THE THEME OF this special issue is software sustainability—the capability of software systems to endure different types of change over their lifetimes. In our special features, you’ll find a discussion of technical solutions aimed at supporting software changes via robust architectures. While the technical side is undoubtedly important, discussions of software sustainability often get tangled up in several thorny business issues, especially in the government acquisition/systems engineering domain, which is where I most often work. Questions that sound easy (but turn out not to be) abound: Who should pay for rework? What are the trade-offs between paying for extra robustness and software quality up front versus handling changes on an as-needed basis after delivery?

These are very large questions that touch on topics not only in this special issue but also in several recent issues: the “twin peaks” model of software requirements and design, which discusses how changes to one are interrelated with the other (Mar./Apr. 2013), and the metaphor of technical debt, which helps us understand trade-offs between short-term goals and long-term quality (Nov./Dec. 2012).

Interestingly, while the research community is further illuminating the complexities of the topic, I’m aware of one company, Advanced Information Services, that has boiled down the business aspects into a very straightforward model for its customers: “firm fixed-price contracting with performance guarantees, including a lifetime warranty on software defects.”

In the domain of US federal contracting for software-intensive systems, where many large projects are configured as “cost-plus” (that is, the contractor is reimbursed for all of its allowed development costs, up to a limit), it’s hard to overstate what a surprise it is to find this type of price guarantee. First and foremost, it’s important to notice how the usual cost-plus contracting approach puts the majority of risk on the acquirer (here, the US government); compare that with how AIS’s firm fixed-price moves the risk to the contractor. (The risk here is based on the fact that if the project turns out to be more complicated or requires more rework than originally expected, the contractor still only receives the originally agreed upon price.)

Founded in 1986, AIS has offices at multiple locations across the US. In 1999, it was a recipient of the prestigious IEEE Computer Society Software Process Achievement Award, “presented to recognize outstanding achievements in improving an organization’s ability to create and evolve software-dependent systems” (www.computer.org/portal/web/awards/spa).

To find out more about how AIS is able to provide this quality guarantee, I spoke with AIS CEO Girish Seshagiri. (For highlights of our discussion, an audio clip is available at www.computer.org/software-multimedia.)

Focus on Defects

The idea for a lifetime warranty against software defects was a natural one, Seshagiri said, because he was spend-
ing time looking at the number of defects on AIS projects post-delivery. This metric is one of the key things that all project teams report to him, and one of his best ways as CEO to get a measure of customer satisfaction from the company’s work.

Often on government projects, the majority of the maintenance effort goes to finding and fixing bugs. If the developer does higher-quality work in the first place, much of that effort can be redirected toward new features and system capabilities, or toward adaptive changes that allow the system to change as conditions do. Seshagiri reasoned that surely this would be an attractive proposition for his government customers.

And it was—but also noteworthy was the effect on his developers. A policy like this sends a message that can be empowering, namely, that developers have to stand behind the work they do. Giving developers the ability to make
decisions that enable them to do this with confidence fosters an environment with not only high expectations but also one where people can be especially proud of their work.

I asked Seshagiri whether he found it necessary to educate his customers about the benefits of a quality-focused approach, on the assumption that this would mean a more expensive contract. But he disputed the basis for the question. Looking across the life cycle, he’s found that the overall cost is actually lower with a focus on high-quality development. And the ability to meet schedule and cost goals becomes more predictable as well, another win for customers. In fact, Seshagiri wondered why software seems to be one of the few disciplines to take the stance that increased quality means higher cost, when so many other industries have discovered that improved quality brings reduced cost, increased market share, and other benefits.

Seshagiri did grant me that government acquisition personnel are responding to several different pressures when they award contracts, of which cost is only one. But in the current climate, high quality is no longer optional. Minimizing the $15 to $40 billion (depending on which estimates you believe) that US taxpayers pay for defect fixing on acquired systems is a worthy goal, but avoiding catastrophic damage to national assets—which could happen from successful cyberattacks that exploit poor-quality systems—is absolutely necessary. As former US Secretary of Defense Leon Panetta has said, “The next Pearl Harbor we confront could very well be a cyberattack that cripples our power systems, our grid, our security systems, our financial systems, our governmental systems” (www.nbcnews.com/id/43828271/ns/technology_and_science-security/t/nations-fight-against-cyber-intruders-goes-local/#.UkH9PYbkt8E). Given the existence of such threats, Seshagiri notes that the future looks very bright for companies that can deliver virtually defect-free software systems with a size of man hundreds of KLOC.

Enabling Technologies?
I asked Seshagiri to talk a bit about how AIS can offer such guarantees: Would he attribute this capability to process, technology, or people? A bit surprisingly to me, he pointed immediately to people issues as the determinant, noting that the work that’s done in software development is overwhelmingly carried out by people on teams: “The crux of the matter is, how do you form a team, how do you motivate that team, how do you manage them to where they can manage themselves?”

AGILE CONFERENCE AWARD
Continuing our relationship with the Agile Conference, IEEE Software sponsored a best research paper award at the event this year in Nashville. The winner was selected by a committee consisting of representatives from our magazine’s editorial board (John Favaro and Maurizio Morisio) as well as from the research track program committee (Mark Ardis and Lucas Layman).

Based on the input of our four expert judges, the award went to “Proposing Regulatory-Driven Automated Test Suites,” by Patrick Morrison, Casper Holmgreen, Aaron Massey, and Laurie Williams. In the words of one judge, “The Morrison paper is a solid new research result, showing that behavior-driven development (BDD) specifications are appropriate for standards-based compliance assessment. It addresses a clear industrial need that has emerged with the growing importance of automated health care information systems.”

Many thanks go to our selection committee for their timely and thorough reviews, as well as to research track chair, Richard Turner, for all his work on arranging this collaboration.

FIGURE B. IEEE Software editorial board member Linda Rising (left) and research track chair Richard Turner (right) were in Nashville to present the best research paper award to author Laurie Williams.
One thing certainly doesn’t help achieve high-performing teams: owing to time-to-market pressures, teams are often forced to work against arbitrary and unrealistic schedules. When this situation occurs, teams fall into a culture of “deliver it now, and we’ll fix it later.” So AIS avoids the root cause by not committing to a schedule without all of the team members’ participation to accurately determine how long the work will actually take. Even if the schedule isn’t what the customer wants, this gives teams the conviction to defend the estimate and explain why it’s realistic and necessary.

This of course requires investment in developer training and effort. For it to work, software developers must be able to estimate, create personal plans for how they will accomplish and track the work, and measure and manage their own quality. But this method avoids the vicious cycle that Seshagiri described: poor quality causes most of the schedule problems; schedule problems end up as quality disasters.

I mentioned that I was surprised that “process” wasn’t the top answer, given that AIS is a CMMI level 5 company. Seshagiri was quick to agree that process can be helpful. But too often, process artifacts can end up being produced by process compliance personnel—not the developers themselves—and then those artifacts don’t actually provide much insight into what’s really being done. As AIS has found, process only helps if the team doing the work takes ownership of the process and can create a process that actually helps get the work done. The team must be involved in deciding what process they’ll use for the work. And that brings us back squarely to people issues: When people have the ability to choose their own planning horizons and the methods used to accomplish the task, the quality they can deliver is amazing.

I asked Seshagiri whether every developer is cut out for, or even interested in, such work. On one hand, I could see this approach being empowering: people give input into estimating delivery windows and have a degree of choice to determine their own right way to get there. But I could also see how this might be daunting because it relies on a set of skills that not every developer has and, in fact, might not even want to attain.
Seshagiri noted that this approach does require a level of professional discipline. You must find people who are predisposed to working in a disciplined way, with an interest in planning their work, tracking their progress and their own defects and understanding where their own problem areas lie. One reason people push back is simply that not all developers have been trained to work in this fashion, which often has more to do with the way developers have learned to work throughout their careers when faced with arbitrary scheduling decisions.

As a recent example, AIS had a 17-person team that delivered 750 KLOC four weeks ahead of schedule with no security vulnerabilities. Half of that team was fresh out of college, which worked to their advantage. Seshagiri notes that if he can reach people before they learn a particular working style and train them on techniques that will help them get the job done and better understand their own performance, he finds that many (but not all) people really do prefer to work that way.

Connecting to Security

Although software security brings its own set of specialized issues and solutions, this type of professional discipline also has an important role in building secure systems, Seshagiri said. Individuals should have an understanding of known problems and existing security vulnerabilities to inform their understanding of what is necessary for a quality system. And this set of problems is fairly well understood: in the US, organizations such as the Department of Homeland Security and the Software Engineering Institute’s CERT program have invested effort into compiling usable repositories of common weaknesses and other security issues.

One of the problems that Seshagiri sees in the industry is an overreliance on testing tools and approaches. These do have benefits, but they can’t replace professional developers who are willing to stand behind and take pride in their own personal work. These developers should be supported by being provided with the necessary knowledge about security vulnerabilities and how they can occur throughout the life cycle. This information should be institutionalized into the way that developers do their work, by being built into the types of quality checks that get deployed as the software is developed, whether it be on checklists for inspections or other approaches. This is where organizational processes can help.

But again, such rigor must also be embedded in a developer’s personal process. For example, every component that an individual developer builds at AIS needs to be zero-defect when it gets past the component unit test phase. If the components are high quality themselves, then AIS can focus the effort on detecting and fixing any issues related to integration—and that also relies on sound design and architecture practices to reduce those defects. For this reason, component quality is another metric that gets tracked and reported up to Seshagiri at the CEO level. His goal is that 80 to 90 percent of the components should be defect-free in integration, systems, and acceptance testing.

Seshagiri acknowledged that they don’t always reach that mark, but a side benefit of aiming for that goal is that when coding is complete, they know what the defect profile will be downstream. And that’s where they get the level of assurance necessary to offer warranties on their code.

Forrest Shull is a division director at the Fraunhofer Center for Experimental Software Engineering in Maryland, a nonprofit research and tech transfer organization, where he leads the Measurement and Knowledge Management Division. He’s also an adjunct professor at the University of Maryland College Park and editor in chief of IEEE Software. Contact him at fshull@computer.org.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.