* XML 2002, Dec. 8-13, 2002, Baltimore, USA. http://www.nwalsh.com/docs/articles/xml2002/lp/
66. N. Walsh and L. Muellner. *DocBook: The Definitive Guide.* O Reilly & Associates, 1999, http://www.oasis-open.org/committe/docbook
67. Ross N. Williams. *funnelWeb.* http://www.ross.net/funnelweb/
68. N. Wirth and J. Gutknecht. *The Oberon System.* Software - Practice & Experience, 19(9), 1989, p. 857-893
69. World Wide Web Consortium. *Mathematical Markup Language.* http://www.w3.org/Math
70. WorldWideWeb Consortium. *Extensible Hypertext Markup Language.* http://www.w3.org/MarkUp
71. World Wide Web Consortium. *XML Inclusions (XInclude).* http://www.w3.org/TR/xinclude
72. WorldWideWeb Consortium. *Extensible Markup Language.* http://www.w3.org/XML
73. World Wide Web Consortium. *XML Schema.* http://www.w3.org/XML/Schema
74. World Wide Web Consortium. *Extensible Stylesheet Language (XSL).* http://www.w3.org/TR/xsl/
75. World Wide Web Consortium. *XSL Transformations (XSLT).* http://www.w3.org/TR/xslt
76. R. Wunderling and M. Zöckler. *DOC++.* http://www.zib.de/Visual/software/doc++/
77. Christopher J. Van Wyk *Literate Programming Column.* Communications of the ACM, Volume 33, Nr. 3, March 1990. p. 361-362

Copyright © 2003 John Wiley & Sons, Ltd.
Prepared using speauth.cls

Draft
Don't redistribute or cite without explicit permission!