In the Zone: The Need for Flexible Roles

Blurring the lines of distinction between team roles is often necessary to adapt to a changing environment.

One of the ways people try to make software development more effective is in the definition of roles: if we can just put the right people in the appropriate positions, with well-understood responsibilities and clearly defined interfaces passing clean information from which effective decisions are made, then our project will be successful. Certainly, that would be nice if we could achieve it. But if precise role definition is so useful, why doesn’t it occur more often? And even if we could predefine exactly what people should do, would it be all that helpful? Maybe not.

Through this column, I have been exploring the concept of software as a knowledge medium and software development as a knowledge acquisition process. We can think of software development activities as making executable both what is known and what is not known. There are four categories of these knowns and unknowns:

The transfer and implementation of previously known knowledge—this may be in the executable software knowledge forms of programming language primitives, library functions, pre-defined objects, reused software, and reused executable work products of any type. Or it may be in the book form of systems documentation, system models, process models, templates, project plans, and even textbooks and methodologies. It may also be brain-resident in the form of experienced developers carrying knowledge forward from previous projects.

The definition of specific unknown quantities—which occurs when the presence of system variables, and perhaps their ranges, are known, but their specific values are not. Often this kind of knowledge acquisition is related to the first type or activity when, in order to use the previously acquired knowledge, we must understand the variance from either the “standard” form (as in functional reuse), or the “experienced” form (as in the experienced developer). In order to make this knowledge usable, some well-defined questions must be answered.

The discovery of unknown quantities—which often happens when, attempting to “prove” the aforementioned knowledge categories, we find things we simply did not expect to find. This is, in fact, the main reason we test systems.

The discovery and proof of previously unknown processes or approaches—which may happen as a consequence of the preceding category or from project...
The Business of Software

The problem with role assignments is that they can become a barrier to flexibility. People, groups, and even organizations can hide behind their job descriptions and play the “not-my-job” game.

post mortems and process initiatives. We discover this when we explore why our previous knowledge didn’t apply when we thought it would, or what kinds of processes might better uncover unknown aspects of similar systems in the future.

Readers of previous columns will recognize these as resolving the first four of the Five Orders of Ignorance.

When we look at explicit and rigorous role definition, it is quite obvious that its effectiveness is an inverse function of variability, as shown in the table here. Rigorously defined processes and their associated roles can only be really effective in the first two cases. They will almost certainly be less effective in the case of discovery, and are quite irrelevant in the case of process discovery since the definition of roles is a primary output of that activity. The effectiveness of role definition is a function of the quantity and types of unknowns present at the time when the roles are being defined. In the first two cases, the intrinsic variability is low. This means there are few unknowns and implementation consists of the tactical activities of either applying what we already know, or obtaining specific answers to the specific questions we already have. When we don’t know what we are going to find, or we don’t know a way to find it, the effectiveness of any predefined role must be suspect.

We can describe projects as mapping onto generalized domains within a development organization. As an example, in Figure 1, there are two intersecting axes. One, the “Requirements Continuum” represents the range of knowledge-factoring activities from the highest level of customer requirements down through the detailed proof of the satisfaction of the requirements (typically in late system testing). The horizontal axis, the “Organization Continuum,” represents the range of the organization’s resources structure allocated to either servicing the customer needs or perpetuating the development organization. It ranges from specifically dedicated resources allocated to servicing a particular customer need to generalized technology support functions that operate across the organization and are shared by all customer constituencies. This example is not definitive of an organizational domain, but it does help to explain the concept of the Zone Defense. A typical mapping of predefined organizational groups on this continuum is given in Figure 2a.

**Man-to-Man Defense**

Figure 2a is described as Man-to-Man Defense, where specific groups are set up with specific
responsibilities, reporting channels, and interfaces to other groups. This works well when there are few variables or unknowns to upset the applecart. The problem occurs when we find activities that do not fit neatly into the predefined roles. This commonly occurs when we discover unanticipated knowledge. None of the groups have been tasked with the responsibility of dealing with these items, since they were unanticipated. So who does them? Unless a group chooses to modify its predefined roles and take over these duties, they will not be addressed. If two groups pick up the tasks there will be some duplication of effort and maybe some contention between the groups. As the teams flex to accommodate the changing responsibilities, they change the dynamic of their assignments. But what happens in a situation where the assignments are constantly changing? What is the value of highly detailed role assignments at the start, when we know they will change to invalidate the assignment?

The problem with role assignments is that they can become a barrier to flexibility. People, groups, and even organizations can hide behind their job descriptions and play the “not-my-job” game.

**Zone Defense**

Taking a leaf from the football gamebook, we can set up groups as more of a Zone Defense. In Figure 2b, we see the Zone Defense version of the same organizational structure depicted in Figure 2a. Instead of explicit roles and interfaces (which will change anyway), the groups are set up with people of the appropriate skills, capabilities, and attitudes. Each skill area has a bias toward a discipline, but the actual task assignments are loosely defined and allocated on an as-needed basis. Some of the characteristics of this intentionally loose structure include:

- The presence of guidelines, rather than explicit role and process definition.
- On-demand task prioritization.
- Teams prepared for changing responsibilities and that have both the mindset and the change management toolset to navigate the constant changes.
- Willingness to back up people across nominal team boundaries. This means that as changing circumstances modify the tasks of (say) the Test/Validation Group, and a higher priority activity is identified, another group may adjust its priorities to backfill on some of the Test/Validation Group’s original responsibilities.
- Wider-ranging prioritization mapping—across the larger group, rather than strictly within the smaller functional groups.
- An expectation of change.
- An atmosphere of proactive cooperation and collaboration.
- Extensive and effective network-
The Business of Software

In many cases, the business of software is as fluid as a game of football or basketball.

- Careful management of the communication infrastructure—neither over- nor under-communicating.
- An acceptance of responsibility and accountability, rather than an assignment of responsibility.

This last point seems to be critical. Many management approaches favor the careful partitioning of work and the precise allocation of responsibilities to both individuals and groups. This is fine when such responsibilities can be defined. In a discovery activity, which all software development is to some degree, this often cannot be done to the necessary level of precision.

The difference between assignment of responsibility and the acceptance of responsibility is subtle but very important. If I am assigned responsibility for a task, but cannot predict what the tasks will be, or I must rely on others to perform tasks for me, I may fail through “no fault” of my own. When the task is not done, I can point to the others who are to “blame” for the failure. Conversely if I, and those around me, accept responsibility, we will not let ourselves fail if we, collectively, are capable of being successful.

This might mean that I change my priorities to cover a task not assigned to me, and perhaps you change your priorities to take on one of my responsibilities. The classical project management assignment process is often too slow to accomplish this in a reasonable timeframe.

The Ant Hill Project

A few years ago, I taught a lab class on project estimation that used some of the most popular estimation tools available. One of the groups attending the class had recently completed a telecomm project. It had taken eight of them nine months, and they delivered the system with zero defects. We plugged the known values from their project into the estimation tools, and the tools were quite unanimous: using any realistic parameters they could not do what they just did. This effort should have taken a minimum of 18 months. Even with that schedule, the tools predicted they should have delivered a large number of defects. But they did not. Obviously the tools were not calibrated to the performance level this team was able to achieve. But how did they achieve this?

Their project setup was very interesting, as shown in Figure 3. Taking a cue from the explorers Lewis and Clark, there were two co-equal leaders of the team. These were skilled and experienced engineers who also happened to be good friends. The other six members of the team had varying degrees of skill and experience. The co-leaders provided extensive technical direction in terms of system architecture, but chose not to focus on or direct the team members’ tasks.

The mandate to these members was simple: work on whatever you think you need to work on. Team members were given almost total freedom to work on whatever they thought were the most critical tasks. If, by working on these tasks, they determined they
should be working on other tasks, so be it. If the team members needed direction they could ask the team leaders. If they felt they were competent to decide what they should work on, they could go ahead and work on it. There was an expectation that team members would discuss their choice of tasks with each other and with the team leaders, but even this was not demanded. Furthermore, a contract was arranged that, for critical items (as determined by the person asking), one of the team leaders would provide an answer within 10 minutes.

The project was set up rather like a client/server application. And as the knowledge from the “server” level (the team leaders) was passed to the team “client” level, the team’s need to employ this channel was reduced. The almost complete lack of task direction meant the team, in full discovery mode, could immediately react to changing circumstances either on an individual or a team basis. The high degree of technical direction, however, meant that peoples’ designs were far more likely to mesh. This is the opposite of traditional time-constrained projects where task management is defined from above and controlled down to 15-minute intervals, but technical direction is left in the hands of individuals. This project resembled more an ant hill than a structured system, with each team member contributing, learning, and discovering at a progressively optimal rate. Their results spoke for themselves.

In Defense of Zones
As well as Zone Defense, there is also Zone Offense. Describing Tex Winters’ “Triangle Offense” basketball coach Phil Jackson said “For the strategy to work, all five players have to be moving in sync so that they can take advantage of the openings that occur when the defense overextends itself.”

In many cases, the business of software is as fluid as a game of football or basketball. We need an overall plan, and we need to know what our roles are to achieve that plan, but we also need to learn to adapt flexibly to a constantly and ever more rapidly changing environment. When we understand that much of what we do is a discovery activity, we see that strict roles and rules may not work. When we can routinely devise project and organizational structures that are intrinsically elastic and can flex to meet the needs of our constantly changing situation, then we will be in the zone.

Phillip Armour (armour@corvusintl.com) is a vice president and senior consultant at Corvus International Inc., Deer Park, IL.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or a fee.

© 2003 ACM 0002-0782/03/0500 $5.00