The Poetry of Code

Abstractions (deservedly) contribute positively ....” Diomidis Spinellis, in his July/August column “The Way We Program,” is right about that. “The writing of comments is prose literature, aiming to tell the story behind the code. The formation of layout through white space is sculpture, seeking to show the code’s hidden structure.” I turned to this article the moment I received my print copy. Spinellis is on the right track, but stops well short of what we need to tell developers. Code is prose literature, but it’s not the comments that need to be readable and inspiring, nor is the white space the communicator of hidden structure. Rather, the executable code must read smoothly and understandably, with all statements in a given scope having the same level of abstraction. Meaningful identifiers are essential, but the developer-reader wants clarity rather than poetry. The executable code must communicate “plainly, succinctly, and unambiguously.” With executable code like that, comments become like footnotes, and white space like ... well, white space, in a good book. Suddenly the code is clear to every “next developer,” compiles smoothly, functions well, and is safe to update and maintain.

Tom Harris
t.harris@ieee.org
http://talkaboutquality.wordpress.com

Diomidis Spinellis responds:
You’re absolutely right in pointing out that the source code’s executable elements form its most important part. The idea behind my column was to use the amount of white space and comments in production code as a way to illustrate the importance we developers place on a system’s source code. Bad code can’t be fixed through commenting and white space, but great code can be made even better through judicious use of these elements.

A Continuum of Methods
I enjoyed Hakan Erdogmus’ perspective on software methodology in “Essentials of Software Process” (July/August 2008). I’ve tried to categorize methods myself, and I agree that it’s quite challenging.

I like your seven dimensions, but I prefer something simpler—with some information loss, obviously.

I recently argued for adopting a Unified Process (UP) in my organization. I came up with one main theme and three dimensions. I view software development methods along a single continuum from predictive to adaptive, each emphasizing differing approaches to exploring requirements, designing solutions, and managing risk. Waterfall visions proceeding through them once, linearly and sequentially. UP and agile methods envision proceeding through them iteratively, elaborating in small increments, until the work product is complete. Agile methods differ from a UP in their emphasis on evolution and adaptation.

So, we can distill three axes of differentiation:

- planned versus evolutionary—the method used to discover and explore requirements and solutions,
- fixed versus flexible—tolerance for change, and
- sequential versus cyclical—the risk management approach.

Waterfall is on the left side of these three axes, UP in the middle, and agile methods on the right.

So there’s my two cents. Keep up the great work. IEEE Software is both informative and a pleasure to read.

Mark Sawers
Software architect
mark.sawers@gmail.com