using SGML, and was formatted using Adept 5.0 from ArborText Inc. Obviously, designing a book that is comfortable to read is not the same as writing a 'FOSI', an output specification for ArborText's Adept product. I hope that the publisher will work hard on improving the layout of the book, but I have my doubts.

Of course, this says nothing about the applicability of SGML to book production, but only about the quality of available SGML tools, or the expertise of the people using these tools. That computers *are* capable of producing more readable and more attractive books is shown by a book co-authored by one of Mr. van Herwijnen's colleagues at CERN, namely *A LaTeX Companion*, by Michel Goossens, Alexander Samarin and Frank Mittelbach. But then, of course, that book was made with LaTeX!

◇ Nico Poppelier
Elsevier Science Publishers
Amsterdam
The Netherlands
Internet:
n.poppelier@elsevier.nl

Book review: *Literate Programming*

Christine Detig and Joachim Schrod

Donald E. Knuth, *Literate Programming*. Center for the Study of Language and Information, Lecture notes no. 27, Stanford 1991. (Distributed by the University of Chicago Press.) xvi + 386 pp., index and comprehensive bibliography.
ISBN 0-9370-7380-6 (pb), 0-9370-7381-4 (hc).

The essence of literate programming?
Say it twice!
— D. Knuth (1993)

This book is an anthology of works by Donald Knuth; it tells us the story of literate programming. It consists of an introduction, a lecture, eight articles, three book excerpts, a program, and a bibliography. John Hobby is responsible for the selection of the contents; the introduction is the only text previously unpublished.

The presented material spans almost 20 years of Knuth's work, in which literate programming developed from concerns about the quality of software description, through first ideas to categorize and improve it, to applications and experience reports based on the methods and tools he has created.

Contents

The collection starts with the *Preface* that presents Knuth's views on the relation between the different texts selected by Hobby. It shows the "red thread" of the book and gives advice on how to read this book. Besides this introduction, the only new material is a paragraph at the start of each text that presents the context of original publication.

The first text, chapter 1, is the lecture given by Knuth in 1974 when he received the *Turing Award*, the most important Computer Science award. Already at that time, Knuth had named the basic principles and motivation of literate programming:

> The chief goal of my work as educator and author is to help people learn how to write *beautiful programs*. [...]
>
> [The goals of correctness and adaptibility] are achieved when the program is easily readable and understandable to a person who knows the appropriate language. [...]
>
> Please, give us tools that are a pleasure to use, especially for our routine assignments, instead of providing something we have to fight against.

In this lecture, *Computer Programming as an Art*, Knuth argues that programming has much in common with music composition. Here we also find the reasoning behind the statement that programming is not a science, but an art. Knuth still holds the professorship for the "Art of Computer Programming" and this chapter shows us basic principles of his whole professional life.

Chapter 2 presents one of Knuth's most cited articles: *Structured Programming with* ***go to*** *statements* (1974). This article must be read in the context of its time: People had just started to develop programs in a systematic, structured way; the scientific community was discussing for the first time how to write long-living programs. It's written in the context of Dijkstra's famous letter "Go to statement considered harmful" and shows that the problem of unstructured programs is not based on language constructs.

Chapter 3 continues with an early effort of Knuth to present a larger piece of code in a readable and understandable way: *A structured program to generate all topological sorting arrangements* (1974). According to Knuth himself, the presentation of this article left much to be desired. He had realized that writing programs intended for critical reading means to make construction and evolution recapitulable. This requires other forms of writing and presentation than the old, machine oriented, style.

Chapter 4 marks a turning point in the story told by this anthology — Knuth takes the step from structured to *Literate Programming* (1984). He formulates the credo of this new paradigm:

> Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.

This new way of thinking is presented by an example; that example used the tool WEB he had created for the work on the programs TEX and METAFONT. WEB supports the intertwining of informal and formal parts: the former marked up with TEX tags, the latter, pieces of Pascal code. These code pieces are organized as refinements. Besides the facilities outlined above, WEB is also burdened with a set of features meant to overcome inherent deficiencies of the underlying programming language Pascal.

Both readers and writers face new requirements and viewpoints with this new programming paradigm. In chapters 7 and 9, two reflections on this aspect are presented: *How to Read a WEB* (1986) and an excerpt from *Mathematical Writing* (1987) that reports from discussions between Knuth and students on the subject of literate programming.

We just skipped chapter 8; there we enter an area well known to our fellow *TUGboat* readers. Two *Excerpts from the Programs for TEX and* METAFONT (1986) show the application of the literate programming paradigm in production-quality software. We read something that seems to come from a textbook on algorithm design, not from real-life programs that are in use at hundreds of thousands of installations.

Chapters 10 and 11 present data to support the claim that literate programming leads to programs that are better maintainable. *The Errors of TEX* (1989) presents the history of TEX, categorizes the errors Knuth made, and makes a thorough analysis of the development process. The data beyond this analysis is the diary, *The Error Log of TEX* (1978–1991). This diary gives an insight into his work situation: programming at night, the switch from SAIL to Pascal, finishing the last TEX version of 1982 at December 31, 23:59.

Back to chapters 5 and 6, where the reaction of the scientific community is representatively outlined. Jon Bentley picks up literate programming in his regular column in the *Communications of the ACM*, *Programming Pearls: Sampling* (1986). (Eventually, this led to the establishment of an actual *Literate Programming* column.) In *Programming Pearls: Common Words* (1986) Knuth presents a WEB solution to a problem posed by Bentley; a review by Malcolm McIlroy follows.

Chapter 12 finally presents Knuth's actual interest in literate programming by giving a programming example in CWEB, the tool he has also chosen for presenting combinatorical algorithms in his newest book *The Stanford Graphbase.* (This book will be a base for the next volume of his major work, the series *The Art of Computer Programming.*)

Review

The collection enlists pieces of work on the topic without "glue", except for a few remarks that relate to the origin of the articles. The reader himself is in charge of finding the relation between them. But especially the first article gives a clue to Knuth's motivation behind all technical aspects: He, now professor of *The Art of Computer Programming*, has always tried to make programming an art.

> [P]rogramming can give us both intellectual and emotional satisfaction, because it is a real achievement to master complexity and to establish a system of consistent rules. [...] My claim is that it is possible to write *grand* programs, *noble* programs, truly *magnificent* ones!

Starting with the metaphor of programming as an analogy to music composition, he later recognized that it's more like writing literature. "The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style."

The anthology reveals that Knuth always made literate programming speak for itself. In his articles, it is always presented by examples, connected to the tool WEB, based on Pascal (later C with CWEB).

The principles of literate programming,

1. integrating informal and formal expressions of the same thing, combining explanation and program code into one document,
2. presenting the software according to the solution's semantic structure, keeping the design method visible until the implementation is finished; yielding — together with the human explanation focus — traceability of the development process,
3. concurrent, independent abstraction hierarchies for informal and formal parts, i.e., sections for documentation and refinements for code,
4. linking definition and usage of entities; in WEB by the form of cross references, index, table of contents, etc.,

5. using modern presentation techniques; including, but not limited to, typography, graphics, formulas, tables, etc.,

have to be recognized by ourselves; they are partly obscured by the concrete details of the used examples and tools.

Free distribution of all required tools and integration with TEX made Knuth's way *the* choice to access literate programming. For a long time, literate programming was totally determined by that orientation, both by concept and problems addressed, but also concerning the tools used. This has influenced at first hand also the reception of literate programming, shown prototypically by the review of McIlroy in chapter 6. Literate programming is taken to be synonymous with `WEB`, the presented programs are attacked for being monolithic and not reusing other modules — caused in fact by the base language Pascal.

In the meantime, more and more people use literate programming not for the creation of academic solutions to small problems, but for their day-to-day work instead. The discussion forum, a USEnet newsgroup and electronic mailing list, shows that literate programming really starts to be a paradigm in the sense of Thomas Kuhn: A new generation starts to use the principles without caring if it's fully accepted in the traditional development process. As Norman Ramsey put it once, it's the time of the "true believers".

The book doesn't go beyond the starting period of literate programming. Neither does it give a reflection on the paradigm itself, isolated from its own development. Nevertheless, Knuth calls for a second generation of work on literate programming. In the comprehensive bibliography, he lists current work of those that follow his direction, but also of those that go different ways: Extension of literate programming to development of large software systems and to the whole software development process is addressed, printed publication is substituted by electronic documents, and different programming language concepts are taken into account as well as separating literate programming from fixed target languages.

Conclusion

As we expect from an anthology, no new material is presented. The book provides a collection of texts that might not have been very accessible to people outside of universities. This gives the chance to gain a clear understanding how Knuth developed the literate programming paradigm from first requirements to its realization, built upon his ideas for structured programming. If you are interested in such a time-spanning view on scientific work, be it for delight only or for interest in the topic itself, this book is a *must*.

But beyond the formation of a paradigm, this book also shows something rare: It provides insights into the thoughts and working of one of the most influential computer scientists of this century — a man who does not only want to gain knowledge, but wants to share it, wants to make it understandable and accessible. This is so important for him that he was willing to spend years of his work on projects for realizing his ideas, and he has created something qualitatively new with a potential not yet fully exploited.

LET'S GO FORTH now and create *masterpieces of the literate programming art!*

◇ Christine Detig
Technical University of Darmstadt
WG Systems Programming
Alexanderstraße 10
D-64283 Darmstadt
Germany
detig@iti.informatik.th-darmstadt.de

◇ Joachim Schrod
Technical University of Darmstadt
WG Systems Programming
Alexanderstraße 10
D-64283 Darmstadt
Germany
schrod@iti.informatik.th-darmstadt.de