The Interplay of Art and Science in Software

Terry Bollinger, The MITRE Corporation

Although history and science have provided lots of important bits and pieces of information about the world around us, we are still pretty short on getting a good overall understanding of what's going on. Take, for example, the most profoundly accurate and predictive physical theory of all time, a collection of concepts and equations that goes by the unwieldy moniker of Quantum Electrodynamics, or QED. Ignoring for the moment the intimidating name, QED is really just an astonishingly compact and simple set of rules for how electrons (the visible parts of atoms) and photons (particles of light) interact.

ULTIMATE PREDICTION THEORY?

Developed by Richard Feynman in the late 1950s, QED is so conceptually complete and accurate in its ability to predict atomic behavior that no one has since been able to show any deviation between its predictions and the actual behaviors of atoms and light. And since atoms and light make up nearly every significant aspect of the world around us (with the exceptions of gravity and nuclear energy), QED in principle could predict just about any sort of event. From events as small as the interactions of two atoms through events as unimaginably complex as populations electing new leaders, all are in principle directly understandable and predictable using QED.

Guest Critic: Terry Bollinger, The MITRE Corporation, 1820 Dolley Madison Blvd., McLean, VA 22102-3481; terrybol@erols.com.

Humans have a funny knack for figuring out how to do something when there is no perfect answer.

The catch

The catch, of course, is the phrase "in principle." In actual practice, QED not only can't show you the forest for the trees, it can't show you the trees for the leaves—or the leaves for the aphids, for that matter. To put it simply, QED is computationally bound in a fashion that can never be fully resolved. Even with the latest algorithms and hardware strategies, the calculations required to fully model small sets of molecules using QED would be too complex. Besides, other less precise but easier-to-calculate methods exist that accomplish much the same thing.

QED modeling is also hampered by a very serious information acquisition problem. In short, a full QED model can be populated only by literally taking an object apart atom by atom—in other words, by obliterating it. Besides being physically impractical, such a procedure would also significantly decrease the chances of getting people to volunteer for QED modeling.

Scientific workaround

QED is by no means alone in the problem of scale-up. In fact, science is not so much a set of absolutely fundamental rules as it is a collection of useful approximations that suffice only for limited regions of the real world.

Thus while QED is useful at the level of individual electrons and photons, it generally gives way to the simplified equations of quantum mechanics for modeling objects the size of atoms and molecules. Further simplifications occur when describing the motions of atoms in a gas, in which the striking complexity of a single atom is represented as nothing more than a billiard ball. Still further up the scale of simplification and approximation is fluid mechanics.

By the time all this reaches the level of a woman paddling a kayak in a spring whitewater race, the meaningful abstractions are no longer isolated to the fluid itself. Instead, they become complex collages of the kayak's motion and angle, the woman's muscle strength and endurance, and the subtle visual clues that the water provides for spotting hidden rocks and currents.

NAVIGATING THE UNKNOWN

It is here that the remarkable nature of human intelligence truly exhibits itself. As does the kayak racer, humans have an uncanny knack for figuring out how to do something in those many situations where there is no perfect answer. It is at this boundary that science truly does merge with art, and progress is possible

Continued on page 125

Continued from page 128

only when they work together.

Rigorous science provides a framework and a solid basis for further analysis. It prevents you from following a hunch into untestable speculation or, worse, into superstitious reliance on factors that are provably irrelevant or that even oppose the hypothesis.

The artistic part of this process lets science move unexpectedly into new currents and previously unmapped understanding. What is most striking about the biographies of truly great physicists such as Albert Einstein and Richard Feynman is how consistently their greatest theories stemmed from pursuing seemingly minor or even irrelevant issues.

Einstein developed his gravitational theories primarily because of his intransigent fretting about the impossibility of a motionless electromagnetic wave, a topic that most physics advisers of his day would have suggested was an irrelevant problem of insignificant consequence. Yet he pursued this odd little side current until it led him to a vast river that completely unraveled conventional concepts of space and time.

Similarly, Feynman spoke in one of his autobiographies of how QED originated in part from his wondering about the behavior of a spinning lunch plate in the cafeteria—hardly the sort of topic that you would expect to revolutionize quantum physics!

SOFTWARE AND THE INTERPLAY OF ART AND REASON

What does this history of physics have to do with software? Allow me to present a peculiar and (I hope) controversial thought: The creation of genuinely new software has far more in common with developing a new theory of physics than it does with producing cars or watches on an assembly line. As a corollary, the goal of a software process improvement should not simply be to "reduce errors" or "increase predictability." Instead, process improvement should seek to make a group of developers collectively smarter than any one of its members.

To understand this proposition, you must be able to view software development, like physics, as beginning with a set of givens or preexisting results. In physics, these givens are the outcomes of earlier experiments and the implications of validated theories, which together form a basis for constructing new, more powerful theories. In software, the givens are the functional capabilities provided by earlier development efforts and the rules of combination and construction implied by mathematics and computation theory. For both physicists and software developers, the availability of previous results provides them with vastly more powerful starting points.

However, there is something more subtle going on here than just providing clever people with more and more powerful tools. The most powerful results of physics and software are the ones whose elegance and simplicity make them into what might best be called *tools of the mind*. By this I simply mean concepts whose clarity of definition and power of application make it easier to discern and explore new ideas that might otherwise be too remote or too deeply buried in complexity to be recognized. The virtual particles of QED and the virtual machines of software are both relatively simple concepts when defined in terms of their underlying premises. Yet neither of them can be easily understood or arrived at without first possessing the tools of the mind provided by earlier work in the same subjects.

An important corollary to the idea of tools of the mind is that reinventing such a tool is decidedly less useful than using it to create new tools and results. For example, a modern physicist would not be much impressed by a student who "discovered" gravity a few hundred years after Newton. Similarly, software developers should not be much impressed by the practitioners of software development who reinvent the same basic software capabilities over and over again.

This is a different way to look at software development, one that emphasizes what I believe is a more effective understanding of what is meant by software reuse. To a software developer who operates in the same way as a physicist, the goal is to collect the most powerful and carefully generalized set of software "theorems" (modules or subsystems) possible for use in constructing still more powerful theorems. To be effective, these collected tools must provide not just functionality, but the simplicity of purpose and power that makes them into powerful tools of the mind.

Such a suite of software components must support multiple levels of working detail, just as physical theories allow analysis at multiple levels: General abstractions give way to more detailed analytical needs. As with the kayak racer, it is this ability of both the physicist and the software developer to ride the wave of earlier thought and enter the domain where science, art, and insight interplay effectively.

The creation of genuinely new software has far more in common with developing a new theory of physics than it does with producing cars or watches on an assembly line.

PROCESS IMPROVEMENT AND INTELLIGENCE

From this it stands to reason that process improvement (that is, the process of helping projects produce good software faster and more efficiently) is more than simply adding good measurement and controls to a process. To be truly successful, process improvement must address the much deeper and uniquely difficult issue of how to distribute creative, intelligent problem solving across a group of heterogeneous individuals and computers.

Stated another way, major process improvement will require increasing the *group IQ* of a collection of people and machines. That way, the group as a whole is faster and more effective than any one of its parts at solving new and difficult development problems.

THEORIES OF IMPROVEMENT

But why is this necessary? Why can't the simple process improvements for replicated, factory-style products work for software, too? Is the group IQ idea

really that important to software process improvement?

We can find the answers in the theoretical background of software process improvement—or, to be more precise, in the *lack* of such a theoretical basis. Too often, the comparison of software development to factory production has been based more on hopes, dreams, and funding strategies than on any profound insight into how software is developed.

Explicitly recognizing the group IQ concept helps avoid "process stupidification," in which a group becomes more stupid as a whole than its individual members.

For example, even a cursory glance at the automobile industry leads you to suspect that the part of that process that corresponds to software development is the design of new car models, not the replication of cars in a factory. Designing a new car model is a complex, risky process. It requires balancing issues such as previous designs, market expectations, new technologies, and artistic factors in ways that can be greatly assisted by computer technology. Nonetheless, this process very much depends on the availability of experienced problem solvers.

Ironically, many people frequently compare software development to the automobile's factory phase, a point in time long after designers have resolved the tricky design issues.

Another way of understanding the importance of group IQ to software development is to recognize that software is a uniquely nonphysical entity, a creation that is as close to being a pure capture of complex intellectual reasoning as humankind has ever developed. No matter what its physical or machine representation, software ultimately captures a vast range of solutions to intellectual problems ranging from how to make a computer perform a very simple operation to abstract reasoning about some of the most profound problems. The quandary of how to add new software solutions into that universe of old and new solutions is itself a strikingly difficult problem. It's not a problem solvable by techniques based on making sure one bolt looks exactly like another. Software development must be both creative and constrained, but the constraints need to provide structure and direction rather than prevent a software solution from traveling the waves of new ideas and innovations.

BUILDING AN APPROACH

We can build "smarter" processes in several ways.

Make group IQ improvement a major goal

Explicitly recognizing the group IQ concept helps avoid "process stupidification," in which a group becomes more stupid as a whole than its individual members. Process stupidification is a particularly strong risk when the metrics for process improvement have been structured around replication. Such metrics can drive the group to perform the same comfortable programming tasks repeatedly to increase productivity and quality metrics. In contrast, the key indicator of increasing group IQ is a group's ability to solve problems that no single person would have been able to solve alone.

Avoid using replication metrics to judge design efficiency

Replication metrics are aimed at ensuring that every piece produced on an assembly line is within some level of conformance to an existing model. The danger in applying such metrics to design-intensive activities is that it can produce behaviors that are directly contrary to the overall efficiency of the resulting products. For example, metrics that focus too tightly on production of lines of code per staff-day can lead to inadvertent or intentional code "sandbagging"—the addition of inefficient or essentially pointless lines of code. Instead, design metrics should focus more on looking for the unknown and potentially deadly risks associated with any completely new design and should encourage looking for the unexpected.

COMPUTER
Innovative technology for computer professionals

Circulation: *Computer* (ISSN 0018-9162) is published monthly by the IEEE Computer Society. IEEE Headquarters, 345 East 47th St., New York, NY 10017-2394; IEEE Computer Society Publications Office, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1314; voice (714) 821-8380; fax (714) 821-4010; IEEE Computer Society Headquarters,1730 Massachusetts Ave. NW, Washington, DC 20036-1903. Annual subscription included in society member dues. Nonmember subscription rate available upon request. Single-copy prices: members $10.00; nonmembers $20.00. This magazine is also available in microfiche form.

Postmaster: Send undelivered copies and address changes to *Computer*, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08855. Periodicals Postage Paid at New York, New York, and at additional mailing offices. Canadian GST #125634188. Canada Post International Publications Mail Product (Canadian Distribution) Sales Agreement Number 0487910. Printed in USA.

Editorial: Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *Computer* does not necessarily constitute endorsement by the IEEE or the Computer Society. All submissions are subject to editing for style, clarity, and space.

Look at the Skunkworks model

The highly classified Lockheed "Skunkworks" project has been responsible for some of the most innovative aircraft designs. It also provides one of the most intriguing models for increasing the efficiency of a design-intensive project.

Published information on the Skunkworks indicates that they use small groups of experienced engineers in a rapid prototyping mode that helps them eliminate dead-end paths early, before such paths become costly to change. Bureaucracy is kept to a minimum, and communications between technical peers is excellent. The Skunkworks seems to work in part because it encourages the innate ability of people to communicate well in small groups, rather than attempting to regiment such communications via an artificial (and often out-of-touch) bureaucracy.

Use people for problem solving

It is astonishing how often methods intended to "improve" a software process actually wind up trying to make people into computers. Since computers have the perfect memories and huge information channels, they need to store most of a group's "memory" and "communications." People don't remember things very well and are great at botching detailed communications, but they are also the only ones who actually solve new problems. The structure of an intelligent group must recognize these complementary skills and make use of both sets. It should avoid having people do repetitive or boring tasks that could be automated.

If your design process is repeatedly performing the same tasks, take a closer look. You are probably falling into the trap of people doing things that the computer should be doing, either because they are comfortable or because no one is stepping back to look for better ways to do the same task.

Build and maintain a suite of starting-point technologies

Developers should view good tools and components as the starting points for development and the keys to rapid development of new tools. This is especially true since the Internet explosion, which has made it possible to find components and evaluation information faster and more easily than was ever possible before.

Software is a discipline full of ironies. No other field of science or engineering has successfully produced machines—"soft" though they may be—of such incredible complexity or sheer utility as the software community. And yet we are also rightly accused of sloppy work, sloppy thinking, and risky development practices. How is this paradox of software productivity possible? How can a community noted for its immature development methods be the very same one that has successfully produced the largest and most complex working machines in human history?

The answer surely lies in part in our unwillingness to recognize the absolutely critical role of the software genius, the woman or man able and willing to navigate the hidden paths and channels at the boundary between software science and art. While physics has always proclaimed its geniuses (although usually belatedly), we in software seem curiously ashamed of ours. After all, geniuses cannot be replicated, and what cannot be replicated cannot easily be scheduled and priced. For a field whose bread and butter is commercial and government contracts, such an admission of dependency on unique individuality and skill is not easy.

And yet we must admit it, even if only to ourselves. If there is any one lesson from the development of powerful physical theories such as QED, it is that real progress begins with the unrelenting pursuit of the annoying paradoxes and dangling threads that ruin the status quo's otherwise tidy appearance. ❖

Terry Bollinger is a principal information systems engineer at The MITRE Corporation. Contact him at (703) 883-5638; terrybol@erols.com.