Send letters to Editor ■ software@computer.org ■ fax +1 714 821 4010

First Virus?

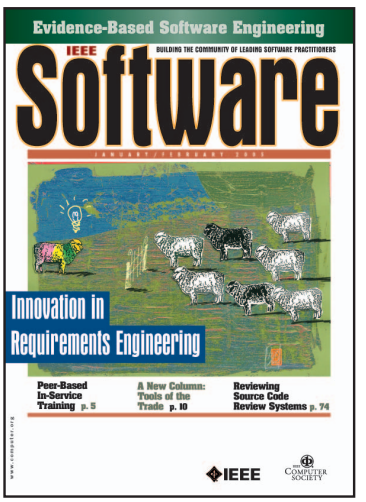
Does anybody know when the first computer virus was created? To get the discussion going, I'll draw a line in the sand: it was no later than 1971. In 1971, as a graduate student at the University of Illinois at Urbana Champaign, I was working on a multiuser PDP-11. I created a process that (a) checked to see if an identical copy of itself was also running as an active process, and if not, created a copy of itself and started it running; (b) checked to see if any disk space (which all users shared) was available, and if so, created a file the size of that space; and (c) looped back to step (a).

As a result, the process stole all available disk space. When users tried to save files, the operating system advised them that the disk was full and that they needed to delete some existing files. Of course, if they did delete a file, my process would immediately snatch up the available space. When they called in a system administrator to fix the problem, he examined the active processes, discovered my "evil" process, and deleted it. Of course, before he left the room, the other evil process would create another copy of itself, and the problem wouldn't go away. The only way to make the computer work again was to reboot.

Of course, 1971 predates the use of the term *virus*. But does anybody know of any earlier instances of virus-like software?

Al Davis
Professor
University of Colorado
adavis@uccs.edu

We welcome your letters. Send them to software@computer.org. Include your full name, title, affiliation, and email address. Letters are edited for clarity and space.



Clarifying ambiguity

Bob McCloskey of the Department of Computing Sciences at the University of Scranton in Scranton, Pennsylvania, pointed out to my coauthor Erik Kamsties and me that we made a mistake with one of the examples in our Quality Time column, "The Syntactically Dangerous *All* and Plural in Specifications" (Jan./Feb. 2005).

The problematic example is, "All of the lights in any room have a single on-off switch." We identified two possible interpretations: (1) each light in any room has its own single on-off switch that isn't shared with any other light, and (2) all the lights in any room share a single common on-off switch.

In the column, we suggested that the first interpretation would be better stated as either (1a) each light in any room has a single on-off switch or (1b) each light in any room has its own on-off switch. Bob pointed out that sentence (1a) means only that each light has a single on-off switch and says nothing about how many lights each switch controls.

Indeed, sentence (1a) allows another interpretation: (2) all the lights in any room share a single common on-off switch! To get sentence (1a) to mean interpretation (1), Bob points out that more words must be added to get sentence (1a′): Each light in any room has a single on-off switch in the room, and each on-off switch in any room controls a single light in the room. Bob points out also that sentence (1b) avoids the problem of sentence (1a) by using the word *own*, which implies that the switch isn't shared with any other light.

Bob is absolutely correct here, and we were somewhat red-faced to have written an ambiguous or incorrect example in an article trying to describe how to avoid ambiguity. We can say only "somewhat," because the interesting thing is that Bob was the first and, so far, the only person to notice the problem since a variant of the example came up in one of my consulting jobs in 1991. I explained the first alternative meaning with sentences corresponding to sentences (1a) and (1b). No one at the company noticed the problem. Moreover, on every occasion since then in which Erik or I has explained the example with the same alternative sentences (1a) and (1b), no one has noticed the problem.

Ambiguity always is only in the eyes of the beholder. Alan Davis's operational, testable definition of ambiguity (from *Software Requirements: Objects, Functions, and States*, Prentice Hall, 1993) drives this point home: "Imagine a sentence that is extracted from an SRS [software requirements specification], given to 10 people who are asked for an interpretation. If there is more than one interpretation, then that sentence is probably ambiguous." Of course, the problem with this test is that, as in software testing, there's no guarantee that the 11th person won't find another interpretation. However, this test does capture the essence of a useful SRS that's unambiguous for most practical purposes. In other words, if everyone involved with a sentence agrees on a single meaning for the sentence, the sentence ain't ambiguous!

In a sense, Bob is the proverbial 11th person who finally found the ambiguity lurking in our example. Perhaps the proximity in time and space of the verbal offering of sentences (1a) and (1b) in opposition to interpretation (2) lulled everyone to accept that sentences (1a) and (1b) were alternative, equally valid expressions for interpretation (1). Perhaps, had we formulated the interpretations and sentences in first-order predicate calculus as Bob did, we or others might have seen the problem earlier. However, the reality is that the overwhelming majority of SRSs are written in natural language. For all of natural language's weaknesses, and because of the weaknesses, we have to work with the weaknesses.

The example we used in the column is an example of both the ambiguity of plural nouns and of the difficulty of establishing correspondences between nouns. Here's an example of the plural ambiguity that avoids the correspondence problem that Bob writes about:

- (3) Each term, students at the University of X take 5 classes.
- (4) Each term, students at the University of X take hundreds of classes. Without domain knowledge about students at universities, you're left wondering about the large difference in the direct objects of these two structurally identical sentences. A little bit of domain knowledge tells us what the sentences mean and suggests less ambiguous ways to express them:
- (3′) Each term, each student at the University of X takes 5 classes.
- (4′) Each term, the students at the University of X take hundreds of classes.

We use this opportunity to alert readers to other descriptions of the plural problem. Attempto Controlled English (www.ifi.unizh.ch/attempto), an unambiguous subset of English in which each sentence has only one meaning, originally outlawed plural because of ambiguities like the ones we reported. The ACE solution is actually workable, because you can express truly plural sentences (such as 4′) in the singular with a collective noun identifying the set of elements the plural noun denotes. For example, you would express 4′ as, "Each term, the set of all students at the University of X takes hundreds of classes." With such constructions ("each student" or "the set of all students"), there's no doubt what is meant.

A glance at any ACE definition or description document is an eye opener, because the commentary explaining the reasons for each restriction is a compendium of ambiguities in English. For her PhD dissertation (available at www.ifi.unizh.ch/attempto/publications/index.html), Uta Schwertel studied ways to allow plural into ACE under restricted and controlled circumstances using *each*, *own*, and other clues as to the intended meaning. "Controlling Plural Ambiguities in Attempto Controlled English" (*Proc. 3rd Int'l Workshop Controlled Language Applications*, 2000; www.ifi.unizh.ch/attempto/publications/papers/claw2000.pdf) describes the essence of her work. We thought that one of her examples of the plural ambiguity—"Three girls lift a table"—was more to the point than any of ours.

Does each of the three girls lift one table or do all three girls together lift one table? We recommend looking at Schwertel's work for her view of the problem and its solutions.

Daniel M. Berry
Professor
University of Waterloo, Canada
dberry@uwaterloo.ca

Erik Kamsties
Carmeq Software & Systems
erik.kamsties@imail.de

"Best practices": Globally accepted, voluntarily adopted standards

One thing engineers do is solve problems, and software engineering is an area that has plenty of problems. Naturally, engineers propose solutions, and to make them attractive, they're tempted to include qualifiers like *best*, *critical*, and *essential*. However, unless the propositions are proven, professional bodies don't accept such terms. For example, let's consider software engineering's "best practices."

Software engineering projects are carried out in partial ignorance. Therefore, each individual project must be viewed as "social experimentation," as Mike W. Martin and Roland Schinzinger said in their book *Ethics in Engineering* (McGraw-Hill, 1996). As long as engineering a software project remains an experiment, there will be confusion. and we must clarify and logically fix the practices.

According to my understanding, only two categories of "practices" exist:

those governed by mandatory or statutory rules and regulations and those governed by other standards and reports.

The first group's document titles often include the words *code*, *act*, and *statutory rules and regulations*. The practices mentioned in these documents have legal binding in the acquirer/purchaser's country. OSHA (Occupational Safety and Health Act) and NEC (National Electrical Code) are typical mandatory practices. These documents contain the "real best practices" accepted by industry, customers, and the general public.

The second type of practices might also be in the form of a code or standard but not be mandatory. I'm not aware of any such code in software engineering. Even international standard ISO/IEC 12207:1995/Amd 2:2004 (Information technology—Software life cycle processes) hasn't yet become a code, and its acceptance varies from project to project depending on the contractual terms.

All other standards and reports acquire legal support when those documents are referred by the purchaser and accepted by the supplier in the contract. Otherwise, they're like any other article, paper, or report. Examples include

- nonmandatory standards,
- recommended practices,
- guides,
- technical reports,
- best practices,
- core activities,
- major activities,
- commitments,
- general conditions of contract, and
- minutes of meetings.

All these nonmandatory documents are invaluable sources of knowledge. It hardly matters whether you call them recommended practices, best practices, guides, or, for that matter, simply activities. It's most important that they deliver consistent performance in industry and can become codes in the future. Maybe no single standard or technical report will become a code or mandatory standard. A combination of them might evolve gradually. ISO 9001 and SA-CMM are the best examples of such evolutionary standards. Numerous organizations in many countries have voluntarily adopted these standards.

Professionals have a duty to generate as many ideas as possible and bring them to the light. Let's not block those ideas. In her letter "What's Best?" (May/June 2004), Ellen Walker said, "If we try to adhere to a scientific approach, relying on solid data to support our decisions, we probably won't publish anything."

For example, let's tolerate the clichés in writing and look for any new ideas therein, until the practices the authors mention develop into accepted standards. Clichés in writing will have to fade out on their own. We can also suggest alternatives. Have we set a moratorium on *paradigm*, *model*, *meta*, *benchmark*, *artifacts*? Of course, we can always add a disclaimer. For example, the SA-CMM declares that

> *[t]here is not "one right way" to implement an acquisition process. The SA-CMM is not engraved in stone. Also, except in a few carefully chosen instances, the SA-CMM does not mandate how the acquisition process should be implemented or who should perform an action; it describes what characteristics the acquisition process should possess. … The SA-CMM should be interpreted in the context of the needs of the organization; just because something is in the SA-CMM does not mean it should be applied automatically.*

Best practices are those standards that are accepted worldwide, willingly implemented, and certified by third parties, and that satisfy all stakeholders.

R.T. Sakthidaran
Professor
KLN College of Engineering, India
sakthidaran@gmail.com