Indirect Communication

A perspective on the challenges associated with the exchange of information

While in elementary school my classmates and I enjoyed playing a game we called “telephone”. The teacher would whisper a phrase to one of the students, who would then whisper it to another student, who then whispered it to another, and so on until the final student would announce the phrase that had been passed from student to student. The class would enjoy a good laugh upon hearing the original phrase from the teacher since rarely did it ever approximate what the final student had announced. Hence, at an early age we learned of the challenges associated with exchanging information through indirect communication.

Something similar to this game takes place in the data processing industry between customers and developers. When I first got into this industry nearly thirty years ago it was common to hear claims about developers who could not hold a conversation with a customer without resorting to techno-babble, leaving the customer bewildered. One of the ways to bridge this communications gap was to put in place proxy agents who could act as interpreters between customers and developers. The job of these agents was to find out exactly what the customer needed and to convey that information to developers in terms developers could understand, as well as to keep the customer informed about what the developers were developing in terms the customer could understand.

There is no direct line of communication between customer and developer with this model, though it continues to be used today. Developers find themselves getting user requirements second or third hand. The more agents between the customer and the developer, the higher the chance that the developer will not completely understand what it is the customer needs and the higher the chance that what is finally delivered to the customer is not exactly what the customer requested. A perspective on the effectiveness of this model is offered by Mary and Tom Poppendieck:

> “The biggest cause of failure in software-intensive systems is not technical failure; it’s building the wrong thing. We know this. Study after study confirms it. And yet we continue to put intermediaries between the development team and its customers. We might call these intermediaries systems analysts, or product owners, but all too often they represent a handover of information. Secondhand information about the problem to be solved handicaps the people making the detailed decisions about how the system will behave. A critical role for product managers and product owners is to build bridges between customers and development team members, so they can talk to each other.”[1]

Not only do we continue to tolerate the separation of customers and developers, but even within an organization we stifle the lines of communication between those groups whose combined efforts contribute to the completion and delivery of the product. Much of the communication between these groups is through written documentation and conveyed using a technique known as handoffs – simply passing the documentation on to the next group, without any verbal exchange at all. Again, Mary and Tom Poppendieck point out the weakness here:

> “Handoffs are similar to giving a bicycle to someone who doesn’t know how to ride. You can give them a big instruction book on how to ride the bike, but it won’t be of much help. Far better that you stay and help them experience the feeling of balance that comes with gaining momentum. Then give some pointers as they practice starting, stopping, turning, going down a hill, going up a hill. Before long your colleague *knows* how to ride the bike, although she can’t describe how she does it. This kind of knowledge is called tacit knowledge, and it is very difficult to hand off to other people through documentation.”[2]

Management personnel in some software development organizations are surprised to learn that the voluminous documentation required by their carefully devised software development process does not by itself succeed in conveying the necessary information to those involved in the process, to the extent that it can facilitate not

1 Poppendieck, *Leading Lean Software Development – Results Are Not the Point*, p. 86
2 Poppendieck, *Implementing Lean Software Development – From Concept To Cash,* p. 77

retuning to preceding steps already presumed to have been completed. Mary and Tom Poppendieck present an alternative to this approach:

> "Sequential development methods, derived from contracting practices, encourage over-the-wall handovers, even though it has been demonstrated many times that concurrent engineering – engaging experts from all steps of the workflow in the initial design – is one of the best ways to eliminate design loop-backs."[3]

Perhaps we need to consider that agents and documentation are not going to resolve this dilemma with the exchange of information, and that steps for correcting the problem will involve both improving the abilities of developers to communicate with customers and changing software development processes to facilitate a higher level of interaction between people.

Jim McDonough – May 23, 2011

3 Poppendieck, *Leading Lean Software Development – Results Are Not the Point* ,p. 111