Editor: **Grady Booch**
IBM, grady@
computingthehumanexperience.com

In Defense of Boring

Grady Booch

THE PURPOSE of good software is to make the complex appear simple.

Complexity is the key factor in the cost of software and the time it takes to develop and evolve it. If you reduce Barry Boehm's software economics model down to its essence, you'll see that this cost/time is a function of the complexity of the system, raised to the power of process times the team, times the tools (and weighted in that order, from the most to the least significant). Having the right tools helps—having the right team helps far more—but whatever you can do to control complexity has the most significant impact on a system's development life cycle. Furthermore, a good process will dampen complexity, while a bad process will amplify it.

There's also a subtle yet important interaction between an organization's process and its team. The very best teams will embody an emergent process that's perfectly tuned to its culture, its domain, and its history. This is the nature of all highly effective teams, wherein the process becomes a part of the atmosphere. However, to paraphrase Garrison Keillor's description of Lake Woebegone, every organization likes to believe that theirs is a place where all their developers are strong, all their code is good looking, and all their system metrics are above average. Nonetheless, on average, the average developer is average. This means that this process/team relationship is far more complex: the stronger the team (and the greater the risk to or value of the system), the less that a high ceremony process is needed. The better the team (and the less risk or value present), there can and there must be a reduction in ceremony.

As it turns out, all of this is very hard to do.

The Dynamics of Complexity

First, as Fred Brooks has told us time and again (and as we need to be reminded time and again), there's an essential complexity to software, a complexity that's inescapable and irreducible. Build yourself a natural language question/answer system, manage the textual and visual brain droppings of about a billion users, craft a vehicle that can semiautonomously explore an alien planet: these are all things that multiple people spend multiple careers trying to get right.

> On average, the average developer is average.

Second, however, there is self-imposed, accidental complexity. Stick your head inside the workings of any 1 to 10 million SLOC system, and most of the time you will see a muddle. Yes, there will be obvious lines of demarcation, faults where you can observe the impact of some major technical or business tectonic shift, and reoccurring fossils in the software's geological levels, laid down by different individuals with different styles over different times. The most significant design decisions are probably visible, evident in the major edifices and reflected in the dark corners of the system. Nonetheless, I've yet to see any ultra-large software-intensive system without some vestigial organs and strange irregularities. This is the very nature of how large systems evolve, be they natural, organic, or human-made.

To that end, the best we can do is simply strive to manage complexity. We can neither reduce nor eliminate a system's intrinsic complexity. From a system's engineering perspective, this is where we apply all the tricks of our trade to devise crisp abstractions, a good separation of concerns, and a balanced distribution of responsibilities. A discipline of steady incremental and iterative executable releases helps to steer a project in the right direction, which is not necessarily the direction first envisioned. A discipline of patterns serves to establish the system's texture and attends to crosscutting concerns. A discipline of refactoring is hence the result of combining the best practices of a rhythm of releases with the motifs of textures. Refactoring helps to take off the sharp, unnecessary edges of a brittle system. When done right, the result is positively, beautifully, breathtakingly boring. As it should be.

0740-7459/13/$31.00 © 2013 IEEE

Smooth Edges

These concepts apply not only to the inside of a software-intensive system but also to its outside. When used as a part of a system of systems, the edges of any subsystem must play well with others, especially with others that didn't even exist at the time you built your system. If a subsystem offers up APIs or services that are awkward to use, too fine-grained, too big, or just plain irregular, then you have a problem. That's not boring, because you'll find you have to force a fit by writing some one-off code that hides the evils of the existing interface, bridges the gaps, and sometimes routes around it, either by jumping across levels of abstraction or replacing some functionality entirely. When the edges of a subsystem are well designed, they are approachable and understandable, they snap together easily with other edges, and their behavior is predictable. Hence, they are boring.

On one hand, we seek to build software-intensive systems that are innovative, elegant, and supremely useful. On the other, computing technology as a thing unto itself is not the place of enduring value, and therefore as computing fills the spaces of our world, it becomes boring. And, that's a very good and desirable thing.

This is the perspective of boredom as seen from inside a software-intensive system looking out. Looking at such a system from the outside in is an entirely different matter. Let us then look at software through the lens of the human experience.

Technological Babysitting

Recently, I was in Silicon Valley, where I did a little shopping. I'm a people-watcher, and a charming young boy, perhaps three or four years old, caught my eye. He was with his father, and the two were apparently waiting for the boy's mother, who was trying on clothes. Time and again, the young boy tried to engage his father's attention, to no avail. Completely frustrated with the interruption of wherever the father's thoughts were taking him, the dad whipped out a smartphone, put on a movie, and shoved it under his son's face. The father continued looking out into space, while the child, slack-jawed, focused on the movie, his face bathed in the usual smartphone glow (a phenomenon I call receiving an iTan).

In the father's defense, he might have been having a Really Bad Day, but I don't think so. Rather, the father was medicating his son with an iPhone. In so doing, using Sherry Turkle's terminology, the father and son could now be alone together. This is a scene I see play out all the time. I'm no longer surprised when, walking along the beach, I see a whale breaching, only to look back at the shore and see a family, heads down in their smart devices, oblivious to the world beyond their screens. I suppose, using a title from the Grant Naylor book, they found their computing experience to be *Better Than Life*.

I am an expert in computing, not in children (although my wife is, as a child and family therapist who was in private practice), and I have no children of my own (although we have been godparents to about a dozen kids and have also brought a single mother and her child into our household for a few years). That said, I recognize when technology is being used as a substitute for reality, and what I was witnessing was one such case. From my perspective, a child needs time to dream, and while tablets and such are useful in moderation, they are never a substitute for human interaction, especially when one is learning how to grow up.

Turkle's *Alone Together* and Carr's *The Shallows* offer some evidence of the effect that technology has upon us when we immerse ourselves inside it, at the expense of being fully present in the world. There is work to be done to deeply, scientifically understand the implications of computing, but nonetheless…look! Squirrel!!!

Sorry, I was distracted there for a moment.

But that's the point. We don't yet know fully the implications of intimate computing on the individual, nor likely will we for a generation or so. While I'm confident that the human spirit will adapt, I'm also certain that all of us—especially children—need some boredom in our life. The intentional use of computing is a good thing, even if that means intentionally not using that technology from time to time, as a sort of digital sabbatical.

As such, we need more boring software, software that's so fundamentally boring that it disappears. If you must have a tablet in a child's face, then devise a killer app that would engage the child and the people in the immediate vicinity in such a way that they're required to interact with one another. Perhaps this might be an augmented reality app for a child's game of I Spy, or counting or spelling games that are contextual to the world around the child. You know, stuff that is part of the boring real world.

Now that's the kind of boring software we need much more of.

GRADY BOOCH is an IBM Fellow and one of the UML's original authors. He's currently developing *Computing: The Human Experience*, a major transmedia project for public broadcast. Contact him at grady@computingthehumanexperience.com.

See www.computer.org/software-multimedia for multimedia content related to this article.