Slicing and Dicing Development

An Analysis of Onshore / Offshore Models

There are many ways to organize your software development organization to take advantage of an offshore capability. This paper examines the pros and cons as well as the reasons behind some of the more common forms of onshore / offshore models.

By Wayne Simmons
Founder
Kata Consulting

Throw Your Requirements Over The Wall

When people think of offshore software development what generally comes to mind is the Throw Your Requirements Over The Wall model where the entire development cycle is performed by an offshore team. In this approach requirements are sent offshore and completed software is returned. Of course that's an oversimplification because there are reviews of the requirements, reviews of the designs, milestone releases, acceptance testing, etc. However the basic premise is that a company's business people are onshore and the developers are offshore.

The main reason a company enters into this type of model is for cost savings. A typical example is the large company that wants to shut down an onshore development facility. Many companies fit into this category after the dot com bubble collapsed.

Another example is the small company that has an ongoing need for custom software but does not have the resources to hire a development team and create the infrastructure necessary to support the team. Today, a startup web-based company may fit into this category as it allows the company to expand (or contract) as business dictates.

The Onshore Experts

This is the classic onshore / offshore model. The onshore portion of the development team consists of business analysts and software designers with expertise in a particular domain that cannot be replicated offshore. These experts create detailed specifications and possibly prototypes which are then implemented offshore. There is greater linkage between the onshore and offshore teams than in the Throw It Over The Wall model. However, there is a clear separation between design and implementation - design is onshore and implementation is offshore.

The main driver for a company adopting this model is still cost savings, but in this case the company is attempting to keep its competitive edge by retaining the onshore experts. The manufacturing world has many examples of this model. A car manufacturer, for instance, may design the car onshore but send the plans to an offshore factory for assembly.

The Forward Focus

The Forward Focus is another onshore / offshore model where the onshore portion of the development team is responsible for new versions of software and the offshore portion is responsible for maintenance of software. When the onshore team creates a new version of the software, responsibility for support and maintenance is handed over to the offshore team.

The main driver here is to increase capacity in the entire development team by maximizing the work of the onshore team. The expensive onshore team will focus on new features and the less expensive offshore team will focus on the more tedious work of fixing bugs.

This is a common way for companies to “test the waters” of offshoring. It allows companies to develop the offshore team over time and perhaps move the creation of new versions offshore eventually. Another upside is that this arrangement is easily reversible if it doesn’t work. It is also less prone to onshore rebellion since most developers are happy to give maintenance work to someone else.

This model has some big negatives though. The first is that there has to be a knowledge transfer with each new software release. This adds to the overhead for both onshore and offshore teams. The second big negative is that the company will *most likely* wind up with double the development infrastructure since each location will need its own development environment. That, of course, depends on the nature of the software being developed.

Follow the Sun

The premise behind the Follow the Sun model is that as one development center lays down its tools for the day another development center is picking them up.

For a service oriented organization this is a perfectly reasonable model. A global brokerage firm, for instance, may have traders in Hong Kong pass their open orders to traders in London at the end of the day. If those orders are still open at the end of the day in London, the trader can pass them to New York and so on.

For a software development organization, this model is simply not practical. The key is the unit of work. An order can easily be passed from one trader to another, however, a section of half written code cannot be passed from one developer to another.

There is one Follow the Sun arrangement that can work extremely well, though. If development and testing are in different locations testing can occur while the developers sleep and vice versa. This effectively creates a 24 hour work day. This arrangement can generate fantastic results for bug fixing. It is less appropriate for new feature testing which requires greater interaction between developers and testers.

The Long Hallway

This model treats the onshore and offshore sites as one virtual team connected by The Long Hallway of the Internet. There is no specific type of work done at either site, however, the overall management typically remains onshore. This model is based on the Open Source community and generally relies on collaboration tools.

The obvious downside to this model is the coordination overhead. The sites may be connected by The Long Hallway, but the time difference is still there. This model generally leads to a lot of conference calls during non-business hours for both sites and requires extra dedication by team members at both sites.

Companies use this model to increase development capacity. Generally any company that deploys this model will do so for short a duration in order to finish a particular project because it is not sustainable in the long term.