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

UML Everywhere

Diomidis Spinellis

flowchart, n.: The innumerate misleading the illiterate. —Stan Kelly-Bootle, The Devil's DP Dictionary

A mechanical engineer who sees the symbol ⊥ in a diagram will immediately realize that a feature is specified to be perpendicular to another. In contrast, a software engineer looking at a diagram's line ending with the symbol ◊ will at best wonder whether it denotes aggregation, as in Unified Modeling Language (UML), a "zero or one" cardinality, as in IDEF1X, or something else invented by a creative academic. Worse, many developers will simply scratch their head in bewilderment.

A standardized and widely used diagramming notation is a sign of a profession's maturity. It simplifies the life of the diverse group of people who read the drawings, it improves the quality of the drawings, and it benefits the profession through network effects. Sadly, in the field of software engineering we've got a long way to travel. Before writing this column, I examined the diagrams printed in the proceedings of the prestigious International Conference on Software Engineering and in this magazine. More often than not, the diagrams employed an ad hoc notation of the authors' invention.

To improve this sad state of affairs, I propose that every one of us should make a concerted effort to use the same graphic notation for drawing all our diagrams. If you don't think that this idea is preposterous, hear the next part: I furthermore propose that for our diagrams we should adopt the graphic notation techniques of UML. If you've managed to remain calm up to this point and you're not yet busily writing an angry letter to the magazine's editor, please read on to see what we'll gain from standardizing to a single notation, and then let me explain why UML isn't so bad compared to your favorite alternative.

Immediate Dividends

The prime benefit of standardizing to UML diagrams **is** that we'll become more effective in reading and understanding those diagrams. Our mind is a superb pattern-matching engine. A standardized notation will quickly embed itself in our subconscious as patterns that we'll immediately recognize for their meaning rather than their shape. As an example, when looking at a dashed line ending in an open arrow, you'll read that as "depends on" in much the same way as you now read 3.1415 as π. Therefore, over time we'll become more proficient in reading UML diagrams, and each time we encounter a diagram we'll save the time needed to familiarize ourselves with its notation.

Our diagrams will also be more expressive, because we'll be able to use UML's (some say excessively) rich set of standardized notations. In diagrams you'll find relationships like a dependency, an aggregation, or a realization expressed using a specific line type and arrow ending, rather than having to second-guess what a generic arrow actually means. Moreover, by using predefined shapes we eliminate the legends

 0740-7459/10/$26.00 © 2010 IEEE

that often accompany ad hoc notations, freeing up space for the actual diagram. And UML's size shouldn't worry us any more than the richness of the English vocabulary. As any teenager can tell you, a few hundred words (or a small subset of UML) is adequate for most practical purposes.

Using UML will also benefit us mightily when we draw diagrams. First of all, after a short learning period, we'll all be able to concentrate on how our diagrams can best convey our ideas, rather than on inventing new notations. Also, by adopting a diagramming standard, we'll strive to improve our expressiveness and clarity through the best use of UML's notations in much the same way as we (should) continuously try to improve our use of the English language.

Furthermore, using a rich standard language like UML forces us to be precise not only in the way we express our thoughts in the diagram, but also in how we think about the problem. Sloppiness in diagramming notation allows for shoddy designs. When a randomly drawn arrow or shape can mean anything, many of us won't bother to pin down its precise meaning, and important aspects of a design will escape our attention.

Besides precision, skillful diagramming also drives the essential simplification of reality—and thereby abstraction. This enhances our expressiveness through a process that the cartoonist Scott McCloud (in his book *Understanding Comics: The Invisible Art*) terms "amplification through simplification": omitted details increase the diagram's applicability.

Moreover, the universal adoption of UML will elevate our design diagrams to a form of literary expression. In much the same way that Shakespeare wouldn't flourish in the age of cave drawings, with ad hoc notations we deprive ourselves of the power of communicating through a shared, sophisticated language. And with UML diagrams all around us—in magazines, conference presentations, and design proposals—we could read and learn from a rich set of examples. It's one thing to read a colleague's first draft of a use-case diagram, and another to learn from the UML diagrams of the Great Masters.

Weighing in

If you would like to give your opinion or read further discussion on this topic, please visit the "Tools of the Trade" blog at www.spinellis.gr/tools.

Second-Order Effects

As more people adopt UML diagrams, network effects will kick in and act as a virtuous self-reinforcing mechanism. Increasingly, more of our profession's members will be able to understand diagrams in much the same way as a building's technician can read the installation's engineering blueprints. Although we might demand from an experienced software engineer to be proficient in several modeling notations, it's unfair to expect the same from a budding developer, tester, or graphic designer. By standardizing to UML, we can easily train all software professionals and other stakeholders with an appropriate course on UML notations. This will greatly improve the way we communicate, because (and please give me some credit for avoiding this cliché until now) "a picture is worth a thousand words."

In addition, the widespread use of UML should result in what educators, psychologists, and sociologists call *internalization*: we will accept UML's notation as our own way of thinking about designs. This process will yield profound changes in the way we design software. When our mental processes deal directly in UML, we'll be able to devise and understand increasingly sophisticated designs. These days most of us would find such designs too complex to comprehend, because our mind must translate a diagram or code into a distinct, imprecise mental picture of the design.

Finally, as everybody takes up UML, we'll also be able to use the quality of a model's diagrams as a signaling mechanism for identifying good designs. Just as a spam email's spelling and grammar mistakes make you realize that the amazing business opportunity it describes is simply a con, you'll be able to quickly estimate a design's quality from the way it uses UML's graphic notation.

That Said ...

This has been a difficult column for me to write. Whenever I bring up this topic, I usually receive lukewarm or even hostile reactions.

I know that UML has many shortcomings (see, for example, http://en.wikipedia.org/wiki/Unified_Modeling_Language#Criticisms). I find it a shame that it hasn't successfully solved the problem of design tool interoperability, and I personally find some of the graphical choices made for its notation difficult to work with. However, I'm convinced that using a standard notation is always better than using an ad hoc one (which is the most common alternative). In the longer term, I'm also sure that we'll reap huge benefits from adopting a single notation instead of battling with many competing ones.

Why use UML and not your favorite (and more elegant) language that's better suited to your application domain while also being supported by an extremely powerful tool and a matching singing and dancing programming language? The answer is simple: the user base (unequivocally established through a Google search) and network effects. In the field of technology, we often see that these considerations trump all others (Betamax versus VHS is a classic example). Also, frankly, it's difficult to believe that a different line-ending shape can have an immense effect on a designer's productivity. So, for the benefit of our profession, let's put our differences aside and agree to use UML's notation, concentrating on the substance of our designs rather than their appearance.

Diomidis Spinellis is a professor in the Department of Management Science and Technology at the Athens University of Economics and Business. Currently he is serving as the Secretary General responsible for information systems at the Greek Ministry of Finance. Contact him at dds@aueb.gr.