Don't Change a Thing

How Adopting a Service Provider Attitude Can Boost Your Career

By Michael DiBernardo

As a student of computer science, there's a significant chance you will end up working in software development after graduation. Despite whether your career path takes you into industry or academia, you're likely to have some kind of interaction with software development companies or organizations, if only in trying to get the most out of a project or collaboration.

Students and experienced software developers alike often assume that a new graduate starting his or her first job will work within an existing development process, without necessarily contributing to it or changing it. The new hire, it's assumed, has enough academic training to contribute to the codebase and intellectual framework around the product, but not to the methodology that's actually used to guide that development. This process can include status-quos for project planning, work estimation, revision control, check-in policies, product architecture, development techniques (such as design by contract, test-driven development, Scrum), testing methodologies, build management, and deployment.

But are software developers wasting their recent graduate employees by keeping them out of the process?

The idea of the "new graduate as apprentice only" reduces both the new hire's experience as an employee and the organization's return on investment of hiring. Many CS graduates of today's universities have significant experience with companies and organizations that have a track record of engineering excellence. Typically, this experience comes from co-op or internship placements, contributions to open-source projects, community or volunteer projects, and research assistanceships or "honors projects" that are supervised by professors or other highly esteemed mentors.

In the course of their internship-level work, new graduates gain experience working with processes that are often more efficient than the ones that are in place at their new employer. Failing to leverage these prior experiences to improve the performance is a major disservice to both the new graduates and the software companies that hire them. On the other hand, there are many things that can go wrong when trying to change processes that are already in place, especially if the person initiating the change is perceived as less experienced. However, by presenting themselves as "service providers," CS graduates can overcome much of the resistance that they would otherwise face. Taking on this role can speed up the process of establishing trust and credibility in a new organization.

The New Employee Trap

Before I discuss the service-provider approach to encourage change, I want to first consider how a new hire, who is eager to improve the status quo of software development, might approach the problem.

Let's say we have a new graduate named Mark, who has just joined a medium-sized organization called Informila on a team of about 50 developers. Within a few weeks of working at Informila, Mark identifies some existing processes that he would like to changed or replaced. Specifically, he finds that using Concurrent Versions System (CVS) as a revision-control system is causing problems that could be obviated by switching to a new system. Mark has already lost some work due to failed commits that submitted partial changes. Having previously implemented a conversion from CVS to Subversion during a co-op term, he believes that he could implement the change in his new organization with very little negative impact.

In a sentence, here's how Mark sees the problem from his perspective:

As a new but experienced team member, I want to make things better by changing the way Informila has always done things.

In other words, Mark views himself as part of the team already, albeit a bit green. However, the other developers at Informila may not feel quite this way just yet. It takes time to become an integrated member of a team, and Mark just hasn't been around long enough.

Another premise we can draw from Mark's statement is that he has a genuine desire to make things better, and to do so, he believes he needs to *change existing processes*. Change always comes at a cost. In this scenario, there are at least two currencies: time and money (which we treat interchangeably) and trust or reputation. The latter currency is much harder to gauge and is often forgotten—but it is a real cost. If Mark has not built up the trust or reputation required within his team to lead change, the other developers will resist it.

Back to the scenario: Mark suggests to his teammates that they migrate their current revision control system to Subversion. He is unable to convince them and their development manager that this is a good idea. Even though Mark has written some sample scripts to demonstrate that he can make the conversion without losing the existing revision history, and he has previous experience and thus some credibility, the team worries that this would be a risky change to their process and isn't worth the potential benefit. Mark decides that he needs to spend more time doing his best to demonstrate his development prowess by writing quality code before he can build the trust needed to make such changes.

The Service-Provider Approach

Thwarted from making any inroads, Mark had decided to operate within his narrow job responsibilities for some time while he slowly builds trust with the team. The problem with this approach is that it will take much longer than Mark expects to build his reputation. Even if Mark is an exceptionally good programmer, his team is already quite competent, and the decision-makers within the organization are more concerned with the performance of the team as a whole than they are with any individual. Mark may not realize it yet, but it will be difficult for him to differentiate himself technically from his teammates.

In the interim, Mark will struggle with the development processes that he strongly feels are inefficient, and as a result, he'll enjoy his work less. This will in turn affect his own motivation to differentiate himself from his peers. There is a better way. In his article "At Your Service: Creating a Service-Oriented Software Engineering Team within your Organization" (*STQE Magazine*, May/June 2001), Robert Sabourin describes how reframing a software engineering team as a service provider to other teams in the organization can greatly improve the ease with which that team is able to enact measurably positive change on the organization as a whole. And there's no reason this idea should be limited to only teams of testers or developers.

Sabourin says that when a service provider is first seeking out ways to earn trust from its customers, it's much simpler to build trust by *adding* services instead of trying to convince the customer to abandon a service provided by some other organization in exchange for one you will provide.

Almost every student of computer science is taught in introductory algorithms courses that a change can be viewed as some combination of an addition and a deletion. We can view the changing of an *existing* process in the same way. There are costs—time and money and trust—associated with removing or "deleting" the old process, and then similar costs for adding the new service. However, it typically requires much less of an investment of trust to add a service, because doesn't disrupt an existing structure. If you can manage to add a service that's useful to the organization and meet a need that was not currently being met before, you will build trust that can then be "spent" to change existing processes. In fact, once you have successfully added a few processes, you may even be asked to change things that you previously petitioned to improve in the first place.

A Matter of Mindset

Let's say Mark decides to take this approach. How does it differ from what he was already doing?

There are two steps to adapting a service-provisioning based outlook. The first is to reflect on the services that you are capable of providing, which may or may not already exist in the company.

For example, Mark is working in a Windows-centric environment, but he's also quite accomplished in Linux development. He has done a great deal of web application development using open source technologies. He is willing to spend some extra hours outside work on side projects related to his job, which is a luxury that many of his older coworkers cannot afford due to other responsibilities. Finally, he has a fair amount of experience in operating and integrating bug-tracking and revision-control systems. Mark decides that within his organization, he is capable of adding value by providing services related to 1) Linux development, 2) web application development, and 3) revision control and bug tracking.

The second step involves finding opportunities to "sell" these services in situations where people want to add to the existing process. This step is fairly simple because most people who are in pain are not shy about expressing their anguish.

There's one sticking point in the second step for many people, though. Most developers will hesitate to add components process that they feel aren't really needed, no matter how loudly others are clamoring for it. This desire to be efficient in the short term can actually hamper one's ability to improve the overall efficiency of the process in the long term, since adding services can be a great way to build up the trust needed to make broader changes.

As an example, Mark has often heard Andy, the director of test engineering, complaining that he has no good way of visualizing the statistics maintained by their current bug-tracking system. Andy would really like an application that could generate charts to show the trend of how the number of bugs opened, re-opened, resolved, and so forth, has changed over the course of a project, especially when each project approaches its deadline, and different branches are required to be maintained.

Mark's first thought when he hears this request is that some other open-source bug-tracking tools already have this capability, and that this functionality could be acquired "for free" by simply switching over. It's a procedure that he has implemented before, and he knows it can be done relatively painlessly, save for the fact that the test team and product owners would need to learn how to use the new system (a time and money cost).

The other approach—writing a program to solve a problem that has already been solved—is something Mark finds highly distasteful. In many respects, writing new software is wasteful compared to simply switching to the other bug-tracking system.

However, Mark reasons that if he were a consultant offering his services to this organization, this perceived need would provide the perfect opportunity to build trust with the client, because it doesn't involve changing any existing processes at all, and it's something the client feels is desperately needed. By meeting the perceived need, Mark can start to acquire the trust currency he needs to make things more efficient in the long run, especially since the director of test engineering is such an influential person in the organization.

Perceive and Be Perceived

It may seem as if adopting a service-oriented "method" isn't really much of a change at all. Mark's intent has not changed. He still wants to make things better within the organization. However, his perspective of what he's fundamentally doing has shifted subtly, and in a way that will improve how others perceive Mark and his actions.

Mark's new statement of action would be something like:

> *As a provider of Linux development, web application development, and revision control and bug tracking services at Informila, I would like to make things better by adding components or processes to fulfill needs that have been expressed by the organization.*

The outlook is totally different. Mark has gone from focusing on himself as an individual to identifying himself as a provider of services. While the shift is strictly one of notation, it removes some of the ego from his perceived role, and allows him to occasionally see the opportunity in courses of action that he might otherwise dismiss as silly or ineffectual.

In my experience, this seemingly superficial change has a very real effect on the kinds of opportunities that one will pursue, and the ways in which one will react when confronted with adversity in these pursuits.

Another shift Mark has made is that he's focused on adding processes to build trust before trying to introduce changes to existing processes. Lastly, he's focusing first on needs that have already been recognized by the organization, before trying to address things he has only noticed himself.

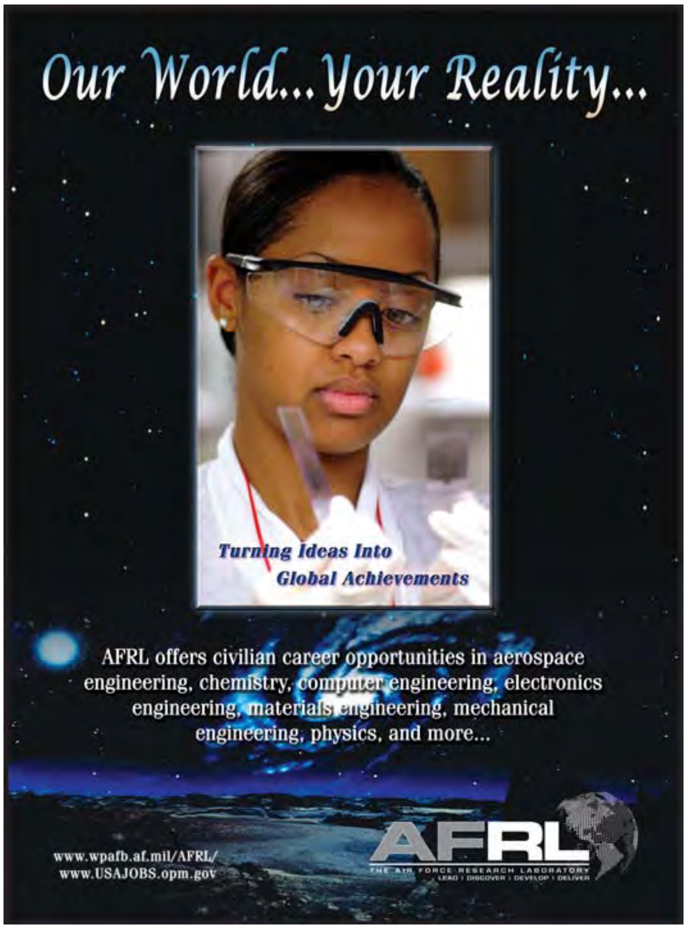
Mark ends up working on the bug charting system for Andy, and has a first version ready within a month. Andy uses the system to demonstrate some of the current statistics to management, and everyone remarks on how much easier the data is to interpret now with the new charts. Despite the fact that Mark has just essentially replicated functionality that was available in another system, by adding this component to Informila's existing workflow, he has made more headway toward garnering trust than he would have been able to accomplish through months and months of focusing solely on fulfilling his declared job duties.

In Perspective

I have met many recruiters, managers, and company owners who have commented that people in technical roles are expected to have excellent technical skills as a prerequisite. However, they are differentiated based on their ability to communicate and how they get along well with others. Seeing yourself—and getting others to see you—as a service provider is a highly valued soft skill. And you might just make your work environment a happier place for yourself while you're at it.

Biography

*Michael DiBernardo (*mikedebo@gmail.com*) is a software craftsmen who writes, teaches, and practices the art of software development. He has worked for the government of Canada, the universities of Waterloo and Toronto, Google Inc., and Novell Canada Inc. He holds a B.Math in Computer Science and Bioinformatics from the University of Waterloo, and an M.Sc in Computer Science from the University of British Columbia.*


Our World...Your Reality...
Turning Ideas Into
Global Achievements
AFRL offers civilian career opportunities in aerospace engineering, chemistry, computer engineering, electronics engineering, materials engineering, mechanical engineering, physics, and more...
www.wpafb.af.mil/AFRL/
www.USAJOBS.opm.gov
AFRL
THE AIR FORCE RESEARCH LABORATORY
LEAD | DISCOVER | DEVELOP | DELIVER

Assisting in the advancement of Ethernet Technologies

The Ethernet Alliance®
from carrier to consumer

Join the community of over 100 member organizations committed to the continued success and expansion of IEEE 802® Ethernet technologies

Being part of the Ethernet Alliance University Program allows students and professors to network with industry and standards leaders, contribute research studies on Ethernet technologies and participate in programs such as the White Paper Challenge and internship programs

EAUP

Visit **www.ethernetalliance.org/university-program** to learn more