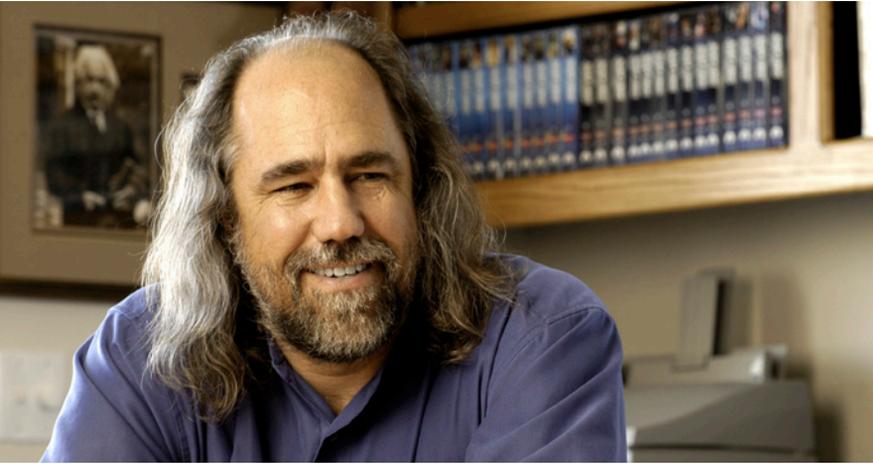# Interview with Grady Booch

CrossTalk: Who are your most significant influences both inside and outside of software?

Well, I've got quite a few, so I'm sort of going to give this to you in a historical perspective.

U.S. Navy Rear Admiral Grace Murray Hopper, a fascinating woman I met some years ago, I still have my nanoseconds from her[1]—and readers who know about that will smile gently. In her lectures, she had this wonderful visual cue she used when telling people about the amazing things in regards to the shrinking of machines. She would have, prior to the lecture, taken telephone wire and cut it into little 11-inch segments and passed it out to people, saying, "Here's a nanosecond." It actually represented the distance that light would travel within a nanosecond—an amazing visualization. I was always touched by her grace, her ability to speak power to truth, and her ability to integrate the technical with the social.

Fred Brooks, of course, has been a tremendous influence to me. Ed Yourdon and Tom DeMarco on the process side, and more recently, folks like Kent Beck, Ward Cunningham and Scott Ambler. I also add to this list Mary Shaw at Carnegie Mellon—she has taught me a great deal about architecture, Philippe Kruchten did as well. The late Randy Pausch;[2] I had the delightful opportunity to meet with him briefly at a conference while he was presenting Alice.[3] And George Walther, who was my instructor at the Air Force Academy. He was the first person who really introduced me to the notion of discovering beauty in software.

Of course Jim and Ivar were tremendous influences on me—we could not have produced the UML without the collaboration from all three of us. We are three radically different personalities and, as I have said publicly, it's amazing that we accomplished what we did without felonies being committed along the way. But I'm delighted that we did, and I honor and respect them and dearly love the time I spent with them. It was a high point in my career.

Outside the software world, Richard Feynman is my absolute hero. His ability to just be a renaissance man, his interest in so many wide-ranging fields, his desire to follow his bliss … he is a role model for me.

CrossTalk: You have written numerous books in your career. Which one do you believe has had the most influence on the DoD software community?

Which one do I think has the most effect? The next one I'm writing. [LAUGHTER]

The next one is titled *The Handbook of Software Architecture*.[4] Again, it goes in my theme of architecture as an artifact and the important role I believe in delivering complex systems. My goal here is to basically document the architecture of 100 interesting systems and describe them. My intent is to capture what we find to be the best practices in architectural patterns that are out there. This is an effort that has been going on for seven years—and I hope I will finish it within the next seven years. It's hard research, but I'm learning a lot of things, discovering things, and inventing things along the way.

As for past work, probably *Object-Oriented Analysis and Design with Applications*[5] was the most significant one because it sort of helped start the effort of unification of the work Jim, Ivar, and I did. It was influential in terms of notation as well as process, and, frankly, in making object-oriented design a household name.

CrossTalk: What is the current impression of the future of the UML—a language you helped create?

Well this is a very timely question because we [at IBM] recently submitted to Object Management Group (OMG) a response to their request for information about the next generation of the UML.[6] I'll begin by saying where I think the UML is, and where the trajectory is going.

First, as one of the original officers of the UML, I am flattered, amazed, stunned, and staggered at the reach the UML has had. It has shown up in places I never, ever anticipated when Jim [Rumbaugh], Ivar [Jacobson], and I began the journey unifying our methods.

What delights me and absolutely tickles me is the realization that the goals of the UML are still very valid today, and UML 2.0 continues to help deliver in that regard. I see the UML being used in places far beyond whatever I anticipated and that is very exciting and very humbling.

Yet that being said, part of our recommendation back to the OMG and the biggest thing I pushed is for a return to the fundamental roots of the UML, which is really two-fold.

First, UML 2.0 is more complex than it needs to be, and I would like to see the UML become simplified over time. And that's not a means of throwing things out and not being backward compatible, it's just a matter of refactoring the language so that there is a common underlining core.

The other thing I would really like to see is return to the roots of the language not being a visual programming language, which has fueled a lot of the model-driven development work. In some domains, it's quite appropriate … but it's a modeling

language … I would like to increase the use in the semantics of the UML relative to things like reverse engineering and mining and reasoning about things as they unfold over time.

One other thing—in terms of where the UML is headed—is that I was blown away recently when I discovered an article called "The Systems Biology Graphical Notation."[7] Apparently it was inspired by the UML as an attempt to build a standard for biologists for modeling things within their world—things like mechanisms within cells and the like. So that's an example of where the UML has extended its reach far beyond whatever I imagined. That's pretty cool, and it also tells me that the language does have staying power; it's going to be around here for a long, long time. We do need to simplify and refactor it.

CrossTalk: If you were in charge of DoD's weapon systems software and infrastructure IT systems, what would be your top initiatives?

It really used to be, decades ago, that the DoD was leading the marketplace in the delivery of software-intensive systems. The harsh reality is that the commercial sector is leading best practices and really pushing the arc relative to software engineering and software development. So, in that regard, the DoD is behind the times. That is not to say that they are not pushing the limits in some areas. The kind of complexity we see in certain weapons systems far exceeds anything one would see commercially, but ultimately, there are a lot of things that the DoD can learn from the commercial world. As I look across the spectrum of systems that are successful and try to find the anti-patterns from those that are unsuccessful, there are three that come to mind and appear to have relevance for success—not necessarily in any order.

There's the leveraging of open-source principles. I know that the DoD has Forge.mil, which is evolving those many ideas of SourceForge, and I very much encourage that notion because there's this opportunity for transparency, visibility of software intensive systems—it has certainly added value in the commercial space. So I would certainly encourage and intensify the use of those open-source platforms.

The next initiative I would bring about would be the collaboration infrastructures. The reality is that the DoD builds software-intensive systems with contractors who are spread across the globe, potentially—and certainly the deployment of these systems is across the globe as well. I'm not sure that the DoD has invested enough. And it's not just the classic Web 2.0 kinds of things like wikis and shared whiteboards and the like. I would also do some exploration in virtual worlds, the kinds of things IBM and myself are trying to push in that space.

The third thing—and I've had some strong initiatives in this— is the whole area of architecture. What drives me to this conclusion is that as I look at the main complaints and pains that virtually every organization has in delivering software-intensive systems, there appears to be a common thread between the architecture and the artifact. So I would go beyond DoDAF [Department of Defense Architecture Framework]. I really like the standard. I think it's effective for what it's intended to be for—really trying to model the enterprise of the warfighter— but, in my personal opinion, I am less confident that it's appropriate for the architecture of the software-intensive systems. So I would certainly begin some initiatives to push for the notion of architecture as an artifact in terms of its representation and its governance of the social organizations around it.

CrossTalk: What is the next big approach to creating software-based systems that is going to make a significant difference?

In terms of the next big approach, I believe it is growth in our understanding of systems engineering.

Traditionally you begin the design saying, "I've got these pieces and let's throw in a processor here and there, and then you software guys go off and do your thing." The problem is you can't, from a systems engineering perspective, treat software as something you can put aside. Rather, it is an intrinsic, essential, universal piece of the system. So I think the biggest change we will see—or the biggest need—is the move toward a recognition that systems engineering needs to incorporate more and more of the practices we know into pure software systems because, in the warfighter's case, these are hardware/software systems—and that means we have to approach them differently than we have in the past.

So how does that manifest itself in terms of actionable things? The real news is that there is work to be done. INCOSE's beginning to embrace these ideas in the emergences of languages like SysML [the Systems Modeling Language] is helping us move along in that direction. But we don't know all the answers, and we're on a journey along the way—that's why I say it's the next big thing we'll have to worry about.

CrossTalk: What new advances and changes in languages and software engineering are on the horizon?

The following is, again, my personal opinion—not that of anybody living or dead or yet to be born, and I say this because it is a controversial one. I've said it publicly and usually I get lots of nasty e-mails after I say it, but my observation is that on the language side we're really at a plateau.

While I tracked what was happening in the language research space, I was really excited about what was going on in aspect-oriented programming. But that seems to have died out, in the sense that people were still dealing with problems in the weeds and it really hadn't risen up to the level of aspects at a higher level of abstraction. So on the language side, I think we're going to see a continuation in most of our existing languages. Look at C++ and you'll see that they have fixed a number of things in the current standards and they've really tried to extend it in some other areas as well—and these are largely incremental, albeit, important changes.

Where I think the biggest changes will happen will be back in the software engineering side. But before I attend to that, let's talk about what pushes us in that direction: What are the forces that cause that change?

There's the presence of legacy and how one addresses that. We have a crushing burden of legacy upon us—and not to put this in a negative light—but the reality is there is a significant capital investment in legacy and that leads us to not

throw these things away, but rather trying to figure out how to interoperate with them. SOA [service-oriented architecture] certainly plays an important role helping them interoperate, but then again legacy—the forces around that—are one issue. Another thing that is pushing up is presence of multi-core. The frequency-scaling wars are over, so we can only begin to boost computational resources by boosting frequencies certain ways. If you have an obviously decomposable parallel problem, multi-core usually fits, but you have a less than obvious decomposition. It's really nasty and hard. Another force I think we're driving up is the whole problem with security—and then the biggest one, perhaps the most dominant to impact us, is the issue of complexity. We are building systems of crushing complexity, so we need some help in that regard.

Those things together I think are pushing us. With regards to what is happening in the software engineering side, the good news is I think we have a good picture for how high-ceremony processes and agile processes work well together. So there's a lot of good information coming out of that world. And, although I think it may be self-serving, I see short-term growth towards the practices around architecture as an artifact, and then the next thing on the horizon is less-so software engineering and more-so systems engineering.

CrossTalk: What are the restraining parameters that hold software engineering back from more breakthroughs?

First off, I don't really believe in breakthroughs. The reality of the progress of science, especially in software, is that changes come from the confluence of many things, where you might reach a tipping point that changes things. But I'm more of one for evolution than revolution. Frankly, I even consider object-oriented design to be an evolutionary thing as opposed to a revolutionary thing.

I think I've actually talked about the true restraining factors already: legacy, inoperability, multi-core security. And the last is complexity. We are dealing with systems that far exceed the intellectual capacity of any single human. We generally lack the notations, processes and measurements to help us deal with that complexity—so that's what holds us back. It's a wickedly hard problem.

CrossTalk: What are we to do with all the DoD systems that were implemented in older object-oriented languages like Ada, Modula, etc., as there are less and less engineers skilled to enhance and maintain the code?

There are older object-oriented languages being used. In fact, I'm engaged in a project that is still using Ada, and I'm excited that they are because it is really well-proven language.

It is a problem, but not one the DoD has alone. I had been working on a project with the IRS … a system that is central to their tax processing has about 500,000 lines of assembly language, a lot of which was written in the '60s and it still exists there. I see systems written in COBOL; I've seen systems written in PL/I. So this is a systemic problem that goes to the heart of the issue of legacy of older systems that I mentioned.

I've actually written and discussed this very topic; what I call the "Nine Things You Can Do with Old Software."[8] One thing you can do is harvesting—which is basically taking these older things and doing the reverse engineering of pieces of them to extract the algorithms, the data structures, things like that, and then rewriting them—but that is so very hard. Another—the most effective thing that I have seen—is the notion of continuous architectural transformation. It requires considerable process discipline and it goes back to the heart of architecture and artifacts. Only a few organizations that I've seen have been really successful. I hold up eBay as a classic example.

CrossTalk: That's interesting. I didn't know there was that much old language out there still being utilized.

Oh yeah, there are gobs of languages. I did a quick calculation asking how many lines of codes do we produce in the world on a yearly basis? It was a low number, but if you make an estimate for the number of software professionals, the number of people that actually code, the number of lines of code per average per year, you end up with around 33 billion lines of code new or modified or produced every year—and I will be honest in saying that's conservative and it's probably off by an order of magnitude. So if you integrate that over the years, it means, at the very least, that we probably have over a trillion lines of code out there—and much of that is still running in these old systems. So the presence of these legacy systems is a reality—and it's not just a problem the DoD has.

CrossTalk: What major changes would you like to see in the DoD to forward software engineering success?

I think the major change is in education. I don't mean to be critical, but in many ways the DoD's expertise has, frankly, been outsourced to its contractors. It is not to say that is a horrible, terrible thing, but a lot of the things that happened in old warfighting systems came through intrinsic expertise inside the DoD. I would strongly encourage the increase of education of the DoD's intrinsic forces with regards to decision engineering and software engineering—and draw back into the DoD more of that intellectual property. Ultimately, delivering for the warfighters is what the DoD is all about, and that requires an intensely educated staff to make that happen. How does one make that manifest? I think there is work to be done in acquisition policy, in processes for delivery in the use of things like DoDAF. I think the DoD itself can lead and should lead this, and it needs to make this change in the interspatial spaces of its training, in its service academies, and in its colleges as well.

CrossTalk: Have you seen anything to suggest that the DoD has gathered that same point of view and that it might be starting to change its perspective and train people differently?

Walt J. Okon and I recently had a conversation on that very topic. I did raise with him the notion of education. I was absolutely ready to dance on the table when he told me that one of his major initiatives, beyond 2.0 reaching closure, is that whole issue of education. Beyond what he is doing, I don't have a lot of insight, but I am certainly encouraged by his efforts.

CrossTalk: Back to the idea of needed training prior to getting ensconced in the industry: How do you see the current state of software engineering in higher education, and where do you think it needs to go?

I've had the delightful opportunity to engage with a lot of different schools. I make a yearly jaunt around the universities—both in the U.S. and other places in the world—to give lectures and the like. I've also had the chance to interact with people both in the ACM [Association for Computing Machinery] and in the IEEE on K-12 and undergraduate and graduate degrees.

What is growing are the interdisciplinary kinds of things like I've seen at CMU, and at USC through Barry Boehm, where systems engineering is coming together and software is an important piece of that.

There is this mental model I use that I speak of as "the laws of software." So if you imagine that we have a surplus of cognitive resources—in other words, human intelligence or human imagination is not a limited resource—we come up with these visions and we have to turn that into "raw running naked" code. The question for me is what separates us from vision, to turn that into raw running naked code—and the answer is there are these things in the laws of software.

You'll see that things move from the computer science-y things, which are very mathematically based and very fundamental, into the things that become more human-oriented—elements like politics and ethics and moral issues. We think we know how to build certain things … the question is should we?

What makes it most difficult to move from vision to execution is something that swirls around the problems of design and the problems of organization. How do I best architect a system? How do I best architect my organization to deliver that system? As it turns out, there's this wonderful, delicious cusp of the technical and the social, and that's where the sweet spot for delivery is in education. How does one attend to the fiercely technical problems, but at the same time be cognitive of the social issues as well? I swear there are days that I go into an organization where I'll show up as über geek and other days I have to show up as Dr. Phil, slapping faces around, saying, "My God, what are you thinking?" So, in terms of where I think things need to go—well, for people delivering software-intensive systems, I think our education system has to attend to that dance between the technical and the social.

> So this is still an exciting place to be. I encourage people who are thinking about this field to recognize that there are a lot of wickedly entertaining, exciting and delicious problems to solve. We're not done yet.

CrossTalk: I was recently in a conversation where we were trying to set up a degree program with a local university for UAS [unmanned aerial systems] and the big argument was hardcore engineers versus interdisciplinary people. I take it you're leaning toward interdisciplinary as a strength?

Well, I say it is very much a strength because if you look at unmanned vehicles, this is a classic systems-engineering problem. There are some wickedly technical problems to overcome, but ultimately I'm delivering a system to be used by humans, to be used in the context of other complex warfighting systems. These are not islands, so I would want to seek out the best ideas from a variety of places. So yes, I can't imagine one considering this other than interdisciplinary activity.

Through the mixtures of putting smart people together in different domains, innovation comes about in unexpected ways.

The final thing I'd offer is, you know, that this is still an exciting discipline. The global economy is in a funk, there's no doubt about it. I've been lecturing recently about the notion of software abundance in the space of economic scarcity, and I'm utterly convinced that the delivery of software-intensive systems is still a major source of innovation and, therefore, economic growth. So this is still an exciting place to be. I encourage people who are thinking about this field to recognize that there are a lot of wickedly entertaining, exciting and delicious problems to solve. We're not done yet.❖

NOTES

1. To learn more about Rear Admiral Hopper (1906-1992)–including her famed nanoseconds–visit <**www.chips.navy.mil/links/grace_hopper/womn.htm**>.
2. Pausch may be known best for his Last Lecture: "Really Achieving Your Childhood Dreams."
3. Alice is a 3-D programming environment.
4. Currently, Booch maintains a blog, <**www.handbookofsoftwarearchitecture.com**>, for The Handbook of Software Architecture, which serves as the repository for ongoing work in an effort that will eventually be published in print.
5. First published in 1991, Booch's book is in its third edition (2007).
6. See < **www.uml.org**> to learn more about UML, the current status of UML 2.0, and the role of the OMG.
7. By Nicolas Le Novère, et al., in the 7 Aug. 2009 edition of *Nature Biotechnology*. The article is available at <**www.nature.com/nbt/journal/v27/n8/full/nbt.1558.html#a1**>.
8. See the Sept./Oct. 2008 edition of *IEEE Software* or listen to the podcast, "Nine Things You Can Do With Old Software," at <**www.computer.org/portal/web/computingnow/onarchitecture**>.