

5 Practical Tips for Enterprise Architects

Apply practical strategies to make distributed development more effective for your organization and move your customers' businesses forward

In his national bestseller *The World Is Flat*, Thomas L. Friedman spends more than 100 pages describing "The Ten Forces That Flattened the World". Most of his examples center on how technology trends such as outsourcing and offshoring have changed and continue to change the world, and in most cases level the playing field for those not previously included. As enterprise architects, we are well aware of the benefits—from cost savings to faster time to market—that our companies are hoping to gain from distributed software development. If your company is engaged in outsourcing, offshoring, or nearshoring, or if your team has been merged or acquired into a situation where software development is now distributed, welcome to the club.

There is no end in sight for distributed development. The current practices of dispersing different activities and phases in the development process will continue to grow. This growth is for reasons ranging from retaining talent, to pursuing a 24-hour workday and shorter development cycles, to everyone's favorite, saving money.

However, a recent Gartner research note entitled, "Globally Distributed Application Development Demands Improved Collaboration," calls out the challenges of distributed development (see Resources). Gartner analysts wrote, "Application delivery and maintenance by a globally distributed team, although potentially cost-effective, poses a risk that comes from geographical and cultural obstacles to collaboration, exposure of intellectual property to out-of-country geopolitical conditions, and from complexity of tracking project changes and progress."

I agree, but I also believe that as enterprise architects we have a unique opportunity—and responsibility—to mitigate some of these risks. Here are five ways we can do our part to make the distributed development initiatives within our own organizations (more) successful.

1. Communicate using a formal design language.

Communication is the number one challenge for anyone working as part of a distributed team. It's no exception with enterprise architects. Many of us were hired because we not only have effective engineering skills, but we also have the ability to communicate effectively—often a key point of differentiation between senior development staff and enterprise architect job descriptions.

As we work with others outside of our locations, it's imperative that our communications stand on their own. PowerPoint slides and ad hoc Visio diagrams don't translate well across the country or around the world. For this reason, we should always describe architecture using formal design languages.

It doesn't matter whether you choose Unified Modeling Language (UML), Business Process Modeling Language (BPML), Alloy, or another design language, assuming you select one appropriate to the design task. The important point is that you use at least one. That way, you can ensure that the models you create will include the relationships and semantics you intended when viewed through the eyes of a reasonably literate reader.

My team has standardized on a specific subset of UML to help us specify, visualize, and document models of software systems, including their structure and design, in a way that is flexible enough to meet our requirements. We create UML 2 class, state machine, activity, component, and package diagrams, and we've also defined the subset of each type of diagram that our entire team must be able to read.

If we desire, we can now take advantage of new Model Driven Architecture (MDA) features, which include OMG's Query View Transformation (QVT) that is used in model-to-model transformations, and support for Object Constraint Language (OCL) 2. These features give us options for using the models as more than communication tools. We use other design tools as well, including XML schemas and business process models (using the BPM Notation), from which we can generate business process execution language for Web Services definitions (BPEL4WS).

A formal design model using a standard modeling language can increase productivity and quality among distributed development teams by enhancing communication, and automating design and code reviews by using audits and metrics at the model and code level. With increasing scrutiny to increase visibility and predictability throughout the software development process, design-level measurement is something I predict we will be asked to do more in the coming months and years.

We don't model everything and we don't try to maintain an up-to-date UML atlas of the system. We choose carefully those parts of any architecture that we want to document with formal design. In doing so, we increase the likelihood that we make good

use of the models and diagrams we do maintain. By having a relatively small set of diagrams and models that are core to the system, we can frame almost any discussion about capability or implementation.

2. Establish, maintain, and mandate architectural standards.

During a recent meeting with our customers, one of them brought up an issue that shouldn't be surprising to us. She said that after funding requirements engineering, quality assurance, and other areas, software development organizations allocate approximately about 40 percent of their budget to development work. Yet development represents 90 percent of the risk of cost overruns, late delivery, and/or outright failure. Her frustration showed as she described a common software development antipattern. Each time her Java developers took on a new project, they went through a software selection process, deciding between Struts, Spring, or Seam; choosing Hibernate, Enterprise JavaBeans (EJB), or Spring; selecting between JavaServer Pages (JSP), Velocity, and JavaServer Faces (JSF); and so on. They almost never did the same thing twice. They were always learning on the job, which meant they didn't have enough experience with the technology choices to deliver good estimates and a predictable outcome.

You run into this same problem with process churn: "Let's try XP on this project," or team churn, "This week, you four will work together." The team never does the same thing twice. As with any team sport, you drill, practice, and play toward ever-improving consistency. Improvement is a series of small steps forward from some baseline. If you keep changing the baseline, you can't get real improvement. To help your distributed team reduce its risk, insist on establishing, maintaining, and mandating a specific set of architectural standards.

Our enterprise architects have mandated that our architectural standards include EJB 3.0, JSP, and Web Services Description Language (WSDL). We have also expressed what at this time is not acceptable to use. Some organizations outsource standard architecture selection and maintenance, choosing something like SourceLabs' SASH Stack, which includes Struts, Access, Spring, and Hibernate.

By standardizing everything from project directory layouts to static analysis tools and unit test tools, we are assured that all of the code written—regardless of where it is written or by whom it is written—will run in our common environment.

If and when someone wants to make a change to our architectural standards, he or she must go in front of an internal architectural group, comprised primarily of enterprise architects responsible for maintaining our existing standards, to discuss it. The reasons for this formality are trapping attempts at duplicate work, capturing the reason for changes, and ensuring that all standards are documented appropriately prior to use. Teams have a way to make changes as needed, but the changes are under control. This way, we avoid having individuals in remote locations working with unapproved standards that may break our existing environment, causing us costly time and resource disruptions. The team treats the standard architecture like an internal product.

Although moving to a common architecture does cost resources (as some amount of work must be done to bring it online), it ultimately makes development less brittle, frees up valuable

resources around the world to work on higher-level business challenges, and improves communication.

The trick, in my opinion, is to avoid thinking that architectural standards are the key differentiator for the business. Architectural standards and models are useful, but we know the real money is in solving problems in the business domain. Let's not get caught in a web of false innovation or recreate the wheel when it comes to architecture. Now and into the foreseeable future the really interesting and much more challenging enterprise architecture work is in the business domain.

3. Create a domain model, and focus on becoming a business domain expert.

If your development teams are still focused on solving technology architecture problems, your IT organization is destined to become a cost sink. Enterprise architects have a responsibility to help turn distributed IT teams into ROI centers by spending the majority of their time and refocusing extended development teams on domain models.

Innovations in the domain model are critical success factors because they solve real business problems. If the domain model reveals that the underlying architecture needs to change, then it is time to fix the architectural model. Until then, however, the architectural model is probably fine once it has been established, maintained, and deployed throughout the organization.

Experts, including Peter Coad, David Taylor, and C.J. Date, believe in domain modeling or domain-level design and have written books on the topic. Coad and Taylor work in the object-oriented (OO) world. Date uses the relational model and business rules. Don't try to make up your own approach. Enterprise architects can follow existing business domain modeling frameworks from these experts, or pick another framework like the approach in Streamlined Object Modeling (see Resources <javascript:openWindowRes();>). The key is to deliver business value on top of that framework by becoming an expert in the business domain, whether that domain is software development, financial services, retail, or another industry sector.

For those of us moving to service-oriented architecture (SOA), this shift means don't begin by looking for and cataloging all of the existing services and components, another common antipattern. Start with the business process the business is asking you to automate. Begin with the BPEL or the BPMN and the metadata or schemas that describe the business. Take the business process models and see if you have services that map to the services they require. It will be more valuable to the business if you start at the level of business process, rather than cataloging the technical services that don't directly provide value to the business. A catalog of services built up as each service is used or created is much more cost-effective than spending a year or years building a catalog full of services that may or may not be used going forward.

Our organization's domain model—based on Peter Coad's modeling in color and OMG style metamodeling—allows enterprise architects and product owners to write and distribute to our dispersed development teams short user stories that include portions of the domain model by reference, building on the team's existing and hard won knowledge. We have also chosen to implement the domain model directly in a service layer that

allows us to more easily express and solve problems in terms of our business domain. Domain models provide the highest value to the business, short of working software. Therefore, the more that enterprise architects focus on making them right and getting them into a unified format that everyone on the extended team can interpret, the faster that IT can become a profit center.

4. Train teams on the architecture standards and domain model.

Once your architectural standards and domain models are defined, create standard policies and train everyone on the team to follow them. For example, formalize check in and check out policies when it comes to code and consider using a rules- and roles-based source code management tool to control processes, which can prevent costly downtime and avoid discrepancies from individuals having code on their desktops.

Within our organization, we think of architecture as a product—just one that is delivered internally—that enterprise architects need to maintain, update, secure, and bug fix. We use a source code repository to facilitate these efforts. By doing so, we help our development teams gain repeatability and predictability because changes are primarily incremental.

We've all seen the statistics. Rework can consume between 30 and 40 percent of the total effort expended on a software project. I believe that non-defect rework, and even some rework associated with defects, is often the result of friction between development organizations. Someone doesn't like the way someone else coded something, and it costs the entire team. Instead of encouraging distributed developers to own code or compete by teams or locations, train them all to follow the architectural standards and domain models. By following the standards and model, they will create an environment where anyone on the team, regardless of location, can review anyone else's code, and knowing they are all part of the same process can improve upon it efficiently. This process helps eliminate friction and allows everyone on the team to focus on solving business problems more quickly.

To ensure your organization is effectively training teams on your architectural and process standards, stay involved. As enterprise architects, we know this stuff cold, and we need to ensure its progress remains visible. Therefore, it's important that we don't just outsource the training of our architectural standards to anyone. Work with internal or external training partners to train in context.

For example, our enterprise architects deliver UML and Borland Together training to our teams. Then our enterprise architects partner to deliver the enterprise architecture, the domain model, and the metamodel in context to our distributed teams. To be most effective, train immediately and in context, before teams are going to use a model, a new approach, or a new technology.

Effective training fosters improved communication throughout distributed teams and pumps morale among enterprise architects because everyone can now more easily see himself or herself as part of one team. Don't forget to keep training dynamic. New employees and outsourcing organizations join teams regularly,

and enterprise architects in mentoring roles can always benefit from refresher courses and/or opportunities to formally present information in front of others.

5. Always consider your customer.

Even in a globally distributed environment, you should create co-located (sub)teams. Enterprise architects and product owners should be distributed equally with the development teams in various locations. Teams should not be stranded in locations where they can't get reasonable architecture and requirements responses quickly. This requirement means enterprise architects will need to travel to maintain their own team, but it's important to have them working closely with each team.

Like many distributed software organizations, we have found Scrum to be a highly effective development methodology to decrease time to market. However, with a growing number of dispersed developers, we have added components that we believe improve communication and pride in ownership. Some are relatively standard in distributed Scrum practice, like Scrum of Scrums and an architecture group managed under Scrum discipline. Additionally, we take the iteration deliverable very seriously.

Historically, individual developers would test a final piece of code on their own box and then send it to quality assurance (QA). Now with every iteration and build of our software, we include the installer, documentation, and any supporting test/demo elements to ensure that we are thinking about customers every time a feature is changed or added. This practice has done wonders to ensure enterprise architects and developers remain focused and accountable for making the end-user experience the best possible and that we're always testing exactly what customers will install. In early iterations, enterprise architects spend significant time working with the team to describe and build out the requirements and architecture for these capabilities. This time is crucial in today's business environment where customers always have another choice.

Taking the First Step

Today's global marketplace has forced all of us to work together, and work better and smarter. Enterprise architects sit in the perfect position to make distributed development more effective for organizations. Using these practical strategies, enterprise architects have the unique experience and ability to communicate effectively through visuals and context the software that will drive our business and our customers' business forward. Let's take the first step. **VSM**

Dan Massey is a chief architect at Borland Software Corporation (www.borland.com). Dan's main focus is Borland's IT management and governance solution that helps organizations accelerate their path to Software Delivery Optimization (SDO). His work includes frequent meetings with enterprise customers, designing approaches to aligning IT work and architecture with business objectives, process-centric planning, Borland's software process enactment metamodel, and the deployment architecture for Borland's IT management and governance solution. Prior to joining Borland, Dan was a TogetherSoft mentor and J2EE architect and developer. Contact Dan at dan.massey@borland.com.

Reprinted with permission from Visual Studio, May 8, 2006. © Fawcette Technical Publications. All Rights Reserved. On the Web at www.enterprise-architect.net. FosteReprints: 1-866-879-9144