Software—Concepts and Tools (1996) 18: 35–46

Software—Concepts and Tools

Analysis of Literate Programs from the Viewpoint of Reuse

Bart Childs
Department of Computer Science, Texas A&M University, USA
e-mail: bart@cs.tamu.edu

Johannes Sametinger
CD Laboratory of Software Engineering, University of Linz, A-4040 Linz, Austria
e-mail: sametinger@swe.uni-linz.ac.at

Abstract. Donald Knuth created the WEB system for literate programming when he wrote the second version of TEX, a book-quality formatting system. Levy later created CWEB, which is based on Knuth's WEB using the C programming language and supporting development using the C and C++ programming languages. Krommes' FWEB is based on CWEB and supports several programming languages. We analyze some parts of these systems from the viewpoint of reuse.

We make reuse comparisons of four elements of the TEX system: TEX, METAFONT, DVItype and METAPOST. We also compare the primary filters (*tangle* and *weave*) of CWEB and FWEB. We analyze the code and integral documentation, considering similarities of chapters, lines and words.

With this study we demonstrate that both code and documentation can and should be reused systematically and that there is a need for methods and tools for doing so. Literate programming and software reuse are by no means in contradiction. However, current literate programming systems do not explicitly support software reuse, even though reuse was common in their development.

Keywords: software reuse, literate programming, TEX, WEB, case study

1. Introduction

The literate programmer should keep in mind that the human reader is as important as the machine reader. Human readers are necessary for maintenance activities, an area of prime importance in the study of software engineering. We agree with Knuth's claim that literate programming is a process which should lead to more carefully constructed programs with better, relevant 'systems' documentation [16]. We take Knuth's style of literate programming as prototypical. It was used in writing the second version of the TEX typesetting system [11, 12] and its related components. This WEB system, as he used it, leads to

- top-down and bottom-up programming through a structured pseudocode,
- programming in sections, generally a screen or less of integrated documentation and code (where section in this use is similar to a paragraph in prose),
- typeset documentation (after all, it was for and in TEX),
- pretty-printed code where the keywords are in bold, user-supplied names in italics, etc., and
- extensive reading aids which are automatically generated, including table of contents and index.

The value of each of these items depends on the programmer, as always. For example, the index mentioned in the last item can be supplemented by user-supplied entries in addition to those automatically generated (which are similar to compiler cross reference lists). If the author does not furnish these, the modifier *literate* cannot be justified, in our opinion. For example, in TEX [12] Knuth entered nearly a thousand extra index entries, of which more than 600 were unique.

Pappas stated that a literate programming approach provides benefits in writing reusable code [23]. He emphasized that reusable software requires "more than just following coding guidelines". Further, "if a software component gives a programmer the impression that it will take almost as much time to

understand ... as it will to write ... (it) will not be reused!" We feel that the quality, locality and integration of documentation that is provided by Knuth's style of literate programming could have a dramatic effect on reuse.

We have chosen Knuth's sources for his TEX system and descendent literate programming systems as examples because:

- they are in the public domain,
- they are commonly available,
- they are well documented,
- they are consistent and complete,
- they are written in Knuth's WEB (and descendent systems) for literate programming,
- they are of reasonable size for our planned investigation, i.e., big enough for serious investigation and small enough to complete the investigation within a reasonable amount of time, and
- Knuth is an experienced, careful, accurate and meticulous literate programmer.

In Chapter 2 we discuss some of Knuth's design decisions in order to clarify some frequent misunderstandings. Chapter 3 contains information about software reuse and why we believe that reuse in literate programming is important. In Chapter 4 we present an overview of the systems we considered for investigation. In Chapter 5 we explain how the results were obtained and then we present the results in Chapter 6. Discussions and interpretations follow in Chapter 7. Finally, a short summary appears in Chapter 8.

2. A View of some of Knuth's Design Decisions

We wish to enable understanding of some of Knuth's decisions because the results of them have often been misunderstood and misused in what we view as unjust criticism.

Knuth released the first version of TEX in 1978. It was written in the SAIL language and was generally available only on DEC 10's and 20's. An enthuastic following developed in the academic and research lab communities with such machines (and a few ports were made to other systems.) A number of limitations in the original TEX system and the fact that DEC halted manufacture of the 36-bit systems led to the decision to rewrite and extend TEX. Knuth included portability as a prime concern and believed that "systems documentation" would be a significant factor.

Knuth surveyed a number of users and concluded that Pascal was the best language choice because it seemed to be everybody's second best language and most versions of Pascal had the facilities to be a reasonable host for writing a 'systems program'. (C was not commonly available at that time, 1980.) Knuth honored many of the 'standard' aspects of Pascal, used some common extensions that were of great benefit, and extended Pascal with some WEB features. There have been a number of posts to news groups of like "literate programming is brain dead because I don't like programming in a monolith". It is obvious that the only reason Knuth's WEB did not use include files is that it was not common in Pascal(s) in 1980! Features of the WEB system that enhanced portability included macros, facilities for converting long, readable variable names to arbitrary compiler limitations, and many others.

Knuth's design decisions were based on making the TEX system portable to a wide variety of systems. He accomodated a number of characteristics that may seem perverse today. For example, he programmed using long variable names that were generally words from the dictionary connected by underscores. Because of the existence of Pascal compilers with arbitrary requirements, the filter that extracts code (*tangle*) produced code that was all uppercase, no underscores, at most eight characters long, and unique in the first seven.

Knuth makes a token payment to the first person to find an error. Updates to the TEX systems are issued periodically. The errors that have been found and corrected are widely distributed and its evolution has been well documented also [15]. We will not address these errors, but that could be a fertile area of study. (Incidentally, Knuth has doubled the 'token payment' with successive revisions of TEX and now maintains it at a rather significant level.)

3. Software Reuse

Software reuse is the process of creating software systems from existing software rather than building them from scratch. Reusable software has many benefits, including the following most common ones [5, 19, 22]:

- reduction of development time and redundant work

- ease of documentation, maintenance, and modification
- improvement of software performance and software quality
- encouragement of expertise sharing and intercommunication among designers
- smaller programming teams for the construction of more complex software systems

In the context of literate programming we are interested in technical aspects rather than in managemental, cultural, organizational, economical or legal issues, which, without any doubt, have a big influence on successful reuse of software. Technical aspects of software reuse have many facets, too, e.g., ad-hoc reuse *vs* institutionalized reuse, black box reuse *vs* white box reuse, code reuse *vs* design reuse, and code scavenging vs as-is reuse. Since McIlroy's vision of standard catalogs in 1969, the term software component has played a major role in the context of software reuse. Many definitions and taxonomies of software components exist, e.g., in [4, 27].

We believe that the idea of literate programming is important in achieving well-documented and structured software systems. The question that arises is: How do literate programming and software reuse fit together? Donald Knuth proposes creating each software system as a piece of literature. Can this literature be cut into components and reused in various contexts? Evaluating this reuse question is necessary toward our goal of determining to what extent reusable components could have been extracted from these literate systems with minimal effort.

4. The Subject Systems

The *TEX* system and the WEB processors were written in the original WEB. We describe these programs rather superficially. A description of most of the inputs and outputs is also furnished to enhance understanding the functions of the programs. The ← indicates that a file is input, → indicates that a file is output, and ↔ indicates interactivity (the terminal). Since the original WEB included a number of features that were designed to compensate for Pascal deficiencies, some of these files would not be included if the system were written in another language, say C. For example, the *pool* file was used because of string handling deficiencies in standard Pascal.

We studied the following versions of the codes: TEX 3.141, METAFONT 2.71, METAPOST 0.63, DVItype 3.4, CWEB 2.99++, and FWEB 1.30a.

4.1 The TEX System

We studied four WEBs from the TEX system: TEX, a book-quality formatting system [11, 12]; METAFONT, a system that enables a programmer/artist to create a family of fonts for TEX [13, 14]; DVItype, a prototypical reader of *dvi* files that are the output of TEX [10]; and METAPOST, a close relative of METAFONT that enables the creation of high-quality graphics as encapsulated PostScript files [7, 8]. An outstanding feature of the TEX system is the complete and careful documentation that it includes. Several of the WEBs were written by Knuth himself and some others were obviously carefully reviewed by him.

4.1.1 TEX. The TEX processor converts a plain text file containing document markup into a device-independent graphics metafile. It inputs a number of other files in this process to get font charcteristics, document styles, etc. The files associated with the execution of the TEX processor are:

- ↔ *terminal*
  Although TEX is often characterized as a batch processor, it is used in an interactive mode most of the time. Small errors can be corrected (but they later have to be changed in the source file), debugging commands can be issued, etc.
- ← → *tex.fmt*
  Each installation of TEX normally has two versions of TEX. The *'ini'* version converts a macro file into a smaller, binary version that has been compiled and only has to be input. The TEX processor inputs this binary file, while it is output by the iniTEX processor.
- ← *tex.pool*
  The *pool* file is a feature of the original WEB that was designed to overcome the lack of portable, efficient handling of varying length strings in Pascal. This is input by iniTEX.
- ← *source.tex*
  The source file is as described above. This document may have structure. The top-level source often is a list of commands that specify which style is to be used and the order of other files to be input, which are the majority of the document.
- ← *style.**

The style file(s) can specify a large number of items such as the macro processor (often LaTeX) article/report/book/..., double column, two-sided printing, etc. Several extensions are used, including *.tex, .sty* and *.cls*.

- → *source.dvi*
  The device-independent file is a graphics metafile. The design of this format was based upon having characters as the primary graphic element and is an efficient representation of the document. It includes a record of when it was created and appropriate checksums for consistency.
- ← **.tfm*
  The TeX font metric file contains size information for each font used in creating the *dvi* file. Additional information that is used includes ligatures, kerning and spacing. The *tfm* file is specific for a font and is not dependent upon the printer that will eventually be used. The checksum for each font is stored in the *dvi* file for later consistency checks for each step of document production.
- → *source.log*
  The log file is a journal that includes a history of the creation of the *dvi* file. This information is a superset of the information that appears on the screen as the TeX processor works.

4.1.2 METAFONT. The METAFONT processor operates in a manner that is similar to the TeX processor and at the same time is quite different. METAFONT accepts as input a source file that is a metadescription of a font (family). It does significant graphics interpretations, solves equations, and handles other items associated with the creation of a consistent family of fonts. The primary outputs are the tfm files that TeX uses as well as files that specify locations of ink for each character in a font for specific printers. The files associated with the execution of the METAFONT processor are quite similar to those in TeX. Significant parts of the input and output would obviously be reused.

4.1.3 METAPOST. METAPOST is a close relative to METAFONT. The METAPOST processor inputs have file layouts much like METAFONT sources. Instead of creating a font which has a family of related glyphs constructed using common strokes, serifs, etc., the output of METAPOST is book-quality figures.

The programmer identifies each figure by a number that is used as an extension for its file name. The output is encapsulated PostScript. The METAPOST processor can invoke TeX to create labels using the same fonts as the intended document.

The input and output of METAPOST is obviously quite similar to that in METAFONT. A significant difference is that the two binary files (tfm and *gf*) are possible with METAPOST but are usually not output.

4.1.4 DVItype. The DVItype processor was created to serve two purposes. First, in the early days when porting TeX was a common activity, it served as a great debugging aid. Second, since it properly reads all possible *dvi* files, it provided a big help creators of programs to input *dvi* files and output printer files. DVItype is a prime example of a program intended for *reuse*. Since it accepts as input the output of TeX, the potential for reuse is obvious.

The files associated with the execution of the DVItype processor are:

- ↔ *terminal*
  See *terminal* in TeX.
- ← *source.dvi*
  See *source.dvi* in TeX.
- ← *setup*
  Ports of DVItype often included the use of a setup file to ease normal use.
- → *source.tfm*
  See *.tfm in TeX.

4.2 WEB Systems

WEB systems support literate programming [16]. They read WEB sources (code and documentation) and act as front ends for the Pascal compiler and the TeX formatting system. The *tangle* processor creates the Pascal file which will eventually become the executable program. The *weave* processor takes the same WEB source and creates a TeX source file which constitutes the documentation of the program.

4.2.1 The *tangle* Processor. The files associated with the execution of the *tangle* processor are:

- ↔ *terminal*
  See *terminal* in TeX.
- ← *source.*web
  The WEB file contains both code and documentation, i.e., a literate program.
- ← *src_ch.ch*
  The change file can be used optionally to make changes to the WEB file without acutally modifying it. This is often used for porting a

system to various machines. Notice that the prefix for the change file is not necessarily identical to the prefix of the WEB file.

- → *source.pas*
  The Pascal source code file contains all the code of the WEB file. *tangle* collects the code, orders it correctly and outputs it ready for compilation.
- → *source.pool*
  The string pool file's purpose is to make the handling of variable-length strings less tedious. (Standard Pascal does not have variable-length strings.)

4.2.2 The *weave* Processor. The files associated with the execution of the *weave* processor are identical to those of the *tangle* processor, except that the main output is a TEX file.

4.2.3 CWEB and FWEB. Knuth defined and created the WEB system and then Levy created the CWEB system as a relatively direct translation [17, 20]. Although the CWEB system is based on Knuth's original WEB, reasonable assumptions were made about the C language rather than Knuth's lowest common denominator assumptions about Pascal (which Knuth made solely to make a portable, maintainable system). Levy and Knuth now maintain CWEB.

The FWEB system is a direct descendent of CWEB, with changes that are more typical of *reuse* [1, 18]. FWEB is a significant extension of CWEB for a specific reason. Krommes' FWEB is multilingual, crunching numbers in Fortran on supercomputers and doing interpretive graphics using C and X-windows. Thus Krommes built Fortran, Fortran 90, C and C++ support into FWEB.

There are several other WEB systems, but we feel little could be gained by extending these studies to them.

4.3 Expected Similarities

There are two types of relationships between the mentioned programs that are reasons to expect reuse. These are:

- The programs are part of a system and/or operate in the same fashion.
  TEX and METAFONT operate in similar fashions. Also, TEX accepts as input the output of METAFONT. The detailed description of the input and output files of these programs is an area where we expect significant reuse. Browsing the sources of TEX and METAFONT in book form [12, 14] reveals many chapters with the same title. Similarities in these chapters are obvious even from just turning the pages.
  The *tangle* and *weave* programs are even more similar as they share identical inputs. These will be analyzed only in the context of different WEB systems for different high-level languages (for space reasons). The *tangle* and *weave* processors input WEB source and change files. Sewell also noted similarities in the way tables are allocated in memory [25]. There was no black-box reuse in the original WEB system. The file *common.web* is appropriately named and contains sections that are used by both *tangle* and *weave* in the CWEB (and FWEB) systems.
  DVItype is a program that inputs and interprets the output of the TEX program.
- The programs were modified to create related programs or to significantly extend the functionality of the original programs.
  METAPOST is a modification of METAFONT as the components of FWEB are modifications of CWEB. These processors also utilize large arrays that contain all the elements they are manipulating. Regarding reuse from CWEB to FWEB, it is clear that *cweave* and *fweave* should be closer than *ctangle* and *ftangle*, because of Fortran's unique nature. A large part of the *ftangle* source is required by the record (rather than stream) orientation of Fortran.

Note that the *C*WEB and *F*WEB systems were written using C, while all other programs above were based on Pascal. Also in the *C*WEB distribution, the WEB files use the terse extension of *tt.w*, as one might expect in a UNIX-oriented system. We use the extension *tt.*web in the discussion of all WEBs.

5. Reuse Measures

We compared sources to determine measures of reuse. These comparisons were done using the UNIX utility *diff*. In most cases there was some preprocessing; this will be discussed later. Reuse can be identified at a number of different levels, i.e., words, phrases, sentences, lines, paragraphs, sections and chapters. The following list is a discussion of factors affecting reuse at these levels. We use a **bold** font for the levels of our primary results (lines and words) and *an italic* font for levels that we used in indirect fashions (chapters).

- **Words** — Individual words seem to be a trivially fine granularity. There will be some obvious reuse of articles, keywords, ... Knuth reused many sentences with some editing, such as inserting or deleting a parenthetic expression. We feel that reuse at the word level compared to reuse at a higher level of granularity is indispensable.
- Phrases — A literate program can be viewed as a system of structured pseudo-code. The meaning of a section of code should be rather explicitly indicated by the pseudo-code name of a section of code, like *@<Set initial values of key variables@>*. This string, for example, appears 35 times in the source of TEX. The first one is a place holder where code goes and the rest define initializations. This type of reuse is also reflected in word reuse counts.
- Sentences — This was considered but discarded in part due to Knuth's attention to detail and consistency. For example, the TEX book has a lion that decorates the beginning of each chapter, while the METAFONT book has a lioness. In an error message where all relevant help that Knuth could anticipate has already been given, he suggests the TEX user emulate Hercule Poirot and at the same point suggests that the METAFONT user emulate Miss Marple. Strict sentence comparisons would not sense the similarities, but word and phrase comparisons would.
- **Lines** — Lines pose the same problems as sentences. We use lines in the sense that they appear in common editors like *vi* and *emacs*. The use of existing tools like *diff* is also advantageous. We have observed that in many cases where a code is reused, but edited, the original line structure is often kept. This obviously contributes to high reuse indicators.
- Paragraphs — Paragraphs are consecutive lines separated by blank lines, headings, etc. We feel this is adequately covered by line reuse in the previous item. If we also calculated the lengths of runs of identical lines, then this would be a better indicator than paragraphs *per se*. This is discussed in more detail in [6].
- Sections — In a book model, WEBs are characterized as having chapters, sections and paragraphs. In the FWEB model, Krommes introduced major sections to enable finer granularity. The titles of major sections appear in the table of contents with the enclosing chapters, but indented. Knuth used the terms section and module interchangeably and called what we call chapters 'major sections'. We avoid the use of 'module' to avoid conflict with its use in Ada, Fortran 90, Modula-2, etc.
- *Chapters* — A chapter may contain sections, and its title appears in the table of contents. The output of a chapter (in the typeset documentation) always starts on a new page. The title of the chapter is presented in bold type, but not with the great emphasis that is normal in books. We did not expect to find identical chapters, except for the titles.

$$R = (1 - (E / T)) \times 100$$

R	reuse level from comparing lines. 0 means no reuse, 100 means everything (all lines) of *file a* had been reused in *file b*.
E	number of (edited) lines to be changed or deleted from *file a* in order to get contents of *file b*
T	total number of lines of *file a*

Figure 1. Reuse Measurement.

We base our comparisons on lines and words because of the simplicity of calculating these. We found it difficult to take semantic information into account. Comparing lines and words gives a good indication of reuse. Obviously, high line reuse indicates much reuse; low line reuse with high word reuse also indicates much reuse, but with local modification. Finally, if both line and rd reuse are low, then apparently there was not much reuse. The folloeing subtle differences affect these measures of reuse:

- Single words can be exchanged (such as TEX, METAFONT, DVItype, etc.).
- Nonbreaking blanks (often called a hard space) may appear where a space is to be output.
- Line breaks can be changed.
- The order of chapters and sections can be different.
- User-supplemented index entries can be different.
- Sentences can be changed in syntax (e.g., word ordering) without any change in the semantics.

We compared chapter titles of the systems under consideration to find the candidate sections for comparing line and word reuse. This gives a first indication of similarities. For example, TEX consists

of 55 chapters, while METAFONT contains 52. Twenty-six chapter titles are identical. These chapters have been investigated in detail. There is likely some reuse in the other chapters, but we have concentrated on those with identical titles.

We copied each chapter to a unique file and used *diff*. This yields *add*, *change* and *delete* information that can be applied to change *file a* to *file b*. An indication of how much of *file a* is reused in *file b* is the total number of lines in *a* less the number of lines that need to be changed or deleted to create *b*. The reuse level (of *file a* in *b*) is shown in Figure 1.

As empty lines are considered to be equal, the reuse level, naturally, is greater than zero, if empty lines appear in both files. Thus, it is crucial that empty lines be eliminated before the reuse level is determined. Of course, for two equal files the result of R is 100. When empty lines are eliminated then, R is usually zero for nominally different files.

Lines can be similar and differ by as little as a word or punctuation. Therefore, we replaced each blank by a newline, and the values of R increased, as expected. Our definition of word reuse, R_w is based upon these one-word-per-line files. Obviously, different files can have non-zero reuse levels because the same words can appear in both files. We denote R_l and R_w as reuse levels considering lines and words, respectively.

R_w is usually slightly higher than R_l. Sometimes, however, R_w is significantly higher than R_l. This is the case when reused text has been modified extensively, which leads to differing lines (lowering R_l) while retaining many of the same words (lowering R_w less than R_l.)

We will demonstrate the suggested kind of reuse evaluation by studying the first paragraph of the chapter “Introduction to the syntactic routines” of TEX and METAFONT. The text of TEX in Figure 2 contains 12 lines and 128 words. The text of METAFONT contains 13 lines and 135 words. Identical lines are marked with ‘=‘. Words that do not appear in both systems are striken through. (Words are taken to be any sequence of characters delimited by whitespace.) To transform the text of TEX to the text of METAFONT, 9 lines or 30 words have to be changed. This results in a line and word reuse of R_l = 25.0% and R_w = 76.6%. Note that the high difference between R_l and R_w indicates modification and polishing of the source.

TEX - METAFONT	lines	words
LTEX	21,541	122,137
LMETAFONT	20,481	109,307
RTEX/METAFONT	14.3%	21.5%
RTEX(33.4%)/METAFONT	42.8%	60.7%

Table 1. Reuse level of TEX in METAFONT.

```
TEX:
   @* \[21] Introduction to the syntactic routines.
=  Let's pause a moment now and try to look at the Big Picture.
   The \TeX\ program consists of three main parts: syntactic routines,
=  semantic routines, and output routines. The chief purpose of the
=  syntactic routines is to deliver the user's input to the semantic routines,
   one token at a time. The semantic routines act as an interpreter
   responding to these tokens, which may be regarded as commands. And the
   output routines are periodically called on to convert box and glue
   lists into a compact set of instructions that will be sent
   to a typesetter. We have discussed the basic data structures and utility
   routines of \TeX\, so we are good and ready to plunge into the real activity by
   considering the syntactic routines.

METAFONT:
   @* \[30] Introduction to the syntactic routines.
=  Let's pause a moment now and try to look at the Big Picture.
   The \MF\ program consists of three main parts: syntactic routines,
=  semantic routines, and output routines. The chief purpose of the
=  syntactic routines is to deliver the user's input to the semantic routines,
   while parsing expressions and locating operators and operands. The
   semantic routines act as an interpreter responding to these operators,
   which may be regarded as commands. And the output routines are
   periodically called on to produce compact font descriptions that can be
   used for typesetting or for making interim proof drawings. We have
   discussed the basic data structures and many of the details of semantic
   operations, so we are good and ready to plunge into the part of \MF\ that
   actually controls the activities.
```

Figure 2. Sample paragraph comparison.

Note that some single words (*that*, *be*, *of* and *the*) are considered as being reused. This may sometimes result in a slightly higher R_w than is justified (even if these words were actually reused when editing the text). Not considering these four words would result in R_w = 73.4%.

Note that despite its high similarity, there are only three identical lines in the METAFONT paragraph. The first line would have been identical except for the inclusion of the chapter number.

6. Results

We present the results of most of our comparisons in tabular form. The tables contain reuse levels for lines and words, lengths, and in some cases are indexed by chapter title. Detailed comparisons of some items are included. We use the following notation in presenting the results of the comparisons and computing the reuse level:

- $L_{file\ a}$ – length of *file a*, i.e., the number of non-empty lines
- $R_{file\ a/file\ b}$ – reuse level of *file a* in file *b*, i.e., how much of *file a* was reused in *file b* (in percent).
- $R_{file\ a(x.x\%)/file\ b}$ – reuse level of *file a* in *file b* by considering only *x.x* percent of the lines of *file a*. (These are the lines of the chapters that appear with the same title in *file b*.)

6.1 TEX and METAFONT

TEX contains about 21,500 lines and 122,000 words. METAFONT consists of about 20,500 lines and 110,000 words. TEX and METAFONT are divided into 55 and 52 chapters, respectively; 26 of these chapter pairs have matching titles. These chapters contain 33.4 percent of the lines of the TEX system. Table 1 shows the resulting reuse levels. 14.3% of the lines and 21.5% of the words of TEX are reused in METAFONT. Of the 26 chapters with matching titles, 42.8% of the lines and 60.7% of the words are reused in the corresponding chapters in METAFONT.

Table 2 compares in detail the chapters that appear in both systems. The first two columns present the reuse levels of lines and words. The right two columns specify the total number of lines/words of the various

DVItype - TEX	lines	words
$L_{DVItype}$	2,136	13,606
L_{TEX}	21,541	122,137
$R_{DVItype/TEX}$	18.8%	32.1%
$R_{DVItype(34.9\%)/TEX}$	53.8%	75.2%

Table 3. Reuse level of DVItype in TEX.

chapters in the TEX system. Remember that the definition of reuse stated earlier indicates that all these chapters are *reused*. The results in these tables indicate the extent of reuse within these chapters. The high reuse values of the chapters entitled 'Character Set', 'Input and Output' and 'Reporting Errors' were expected (see Section 4).

TEX and METAFONT contain lines (even paragraphs) in which the only difference is a replacement of the word TEX with the word METAFONT. Also, there exist sentences that have been improved by a change of word ordering or by inserting or deleting single words. Additionally, METAFONT has many index entries that are not in TEX, but these do not affect the reuse level.

We investigated several disturbances that did not raise the reuse level as much as we had expected. For example, we removed index entries and replaced the words TEX and METAFONT with the string xxx. The total reuse level for lines increased from 14.3% to 14.8%, and for words from 21.5% to 21.6%. The levels for similar chapters increased from 42.8% to 44.7% and from 60.7% to 61.3%.

The numbers in the table, therefore, are a lower bound of the reuse level. This fact is also expressed in the rather high difference between R_l and R_w.

The example in Figure 2 clearly demonstrates how much care was taken in adapting reused text.

6.2 DVItype and TEX

Six of the 15 chapter titles in DVItype are in TEX. The descriptions of the character set and the device-independent file format have a reuse level of about 70 percent (similar to Table 2). These two chapters comprise one fourth of DVItype. There are other chapters not appearing in TEX but in other tools like gftodvi and gftype (which we did not include for space reasons). Table 3 presents the reuse levels of DVItype in TEX.

When Knuth created a data structure, he would immediately write routines to output the data structure

Common TEX & METAFONT Chapters	R_l	R_w	T_l	T_w
Introduction	63.1%	82.4%	377	3,152
Character Set	81.6%	76.9%	206	1,097
Input and Output	81.1%	93.4%	301	2,333
String Handling	71.5%	90.2%	246	1,574
On-line and off-line Printing	58.4%	72.0%	291	1,581
Reporting Errors	82.7%	93.3%	359	2,022
Packed Data	61.3%	82.7%	124	767
Dynamic Memory Allocation	72.1%	85.1%	265	1,822
Memory layout	69.7%	75.9%	195	1,014
The hash table	19.4%	53.2%	309	1,368
The command codes	0.0%	19.9%	163	1,367
Saving and restoring equivalents	3.1%	8.4%	291	1,803
Token lists	16.8%	34.7%	161	1,027
Intro to the syntactic routines	26.7%	61.4%	86	645
Input stacks and states	47.8%	60.1%	429	3,121
Maintaining the input stacks	47.5%	73.5%	139	729
Getting the next token	24.1%	39.2%	465	2,653
Expanding the next token	7.3%	18.3%	477	2,772
Conditional processing	17.1%	38.1%	345	1,433
File names	62.8%	84.6%	433	2,565
Font metric data	25.8%	46.1%	802	4,949
(Un)dumping the tables	30.1%	53.6%	435	1,696
The main program	56.7%	83.4%	208	1,073
Debugging	71.9%	89.2%	64	344
System-dependent changes	80.0%	97.6%	10	83
Index	64.3%	96.1%	14	137
Total	42.8%	60.7%	7,495	43,127

Table 2. Reuse level of TEX in METAFONT by chapters.

METAFONT - METAPOST	lines	words
$L_{METAFONT}$	20,481	109,307
$L_{METAFONT}$	20,460	104,375
$R_{METAFONT/METAPOST}$	63.4%	67.0%
$R_{METAFONT(80.8\%)/METAPOST}$	79.5%	85.1%

Table 4. Reuse level of METAFONT in METAPOST.

cweave - fweave	lines	words
L_{cweave}	3,726	18,456
L_{fweave}	6,103	24,640
$R_{cweave/fweave}$	8.8%	39.6%
$R_{cweave(60.6\%)fweave}$	14.6%	64.8

CWEB's common-FWEB's common lines words
L CWEB's common 1,143 6,726
L FWEB's common 4,070 17,703
R CWEB's common (100%)/ 12.9% 45.7%
FWEB's common

Table 5. Reuse level of *cweave* in *fweave*.

Table 7. Reuse level of CWEB's common.web in FWEB's common.web.

with appropriate annotations. DVItype has that functionality. We note that similar patterns are common in today's books on languages for object-oriented programming. Also, this is common in application libraries in a number of areas.

6.3 METAFONT and METAPOST

The highest reuse in our studies resulted from comparing METAFONT and METAPOST. More than 60 percent of METAFONT (20,000 lines and more than 100,000 words) is reused in METAPOST. METAFONT has 52 chapters; METAPOST has 49. Of these chapters, 44 with the same title appear in both and 24 have a reuse level higher than 90 percent. Except for three chapters, all the other chapters have a reuse level higher than 70 percent. These results are presented in Table 4.

6.4 CWEB and FWEB

The differences of the reuse levels of *cweave* in *fweave* should be noted because of the difference between R_l and R_w. This indicates text scavenging, significant reuse and slight modifications (see Table 5). The results are similar for *ctangle* and *ftangle* (see Table 6).

Note that *fweave* and *ftangle* are significantly larger than *cweave* and *ctangle*, respectively. The reason for this is that FWEB deals with Fortran and several other languages, whereas CWEB deals with C and C++.

CWEB and FWEB employ black-box reuse in extracting common parts of the *tangle* and *weave* processors. These were collected in *common*.web. Thus, besides comparing *tangle* and *weave*, investigating *common*.web reveals some more reuse.

ctangle - ftangle	lines	words
$L_{ctangle}$	1,283	6,528
$L_{ftangle}$	5,649	22,335
$R_{ctangle/ftangle}$	6.2%	29.4%
$R_{ctangle(58.9\%)/ftangle}$	10.6%	49.8%

Table 6. Reuse level of *ctangle* in *ftangle*.

All the chapters in CWEB's common part also appear in FWEB. The difference between R_l and R_w is noticeable (see Table 7).

7. Discussion

Code and documentation were reused in the systems studied. This was done primarily by code and documentation scavenging. There are significant differences between line and word reuse due to extensive word-smithing on many segments of code and documentation to present information in the best possible manner.

Each system was created as a self-contained, homogeneous work. To achieve this, reused parts from other systems sources were reworked and adapted carefully. Such adaptations included changing the system name (e.g., TEX to METAFONT), changing the word order or modifying single words for better layout results. Often these adaptations were real improvements, like the addition of index entries. This is white-box reuse at its best. Black-box reuse offers large production gains. This was done in the CWEB and FWEB systems. The lengths of CWEB's *common*.web, *ctangle*.web and *cweave*.web are 54, 162, and 52 kilobytes, respectively.

The following question arises: Would the demonstrated degree of reuse and adaptation have been possible without scavenging code and documentation? Our answer to that question is simple no! We believe that it is obvious that writing and documenting a software system from scratch will lead to different program and documentation structure than building it by reusing existing components and that documentation quality is likely the key component to being able to be effective in reuse. It should be noted that each of the codes studied were relatively self-contained.

Object-oriented development systems were not readily and widely available at the time TEX and the original WEB system were built. If they had been implemented in an object-oriented manner, classes would likely have been reused by building subclasses, not by direct modifications. Documentation needs a similar approach to adaptation and reuse without direct modification, e.g., by means of object-oriented documentation [24].

Knuth's WEB has an include facility which could have been used to facilitate more black-box reuse. However, he chose white-box reuse in a number of instances where black-box reuse would have been

easily possible. For example, a chapter could have been done separately that included documentation and routine interfaces for tfm files. The routines could have been used in a black-box fashion. However, they were integrated and edited (converted into white-box). Code and documentation could have been developed that included data structures, input routines and output routines. METAFONT would not use the input routine and TEX and DVItype would not use the output routine. As we mentioned earlier, he improved (and specialized) documentation and added index entries (apparently to make it more usable).

WEB systems include adaptation features through *change* files. This allows changing code and/or documentation while maintaining a *canonical form.* This is a rather limited way of adaptation, but proved to be effective for porting purposes. Porting was additionally supported by index entries for sections that might have to be changed due to system dependencies. For example, in TEX 61 such index entries were supplied. The primary intent of *change* files was to support portability and to keep the base versions of major programs intact. This was successful, and new releases of TEX programs often require the change of only one line in the change file, the one which includes the version in the banner.

We pose a question of what would be the changes in rewriting some of the standard works in a literate fashion. Kernighan and Ritchie's (K&R) guide to the C programming language is not a standard, but it is an authoritative guide [9]. The title of Appendix B is "Standard Library". Paragraphs from this section should be of central importance in black-box reuse.

We note that K&R tried to maintain the brevity of the first edition. When we created 'literate versions' of paragraphs from this appendix, we had the same feelings that are reflected by two quotes from Thimbleby [26].

> I was surprised how the original commentary (which looked all right embedded in code) looked insubstantial when set apart in the literate style.
>
> A literate programming style is not, to my mind, what literate programming is all about. How literate programming is done, and how easily it can be done and redone, changes the way one programs. It provides new incentives. There is an incentive to make code and documentation consistent (by developing code and documentation concurrently). There is an incentive to explain, and hence understand what you are doing ...

We agree with Thimbleby's statements. Literate programming is more than just integrating source code and documentation. The care with which documentation is done may well affect how much black-box reuse should be done in each application.

We have studied literate programs from the viewpoint of reuse and observed the following:

- The TEX and WEB systems were implemented in a literate manner. These are medium-sized systems. We believe that literate programming is an excellent methodology for the development of large software systems.
- Significant reuse was common in the investigated systems, even though mostly white-box reuse. Some black-box reuse was done in the CWEB and FWEB systems. The absence or low degree of black-box reuse was due to the choice of the programming language (Pascal) more than to literate programming. Reuse and literate programming can coexist comfortably.
- The examples presented show that both code and documentation were reused, modified, extended and adapted.
- Hobby's reuse of METAFONT, Levy's reuse of WEB, and Krommes' reuse of CWEB are ample proof that excellent documentation aids in the reuse of software.
- Levy and Krommes' newer WEB systems demonstrate that reusable literate components were extracted from Knuth's WEB. Significant reuse of elements of TEX are in today's word processors. The hyphenation algorithm from TEX is widely (re)used. This was obviously done in a white-box fashion.

We note that emphasizing reuse makes it difficult to produce software systems as self-contained books (such as Knuth's *Computers and Typesetting* series [11, 12, 13, 14]. Significant effort was made to keep these volumes in harmony. The lion with Hercule Poirot (in TEX) and the lioness with Miss Marple (in METAFONT) are probably the most obvious example. When software systems are written like this and with so much care, it can really be a pleasure to read them. Bentley commented about the pleasure of reading them in much the same fashion one would read an entertaining novel [2]. Knuth has pointed out that the reading aids of the WEB style also makes it more tractable to read a portion of a code without reading the whole code. We feel this is a significant advantage

in maintenance activities (and maintenance is the dominant cost factor of software).

Building software systems out of reusable parts will lead to thinner books with more references, which makes sequential reading less pleasant. But it will help in making reading more efficient. We argue that code and documentation must be designed for reuse. Some effort would be necessary in order to extract common information in the systems we investigated and provide it in a way that it could be reused (other than by text scavenging). The advantages of these efforts would be similar to those achieved when reusing pure code (see Section 3), e.g.:

- Errors needed to be corrected only once rather than redundantly.
- Improvements of the documentation would affect all systems, e.g., more index entries, style.
- Parts of the TEX systems could be reused in other software systems also (including code and documentation without the need of direct modifications).

The documentation should have the same degree of black-box reuse as code. Current techniques and tools do not sufficiently support this.

8. Conclusions

We have investigated some TEX and WEB systems for reuse. These systems have been implemented as literate programs. Therefore if they incorporate reuse, they illustrate reuse of both code and documentation. Most of this reuse was of the white-box variety. We determined reuse levels by investigating the chapters of the systems with the same (or similiar) titles. Then we made a comparison based on lines and words. The different results achieved by comparing lines and words indicate that most reused components were carefully edited and adapted. The process is not unique to the study of literate programs.

We conclude the following:

- White-box reuse is important and was common in these codes.
- White-box reuse impacts on both code and documentation and cannot be ingnored with either.
- The only examples of black-box reuse in these systems was in the newer WEB systems.

Knuth had an include facility in WEB. Although this was not in an object-oriented system, he could have used this facility to do black-box reuse in a number of instances. He chose white-box reuse. In several instances where black-box reuse was an option, he made the documentation specific to its context.

The first author ported the TEX systems to several computer systems and is convinced that Knuth's attention to detail helped considerably. He also acknowledges that today's systems are quite different.

These systems are appropriate for further study that might lead to a better understanding of where it is appropriate to do white-box or black-box reuse. For example, these questions remain open:

- Is it possible to determine the effect of documentation on successful reuse? (It obviously does facilitate reuse.)
- Does literate programming affect the level of reuse? (This could not be answered by studying these systems because no programming team was ever involved.)
- Does object-oriented programming reduce the importance of white-box reuse?
- Does object-oriented programming change the requirements of associated documentation?

Acknowledgements: We are particularly grateful for the constructive criticism of the reviewers, which has greatly improved this paper and our presentation of the material.

References

1. Avenarius A, Oppermann S (January 1990) FWEB: A Literate Programming System for Fortran 8X. ACM SIGPLAN Notices, Vol. 25, No. 1, pp. 52–58
2. Bentley J (May 1986) Programming Pearls—Literate Programming. Communications of the ACM, Vol. 29, No. 5, pp. 364–369
3. Biggerstaff TJ, Perlis AJ (1989) Software Reusability, Vol. I: Concepts and Models, ACM Press
4. Booch G (1987) Software Components with Ada: Structures, Tools, and Subsystems. Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA
5. Braun C (1994) Reuse. In [21] pp. 1055–1069
6. Childs B, Sametinger J (November 1996) Reuse Measurement with Line and Word Runs. TOOLS Pacific '96, Melbourne, Australia
7. Hobby J (April 1992) A User's Manual for METAPOST. Computing Science Technical Report No. 162, AT&T Bell Laboratories
8. Hobby J (September 1992) Introduction to METAPOST. EuroTEX '92 Proceedings, pp. 21–26, TEX Users Group

9. Kernighan BW, Ritchie DM (1988) The C Programming Language. 2nd ed., Prentice Hall, Engelwood Cliffs, NJ
10. Knuth DE, Fuchs DR (April 1986) TEXware. Stanford Computer Science Report 1097
11. Knuth DE (1986) The TEX Book. Volume A of Computers & Typesetting, AW
12. Knuth DE (1986) TEX: The Program. Volume B of Computers & Typesetting, AW
13. Knuth DE (1986) The METAFONT Book. Volume C of Computers & Typesetting, AW
14. Knuth DE (1986) METAFONT: The Program. Volume D of Computers & Typesetting, AW
15. Knuth DE (July 1989) The Errors of TEX. Software—Practice and Experience, Vol. 19, No. 7, pp. 607–685
16. Knuth DE (1992) Literate Programming. Stanford University Center for the Study of Languages and Information, Leland Stanford Junior University
17. Knuth DE, Levy S (1993) The CWEB System of Structured Documentation, Version 3.0. AW
18. Krommes J (Feb. 1990) FWEB (Krommes) vs. FWEB (Avenarius and Oppermann). TEX-hax, Vol. 90, No. 19
19. Krueger W (June 1992) Software Reuse. Computing Surveys, Vol. 24, pp. 131-183
20. Levy S (January 1993) Literate Programming and CWEB. Journal on Computer Language, Vol. 10, No. 1, pp. 67-70
21. Marciniak JJ (Editor-in-Chief) (1994) Encyclopedia of Software Engineering. Vol. 1, John Wiley & Sons
22. Mili H, Mili F, Mili A (June 1995) Reusing Software: Issues and Research Directions. IEEE Transactions on Software Engineering, Vol. 21, No. 6, pp. 528–562
23. Pappas TL (Frank) (5–8 March 1990) Literate Programming for Reusability: A Queue Package Example. Proceedings of the Eighth Annual Conference on Ada Technology, Atlanta, GA, pp. 500–514
24. Sametinger J (January 1994) Object-Oriented Documentation. ACM Journal of Computer Documentation, Vol. 18, No. 1, pp. 3–14
25. Sewell W (1989) Weaving a Program: Literate Programming in WEB, Van Nostrand Reinhold
26. Thimbleby H (June 1989) A Review of Donald C. Lindsay's Text File Difference Utility diff. Communications of the ACM, Vol. 32, No. 6, pp. 752–755
27. Wegner P (July 1984) Capital-Intensive Software Technology, IEEE Software, Vol. 1, No. 3, reprinted in [3]