Principle 5: Information flow
Material for peer review

Norman Ramsey

Prologue

Principle 5 (Information flow) helps us put words in the right place. In particular, it helps us choose helpful words for the beginning and the end of each sentence. But these choices, if taken serious, must also involve two other principles: Principle 6 (Emphasis), which tells us to put the most important information at the end of a sentence, and Principle 7 (Coherence), which tells us to begin a paragraph's sentences with coherent subjects. Because all three principles influence the placement of words within a sentence, all three contribute to the analysis below.

Context of the text

The original text is the opening paragraph of Chapter 8 of my doctoral dissertation, which I wrote in 1992, before I had begun to study any of the methods we are learning in class. The dissertation is called *A Retargetable Debugger*, and the chapter title is "Stack Walking." The preceding chapters, 1 to 7, provide considerable contextual information:

- Chapter 2 presents the user's experience of the debugger `ldb`, and it defines the technical terms *current focus* and *stopping point*. It also shows example summaries of the call stack, and it mentions walking the stack.
- Chapter 3 describes *abstract memories*.
- Chapter 4 discusses *symbol tables*.

The intended audience were primarily the members of my dissertation committee, but I also hoped that the dissertation might be read by others who are interested in details of programming-language implementation, and especially by others who might wish to implement a debugger.

My purpose in writing this paragraph is to focus my reader's attention on two claims: that walking the stack requires both machine-dependent and machine-independent code, and that my design limits the machine-dependent code to a small number of methods.

The revised text as sent to the peer reviewer

R1 `ldb`'s users can debug any active procedure by moving the current focus to the appropriate stack
R2 frame. A stack frame provides the context for commands; this context includes a scope for name resolution and an abstract memory against which to
R3 print variables and evaluate expressions. Stack frames are implemented using mostly machine-independent code, which may, for example, use the symbol table to build a scope for a stopping point.
R4 But stack frames are also implemented using
R5 machine-dependent code. This code, which knows the target's calling sequence, walks the stack and finds out where control has stopped, restores registers from the stack, and builds an abstract mem-
R6 ory. The code knows how frames are related to their callers' frames and where registers are
R7 saved within a frame. To separate such machine-dependent code from machine-independent code,
R8 `ldb` uses Modula-3 subtyping. Subtyping helps `ldb` isolate the machine-dependent code in a few methods.

Self-review

Level of mastery:

- Sentences R1, R2, R5, and R8 respect the principle. In sentences R3, R4, and R6, I have chosen to allow the principle of coherence (coherent subjects) to trump the principle of information flow.

  Sentence R7 does not really follow either principle, but I was unable to make the beginning any better. The ending of R7 does, however, apply Principle 6 (Emphasis).

Summary judgement:

- The text says almost exactly what I want it to say. And through a combination of information flow, coherent subjects, and emphasis, I'm almost completely happy with the placement of words at the beginning and end of each sentence.

Model Self-Assessment of Writing Principles
Principle 5: Information Flow

Norman Ramsey

January 2017

Prologue

Principle 5 (Information flow) helps us put words in the right place. In particular, it helps us choose helpful words for the beginning and the end of each sentence. But these choices, if taken serious, must also involve two other principles: Principle 6 (Emphasis), which tells us to put the most important information at the end of a sentence, and Principle 7 (Coherence), which tells us to begin a paragraph's sentences with coherent subjects. Because all three principles influence the placement of words within a sentence, all three contribute to the analysis below.

Original and revised texts

The original and revised texts are shown in Figure 1.

Context of the text

The original text is the opening paragraph of Chapter 8 of my doctoral dissertation, which I wrote in 1992, before I had begun to study any of the methods we are learning in class. The dissertation is called *A Retargetable Debugger*, and the chapter title is "Stack Walking." The preceding chapters, 1 to 7, provide considerable contextual information:

- Chapter 2 presents the user's experience of the debugger `ldb`, and it defines the technical terms *current focus* and *stopping point*. It also shows example summaries of the call stack, and it mentions walking the stack.
- Chapter 3 describes *abstract memories*.
- Chapter 4 discusses *symbol tables*.

The intended audience were primarily the members of my dissertation committee, but I also hoped that the dissertation might be read by others who are interested in details of programming-language implementation, and especially by others who might wish to implement a debugger.

My purpose in writing this paragraph is to focus my reader's attention on two claims: that walking the stack requires both machine-dependent and machine-independent code, and that my design limits the machine-dependent code to a small number of methods.

Analysis and explanation

Here are the listed elements (a) to (e) of the analysis and explanation called for in the handout.

(a) To show the old and new information, I have put annotated versions of both texts in Figure 2. Old information is shown with dotted underlining and new information is shown with solid underlining.

(b) The original paragraph is very dense with new information. I find it a bit scary, especially sentence O4. The revision is less dense, packing roughly the same amount of new information into eight sentences rather than five. The paragraph is still dense with new information, but given how I've placed the payload in the final sentence, I'm relatively happy with it.

(c) Here are the problems I have diagnosed in my original text:

O1/O2. Information does not flow properly from sentence O1 to O2. I have repaired the problem by moving the phrase "stack frame" to the very end of O1 so that O2 can begin with this information.

O2. This sentence contains four pieces of new information. Also, I'm not entirely happy with using the appositive form

Stack Walking (original)

O1 `ldb`'s users can debug any active procedure by selecting the appropriate stack frame as the
O2 current focus. A stack frame provides the context for commands, in particular, a scope for name resolution and an abstract memory against which to print variables and evaluate
O3 expressions. Much of the work of implementing stack frames is machine-independent, e.g., using the symbol table to build a scope for a
O4 stopping point. But to walk the stack and to find out where control has stopped, to restore registers from the stack, and to build an abstract memory, `ldb` needs to know the target's calling sequence; the location of saved registers within a frame and the relationships of frames to their caller's frames are machine-
O5 dependent. Modula-3 subtyping helps decompose the problem into machine-independent and machine-dependent parts, isolating the machine-dependent code in a few methods.

Stack Walking (revised, after peer review)

R1 `ldb`'s users can debug any active procedure by moving the current focus to the appropriate
R2 stack frame. A stack frame provides the context for commands; this context includes a scope for name resolution and an abstract memory against which to print variables and evaluate ex-
R3 pressions. Stack frames are implemented using mostly machine-independent code, which may, for example, use the symbol table to build a
R4 scope for a stopping point. But stack frames are also implemented using machine-dependent
R5 code. This code, which walks the stack and finds out where control has stopped, restores registers from the stack, and builds an abstract memory,
R6 knows the target's calling sequence. This code also knows how frames are related to their callers' frames and where registers are saved within a
R7 frame. To separate such machine-dependent code from machine-independent code, `ldb` uses
R8 Modula-3 subtyping. Subtyping helps `ldb` isolate the machine-dependent code in a few methods.

Figure 1: Original and revised texts

Stack Walking (original)

O1 `ldb`'s users can debug any active procedure by selecting the appropriate stack frame as the
O2 current focus. A stack frame provides the context for commands, in particular, a scope for name resolution and an abstract memory against which to print variables and evaluate
O3 expressions. Much of the work of implementing stack frames is machine-independent, e.g., using the symbol table to build a scope for a stopping
O4 point. But to walk the stack and to find out where control has stopped, to restore registers from the stack, and to build an abstract memory, `ldb` needs to know the target's calling sequence; the location of saved registers within a frame and the relationships of frames to their caller's
O5 frames are machine-dependent. Modula-3 subtyping helps decompose the problem into machine-independent and machine-dependent parts, isolating the machine-dependent code in a few methods.

Stack Walking (revised, after peer review)

R1 `ldb`'s users can debug any active procedure by moving the current focus to the appropriate
R2 stack frame. A stack frame provides the context for commands; this context includes a scope for name resolution and an abstract memory against which to print variables and evaluate
R3 expressions. Stack frames are implemented using mostly machine-independent code, which may, for example, use the symbol table to build a
R4 scope for a stopping point. But stack frames are also implemented using machine-dependent
R5 code. This code, which walks the stack and finds out where control has stopped, restores registers from the stack, and builds an abstract memory,
R6 knows the target's calling sequence. This code also knows how frames are related to their callers' frames and where registers are saved within a
R7 frame. To separate such machine-dependent code from machine-independent code, `ldb` uses
R8 Modula-3 subtyping. Subtyping helps `ldb` isolate the machine-dependent code in a few methods.

Figure 2: Original and revised texts, annotated

"[comma] in particular [comma]" to explain the "context." I address the problem by splitting the sentence into two independent clauses separated by a semicolon. This enables me to begin the second clause "this context" which is now the old information from the end of the first clause. I believe this change makes the vast amount of new information flow a little more easily.

O3. The sentence begins with new information. Suddenly the topic has changed to the "work of implementing."

Another problem in this sentence has to do with the choice of nouns and verbs. Notice the subject "work" and the main verb "is." Since 1992 I've learned a thing or two, and I believe that "code" (both machine-dependent and machine-independent) is an important character in my story. Here the agent/action/object is "code implements stack frames," and in sentence R3 I've changed the subject and verb accordingly.[1]

Finally, I've put sentence R3 into the passive voice so that I can begin the sentence with "stack frames." This phrase is old information from the end of sentence R1, and it is also a coherent subject with sentence R2.

O4. The sentence not only begins with new information but also contains almost exclusively new information. This sentence becomes sentences R4, R5, and R6.

To make the revision more coherent, I've also removed "`ldb`" as a character and replaced it with the "code" character mentioned above.

O5. The sentence begins with new information and then returns to old information in the middle, before delivering more new information at the end. (At least the most important new information is at the end.) I've addressed the problem by splitting this sentence into two sentences, so that each one (R7 and R8) can deliver an important piece of new information at the end.

[1] And as a bonus, the nominalization "using" becomes the verb "use."

(d) Overall I'm very pleased with the way the revised text moves from old information to new information. Here is my assessment of each sentence:

- R1 works perfectly; it ends with the stack frame, whose implementation is the subject of this chapter, and which is the first part of the next sentence.
- R2 works well enough: it begins with old information from the end of the preceding sentence, and it explains it with one piece of new information (the "context"). Then this information is elaborated with three more pieces of new information. I'm quite happy with the information flow here.
- R3 ignores the new information from the preceding sentence and instead returns to the old information from sentence R1: stack frames. The stack frames are a coherent subject for sentences R2, R3, and R4. In sentence R3, the initial, independent clause works very well: it begins with the old information "stack frames" and introduces a very important piece of new information: they are implemented using mostly machine-independent code.

  The second, dependent clause ("which may, for example...") does not please me so much. On grounds of pure information flow, I would do better to make it a separate sentence ("Such code may, for example..."). But I have chosen instead to keep it as a dependent clause, in part because I want to maintain the coherent subject of "stack frames" for these three sentences R2 to R4.
- R4 moves perfectly from old ("stack frames," "implemented") to new "machine-dependent code."
- R5 begins with old information ("this code") and then introduces *five* pieces of new information. The sentence works only because none of this new information is developed within this paragraph.
- R6 is almost exactly like R5, except it introduces only two pieces of new information.
- The flow of information in R7 is pretty good, but not as good as in other sentences. The two kinds of code are old information at the beginning, but the

new idea of "keep separate" is also pretty close to the beginning. Not so good. But in the context of the entire dissertation, the newest information here is "subtyping," which does appear right at the end.

- The flow of information in R8 is also pretty good. It begins with exactly the new information from the previous sentence, and the most new information "a few methods" is at the end. (I wish I could find a single verb to use in R7 and R8, instead of the two different verbs "separate" and "isolate.")

(e) I've placed important material that I wish emphasized at the ends of sentences R1, R4, R7, and R8. I wonder about possibly splitting R3 into two sentences so I could do so in R3 as well.

Self-review

Problems diagnosed in the revised text:

- The main verb of R2 ("provide") is not an important action.
- The subject and main verb of R8 ("Subtyping helps isolate") are not perfect. What's really important here is the isolation, not helping, and subtyping is just a means to that end.

Level of mastery:

- Sentences R1, R2, R5, and R8 respect the principle. In sentences R3, R4, and R6, I have chosen to allow the principle of coherence (coherent subjects) to trump the principle of information flow.

  Sentence R7 does not really follow either principle, but I was unable to make the beginning any better. The ending of R7 does, however, apply Principle 6 (Emphasis).

Summary judgement:

- The text says almost exactly what I want it to say. And through a combination of information flow, coherent subjects, and emphasis, I'm almost completely happy with the placement of words at the beginning and end of each sentence.

Results of peer review

Elise Deitrick was kind enough to point out some information-flow issues in sentence R5; she felt that the information about the calling sequence came too early. And she pointed out some ambiguity about whether "the code" at the beginning of R6 was the same as "this code" at the beginning of R5. I tried to clarify by changing the beginning of R6 to "this code also knows." In the final revision the "knows" at the beginning of R6 is now old information from the very end of R5.

Elise Deitrick
April 7, 2015

Peer Review – Information Flow - Norman

(1) The text follows the principles we're learning in class, and the right things are important. I feel good about this text.

(2) There is new information too early in a sentence, perhaps even at the beginning

- R5 – "Target's calling sequence" – this might be my ignorance, but I don't think this is mentioned prior and it is very early in the sentence – it comes before old information like the idea of "abstract memory." Would it be possible to move this to the previous sentence? I assume this has something to do with *why* the code is machine-dependent???

(3) I'm convinced by the explanation and it's consistent with what I got reading the text on my own.

(4) The revised text is easier to digest and follow. I think that the emphasis made much more of an impact in the revision.

(5) In R5, "this code" probably means machine dependent code, but then in R6, I am unclear is "the code" is still only the machine dependent code, or the code set as a whole. Just for fun, a possible rewrite to R7, not sure if the information flow is much better: "ldb is implemented such that machine-dependent code is separate from machine independent code using Modula-3 subtyping."