Star Trek, Split Infinitives, and Agile Programming

"Space … the final frontier. These are the voyages of the Starship Enterprise. Its five-year mission: to explore strange new worlds, to seek out new life and new civilizations, to boldly go where no man has gone before."

If you are a reader of CrossTalk, it is a pretty safe bet that you recognize the quote above.

Did you notice the problem lurking in the above quote that has vexed English speakers and writers for almost 200 years? Yes, it is the dreaded split infinitive. It should be "…to go boldly…." The split infinitive is still the subject of disagreement among native English speakers as to whether it is grammatically correct. According to Wikipedia, no other grammatical issue has so divided English speakers since the 19th century, when the split infinitive was declared to be incorrect.

As part of my profession, I write articles—and have done so for over 25 years. I consider myself a pretty decent wordsmith. I have taken several college courses in Technical Writing. Back in 2007, an associate editor at a journal noticed a split infinitive in one of my drafts, and sent me a note jokingly saying, "You should know better." The associate editor was "old school." I, on the other hand, thought nothing about it—because I do not really feel that grammatical rules that originally applied to Latin grammar 2,000 years ago should dictate how I write English today. Modern English usage guides seem to agree—most have dropped their opposition to the split infinitive. Split infinitives were considered wrong 50 years ago, but today they are accepted as correct.

Which brings me to agile programming. I hate to admit it—I was "old school." If the term "2167A" does not make you tremble, you were not developing software 25 years ago. 2167A was a DoD standard, published in 1988. It established, "uniform requirements for software development that are applicable throughout the system lifecycle."

One criticism of 2167A was that it was biased toward the waterfall model. While it did include, "the contractor is responsible for selecting software development methods (for example, rapid prototyping)," 2167A also required "formal reviews and audits" that seemed to lock the vendor into designing and documenting the system before any implementation began. 2167A focused on design documents, rather than the use of accepted CASE tools. Vendors would often use CASE tools to design the software, but then be forced to write 2167A-required documents to describe the CASE-formatted data. The result was that as the system evolved, the CASE tool products could be updated easily, but not the documents. Over time, the design documents required by 2167A became obsolete (assuming they were ever of any real use).

Software development methodologies evolved. In the 1990s we had the CMM®. I was and am a CMM proponent, but let me quote from Wikipedia, "The model was originally intended to evaluate the ability of government contractors to perform a software project. It has been used for and may be suited to that purpose, but critics pointed out that process maturity according to the CMM was not necessarily mandatory for successful software development. Real-life examples where the CMM was arguably irrelevant to successful software development include many shrinkwrap companies (also called commercial-off-the-shelf or "COTS" firms or software package firms). Such firms would have included Claris, Apple, Symantec, Microsoft, and Lotus. Though these companies have successfully developed their software, they have not considered, defined, or managed their processes as the CMM described as level 3 or above, and so would have fitted level 1 or 2 of the model."

In other words, companies using CMM could develop good software, but good software did not depend on the company using CMM.

You see, the role and nature of software development keeps evolving. Early languages had different rules, different purposes, and different focuses. It is hard to imagine agile development in COBOL and Fortran. Modern languages, like C#, Ruby, Perl, and Java have a very close relationship with component libraries, system-supplied support libraries, and higher-level abstractions. It becomes more and more possible to quickly develop something that works—and still have quality software. As long as the software meets good software engineering standards—reliable, understandable, modifiable, and efficient—agile development can get the job done.

Just like Latin grammar rules, perhaps older software development rules are sometimes a bit less relevant nowadays. Back in the 90s, the mere mention of agile sent large-scale software developers screaming "disbeliever" and running for holy water to throw on the heretics. 20 years later, it is a generally accepted way to perform large-scale system development and achieve quality results.

It is all about meeting customer needs and producing quality software. I am not saying the old ways will not work anymore—I am just saying the new ways work, too.

It is the 2010s. Maybe it is time for you "to boldly go" where you have never gone before. Split infinitive or not.

David A. Cook, Ph.D.
Stephen F. Austin State University
cookda@sfasu.edu