Chapter 13
Defining and Documenting the Development Process

13.1 The Manager's Roles and Responsibilities in the OO Software Process

Throughout this book we have been emphasizing the link between OO and controlled development. OO techniques by themselves do not include progress reviews, extensive documentation, or bi-directional requirements traceability although such features are necessary to make any significant development successful. To address such topics, we need detailed, repeatable documentation that can guide and control our work. The process is the fundamental way of implementing that control. A process is a description of the steps required to implement some goal, usually part or all of a method. Processes transform textbook theories and method descriptions into real action steps. Documented processes enable the development team to consistently apply and benefit from the application of OO techniques. It is essential to realize that processes are codified steps that describe a particular organization's way of achieving development goals. This means that processes cannot be acquired off the shelf, but must rather be developed over time [Fayad97a]. As shown in Figure 13.1, detailed processes are dependent on the application area, object-oriented methods, tools, and languages in use.

Figure 13.1: General Processes Must Be Tailored to Your Projects.

Fayad points out that “Management must support the move to process-based development. This implies that process must not be abandoned when schedule pressures loom or process costs initially slow some development phases. Processes are especially important for new object-oriented development teams. Even in a well-organized group, new methods and tools introduce confusion. Individuals will often perceive themselves as less skilled than before and the routines they had established with others will certainly change. Management must make sure that establishing process-oriented development will allow team members to contribute positively. It is management's job to show how processes will help achieve the overall goals of the organization and how each team and its members fit into the big picture. But perhaps the hardest challenge management has in promoting processes is to make sure that people do not view processes as weapons to be used against them. Promoting this view requires a change in management's thinking from the individual as the basic unit to the team, and from individual performance measurement to process measurement. Process measurement will highlight problems and errors in the process. If these measurements are used for performance reviews rather than process improvement indicators, the process is doomed to fail” [Fayad97a].

13.1.1 Measure Processes rather than People

Process orientation has developed a somewhat tarnished reputation because of the way it has been implemented. The goals of process orientation are to improve reliability and efficiency, thereby increasing quality. Too often, organizations try to "install" quality by the use of techniques such as processes. With
the increased emphasis on process as technique, problems may arise [Laitinen98]. Process paralysis and losing sight of the goal of creating products are common traps. Attempting to use off the shelf processes, devaluing skill and experience in favor of processes, and putting "experts" in the position of defining and imposing processes all contribute to the failures that have damaged the reputation of quality management. Having installed processes, some clever management types often decide to "speed" things up by setting goals above the current statistical average for processes. This is destructive. The only way to improve goals is to change the process. Processes are tools that, if done with a sincere organization-wide approach, can help improve work quality and increase productivity [Laitinen98].

If management succeeds in creating a process-oriented approach, the next logical step is to base management on the use, improvement, and measurement of processes. This is a radical step, and we cannot do full justice to the idea in this short section. We refer the reader to sources that expand upon this idea [Deming 86; Latzko 95; Senge 90]. We lay out the basic idea by reiterating that effective development is a team effort. Just as the misuse of process data will cause people to subvert error reporting, a system of processes that allow one part of the system to succeed at the expense of another will also be destructive. Processes help organizational systems run effectively, and to do that, processes cannot be viewed in isolation. When we suggest a different approach to managing people and processes, we do not mean that traditional issues such as absenteeism and personal responsibility should be ignored. Rather, by focusing on the process, a fairer evaluation of people can be made. And since coordinating people to reach a goal that cannot be met by individual action is one of the prime tasks of management, measuring processes is an excellent measure of management itself [Laitinen98].

**13.2 The Top Five Excuses for No Process Documentation**

Process orientation is hard to adopt. Processes are commonly seen as extra bureaucracy that only serves to make a project less effective. In far too many cases, this perception is correct, and process adoption is resisted. Even if the organization is sincerely committed to adopting a process oriented approach, many excuses will be offered at first. We list a few in Figure 13.2 and discuss them in detail in [Fayad97a]. Many others will require much effort to overcome. While there may be some truth in the excuses, they are still excuses rather than valid reasons for lack of documentation. The emphasis must be made to move forward.

**Figure 13.2: Top Five Excuses For No Process Documentation**

Our short answers for the above listed excuses are that undocumented procedures are not scalable or transferable. Using repeatable processes reduce avoidable mistakes, and free developers up for more creative, less routine work. Processes are meant for the people who implement and use them. Other uses are usually, and rightly, considered busywork, and such processes will end up buried on shelves, unread and unused. There is no ideal time to start implementing processes, but documenting processes while performing them gives them the best chance of being accurate and useful.

**13.3 Where to Start and How?**

Very few organizations have established a set of defined processes for software development. Those groups having processes often have not spent the time and money to do real process assessment and improvement. It is common, in our experience, to see "processes" that are merely lists of rules in a somewhat arbitrary order. In many organizations, especially those trying to conform to the Software Engineering Institute's Capability Maturity Model, turning everything into a process has become a goal in
itself. We believe this is the wrong approach. Software development organizations exist to develop software rather than processes [Fayad97a]. The intent of the SEI [Paulk95] and other process improvement programs is not to change focus from developing software to developing processes, but instead to use processes and process improvement to better develop software.

13.3.1 The Trouble With Process Assessment

Process Improvement Models

Software process improvement programs begin with assessment. This activity is intended to give a development organization a sense of where it stands in terms of software production skills. In most assessment models, the organization evaluates its development capability against a set of "best practice" that are supposed to be found in effective organizations. The number of practices, their mastery, and their level of integration into the development organization determine the organization’s assessment score. There are a number of process improvement initiatives, of which, the best known are SEI’s CMM, SPICE, the United States Department of Defense SDCE, ISO 9000, ISO/IEC 12207. Some programs allow self-assessment while others require outside certification.

The Software Engineering Institute’s Capability Maturity Model (CMM) is one of the best known and most widely discussed software process improvement model. It defines five levels of organizational maturity, from initial or chaotic to optimizing. Each increasing maturity level, starting at level 2, has associated with it a set of key process areas. For example, level 2 includes (among other things) requirements management and project planning as key areas. Level 3 includes training and peer reviews. Levels 4 and 5 include software quality management and defect prevention respectively. Each level also includes the process areas of its lower levels.

SPICE, Software Process Improvement and Capability Determination, was developed as an international meta-standard under ISO/IEC that doesn’t aim to replace other standards but to provide a benchmark for current and future process improvement initiatives. SPICE assessment recognizes two categories of software engineering practices: base practices which are essential practices of specific procedures, and generic practices which are applied to any process. It lists five process areas of concern: customer-supplier, engineering, project management, support, and organization. Capability levels for base practices range from 0-not performed, to 5-continuously improving.

ISO 9000 is a set of international standards that mandates the existence and use of written procedures and requires assessment for certification by an outside organization. The idea of the standard is to produce written processes that are consistently followed and can be continuously improved. ISO 9000 certification is widely used in Europe and is becoming increasingly a minimum requirement for doing business there.

ISO/IEC 12207 is a relatively new standard that provides software life cycle process definitions. While it does not require outside assessment or certification, it has strong influences from the United States Mil-Std 2167a and 498. Therefore, it may be assumed to be used as a basis for contractual agreements between suppliers and customers.

Problems With Assessment

One difficulty with assessments is their cost. According to El Emam and Briand [Emam 97] the initial assessments can be costly. Reports for CMM assessments and improvement range from $100 per person to many times that. The average time required for an organization to move from level 1 to level 2 was 30 months and from level 2 to level 3 25 months. The average cost in time for SPICE assessments was
110 person hours, but it varied up to 824 person hours. For ISO 9000 certification, the studies indicate a varying person-month effort based on the number of people in the organization and the degree of pre-existing compliance with ISO 9001. The least effort reported was 13 person-months of effort for an organization that was 85% compliant. While the organizations listed in the study all reported improvements, only companies that had positive outcomes were included. Currently there is little data that verifies assessment and improvement always work. Assessments, then, are expensive, and in an immature organization the results of such an assessment are likely to be meaningless. For example, if a department does not use configuration management to keep track of product versions, there is no process to assess. Until configuration management is put in place, and used for a while, assessment of this facet of development will have significant cost but will not even serve as a baseline for further measurement. Time and effort could be better spent in acquiring and implementing a configuration management system. Even in more mature organizations, it is not necessarily valuable to do such assessments. For smaller organizations the cost may be prohibitively expensive, and the results point not at specific issues that will help the organization improve but rather at ways in which the organization falls short of meeting CMM standards [Fayad97b].

The Software Engineering Institute's Capability Maturity Model (CMM) is a valuable theoretical model of levels of maturity in an organization. One problem with it, however, has been its general acceptance as meaningful standard for progress. As of this writing, a decade after the CMM was published, most development organizations are still at level 1, the bottom level, and only two organizations in the world are accepted to be at level 5. Most of the world's best known commercial software is produced by organizations at or below level 3. And yet, despite this apparent lack of progress with respect to the CMM, nobody would want to move back to the software of a decade ago. Given this somewhat dismal correlation between the CMM and the state of software development, it is surprising that organizations are being encouraged (or required) to be certified at CMM level 3 in order to get government contracts [Fayad97b].

There is a certain orthodoxy about placing organizations in various levels that does not reflect the reality of software organizations. As reported by El Emam, many organizations find the early adoption of certain processes, such as change control and code reviews, more effective than adopting them in the recommended sequence [ElEmam97]. Bamberger [Bamberger97] reports that when she works with clients, she helps them look more at the essence of the ideas of the CMM rather than the explicit maturity levels involved so they can get control of their software projects. At lower levels then, getting a start in controlling projects is more important than orthodox progression through levels of maturity. Further, few level 1 software organizations believe they are at level 3 or above. While they would like to be more "mature" as the SEI defines the concept, gaining such certification should not be a primary goal.

Another problem with process improvement orthodoxy (and not just for CMM) is that there is still no causal evidence that software process improvement initiatives (SPIs) always work [ElEmam97; Jones 96]. As stated above, the preponderance of success stories to date only proves that organizations are not especially willing to report on costly failures. Even more interesting is possibility that other factors may be involved. Candidate factors include improvements in software tools, the "startup effect" in which new initiatives get much more highly qualified and motivated people than standard projects, and the idea of "heroic efforts" [Bach 95]. We suspect, however, that as more studies come out, a slightly different perspective will emerge. In the past, for example, various worthwhile programs such as TQM and quality circles have been adopted and subsequently discarded. Upon analysis, it wasn't the programs that failed; it was their application and the expectations that people had for them. TQM doesn't work as a technique; it only works as a fundamental approach to quality. Quality circles, likewise, are worthless without other enabling organizational changes. In the same way, SPI as technique may prove to be of little value. However, if processes are looked upon as tools, and if the effect of process improvement is to see how work
is done and how tools might be used to improve the work, then SPI, we predict, will have a positive effect [Fayad97b].

Small and medium sized companies have been put off by the presentation of software assessment and SPI. There is no question that assessment can be costly, and for a small, lean organization, the overhead involved in meeting and verifying CMM or ISO 9000 criteria can be prohibitive. ISO 9000, for example, lists dozens of conditions and types of documentation required for certification. For these groups, if ISO 9000 certification is required, they may elect to buy an off-the-shelf solution that meets the letter but not the spirit of the requirements. And for a small group, the CMM level 3 requirements of training programs and peer reviews might be nonsensical. As Bamberger notes, the way CMM is presented often makes smaller organizations feel it offers them no value [Bamberger 97]. Moreover, smaller organizations cannot afford the two to three year duration it normally takes to reach CMM level 3. If CMM certification becomes required for contract awards to subcontractors, then rather than fostering improvement, it will more likely become a meaningless regulation [Fayad97b].

Process improvement, then, must be tailored to the organization and with the goal of improving systems. The best part of assessment with respect to various standards is that a smart organization can use the assessment as a framework to evaluate how projects are done. And then, by conscious analysis rather than slavish adherence, the organization can plan and take steps that will improve its operation. We feel that the focus on the existence of processes, at least in the way that many people apply CMM and ISO 9000 assessments, tends to over-value the technique at the expense of the goal. As we stated earlier, processes are tools that help with solutions rather than solutions themselves [Fayad97b]. Fayad97b discusses several problems with assessment models in detail. The Communications of ACM’s Forum, April 1998 has several responses to a number of questions related to this issue [Forum98].

13.3.2 Process Paralysis

Because of the hype and pressure to improve processes, it is easy to move into "process paralysis" paralysis. Process paralysis, as defined by Yourdon, [Yourdon87] is when the project team can become thoroughly overwhelmed by the new technology and gradually end up spending all of its time: (a) trying to understand the new technology, (b) arguing about the merits of the new technology, or (c) trying to make it work. At the micro process level, this paralysis can cause groups to forget that they are developing software rather than processes. Part of these problems can be attributed to a misunderstanding of what a process is [Laitinen89].

13.4 How to use processes as a baseline for improvement

A baseline is a formally agreed upon specification that then serves as the basis for further development [IEEE87]. The baseline is used as the measured starting point of each of the documented processes. Baselines provide indicators for major milestones. Milestones can be described as major development events and baselines are major milestones [Bennatan95]. The baseline identifies the weaknesses and strengths of each of the existing processes. A complete baseline includes all the improvable facets of documented and tracked processes (see Figure 13.3).

Figure 13.3: Defined Processes Are Baselines For Improvement
13.5 Software Process Hierarchy

We categorize process hierarchy in two ways the organizational level and the process granularity. We can identify a software process hierarchy in three major levels, the industry/government level, the company level, and the project level, as shown in Figure 13.4. The industry level includes the well-known government and commercial standards for process definition. For example, the Software Engineering Institute's Capability Maturity Model describes process models applicable to the whole software development community. It is much more of a description of a way to view the software process than a description of how to do processes. Government standards, in contrast, tend to be more prescriptive. The new standards, MIL-STD-498, J-STD-016, and the U.S. Commercial Standard 12207 (ISO/IEC 12207) describe which processes should exist at the macro level and what components software life cycle process should (or must) consider. Naturally, these standards do not spell out step-by-step details.

Humphrey [Humphrey 89] uses yet a different classification: universal, atomic, and worldly. The universal describes basic process steps at a general level, e.g., the spiral or waterfall models. The atomic level might include the object oriented method used, and the worldly includes the specific and detailed level of tasks.

At the company level, the processes spelled out in detail are usually those that concern compliance with regulations and similar legal matters. Most company-wide processes will be relatively general, stating how divisions or departments will interact and what general life cycle issues will be observed. The company's software engineering objectives will be spelled out at this process level. In some cases, processes at the company level will essentially be policy statements.

There is a significant difference between a policy and a procedure. A policy sets the goals for an organization and may provide guidelines to reach the goals. A policy does not provide detailed steps and descriptions of operations. Since it is a high level statement, it generally doesn't undergo continual revision and modification. A process is more detailed, has expected inputs and outputs, and is used as a tool to achieve the goals of the applicable policy.

The project level processes are specific to individual projects. At this level, processes tend to be specific and the proportion of micro-level processes is much higher. This level contains processes identifying groups and project duration. Many processes will be customized to specific projects with their own particular personnel and equipment.

Since there is no unequivocal standard for the hierarchy, we somewhat arbitrarily arrange processes into macro, mini, and micro levels. Macro processes cover the whole software development life cycle. A macro process would include the software development methodology, like the waterfall model or the spiral model. It would also cover the fundamental approaches used, such as the OMT or Unified development. A mini level process would include the methodology used to do analysis or design or a process existing in a phase of the development cycle. An example of this might be the OMT analysis method. A micro process is one that covers a specific task and lasts for a short period of time within a project phase. Examples of micro processes include defect tracking, object identification, or risk assessment. All processes describe "who, what, and when," but lower level processes also have more on "how."

Figure 13.4: Software Process Hierarchy
13.6 How To Document The Processes
And How To Tailor General Processes To
Your Project

13.6.1 First Steps

Defining and documenting processes is neither quick nor cheap. For an organization without a
great deal of experience, the costs and disruption of implementing processes can be a risk. However, it is
difficult to improve development capabilities without using well-defined processes. We recommend a slow
start, documenting existing operations and reviewing any written processes that exist.

Defining a suitable format for process documentation is an important first step (refer to Figure
13.5). We have found that a combined graphical/text format was easy to understand, communicate, and
maintain. The graphics provides sufficient abstraction and decomposition for easy communication with
developers, management, and customers. The textual descriptions complement the graphics by providing
additional details. Section 13.7 gives more details on a symbolic notation for process documentation.

Figure 13.5: Integrated Graphical/Textual Documentation
Recommended

A process describes a series of tasks to be done to achieve a specific goal. It tells who does what,
when the tasks are done, and how to do them. The task descriptions list conditions required to start and
continue, and the criteria for task completion. They describe measurements taken during the process for
the purpose of control and reporting. Processes exist at different levels, from detailed descriptions of short-
term, single-person tasks to high level descriptions of the development process. The purpose of defined
processes is to have a repeatable and verifiable way of achieving a goal.

Micro processes, those that describe specific, detailed tasks, look much like cookbook recipes,
while macro processes, which describe operations and interactions of larger systems, cannot be as specific.
For example, your software development process may be based on the spiral model. The documentation of
that process will not contain detailed steps and interaction.

The process's definition concentrates on what and when software products are required. For
example, what level of documentation is required for a preliminary design review? Which graphics are
incorporated into our requirements specification? What is the exit criterion for object-level testing? We
found that by precisely defining the software products required for each development phase, each developer
can maximize individual contributions and still maintain system consistency.

The software process must be defined with sufficient detail that any competent developer outside
the scope of the current project could correctly answer the question "what's next?" Specifically, the process
defines the intent, techniques, entry and exit criteria, and appropriate quality standards for each step of the
development. Figure 13.6 shows appropriate process details.

Figure 13.6: Identifying Appropriate Process Detail
13.6.2 Recommendations for Documenting Object-oriented Processes

Having stated general principles, we now suggest items to remember when documenting OO processes. First, specify entry, exit, feedback, task (who, what, when) measurement, and approvals. These are essential components of any process documentation. Then review the list below for other essential pieces.

• Concentrate on tracing inputs to outputs. Traceability is one of the most important traits of successful analysis and design. One of the ways to ensure it is to make sure that inputs and outputs are connected in some understandable way. It is not sufficient to merely list inputs and outputs.

• Identify all reviews. According to the development method you choose, reviews will be part of the process. You should be able to identify in your processes the points at which reviews take place. Part of the review process is to define the expected products. These may be completed work packages or more significant items. There is usually no point in having a review unless its outcome indicates some action to take. This may be to proceed to the next step, re-do the previous step, or to add another step to the process. In any case, the approval authority required to take action must be defined in the process documentation.

• Identify where configuration management occurs. CM defines versions, revisions milestones, and components of a project. These are all items to be documented in the development process.

• Consistently specify all roles involved. Developers, SQA, testers, customers, marketing, sales, and management will have functions in any project, although not all phases will include all groups. A consistent method of identifying the roles will make the process documentation more understandable.

• Map method products to each software activity. OO development will have many outputs: graphics, tables, text, configuration management object, code, and other repositories. Documentation should identify what product is used in what software development activity. For example, object interaction diagrams would be mapped to the design phase and would contain graphic depictions of message flow between objects.

• Identify what/when from CASE tools. CASE tools in order to be useful, must produce output that is integral to the process. Interface descriptions, for example may be one of the outputs. These should be identified in process documentation. If some action depends on the output of a CASE tool, this should also be described.

• Identify appropriate OO notation for documentation. Having decided on an OO development method, it is essential that the documentation describe the method using the same notation as is used in analysis and design. Output from CASE tools may be the way to get this notation into the documentation.
Specify what level of coordination must exist between objects. This information may be documented with the help of interaction diagrams, object flow diagrams, state transition diagrams, and interface descriptions. Again, CASE tool output may provide these items.

13.7 Process Documentation

13.7.1 Symbolic Notation

We present here an introduction to a symbolic notation for process documentation. The advantage to a symbolic notation is that it is easier to follow than the equivalent textual description. Detailed text items can be attached to the symbolic notation for further clarification, if necessary. Missing or overly complex steps tend to show up quickly in symbolic notation, as do unintended loops and other bottlenecks. Figure 13.7a and 13.7b show notation scheme that will aid in reading the process. Appendix A has complete examples using the following notation to illustrate several processes.

1. Participants Chart identifies roles where the persons performing the process are showing between the horizontal lines.

2. Process Step Name contains three components: a) Input description that describe process inputs, such as tools and information, b) Name of the process step and a procedure, and c) Output description that describe process outputs, such as products.

3. Transition Arrow indicates the transition from one process step to the next step.

4. Decision Diamond indicates that a decision must be made before the next step.

5. Filled Process Box decomposes further to sub-processes or describes a step of a procedure that has further detail provided at the lower level.

6. Parallelogram contains two or more process boxes. These process boxes may be concurrent or may be performed in any order.

Figure 13.7a: Process Symbolic Notation

Figure 13.7b: Process Symbolic Notation

13.7.2 Template

The following template gives a layout for documenting the essential elements of a process. It can be used along with the symbolic notation to document a process fully. Appendix A has a complete example of using the following template.

Process Documentation Template:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Title</td>
<td>Defines process name, such as Software Inspection Process</td>
</tr>
</tbody>
</table>
Figure 13.8: Process Documentation Template

13.8 Impact Of Object-Oriented Development Processes On Managing The Project

For an organization just starting the move to process oriented engineering, the effort can be significant and the impact can be disconcerting at first. One of the major problems with any significant change is reorienting the experience of managers and developers to the new norms. We especially expect management to understand trouble signs in a project, but both the ramp-up of learning new methods and the changing types and duration of the methods make trouble spotting difficult. One thing is certain for object-oriented development: more time will be spent in analysis than in traditional development. Generally, the design phase is also longer. While we expect these costs to be offset by lower overall development time and reduced maintenance time, the cost changes must be understood.

We describe in chapter 8 training approaches that can help the transition to OO development. While training is often regarded as an extra cost item or an employee benefit, we feel that there is so much to absorb in moving to object-oriented processes that formal training is absolutely necessary. In fact, training should be considered a strategic advantage of the organization. The down side, of course, is that training is expensive, both in direct cost and in personnel availability, and it must be conducted over an extended period.
One of the firm requirements of significant object-oriented development is that analysis and design artifacts (for example, the interfaces between objects) be firmly defined and controlled. For significant development, CASE tools must be employed to help ensure such control. We discuss CASE tool selection in chapter 6, so we only mention here that the cost of the tools and time of selecting and learning them is a significant project impact. In fact, it is unlikely that integrating CASE tools into the development process is a one-time cost. Each project will have different needs and hence different object modeling requirements. As experience grows with a modeling method, the CASE tools will be used differently.

The use of iterative development models (necessary for OO) will change the need, character and frequency of reviews. As an example, it may be reasonable for an analysis artifact that has gone through multiple iterations to have a more limited inspection than one that is just starting out. As discussed in chapter 15, new technology makes it possible to change the number of people participating in inspections, so that larger, more complex objects may be inspected by more people than would be possible in a conventional inspection meeting.

We have talked at length about the impacts of OO development methods on the management of the project. Here we state again that moving to OO methods will take time, effort, and money. Moving to a process-oriented software engineering will be similarly costly. If there were no advantages to process improvement and OO development, then it would be extremely foolish to undertake the changes. But experience has shown that process improvement and object-oriented development have substantial long-term benefits, once mastered, throughout the lifecycle of the projects. Object-oriented development helps fulfill the fundamental goals of software engineering. As figure 13.9 below details, Object-oriented development provides formal mechanisms to address such principles as,

- Confirmability. By providing information in the model to verify correctness.
- Abstraction. At various levels of detail, to extract only essential information about the system.
- Uniformity. Elimination of inconsistencies and unnecessary differences.
- Completeness. Assuring that all requirements are satisfied.
- Localization. Assuring that related resources are in proximity.
- Modularity. Software is divided into manageable portions.
- Information hiding. Limiting access to information to those functions that require it, and then only by well defined access means.

Figure 13.9: Incorporate OO Principles Into Software Processes

13.9 Process Criteria And Essential OO Process Elements

13.9.1 Macro Processes
Organizations new to iterative development and OO technology should probably not develop elaborate macro processes in advance. (This does not imply that you should develop without a software development plan.) Whether the software process swirls in a spiral or spurts like a fountain is far less important than developing a strategy for estimation and control. The work package concept, estimating tools, and a way of assessing progress and necessary change should be documented and modified as development progresses. Some of these areas may never achieve a cookbook-like level of repeatability. For example, the spiral model shows prototyping and risk analysis at every full turn of the spiral, but the order in which these actions occur is usually not critical and trying to specify the order in the process is most likely a waste of time. This does not mean, however, that macro process actions should be ignored, undocumented, or not measured for improvement. Macro processes tend to control most or all of a software lifecycle, and almost always extend across functional boundaries of the organization.

13.9.2 Mini Processes

OO analysis, design, and development require experience to master. As you progress on a project, attempt to capture the phase level processes used, and review regularly to see how they are changing. Do not attempt to codify the process too soon, especially analysis and design, since they will be evolving as experience with it grows. This does not suggest that you have a haphazard approach to analysis or design. The analysis and design methods must be in place before you begin; appropriate training must have taken place; and an object/method expert must be available to help keep analysis on the right track. However, the details of using the methods and the sequence of work will vary as experience grows. This same advice is true of all the mini processes of development. If you are new to OO methodology, the way you use it will be changing rapidly. It will already be expensive enough to take the time and effort to capture the processes used without trying to formalize them too soon. At first you may want to develop usage notes or other less formal guides. Similarly, avoid gathering too many metrics at early stages. Not only are they expensive and disruptive, but since a new process is hardly likely to be in statistical control, all but the grossest measurements will have little or no meaning.

13.9.3 Micro Processes

Attending to micro processes may be every bit as important as the "higher" level processes: they form the day-to-day systems that take most of the time and effort in development. Because "everybody knows how to do micro processes," and they tend to cover unglamorous tasks, they are often the most overlooked and least organized portion of the process hierarchy. Besides, people are doing these tasks regularly, as part of their normal activities, even if the tasks are not formally documented. In fact, documenting and improving micro processes can dramatically smooth the development operation and allow more resources to be focused on the new features of OO engineering. A good micro process gives specific steps to accomplish a task. Each task should have a clear set of preconditions for starting the task, and clear indications of task completion. Using a schematic approach to documenting the process can help avoid gaps, redundancies, and unintended loops.

Be especially sensitive when creating micro processes to the level of skill required to perform it, and the fragility of the process tasks. If performing a routine task requires expert skill, then the process should be analyzed to see if it can be simplified. Note that some processes legitimately require high skill levels to execute, but sometimes the process is unnecessarily complex. By fragility, we mean the likelihood that a person familiar with the process can perform tasks incorrectly(thus causing serious delay(or can perform the tasks incorrectly without clear error indications. An example of a fragile task might be a data entry procedure in which a single keystroke error requires starting the task over from the beginning. Another example might be a program build task in which a syntax error in a source module doesn't stop the build but merely prints an error message on scrolling output. This latter error might not be discovered for
some time if the previous version of the executable is in the location where the updated executable was to go.

While the warnings about formalizing processes too quickly don't usually apply to micro processes (they should be specific/process paralysis and over "processizing" should be avoided. The benefit of turning some actions into a process, or the liability of not codifying some set of actions, should be compared to initial and continuing costs of having the process.

13.9.4 Who Does the Process?

The people performing the process must own the process. 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, programmers, quality assurance, and testers. The team should meet regularly to determine what, how, and when to document. Obviously, the people who will perform the processes should be included in meetings.

There are a number of issues regarding who does 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 do the same work.)

Second, it is important to recognize that the people performing a task may well know what they are doing. The knowledge of the people doing 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 just for an idealized group with skills and attendance rates that don't match reality. A corollary to this is that process "experts" should not create the processes in isolation: experts tend to create processes so complex and fragile that they can't be used by mere mortals.

13.10 How To Measure Progress By Using OO Software Development Processes

13.10.1 Process Improvement

Implementing a process will probably result in an initial reduction of performance for the obvious reason that people will be adapting to new procedures. (Why this perennially shocks everyone is one of the mysteries of organizational life.) We must watch, however, for process paralysis. If we detect it, we must get the project back on track by emphasizing that our goal is a product rather than a process. Consultants and outside process improvement teams may help with mired groups. Remember that processes, once defined, will undergo incremental evolutionary change and that changes in the volume of work to be handled, or the technology underlying the process, may cause fundamental changes in the process. For example, a process for defect tracking on a small project might be wholly inadequate for a large project. As
we stated above, we fully expect the mini level processes to change as we learn more about OO and about the project itself. With process evolution or changes in environment, the whole process hierarchy should be examined for improvement possibilities. The project's process group should coordinate with the company's process improvement group.

When working on process improvement, remember that reactive change will make things worse rather than better. In most cases, people are impatient with the time it takes to make a process better, and they identify every variation in process operation as a problem to be fixed. The result is chaos. A cycle of change, measurement, and evaluation must instead be followed. Deming makes this point forcefully [Deming 86].

13.10.2 Process Metrics

People tend to invest mystical properties in statistics that are sent to managers. "We just collect it and let management decide," is often the attitude. Managers, on the other hand, feel compelled to ask for statistical data, even if they don't use it or methodically review it. They somehow feel they aren't doing a proper job—or think they appear to be uninterested—if they don't ask for and save piles of data. This just proliferates the collection and generation of useless data. Certainly in micro level processes, statistics should be more oriented toward informing the people doing the process so they can control and improve it. Moreover, at the micro level, some data measurement may come and go as needed [Laitinen98].

13.11 What You Should Look For And What Actions To Take

13.11.1 What a process is and is not

Let us first describe what a process is not. A process is not a replacement for experience or skill. It does not turn untrained personnel into effective workers. Process development is not a make-work project for incapable employees. It does not solve all operational problems, and "processing" a development quickly will not result in improvement. It is not, in itself, the goal of a software development organization. A process does not guarantee error-free development, nor does it prevent false starts or dead ends. A process is not a code word for a set of rules that management imposes upon subordinates. In fact, it is not merely a set of written rules. Having said this, we can begin to examine the properties of object-oriented processes.

13.11.2 Process Properties related to Object-Oriented Methods

- Philosophy. Any software system has three different perspectives: structural, functional, and behavioral (or data, transformation, and control). The structural perspective concerns the decomposition of the system into a set of interacting and/or nested components and the allocation of data, functions and behavior related to these components. The functional or process-oriented perspective is the transformations on data without concern for where (structure) or when (behavior) the transformations occur. The behavioral or control-oriented perspective is the system's response to external events, the propagation of control through the system, and the scheduling of work items (functions). This criterion examines the ability to model process for each the three different perspectives.
• Simplicity. This criterion covers those attributes of the object-oriented technique that provide various modeling aspects of the problem domain in the most understandable manner. This criterion measures technique complexity in terms of number of process steps, notational aspects, constraints and design rules. In addition, this criterion is also concerned with the familiarity of the modeling notations of the technique to the developer (user).

• Tailorability is the customizability of a technique to a domain. Aspects of the object-oriented technique can be altered to tune the technique to fit a particular purpose, organization, individual needs, or a particular problem domain. This property is very useful if the object-oriented technique can be consistently extended or tailored to fit a particular domain. Extensions are either notational changes or process modifications that incorporate development expertise or implementation needs. This criterion addresses implicitly three other criteria: scalability, decomposition, and incremental growth. Scalability addresses reducing or increasing the technique’s notation and process according to the problem size. Decomposition is a special case of scalability where decomposition addresses breaking the problem into manageable components. Incremental growth has two aspects: (1) the ability to produce models that allow the development of the product in increments, and (2) the technique should have the ability to produce increments of its own deliverables allowing delivery of pieces of the object-oriented analysis or object-oriented design.

• Measurability. Measures are quantities that can be evaluated to determine whether a software process meets a particular criterion or to evaluate the degree to which a software process exhibits a certain characteristic, quality, property, or attribute. Repeatability. If two groups come to the same set of models (output) for the same problem domain, this is repeatability.

• Testability is the extent that one can identify an economically feasible technique for determining whether the developed software products (requirements specifications, design, operational software) will satisfy the specification. To be testable, requirements specifications must be specific, unambiguous, and quantitative wherever possible [Boehm84]. Reusability is the degree to which one project’s software development products can be reused on another development project. Reusable items include requirements specifications, design artifacts, code, test cases, plans, etc. Then reusable software products are software products developed in response to the same requirements for one application that can be used, in whole or in part, to satisfy the requirements of another application.

• Traceability is a measure of how well implementation traces to design artifacts and to satisfy a certain set of requirements. Traceability has five elements: (1) The document in question contains or implements all applicable stipulations of the predecessor document, (2) A given term, acronym, or abbreviation means the same thing in the software documents, (3) A given item or concept is referred to by the same name or description in the software documents, (4) All the material in the successor document has its basis in a predecessor document, that is, no untraceable material has been introduced and (5) The two documents in a traceability relationship do not contradict one another (Adapted from [DoD-STD 88]).

• Maintainability. This criterion refers to the ability of the object-oriented technique work products to accommodate changes (for corrective maintenance), to adapt the product to a changed environment (for adaptive maintenance), or to add new requirements (for perfective maintenance). Information hiding or abstraction and precise and up-to-date documentation increase flexibility and play major roles in overcoming the maintenance problem. This criterion examines: (1) if the
object-oriented technique employs information hiding and abstraction and (2) if the technique provides documentation aids.

- Abstraction is a representation of something that identifies the essential details while omitting non-essential details. Abstraction can exist in many different levels. High-level abstraction focuses on general characteristic of the abstract thing and low level abstraction reveals more details of the abstract thing. There are many types of abstractions: functional abstraction, data abstraction, behaviors abstraction, process abstraction, object abstraction, and others.

- Information hiding focuses on making certain details of an abstract thing inaccessible [Parnas 72]. Information hiding is the deliberate hiding information to promote: (1) abstraction and readability (e.g., by suppressing detail until the appropriate time), (2) modifiability (e.g., by preventing the unauthorized use of private information that is subject to change without notice), and (3) portability, reusability, and extensibility (e.g., by limiting the impact of change). A better choice of words for information hiding would be "information protection". The concept of that information should only be granted on a need to know basis.

- Verification is a process of confirming or substantiating that implementation meets the requirements. This process involves demonstrating that the models are consistent within and across phases. Verification is the act of reviewing, inspection, testing, checking, comparing. Verification also is the formal proof of program correctness. This criterion refers to technique processes used to determine the degree to which work products fulfill the requirements.

- Validation is a process of confirming or substantiating that development meets the needs of the customer. Validation answers the question, "Am I building the right product?" [Boehm 84]. Then, this criterion assesses the degree to which the object-oriented technique provides for demonstrating that the end system (product) satisfies the user's requirements. Degree of Formality. The formality of the technique depends on the formality of three major aspects of the technique: the notations (notation formality), the relationship between phases (relationship formality), and the verification of this relationship (proof formality). The formality of notations is the most obvious to satisfy.

- Performance Engineering addresses the ability of a computer system to perform its functions, for example, response time, throughput, and number of transactions. Then the goals of performance engineering are to: (1) ensure timely response to externally generated events; (2) guarantee periodic processing; and (3) maximize system throughput without sacrificing the system reliability and maintainability. Issues related to performance engineering are: resource allocation, throughput bottlenecks, processing delays, communication delays, time constraints, and others. Performance engineering must begin in the analysis phase of the software development (performance analysis). Performance engineering techniques falls into two categories: (1) ad hoc, such as prototyping and (2) formal performance estimation using analytical models, simulation, and mathematical proof.

### 13.11.3 Process Capture

Capturing existing processes can be very difficult. The danger is that one will record too little relevant data (assuming that everyone knows all the steps) or that one will record far too much data, making the essential features of the process murky. The questions one must ask include, how do you know when to do it.
• Are there variations in priority that affect the sequence of steps?

• What critical factors are there in the process? For example, are there tasks that are easy to do incorrectly, hard to tell if they have been done correctly, or that if done incorrectly can cause the failure of this or downstream processes? For example, in building software for production, if development and debug flags are not turned off, it may not be recognized immediately, but security can be compromised and performance or even operation can be degraded for customers.

• How do you know if you've done the task properly? How do you know when to start?

• Process, vs. heuristic vs. policy vs. plan. Is this really something that should be a process? For example, in debugging there are a number of heuristic techniques used, but usually there is not a step-by-step method that guarantees finding and proper analysis of the defect. Consider not turning it into a process. As a corollary, what benefit accrues by turning something into a process? Is it useful or does it merely increase bureaucracy? (Just because it was a process on another project does not mean it should be one on this project.)

• How do outputs vary? By volume, type, etc,. Are there different outputs? Following the process must yield the same output each time or the process is ambiguous. It may be that the process is not understood at the proper level.

• What are the upstream processes that feed into this process? Are any upstream processes skipped sometimes? Are there more than one that feed in? Are there multiple processes that can start this process?

• What are the downstream processes this feeds into? Is there a fan-out? Are there different processes that can be started by the same output? Are there cases where a following process will be skipped depending on the output of this process?

• Basic metrics: How often is this process done; how long does it take?

• How scaleable is this process? E.g., can you double the process and still use it? A ten times volume increase? Can more people be added to do this process, or does another group have to be formed? Can you easily have others do it, or does it require specialized skill? How do you know when the process is overloaded?

• Can this process be meaningfully extended? This question should be reformulated to ask how general the process is. For example, a source code inspection process can be extended to do document inspections, but what each type inspection looks for and metrics are kept will be different. At a micro level, they might be considered two different processes. At this point, it makes more sense to ask, is this micro process part of a family of similar processes, and if so, are they coordinated as much as possible?

13.12 How To Manage Multiple Processes
[Fayad98] discusses the following two approaches in detail:

- **Complete uniformity is not always desirable**

  The fact that different groups have a number of similar processes leads to the ideas that each group's process work should be made 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 which would suggest different procedures [Fayad98].

- **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. It 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 all too often [Fayad98].

### 13.13 Tips and Practical Advise

13-1 **The development process must be documented.**

   More specific detail is needed than what is included in the published technique. You must also describe how the process has been adapted to your project. The project's level of detailed decisions should be recorded. The project process should provide "what's next?" to the team members.

13-2 **Use a project process team.**

   Using a team with members experienced in creating processes is important. They will assist in producing a workable process, but they do not dictate the process to the development team. The development team should own and enforce processes. Adapting a new development technique will undoubtedly change the way developers do their jobs, and a documented development process helps them understand what is expected of them during each development phase. It is also important that team members buy into the new development technique. The manager must insure that they understand the development technique and how it has been adapted to their specific project.

13-3 **Measure Processes rather than people.**

   Processes make a framework that defines appropriate action and efficiency. Effective execution of a process is usually easier to measure and more accurate. It reduces opportunities for an employee to look good at the expense of coworkers. Inter-group conflicts can often be traced to conflicting expectations and requirements. A process-oriented evaluation can show that the conflict is built in. Measuring processes does not mean that individuals no longer have responsibility for performance, and it does not suggest that managers can overlook traditional
personnel issues. It does, however, provide a clear path of action when employees identify process failings.

13-4 Software development processes map the abstract theories of the OO technique into concrete and repeatable actions.

A well-defined process is essential. Our experience in applying a new OO technique emphasizes the importance of having a well-defined software process. Software development processes insure that the software products have a consistently high quality, and using an unfamiliar development technique makes them even more important.

13-5 Processes cannot be acquired off-the-shelf but must be developed over time.

Trying to buy a package of processes virtually guarantees a bad fit, with more work required to adapt the processes than is gained by the purchase. Process experts may help, and for any meaningful measurement, statistical expertise is a must, but the process must reflect what its users do and must reflect the environment in which it is used.

Process prototyping and analysis can help process development. Work flows, state transition diagrams, and even Petri nets may aid in process analysis [Holdsworth94].

13-6 Warning: Poor Process Use is Crippling.

Processes are tools, nothing more. Too often they are used in place of effective management or even as punishment ("Keep those programmers in line!"). Installing process orientation does not cure most organizational problems. Be cautious about trying to use processes as a tool for replacing experienced people with less experienced (and less costly) workers. It usually doesn’t work.

Processes can, however, be a very useful tool for training less experienced workers.

Process paralysis can destroy the organization's focus and can drive out the most productive developers.

Too many processes at the wrong level trade suffocating bureaucracy for chaos; too few processes overlook critical areas and encourage sub-optimization; wrong processes stunt efficiency and hurt morale.

13-7 Process-oriented development lets team members contribute effectively

One of the most important functions of a process is to make the team members think about what should be done and how to do it. A process should not be a set of rules imposed from the outside and passively followed. Process improvement lets the team take a bigger role in shaping work. It helps them understand the work of other team members. Interestingly enough, process-orientation can also help management better understand and more appropriately value the roles all team members play.

13.14 Summary

Effective use of object-oriented development requires strong coordination at every phase of development. At the macro process level, an iterative approach brings out the advantages of object technology, while a more staged waterfall-like approach obscures its advantages. A major portion of the transition to OO technology is the adoption of mini processes, such as choosing methodologies for analysis, design, and development. In many cases, implementation of mini level processes will signal the most radical departure from the previous development environment. Much of our book has been devoted to the evaluation and acquisition of the tools and methodologies that mini processes will organize. The costs and risks of these new tools and methods are high and considerable attention must be paid to embedding them
into processes that make them assets rather than threats. At the macro and mini level, process improvement is as often process implementation.

Some of the greatest inefficiencies in an organization exist at the micro process level. The repetitive tasks that consume so much of any software development project are often wells of frustration for the people that must perform them. We feel that some of the greatest gains can be made by analyzing and improving micro level processes. At the micro level especially, process improvement asks the people who do everyday tasks to look at their work with the mandate to make it more efficient and less frustrating. Providing expert help to analyze and guide process improvement can reduce the conflicts that characterize sub-optimization.

Finally, adopting a process orientation implies thinking about how the organization works. When this is done, it creates an environment for process innovation with its opportunity for dramatic improvement. As object-oriented technology advances, it provides the basis for new modes of development, such as reusable components (often in the form of third party class libraries) use of design patterns, and more radically, application frameworks. We are just beginning to see the impact of these new development modes. But in order to take advantage of them, we must have an understanding of how we work, and the best way to get that understanding is by looking to processes.

References

[Bach95] Bach, J., “Enough About Process: What We Need are Heroes”.

[Bamberger97] Bamberger, J., “Essence of the capability maturity model”.

Management Practices and Techniques, John Wiley & Sons, New York,
1995.

[Boehm84] Boehm, B.W., "Verifying and Validating Software Requirements
and Software Requirements Engineering, IEEE Computer Society Press,

[Boehm88] Boehm, B. W., "A Spiral Model of Software Development and

Davenport, Thomas, Process Innovation: Reengineering Work Through


[Emam97] El Emam, Khaled and Briand, Lionel, "Costs and Benefits of
Software Process Improvement," in Better Software Practice for Business
Benefit: Principles and Experience, IEEE Computer Society Press,

Department of Defense. Defense Systems Software Development, DoD-STD-


[Fayad97b] Fayad, M. E. and Laitinen, M. D., "Process Assessment: Considered