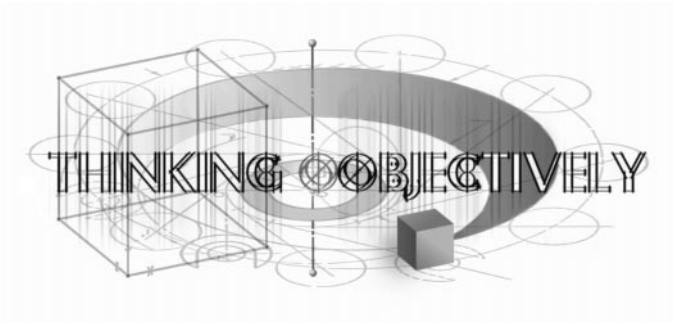

Mohamed E. Fayad

The Art of Managing Multiple Processes

The people performing the process must own the process [1]. While a process group may be essential in helping to capture a process or analyzing a process, it is the group using the process that is likely to have the best understanding of its purpose and appropriate use. In fact, we recommend forming a process team to document exactly how to apply OO techniques. This team should consist of members from all phases of software development—systems analysts, designers, coders, quality assurance, and testers. The team should meet regularly to determine what, how, and when to document. Obviously, the people performing the processes should be included in meetings.

There are a number of issues regarding the delegation of the processes. First, processes do not usually eliminate the need for skill and experience. For example, if defect tracking has been done on an ad-hoc basis with software developers doing the work, implementing a process would not necessarily allow the task to be done by clerical staff. The judgment of the developers would still be required, but the reporting functions could be done by others. (In some relatively rare cases where the use of highly trained people compensates for a total lack of process, defining and implementing a process can allow less-experienced people to do the same work.)

Second, it is important to recognize that the people performing a task may know well what they are doing. The knowledge of the people doing the tasks must be used as the basis for defining the new process. This suggests that when process specialists or team members from other groups are brought in to help define the process, they should not assume a superior attitude. The process must also work for the group doing it, not for an idealized group with skills and attendance rates that do not match reality. A corollary to this is that process "experts" should not create the processes by themselves; experts tend to create processes so complex and fragile they cannot be used by mere mortals.

Complete uniformity is not always desirable. Two issues arise in the management of multiple processes:

- Each project-level process may be different.
- Each project-level and micro-process is similar.

The fact that different groups have a number of similar processes leads to the ideas that each group's process work should be absolutely uniform and that each group should have the same number of processes. This is not necessarily correct.

Different groups may be in different phases of transition to new development methods, suggesting different procedures. A group maintaining a reasonably stable product over time have different needs and procedures than a group developing a new product. There is little point in destabilizing a working process, assuming its efficiency is adequate, in order to establish conformity with a new process. The cost is often prohibitive. Obviously, a small group working on a small application will not require the same level of logistical and coordinating effort a large group on a large project might need. Creating a process means thinking about what is to be done and how to do it. Having

people who use the process as a tool to achieve an end, and having accurate documentation and measurement is much more important than some ideal of process uniformity.

It is common for development equipment and software to change during the lifetime of an application, and it is often desirable or necessary to change the processes in response. Any change carries some risk, and in order to minimize the risk, it makes sense to prototype changed processes before fully deploying them. However, even with prototyping there can be problems due to environment. For example, defect tracking might be done by a single person in the prototype stage or in a small group, but if the volume of defects increases, more people might be needed and some form of coordination between them would be required. Part of a solution might be to acquire a commercial problem-tracking program. But almost any program contains an implied usage method that might differ from the ones established by the group. And the implied procedure, along with the changes required by increased defect volume, would require a few iterations to make a workable process. A prototyping effort has more flexibility and can react to changes more quickly than if the whole organization has adopted the new process, and overall time can be saved. Before committing to the new program and its new process some thought must be given to the effect of the change, and the new procedures should be prototyped.

Process experts and process teams can help identify process issues, but they should not be the ones who create the process and hand it, fully completed, to the people who will use it to define the process. A working process is not a thought experiment. It cannot be imposed because it looks good or has some apparently desirable property like orthogonality.

This is why the people who use the process must own the process; they will be using it. By owning a process, we mean those who use the process must be the ones who define and modify it. The users will define the primary metrics collected from the process and these metrics must first be used by the process owners for control and improvement. Management information must be a secondary goal of such data collection. To counter objections before they are voiced, we are not saying that processes should not be subjected to outside review or benefit from expert assistance. Certainly, the identification and analysis of statistical data gained from the process will require the help of a trained statistician. Process experts should be available to identify errors such as procedures that can deadlock or loop indefinitely. They can identify risky areas such as long feedback loops and ambiguous states, where, for example, it is not clear if a task has been properly completed. Expert assistance and system reviews also are useful in reducing sub-optimization.

Sub-optimization. One of the major concerns when dealing with multiple processes is sub-optimization, that is, simplifying a process or making it more efficient at the expense of up- or downstream processes. This is common between different functional groups. The classic example of software sub-optimization is cutting corners during development in order to meet schedule, thus dramatically increasing the work needed during testing. This tactic still works. Oddly enough, as a quality assurance (QA) team pulls out an increasing number of defects, some still assume that those testing are inefficient because they could not immediately find all the layered problems. Thus, the QA department is blamed for release delays rather than the software developers who apparently made their deadlines.

There are two requirements for reducing sub-optimization: management involvement and whole system analysis. Given that sub-optimization is a zero-sum game often existing between separate groups, and given that separate groups are unlikely to work for a mutually beneficial solution without management involvement, the active involvement of upper management is without question. Frequently, basic business processes must be modified.

An example might be reevaluating the cost of software development to include maintenance and support. This reevaluation may require personnel shifts, retraining, and special information system work to add to inter-group information sharing. At this level of process improvement, only senior management can make the changes. Likewise, a process improvement that reduces headcount in a group is unlikely to be recommended by

members of the affected group. Only upper management can act appropriately. Davenport [2] states that the competitive climate in most companies, especially at higher management levels can work against efforts at process improvement since suboptimization is often equated with greater personal power. At these levels, there must be a reward for process improvement that compensates for potential loss of functional control.

Given management involvement, the analysis of the system as a whole can be beneficial. In the previous QA development example, an analysis of the system would certainly show that upstream changes in the development phase reduces the efforts in the downstream QA testing phase. Although this example is obvious, the development process changes cause a different product to be delivered for testing. The testing phase, in turn, changes from a desperate search to uncover layer upon layer of defects to a more orderly and consistent verification and validation of the system. In fact, it might be found that by careful analysis of process inputs and outputs, substantial time and effort is saved. Analysis also shows what constraints exist between processes. For example, if the development requires new software to interface with a legacy system, then the interface might be more complex and harder to test than a completely new development.

Process merging. Use existing processes and work habits. Build new processes from these existing processes or extend existing processes by adding new functions. Improvement is most often based on experience. This implies costs of continuing analysis and maintenance. When looking at what to turn into a process, consider the risks. The risks associated with the actions, the risks associated with developing a process (continually changing operations may risk becoming quickly obsolete or prohibitively expensive to processize) and the risks of failing to implement a process.

Configuration management. It is important to treat processes like other critical development tools, and therefore all processes should be put under configuration management (CM). In this case, CM may be a category of document management, or it may include configuration control of software tools used in the processes as well. CM, because it becomes possible to reconstruct the processes used in the development environment, is necessary to analyze procedural problems during the lifetime of the project. In a multiyear project, only complete CM or processes will provide enough data. Even if contractual or legal requirements do not require process CM, it is still a good practice. C

References

1. Fayad, M.E., Laitinen, M. *Transition to Object-Oriented Software Development.* Wiley, New York, 1998, to appear.
2. Davenport, T. *Process Innovation: Reengineering Work Through Information Technology.* Harvard Business School Press, Boston, 1993.

Mohamed E. Fayad (fayad@cs.unr.edu) is an associate professor in the computer science department at the University of Nevada, Reno.