Code is often the stream-of-consciousness thinking process a programmer is using in real time to understand and solve the problem.
reinforced, by reworking the code to get it “right,” the more the code becomes de facto “right” in the programmer’s mind. To some extent when you say to a programmer, “You need to change your programming style,” you are saying, “You need to change how you think.” Most of us strongly resist others telling us that our way of thinking is wrong and we should change it to their way.

It always ends up about people understanding things ….

Phillip G. Armour
Corvus International
www.corvusintl.com

Diomidis responds: I agree with most of your points. Interestingly, in various languages, syntax is becoming less mutable. Some languages, like Python and Haskell, give meaning to whitespace. Some IDEs, like VBA, force their own formatting. There’s a fine distinction between necessary and unnecessary variability. When we remove unnecessary variability, the next question is this: if there’s only one way to write this, why should a human write it?

Regarding your explanation on resistance to style issues, my position differs. Most style bugs I encounter concern absence of style and consistency, not the following of a specific train of thought. A sane person wouldn’t defend an erratic coding style. When I point out style inconsistencies, people will reply that they’ll fix them later or that they aren’t important. They don’t defend their style choices, but try to excuse their behavior. There are of course also style zealots, who will stand behind their own strange style choices. My impression is that these are becoming a rare breed. (Or, as I’m getting older, I’m less in contact with such immature behavior.)

Technical Debt

Thank you for your IEEE Software “From the Editor” article regarding technical debt (“Perfectionists in a World of Finite Resources,” March/April 2011). It was fun to read and it was a trigger to reflect about this topic again.

Dealing with technical debt in software reminds me of dealing with the national budget—later, everything will be much better, but now, we need to add this feature and spend this money. ;-) In both cases, there’s compound interest to pay and people still don’t seem to understand the consequences. Hopefully this will change.

But there seems to be some movement in the industry.

Years ago, developers made shortcuts to implement requirements in time and nobody else cared too much. Project managers cared for their due dates, not for “technical details.” Engineering had to fight to get some time for “fooling around” with technical stuff. This was the first “buffer” in a project that was used up by other tasks as you might have expected.

Today, this looks a bit different to me.

Why? Because some years ago, we broke down the walls between engineering, quality, product management, and business management (through agile processes). This resulted in a much better mutual understanding. As a consequence, all disciplines are more sensible regarding a decision’s cost. Putting more features in the current release will lead to more effort and cost for the features in the next release. This is something that product and project managers understand.

And that’s the reason why I don’t like the term technical debt at all; it implies a technical issue but it’s really a product and project issue. Technical debt is a risk for the product, and in the long run, for the company! This makes technical debt visible and vital for management.

Understanding this, I don’t have to fight with management to get time to fix technical debt. I just need to present the cost of not doing it. Then we talk about an economical topic that management understands much better.

Stefan Braun

Forrest responds: Thanks for your thoughtful comments. Any metaphor illuminates certain aspects of an issue but obscures others. It sounds like we agree that the “debt” part of the metaphor is useful in that it highlights the ongoing costs of not paying attention to some issue. In previous discussions, I’ve found the “technical” aspect useful in that it highlights that many of the issues that we’re most interested in have technical causes within the system design or code, as well as technical fixes. But I can understand that other phrasings work better in other environments. And yes, you are absolutely right in that getting the time and resources to retire such debt is a management issue! ☻