Literate Programming to Enhance Agile Methods

Vreda Pieterse, Derrick G Kourie and Andrew Boake

Department of Computer Science, University of Pretoria, South-Africa
vpieterse@cs.up.ac.za

Abstract. In this position paper, after explaining the essentials of literate programming, we argue that a literate programming style is consistent with the values espoused by agile software development; and that the application of literate programming in the context of an agile software development methodology is likely to enhance both the quality and lifespan of the final product.

Keywords: Literate Programming, Literate Extreme Programming

1 Literate Programming Essentials

Knuth [1] uses the term "Literate Programming" (LP) to describe his approach to program design. Rather than seeing the program as instructions to a computer that include comments to the reader, it should be seen as an explanation to a human with corresponding code between "code delimiters" [2]. For a program to be literate it should have the following attributes:

- **Literate Quality**: An artistic creation that explains the solution to a human by crisply defining its components and delicately weaving them together. [1]
- **Psychological Structure:** The program is organized in such a way that the reader is naturally led to an understanding of the decisions that shaped the code. [3]
- **Integrated Documentation:** Documentation of the program is seen as an integral part of the literate program that is developed around the code.
- **Table of Contents, Index and Cross References**: The document must have a table of contents, an index, as well as cross references between related modules within the program. The automatic generation of this information is important [4].
- **Pretty Printing:** Indentation, font styles text colours etc. should be judiciously applied to improve the readability and ease of understanding of code.
- **Verisimilitude:** The generation of executable code and the production of the human readable literate version of the program should be automatically extractable from the same source document. [5]

The first Literate Programming Environment (LPE) called WEB, was designed by Knuth [1] in 1984 at the advent of the procedural programming era. WEB used two processors called TANGLE and WEAVE to convert the original source document respectively into an executable program (which could be *executed* using a standard Pascal compiler), and into a publishable, human readable program (which was *printed* using TEX)

J. Eckstein and H. Baumeister (Eds): XP 2004, LNCS 3092, pp. 250-253, 2004.

Soon a number of similar LPE's (such as those shown in table 1) evolved to support other programming languages or to produce documents using other typesetters.

Table 1. Some Early LPE's

LPE	Year	Language	Typesetter
WEB [1]	1984	Pascal	TEX
CWEB [6]	1986	C	troff/nroff
Literate Program Browser[3]	1987	SmallTalk-80	troff
FWEB [7]	1990	FORTRAN8X	TEX
APLWEB [8]	1993	APL	TEX

In 1990 van Wyk [5] commented that the general acceptance of LP would not be possible before a universal LPE could be marketed. In the light of the technology of the time, most developers accepted this as the death knell of LP. Some valiant supporters of LP however continued to build adaptable LPE's (such as those shown in table 2) that were able to support a variety of programming languages, and to produce documents in various specified formats.

Table 2. Some Language independant LPE's

LPE	Year	Languages used
LIPED [9]	1992	Assembler, Pascal, Clipper
CWEB [10]	1993	C, C++, ANSI C, Java
VizAuthor [11]	1996	APL, Pascal, C, C++, Java
Leo [12]	2002	Java, C, C++, Pascal, Fortran, Perl, Icon, Python, SmallTalk, Cobol, etc.

In the meantime object oriented (OO) development emerged. Commercial integrated development environments (IDEs) supporting OO implemented LP concepts, albeit without specifically supporting LP. Consequently:

- Pretty printing in editors has become a matter of course.
- Tools like javadoc in the JDK support integrated documentation and create hyperlinked documentation with an index and table of contents.
- Tools like Rational Rose show different views of program design, supporting psychological ordering of the program components.

2 Justification of Literate Extreme Programming

Coding done in a literate style promotes the values associated with agile methods.

- **Individuals and interactions over processes and tools**: LP is a communication-oriented programming method [13] and can aid communication better than the pure source code even though intention revealing.
- **Working software over comprehensive documentation:** LP supports comprehensive documentation without wasting time on synchronising documentation with the code. LP improves the chances of creating working software sooner. [1]

- **Customer collaboration over contract negotiation**. LP provides a more readable form of the program, facilitating better customer collaboration.
- **Responding to change over following a plan**: LP promotes simplicity of design by insisting on a psychologically friendly structure. Complete up-front analysis, design and documentation violate the LP concept of integrated documentation. Thus, LP shares with the agile methods the concern that too much planned up-front analysis and design limit the ability to respond to change.

Application of LP is likely to improve the product in terms of the following:

- **Communication:** Reenskaug and Skaar [14] experienced improved communication through written documents while applying LP in practice. XP on the other hand is designed to enhance verbal communication. These communication modes can potentially augment one another.
- **Quality:** Many authors have reported improved code quality when applying LP [2], [6], [15]. While not denying that XP may produce quality code, adding LP to XP is likely to further enhance the quality.
- **Documentation:** Much has been said about the advantages of having proper documentation for programs [16], [17], [18] and about the disadvantages of having documentation that does not match the system. [6], [16], [18]. LP emphasises these advantages and diminishes likelihood on incurring the disadvantages through better consistency between code and documentation [19]. When using LP, the XP team no longer has to minimize documentation to avoid its disadvantages. Instead, they are enabled to reap the benefits of its advantages.

Added benefits of applying Literate Extreme Programming include the following.

- **Retention of knowledge**: In an XP context, the knowledge about the system resides mostly in the memories of the team members. However, no-one remembers everything all the time. Information recorded in the literate program remains available to refresh a team member's memory when the code is revisited.
- **Accelerated distribution of knowledge**: Using verbal as well as written communication will improve the quality of the information that programmers retain. Access to the documented information can also save time later, because the learning curve to understand the code will be flattened by the presence of this information.
- **Newcomer Integration:** The improved quality of the code and the enhanced recorded documentation will help newcomers to be productive in the team sooner.
- **Customer confidence:** Because the knowledge about the system is recorded in ways that ensure continuity, the customer will enjoy greater peace of mind.
- **Outsourcing:** Applying XP in outsourced development is generally discouraged. Adding LP will allow the delivered project to be resumed by a different team. Inclusion of LP could therefore improve XP's applicability to outsourced coding.
- **Scalability:** The agile value set and practices are best suited to teams of less than 50 people who are engaged in projects that are not life-critical [20]. We feel that it will be possible to alleviate concerns about team size, project size and project character that are often associated with XP, by adding aspects of LP to it.

3 Conclusion

We have argued that LP is consistent with and supportive of agile methodologies. The incorporation of LP into a methodology such as XP will enhance communication and improve the software quality. We are cognizant of the commonly observed fact that programmers are not enthusiastic about documentation [18]. Nevertheless, given the cited evidence that LP improves both quality and communication, it seems worthwhile to pursue an empirical study to build a good LPE and to measure the extent to which it can find acceptability amongst extreme programmers. Work in this direction is under way.

References

1. Knuth D.E. *Literate Programming*. The Computer Journal, 27 (2) (1984) 97-111
2. Williams R. *FunnelWeb Tutorial Manual*. Online: http://www.ross.net/funnelweb/tutorial/intro_what.html visited 2003/01/05 (2000)
3. Beck K., Cunningham W. *Expanding the Role of Tools in a Literate Programming Environment*. Presented at CASE'87. Boston Mass. online: http://c2.com/doc/case87.html Visited: 2003-12-19. (1987)
4. Denning P.J. *Announcing Literate Programming*. Communications of the ACM, 30 (7) (Jul 1987) 593
5. Van Wyk C.J. *Literate Programming : An Assessment*. In: Literate Programming. Communications of the ACM, 33 (3) (March 1990) 361-365
6. Thimbleby HW. Experiences of 'Literate Programming' using CWEB. Computer Journal, 29 (3) (June 1986) 201-211
7. Avenarius A., Oppermann S. *FWEB: a literate programming system for Fortran8x*, SIGPAN Notices, 25 (1) (1990) 52-58
8. Dickey LJ. *Literate programming in APL and APLWEB.* APL Quote Quad, 23 (4)11.(1993)
9. Bishop J.M., Gregson K.M. *Literate Programming and the LIPED Environment.* Structured Programming. 13 (1) (1992) 23-34.
10. Levy S. Literate Programming and CWEB. Computer Language, 10 (1) pp 67-70. (1993)
11. Pieterse V., Bishop JM. *Visualization of Programs in Textbooks* Online: http://www.literateprogramming.com/visualze.pdf Visited 2004/01/08 (May 1996)
12. Ream E.K. *Leo Literate Editor with Outlines.* online: http://www.3dtree.com/ev/e/sbooks/leo/sbframetoc_ie.htm visited 2004/03/28, (Dec 2002)
13. Beck K. *Extreme Programming Explained*, Addison-Wesley, (1999).
14. Reenskaug T., Skaar AL. *An environment for literate Smalltalk programming*. OOPSLA 1989 Proceedings, New Orleans, (1989) 337-345
15. Lindsay D.C., Thimbleby H. *A File Difference Program*. In: Literate Programming. Communications of the ACM, 32 (6) (June 1989) 740-755
16. Kotula, J. *Source Code Documentation: An Engineering Deliverable* Online: http://csdl.computer.org/comp/proceedings/tools/2000/0774/00/07740505abs.htm Visited: 2004/01/08 (2000)
17. Hyman M. *Literate C++*, Computer Language. 7 (7) (Jul 1990) 67-69
18. Parnas D. *Software Aging.* In: Software Fundamentals. Addison-Wesley, (2001)
19. Shum S., Cook C. *AOPS: an abstraction-oriented programming system for literate programming*. Software Engineering Journal, 8 (3) (May 1993) 113-120
20. Williams L., Cockburn A. *Agile Software Development: It's about Feedback and Change*, Computer, 36 (6) (June 2003) 39-41