Editor: Diomidis Spinellis ■ Athens University of Economics and Business ■ dds@aueb.gr

The Way We Program

Diomidis Spinellis

If the code and the comments disagree, then both are probably wrong. —Norm Schryer

I can still remember the first time I laid eyes on production-quality source code. This was in the early 1980s, and the code was the BIOS listing of the original IBM PC. The 5,940 lines of code spanned 80 neatly typeset pages in a three-ring slip-covered binder. Two things made a lasting impression on me. The first was the elation of being able to read, understand, and learn from the code that made a real machine tick. This might have sparked my current practical and research interests in open source software. The second was the way the code was commented. The BIOS was written in 8086 assembly language, and almost every line had a comment on its right-hand side. By poring over the code and its comments, I learned 8086 assembly, programming style, and the PC's hardware architecture.

Comments, identifiers, and whitespace

So how important are comments in the programs we write? To answer this question, I set out to measure the percentage of source code size occupied by comments in a few programs. It then occurred to me that comments are only one of the mechanisms we developers use to communicate with our colleagues through the source code. Other methods include creating meaningful identifiers, laying out the program with whitespace, using language-provided high-level abstractions, and developing our own abstractions. To get a feeling of how important these mechanisms are, have a look at Figure 1 listing a Basic interpreter implemented in 1,536 bytes. I wrote that program with the express goal to communicate as little information as possible, in order to submit it to the 1990 International Obfuscated C Code Contest (www.ioccc.org/years.html#1990_dds). The code uses single-letter identifiers, employs minimal whitespace, and doesn't contain any pesky comments. This is the type of code that can keep us awake at night—a literal and metaphorical nightmare.

To see how we use these three mechanisms in practice, I took 30 programs of various sizes and measured what percentage of their source code consisted of comments and whitespace. I also calculated the percentage of source code size devoted to meaningful identifiers by counting the number of (nonlibrary) unique identifiers, then deriving the minimum number of characters required for expressing them, and finally obtaining the corresponding source code size savings obtainable from using the smallest possible identifier names. (It turns out that for programs up to 15 KLOC, you can get away with two-character identifiers. For all but the three largest systems I measured—the Linux, Solaris, and FreeBSD kernels—three-character identifiers are perfectly adequate. And this calculation is conservative, because I ignored the savings possible from reusing identifiers in different scopes. Malfeasant programmers and language designers, take note.) Figure 2 shows a graphical summary of this small study. (The remaining percentage of each program's source code composition

```
#define O(b,f,u,s,c,a)b(){int o=f();switch(*p++){X u:_ o s b();X c:_ o a b();default:p--;_ o;}}
#define t(e,d,_,C)X e:f=fopen(B+d,_);C;fclose(f)
#define U(y,z)while(p=Q(s,y))*p++=z,*p=' '
#define N for(i=0;i<11*R;i++)m[i]&&
#define I "%d %s\n",i,m[i]
#define X ;break;case
#define _ return
#define R 999
typedef char*A;int*C,E[R],L[R],M[R],P[R],l,i,j;char B[R],F[2];A m[12*R],malloc
(),p,q,x,y,z,s,d,f,fopen();A Q(s,o)A s,o;{for(x=s;*x;x++){for(y=x,z=o;*z&&*y==
*z;y++)z++;if(z>o&&!*z)_ x;}_   0;}main(){m[11*R]="E";while(puts("Ok"),gets(B)
)switch(*B){X'R':C=E;l=1;for(i=0;i<R;P[i++]=0);while(l){while(!(s=m[l]))l++;if
(!Q(s,"\"")){U("<>",'#');U("<=",'$');U(">=",'!');}d=B;while(*F=*s){*s=='"'&&j
++;if(j&1||!Q(" \t",F))*d++=*s;s++;}*d--=j=0;if(B[1]!='=')switch(*B){X'E':l=-1
X'R':B[2]!='M'&&(l=*--C)X'I':B[1]=='N'?gets(p=B),P[*d]=S():(*(q=Q(B,"TH"))=0,p
=B+2,S()&&(p=q+4,l=S()-1))X'P':B[5]=='"'?*d=0,puts(B+6):(p=B+5,printf("%d\n",S
()))X'G':p=B+4,B[2]=='S'&&(*C++=l,p++),l=S()-1 X'F':*(q=Q(B,"TO"))=0;p=B+5;P[i
=B[3]]=S();p=q+2;M[i]=S();L[i]=l X'N':++P[*d]<=M[*d]&&(l=L[*d]);}else p=B+2,P[
*B]=S();l++;}X'L':N printf(I)X'N':N free(m[i]),m[i]=0   X'B':_ 0 t('S',5,"w",N
fprintf(f,I))t('O',4,"r",while(fgets(B,R,f))(*Q(B,"\n")=0,G()))X 0:default:G()
;}_ 0;}G(){l=atoi(B);m[l]&&free(m[l]);(p=Q(B," "))?strcpy(m[l]=malloc(strlen(p
)),p+1):(m[l]=0,0);}O(S,J,'=',==,'#',!=)O(J,K,'<',<,'>',>)O(K,V,'$',<=,'!',>=)
O(V,W,'+',+,'-',-)O(W,Y,'*',*,'/',/)Y(){int o;_*p=='-'?p++,-Y():*p>='0'&&*p<=
'9'?strtol(p,&p,0):*p=='('?p++,o=S(),p++,o:P[*p++];}
```

Figure 1. A complete Basic interpreter as a code blob, written with the goal of communicating as little information as possible.

is what the compiler actually requires to create the executable code.)

I found the results surprising. In a typical system's source code, more than half the code serves not as instructions to the computer but as a communications vehicle targeting developers. Furthermore, although we often discuss the importance of comments (this was my original idea for this column), it seems that programmers devote almost the same area of screen real estate to meaningful identifiers and even the humble whitespace. Therefore, all three mechanisms appear to be equally important. In addition, although I expected that the measured composition of a program's elements would differ according to a program's size, my data didn't show this variation for programs spanning thousands to millions of code lines. It seems that for any nontrivial program, each of these three mechanisms carries equal weight; we could well posit that this reflects a stylistic equilibrium we reached through evolution. Finally, although the ratio of code serving the developers rather than the compiler seems high, note that it doesn't even include the overhead of code employed for building and using various abstractions, such as method definitions and calls. Although some abstractions can result in more compact code, I feel that on balance abstractions (deservedly) contribute positively to a system's source code size.

Implications

Nobody in their right mind, I think, would ever dream of writing or even representing the Linux kernel as a blob (at least not before getting ideas from Figure 1). Nevertheless, the large proportion of the source code explicitly targeting developers is profoundly significant to me, because it confirms my belief that source code is the most important artifact of the software development process. Programming is not coding. Programming is not the mechanical transfer of a software design into a form the computer can execute. As many others have observed, programming is an art, and the three elements I've measured are separate, identifiable artistic expressions. The writing of comments is prose literature, aiming to tell the story behind the code. The formation of layout through whitespace is sculpture, seeking to show the code's hidden structure. Finally, the choice of meaningful identifiers is almost stylized poetry, a type of creative communication through a few words adhering to a specific form. Our overarching goal is to communicate effectively—plainly, succinctly, and unambiguously.

The large amount of creativity that goes into software source code has several practical implications. It means we can get great code by hiring talented developers and compensating them as we would pay a great artist (ideally not the archetypal starving one). It also means that we should

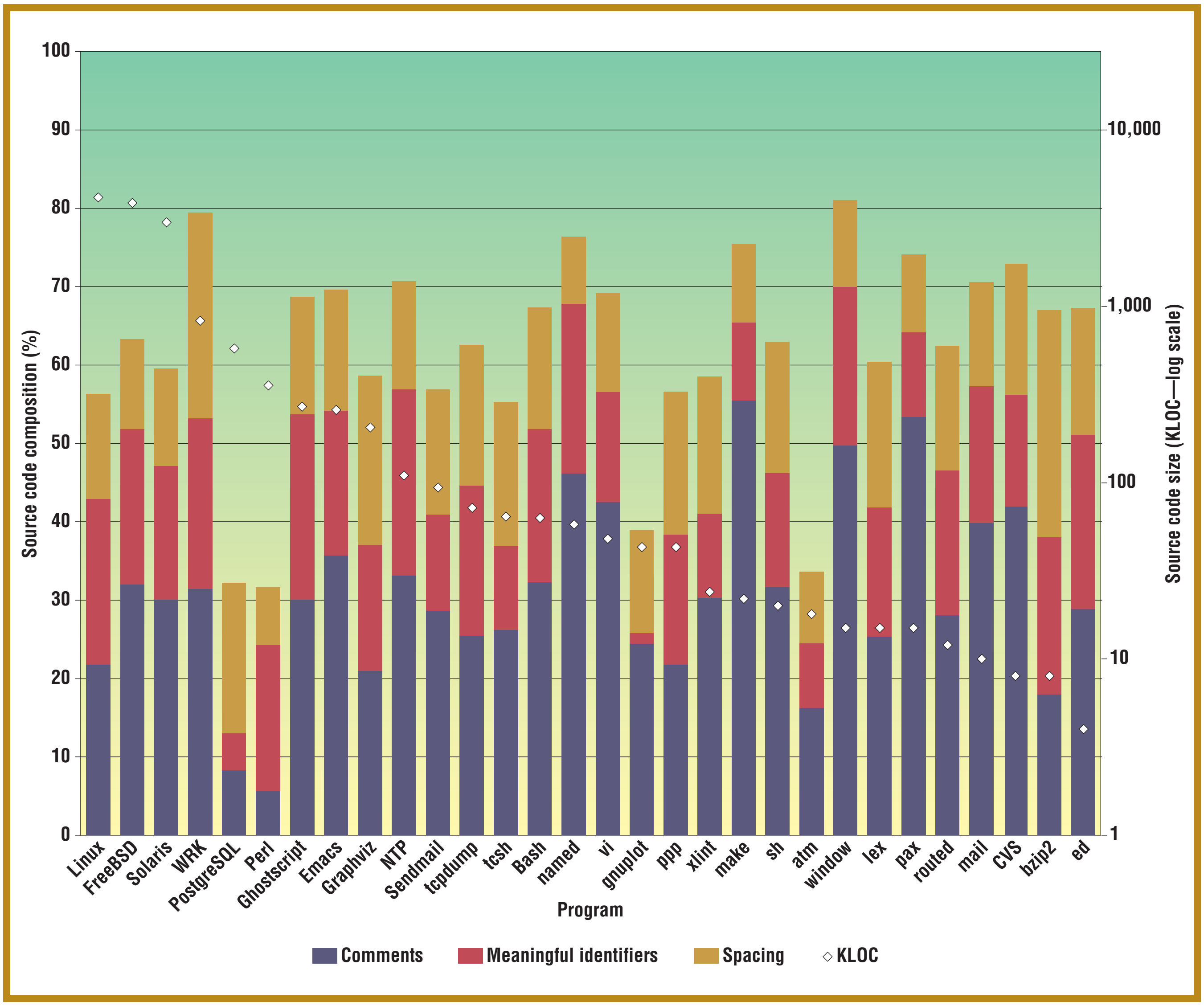

Figure 2. Developer-oriented elements in the composition of 30 programs' source code. Typically, more than half of the source code facilitates developer communications rather than delivering instructions to the computer.

take care of code, treating it as a prized possession. We should learn, respect, apply, and preserve style guidelines and naming conventions; we should treat each comment as part of an essay that will be marked by the most exacting English teacher we've ever had.

To paraphrase William Ward: The mediocre code compiles. The good code runs. The superior code passes tests and inspections. The great code inspires.

Diomidis Spinellis is an associate professor in the Department of Management Science and Technology at the Athens University of Economics and Business and the author of *Code Quality: The Open Source Perspective* (Addison-Wesley, 2006). Contact him at dds@aueb.gr.