Enhancing the Unified Process:
Software Process for the Post-2000 (P2K) World

A Ronin International White Paper

Scott W. Ambler
President
Ronin International Inc.

This Version: March 15, 2000

Copyright 1999-2000 Scott W. Ambler
Acknowledgements

I'd like to thank Doug Smith, Information and Education Architect at Blueprint Technologies (http://www.blueprint-technologies.com) for his valuable feedback regarding this white paper.

If you would like to provide feedback, please send an email to scott.ambler@ronin-intl.com with the subject "Feedback regarding Enhancing the UP". Thanks in advance, and I'm looking forward to hearing from you.
1. What is The State of the Software Industry?

In short, not too good. Software projects are usually late, are usually over budget, are likely to be cancelled, and if not then the final software rarely meets the expectations of their users. Why is this? Is it the compilers that you are using? That doesn’t seem right. Is it the database management systems that you store information in? Nope, that doesn’t sound right either. I know, it must be that you are not using the latest and greatest programming language, component library, or CASE tool… ummm… ahhhh… that cannot be right. It's not a technology problem, it’s a process problem. If you do not understand how to develop software then chances are very good that you won’t be very successful doing so, and our industry’s 80-90% failure rate at large scale, mission-critical projects is an excellent indication that we need to change our ways.

Step back for a minute. Pretend you want to have a house built and you ask two contractors for bids. The first one tells you that using a new housing technology he can build a house for you in two weeks if he starts building first thing tomorrow and it will only cost you $100,000. This contractor has some top-notch carpenters and plumbers that have used this technology to build a garden shed in the past and they’re willing to work day and night for you to make this deadline. The second one tells you that she needs to discuss what type of house you would like built, and then once she's comfortable that she understands your needs she'll put together a set of drawings within a week that you can then review and provide feedback on. This initial phase will cost you $10,000 and once you decide what you want built she can then put together a detailed plan and cost schedule for the rest of the work. Which contractor are you more comfortable with, the one that wants to start building or the one that wants to first understand what needs to be built, model it, plan it, then build it? Very obviously the second contractor has a greater chance of success at delivering a house that meets your actual needs.

Now assume that you're having software built, something that is several orders of magnitude more complex and often more expensive than building a house and assume once again you have two contractors that want to take the exact same approach. Which contractor are you more comfortable with? I hope the answer is still the second one, the one with a sensible process. Unfortunately, practice shows that the vast majority of the time organizations appear to choose the approach of the first contractor, that of hacking. Of course, practice also shows that we have roughly an 85% failure rate, do you think the two phenomena are related? I think so.

To make matters worse, things are getting more complicated within the software industry. Now that the Year 2000 (Y2K) crisis is past us, for the most part, organizations are now dealing with the realities of what we at Ronin International like to call the Post-2000 (P2K) world. In this brave new world you will:

• Develop e-commerce applications at internet-speed using object-oriented and component-based technologies such as CORBA, DCOM, and Enterprise JavaBeans (EJB).
• Leverage the ability of existing and newly hired staff, particularly important in an environment where there is an increased shortage of highly-skilled labor
• Leverage legacy software using enterprise application integration (EAI) techniques
• Model your environment and software using the Unified Modeling Language (UML) notation
• Apply common architectural, analysis, design, and process patterns
• Follow an incremental and iterative development approach

2. What is a Software Process?

What is a software process? A software process is a set of project phases, stages, methods, techniques, and practices that people employ to develop and maintain software and its associated artifacts (plans, documents, models, code, test cases, manuals, etc.). Furthermore, not only do you need a software process, you need one that is proven to work in practice, a software process tailored to meet your exact needs. In
this white paper I present the three leading processes – The Unified Process, The OPEN Process, and The Object-Oriented Software Process (OOSP) – comparing and contrasting them and arguing that you need to take the best of breed from each to be successful.

3. Why Do You Need A Software Process?

An effective software process will enable your organization to increase its productivity when developing software for several reasons. First, by understanding the fundamentals of how software is developed you can make intelligent decisions, such as knowing to stay away from SnakeOil v2.0 the wonder tool that claims it automates fundamental portions of the software process. Second, it enables you to standardize your efforts, promoting reuse and consistency between project teams. Third, it provides an opportunity for you to introduce industry best practices such as code inspections, configuration management, change control, and architectural modeling to your development organization.

An effective software process will also improve your organization’s maintenance and support efforts in several ways. First, it should define how to manage change and appropriately allocate maintenance changes to future releases of your software, streamlining your change process. Second, it should define first how to smoothly transition software into operations and support and then how the operations and support efforts are actually performed. Without effective operations and support processes your software will quickly become shelfware.

Why adopt an existing software process, or improve your existing process using new techniques? The reality is that software is growing more and more complex, and without an effective way to develop and maintain that software the chances of you succeeding will only get worse. Not only is software getting more complex, we're also being asked to create more software simultaneously. Most organizations have several software projects currently in development and have many times that in production, projects that need to be managed effectively. Furthermore, our industry is in crisis, we're still reeling from the simple transition from using a two-digit year to a four-digit year, a "minor" problem with an estimated price tag of $600 Billion worldwide. The nature of the software that we're building is also changing, from the simple batch systems of the 1970s that structured techniques are geared towards to the interactive, international, user-friendly, 7/24, high-transaction, high-availability online systems that object-oriented and component-based techniques are aimed at. And while you're doing that you are asked to increase the quality of the systems that you're delivering, and to reuse as much as possible so that you can work faster and cheaper. A tall order, one that is nearly impossible to fill if you can't organize and manage your staff effectively, and a software process provides the basis to do just that.
4. The Requirements for a Software Process

Just as you start a software project by defining its requirements, you should also start discussing a software process by defining its requirements. Luckily this is already done for you in the form of the Capability Maturity Modeling (CMM). The Software Engineering Institute (SEI) at Carnegie Mellon University has proposed the CMM, a framework from which a process for large, complex software efforts can be defined. The CMM defines five maturity levels, described in Table 1, evolutionary plateaus toward achieving a mature software process, that an organization can attain with respect to the software process. According to the CMM, to achieve a specific maturity level an organization must satisfy and institutionalize all of the key process areas (KPAs), described in Table 3, for that level and for the previous levels. Sponsored by the United States Department of Defense (DoD), the CMM has been adopted by hundreds of organizations worldwide that aim to improve the way that they develop software.
<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
<th>Characteristics</th>
</tr>
</thead>
</table>
| 1. Initial | The software process is ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics. | • Over-commitment is common.  
• During a crisis, planned procedures are abandoned and project teams revert to coding and testing.  
• Success depends on having an exceptional manager and a seasoned & effective team.  
• The software process is effectively a black box to the user community. Resources go in and software potentially comes out. |
| 2. Repeatable | Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications. | • The planning and management of new projects is based on experience with similar projects.  
• Process capability is enhanced at the project level by establishing basic process management techniques.  
• Software requirements and deliverables are baselined.  
• Processes often differ between projects, reducing opportunities for teamwork and reuse.  
• The user community is provided visibility into the project at defined occasions, typically the review and acceptance of major project deliverables, allowing limited mgmt control. |
| 3. Defined | The software process for management and development activities is documented, standardized, and integrated into a standard software process for your organization. All projects use an approved, tailored version of the process. | • A standard process is used, with possible tailoring, on all projects.  
• Management has good insight into technical progress on the project.  
• Defined processes allow the user community greater visibility into the project and enable accurate and rapid status updates. |
| 4. Managed | Detailed measures, called metrics, of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled. | • Productivity and quality are measured for important software process activities across all projects.  
• The user community can establish an accurate, quantitative understanding of the software process capability of your organization/team and the project risk before the project begins. |
| 5. Optimized | Continuous process improvement is enabled by quantitative feedback from the software process and from piloting innovative ideas and technologies. | • Innovations that exploit the best software engineering practices are identified and shared throughout the organization.  
• The software process is improved by changing common causes of inefficiency.  
• Disciplined change is the norm, not the exception.  
• The user community and the software organization work together to establish a strong and successful relationship. |

Table 1. The five CMM maturity levels.
<table>
<thead>
<tr>
<th>Key Process Area</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Defect prevention.</strong> Analyze defects that were</td>
<td>5</td>
</tr>
<tr>
<td>encountered in the past and taking specific actions</td>
<td></td>
</tr>
<tr>
<td>to prevent the occurrence of those types of defects</td>
<td></td>
</tr>
<tr>
<td>in the future.</td>
<td></td>
</tr>
<tr>
<td><strong>Integrated software management.</strong> Integrate both</td>
<td>3</td>
</tr>
<tr>
<td>software engineering and management activities</td>
<td></td>
</tr>
<tr>
<td>into a coherent, defined process for each project</td>
<td></td>
</tr>
<tr>
<td>that is tailored from the standard software process</td>
<td></td>
</tr>
<tr>
<td>of your organization. This involves developing the</td>
<td></td>
</tr>
<tr>
<td>project’s defined software process and managing the</td>
<td></td>
</tr>
<tr>
<td>software project based on this process.</td>
<td></td>
</tr>
<tr>
<td><strong>Intergroup coordination.</strong> Participate with other</td>
<td>3</td>
</tr>
<tr>
<td>teams and groups throughout your organization to</td>
<td></td>
</tr>
<tr>
<td>address the requirements, objectives, and issues</td>
<td></td>
</tr>
<tr>
<td>that are applicable to your entire organization.</td>
<td></td>
</tr>
<tr>
<td><strong>Organization process definition.</strong> Develop and</td>
<td>3</td>
</tr>
<tr>
<td>maintain your organization’s standard software</td>
<td></td>
</tr>
<tr>
<td>process, along with related process assets such as</td>
<td></td>
</tr>
<tr>
<td>descriptions of software life cycles, process</td>
<td></td>
</tr>
<tr>
<td>tailoring guidelines and criteria, your organization’</td>
<td></td>
</tr>
<tr>
<td>s software process database, and a library of</td>
<td></td>
</tr>
<tr>
<td>software process-related documentation.</td>
<td></td>
</tr>
<tr>
<td><strong>Organization process focus.</strong> Develop and maintain</td>
<td>3</td>
</tr>
<tr>
<td>an understanding of your organization’s and project’s</td>
<td></td>
</tr>
<tr>
<td>software processes and coordinating the activities</td>
<td></td>
</tr>
<tr>
<td>to assess, develop, maintain, and improve these</td>
<td></td>
</tr>
<tr>
<td>processes. This responsibility should be assigned to</td>
<td></td>
</tr>
<tr>
<td>a permanent team within your organization.</td>
<td></td>
</tr>
<tr>
<td><strong>Peer reviews.</strong> Methodical examination of project</td>
<td>3</td>
</tr>
<tr>
<td>deliverables by the developer’s peers to identify</td>
<td></td>
</tr>
<tr>
<td>potential defects and areas where changes are needed.</td>
<td></td>
</tr>
<tr>
<td><strong>Process change management.</strong> Define process</td>
<td>5</td>
</tr>
<tr>
<td>improvement goals and proactively and systematically</td>
<td></td>
</tr>
<tr>
<td>identifying, evaluating, and implementing</td>
<td></td>
</tr>
<tr>
<td>improvements to your organization’s software</td>
<td></td>
</tr>
<tr>
<td>process on a continuous basis.</td>
<td></td>
</tr>
<tr>
<td><strong>Quantitative process management.</strong> Establish goals</td>
<td>4</td>
</tr>
<tr>
<td>for and then measure the performance of a project’s</td>
<td></td>
</tr>
<tr>
<td>defined software process. It is critical that the</td>
<td></td>
</tr>
<tr>
<td>performance of individuals are measured, to aid in</td>
<td></td>
</tr>
<tr>
<td>their professional development, but that the</td>
<td></td>
</tr>
<tr>
<td>information not be used to their detriment.</td>
<td></td>
</tr>
<tr>
<td><strong>Requirements Management.</strong> Establish, document,</td>
<td>2</td>
</tr>
<tr>
<td>and maintain an agreement with the user community</td>
<td></td>
</tr>
<tr>
<td>regarding the requirements for the project.</td>
<td></td>
</tr>
<tr>
<td><strong>Software Configuration Management.</strong> Establish and</td>
<td>2</td>
</tr>
<tr>
<td>maintain the integrity of the project deliverables</td>
<td></td>
</tr>
<tr>
<td>throughout the entire life cycle.</td>
<td></td>
</tr>
<tr>
<td><strong>Software product engineering.</strong> Perform the</td>
<td>3</td>
</tr>
<tr>
<td>engineering tasks to build and maintain the software</td>
<td></td>
</tr>
<tr>
<td>in accordance with the project’s defined software</td>
<td></td>
</tr>
<tr>
<td>process and appropriate methods and tools.</td>
<td></td>
</tr>
<tr>
<td><strong>Software Project Planning.</strong> Develop and</td>
<td>2</td>
</tr>
<tr>
<td>negotiate estimates for the work to be performed,</td>
<td></td>
</tr>
<tr>
<td>establish the necessary commitments, and define the</td>
<td></td>
</tr>
<tr>
<td>plan to perform the work.</td>
<td></td>
</tr>
<tr>
<td><strong>Software Project Tracking And Oversight.</strong> Provide</td>
<td>2</td>
</tr>
<tr>
<td>visibility into the actual progress of a project so</td>
<td></td>
</tr>
<tr>
<td>that management can take effective and appropriate</td>
<td></td>
</tr>
<tr>
<td>action.</td>
<td></td>
</tr>
<tr>
<td><strong>Software Quality Assurance.</strong> Review and audit the</td>
<td>2</td>
</tr>
<tr>
<td>project deliverables and activities to verify that</td>
<td></td>
</tr>
<tr>
<td>they comply with your adopted applicable standards,</td>
<td></td>
</tr>
<tr>
<td>guidelines, and processes.</td>
<td></td>
</tr>
<tr>
<td><strong>Software quality management.</strong> Define quality</td>
<td>4</td>
</tr>
<tr>
<td>goals for software products and the establish plans</td>
<td></td>
</tr>
<tr>
<td>to achieve these goals. Monitoring and adjust</td>
<td></td>
</tr>
<tr>
<td>software plans, software work products, activities,</td>
<td></td>
</tr>
<tr>
<td>and quality goals to satisfy the needs of the user</td>
<td></td>
</tr>
<tr>
<td>community for high-quality products.</td>
<td></td>
</tr>
<tr>
<td><strong>Software Subcontract Management.</strong> Select and</td>
<td>2</td>
</tr>
<tr>
<td>effectively manage qualified software subcontractors.</td>
<td></td>
</tr>
<tr>
<td><strong>Technology change management.</strong> Identify, select,</td>
<td>5</td>
</tr>
<tr>
<td>and evaluate new technologies and incorporating</td>
<td></td>
</tr>
<tr>
<td>effective technologies into the organization.</td>
<td></td>
</tr>
<tr>
<td><strong>Training program.</strong> Identify the training needed</td>
<td>3</td>
</tr>
<tr>
<td>by your organization, projects, and individuals,</td>
<td></td>
</tr>
<tr>
<td>and then develop or procure training to address the</td>
<td></td>
</tr>
<tr>
<td>identified needs.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. The Key Process Areas.

One strength of the CMM, at least from a management point of view, is that it distinguishes between immature and mature software organizations. Immature software organizations are typically reactionary and
have little understanding as to how to successfully develop software. Mature software organizations, on the
other hand, understand the software process, which enables them to judge the quality of the software
products and the process that produces them. Mature software organizations have a higher success rate
and a lower overall cost of software across the entire life of a software product than do immature software
organizations.

Why should your organization strive to increase its software maturity? The answer is two-fold: first, the
greater the software maturity of your organization, the greater the quality of the software products that it
produces. This is a direct result of the increased focus on management and project deliverables as the
maturity of your organization increases. Second, as the software maturity of your organization grows, there
is a corresponding reduction of risk on your software development projects, the result of increased
management control and use of measurements (metrics) to understand and manage the software process
and the products that you produce.

Although the SEI CMM focuses on software development, it also touches on production issues that are an
important aspect of the software process. This is important because effective software professionals take
into account the needs of maintenance and support when they are developing software, recognizing that
development is merely one small portion of the overall lifecycle of the systems that they create. Roughly
20% of your organization's software budget is allocated for new development whereas 80% is generally
allocated to maintenance and support efforts. Granted, with the predominance of the Y2K crisis it is likely
that many organizations have allocated far more that 80% of their budget to maintenance in recent years, but
that's an artificial blip that should be discounted. Regardless, recognizing that maintenance and support are
a significant concern to most organizations, the scope of this paper will be the entire software process and
not merely the development process.

To better put the software process into perspective, Figure 1 depicts the scope of the various approaches to
process, showing that the development process is a subset of the software process which in turn is a subset
of the enterprise processes of your organization. The figure also indicates the key factors that influence
your process, including your organization's culture, your architecture, the tools that you use, the standards
that you follow, the legislation that is appropriate to your firm, and the external processes of the
organizations that yours interacts with (such as your customers). The points to be made are that while your
organization's software process is an important and complex thing, it is only part of the overall picture and
that it can be influenced by factors that are not under your complete control.
5. The Leading Processes

Now let's discuss the three leading processes on the market:

- The Unified Process
- The OPEN Process
- The Object-Oriented Software Process (OOSP)

5.1 The Unified Process

The Unified Process (Kruchten, 1999) is the latest endeavor of Rational Corporation, the same people who introduced what has become the industry-standard modeling notation, the Unified Modeling Language (UML). The heart of the Unified Process is the Objectory Process, one of several products and services that Rational acquired when they merged with Ivar Jacobson’s Objectory organization several years ago. Rational enhanced Objectory with their own processes, and those of other tool companies that they have either purchased or partnered with, to form the initial version (5.0) of the Unified Process officially released in December of 1998.

Figure 2 presents the lifecycle of the Unified Process, made up of four serial phases and nine core workflows. Along the bottom of the diagram you see that any given development cycle through the Unified Process should be organized into what Rational calls iterations. Although I would argue that the term increments is likely a better term than iterations, the basic concept is that at the end of each iteration you produce an internal executable that can be worked with by your user community. This reduces the risk of your project by improving communication between you and your customers. Another risk reduction technique built into the Unified Process is the concept that you should make a go/no-go decision at the end of each phase – if a project is going to fail then you want to stop it as early as possible in its lifecycle. This is an important concept in an industry with an 85% failure rate.

---

1. Other processes such as Extreme Programming (XP), Catalysis, and Dynamic System Development Method (DSDM) are also popular but were not considered for discussion in this paper.
The Inception phase is where you define the project scope and define the business case for the system. The initial use cases for your software are identified and the key ones are described briefly. Use cases are the industry standard technique for defining the functional requirements for systems, providing significant productivity improvements over traditional requirement documents because they focus on what adds value to users as opposed to product features. Basic project management documents are started during the Inception phase, including the initial risk assessment, the estimate, and the project schedule. As you would expect, key tasks during this phase include business modeling and requirements engineering, as well as the initial definition of your environment including tool selection and process tailoring.

The Elaboration phase focuses on detailed analysis of the problem domain and the definition of an architectural foundation for your project. Because use cases aren’t sufficient for defining all requirements (Ambler, 1998b) a deliverable called a Supplementary Specification is defined which describes all non-functional requirements for your system. A detailed project plan for the Construction Phase is also developed during this phase based on the initial management documents started in the Inception phase.

The Construction phase, often the heart of software projects (typically to their detriment) is where the detailed design for your application is developed as well as the corresponding source code. I say that projects focus on this phase to their detriment because organizations often do not invest sufficient resources in the previous two phases and therefore lack the foundation from which to successfully develop software that meets the needs of their users. The goal of this phase is to produce the software and supporting documentation to be transitioned to your user base.

The purpose of the Transition phase is to deliver the system to your user community. There is often a beta release of the software to your users, typically called a pilot release within most businesses, in which a small group of users work with the system before it is released to the general community. Major defects are identified and potentially acted on during this phase. Finally, an assessment is made regarding the success of your efforts to determine whether another development cycle/increment is needed to further enhance the system.
The Unified Process has several strengths. First, it is based on sound software engineering principles such as taking an iterative, requirements-driven, and architecture-based approach to development. Second, it provides several mechanisms, such as a working prototype at the end of each iteration and the go/no-go decision point at the end of each phase, which provides management visibility into the development process. Third, Rational has made, and continues to do so, a significant investment in their Rational Unified Process product, an HTML-based description of the Unified Process that your organization can tailor to meet its exact needs. At the time of this writing the current version is 5.1.1 and version 5.5 should be available soon.

The Unified Process also suffers from several weaknesses. First, it is only a development process. The current version of the Unified Process does not cover the entire software process, as you can see in Figure 2 it is very obviously missing the concept of operations and support. Second, the Unified Process does not explicitly support multi-project infrastructure development efforts such as organization-wide architectural modeling, missing opportunities for large-scale reuse within your organization. Third, the iterative nature of the lifecycle is foreign to many experienced developers, making acceptance of it more difficult, and the rendering of the lifecycle in Figure 2 certainly doesn’t help this issue. The iterative nature of the Unified Process is both a strength and a weakness. Finally, because Rational’s approach to developing this process was initially tools-driven the resulting process is likely not yet sufficient for the complex needs of modern developers. You can’t automate everything and even if you could Rational doesn’t have a complete tool suite anyway. For example, I have yet to hear about a Rational help desk product, a Rational user interface design tool, or a Rational data modeling tool. Because Rational is a marketing-driven company there is little motivation to include aspects of the software process that they don’t sell tools for.

5.2 The OPEN Process

The OPEN consortium, a group of individuals and organizations promoting and enhancing the use of object-oriented technology, has developed the OPEN Process, a comprehensive software process. Like the Unified Process, the OPEN Process is aimed at organizations using object and component technology, although it can easily be applied to other software development technologies. Also similar to the Unified Process, OPEN was initially created by the merger of earlier methods: MOSES, SOMA, Firesmith, Synthesis, BON, and OOram. The OPEN Process supports the UML notation, and the Object Modeling Language (OML) notation, and any other OO notation to document the work products the OPEN process produces. The shape of the lines and bubbles may change, but the fundamentals of the software process remain the same.

The contract-driven life cycle of the OPEN process is depicted in Figure 3, taking a more traditional “bubbles and lines” approach as compared to the “humpback” lifecycle of Figure 2. Each OPEN activity is shown as a rectangle, similar conceptually to workflows in the Unified Process, with unbounded activities shown as rounded rectangles and tightly bound activities as simple rectangles. The activities of the OPEN are subject to stated contracts, making OPEN a responsibility-driven process. Activities describe the overall process architecture; these are complemented by smaller scale tasks which are project management focussed. Both activities and tasks describe what is to be done. How it is to be done is described by a set of techniques.
Figure 3. The OPEN Contract-Driven lifecycle.

The left-hand side of the Figure 3 represents the activities for a single project, whereas the activities in the right-hand side represent cross-project activities. This is a major departure from what you've seen in the Unified Process: the lifecycle of the OPEN Process explicitly includes activities outside of the scope of a single project. This is called programme management in the OPEN Process, a programme being a collection of projects and/or releases of an application or suite of applications. Common programme management activities would include organization-wide architectural modeling efforts, process improvement efforts, and standards/guidelines development and support. This sort of effort is often called Enterprise Management or Infrastructure Management within software organizations.

OPEN can be tailored to different projects, organizations and domains. The selection of appropriate tasks and techniques for your specific project is part of the process of tailoring OPEN to your specific requirements something that happens as part of the Project Initiation activity. It has strong support for process modeling, project management and requirements capture as well as traditional OO "analysis and design".

The OPEN Process benefits from several strengths. First, at the time of this writing it is the most comprehensive of the three processes presented in this paper, including a cradle-to-grave approach to the lifecycle of a project and a multi-project view to software that reflects the actual environment of most
organizations. Second, it is the brainchild of a wide variety of practitioners and academics, all of whom are coming to the table with different experiences, skillsets, and backgrounds. This breadth of talent is one reason for the OPEN Processes comprehensiveness and has resulted in a smorgasbord of development techniques from which to choose. Third, it isn't constrained by what can be automated or what can be easily marketed to an existing customer base: the OPEN Consortium is free to call it like they see it.

The OPEN Process does suffer from one serious drawback: ineffective marketing. Remember the ill-fated Object Modeling Language (OML) from the OPEN Consortium, the notation that ran against the Unified Modeling Language (UML) several years ago in the race to become the industry standard notation? The OML was superior in many respects to the UML, but it unfortunately didn't garner the market mindshare that the UML did. My fear is that once again the best candidate will be out marketed by the second best, which more often than not is the norm in our industry. Time will tell.

5.3 The Object-Oriented Software Process (OOSP)

Figure 4 depicts the lifecycle of the Object-Oriented Software Process (OOSP), comprised of a collection of process patterns. A process pattern is a collection of general techniques, actions, and/or tasks (activities) that solve a specific software process problem taking the relevant forces/factors into account. Just like design patterns describe proven solutions to common software design problems, process patterns present proven solutions to common software process problems. Process patterns were originally proposed by James Coplien in his paper "A Generative Development-Process Pattern Language" in the first Pattern Languages of Program Design book (Addison-Wesley, 1995). I believe that there are three scales of process patterns, phase process patterns, stage process patterns, and task process patterns. A phase process pattern depicts the interactions between the stage process patterns for a single project phase, such as the Initiate and Deliver phases of the OOSP. A stage process pattern depicts the tasks, which are often performed iteratively, of a single project stage such as the Program the Model stage. Finally, task process patterns address lower-level process issues, such as the Reuse First process pattern that describes how to achieve significant levels of reuse within your organization and the Technical Review process pattern which describes how to organize reviews and inspections. The three scales of process patterns are conceptually similar to the scale of other types of patterns: in the modeling world you have architectural patterns, design patterns, and programming idioms.
An important feature of a process pattern is that it describes what you should do but not the exact details of how you should do something. Process patterns are an excellent mechanism for communicating approaches to software development that have proven to be effective in practice. When applied together in an organized manner process patterns can be used to construct a software process for your organization, as you see with the OOSP lifecycle of Figure 4. Because process patterns do not specify the exact details of how to perform a given task they can be used reusable building blocks from which you may tailor a software process that meets the specific needs of your organization.

In Figure 4 you see that there are four project phases within the OOSP – Initiate, Construct, Deliver, and Maintain and Support – each of which is described by a corresponding phase process pattern. There are also fourteen project stages in the OOSP – Justify, Define and Validate Initial Requirements, Define Initial Management Documents, Define Infrastructure, Model, Program, Test In The Small, Generalize, Test In The Large, Rework, Release, Assess, Support, and Identify Defects and Enhancements – each of which is described by a stage process pattern. Project stages are performed in an iterative manner within the scope of a single project phase. Project phases, on the other hand, are performed in a serial manner within the OOSP. This is conceptually very similar to what you see in the Unified Process of Figure 2, which is comprised of four serial phases and nine workflows that are performed in an iterative manner.

As you can see, there is more to the OOSP than just its phases and stages. The “big arrow” at the bottom of the diagram indicates important tasks critical to the success of a project that are applicable to all stages of development. These tasks include quality assurance, project management, training and education, people management, risk management, reuse management, metrics management, and deliverables management. The important thing to note is that several of these tasks are applicable at both the project and cross-project (programme) levels. For example, risk management should be performed for both a single project and for your portfolio of software projects. It’s no good if each project is relatively risk free but as a collection they become quite risky. Deliverables management, which includes configuration management and change control functions, is also applicable for both a single project and for a collection of projects because the collection of projects may share a single artifact, such as common source code. Infrastructure management – where you evolve your corporate processes, standards, guidelines, and architectures – is obviously applicable across several projects but will also apply to a single project because that project team will need guidance and support to use your organization’s software infrastructure.
The process patterns of the OOSP reflect the experiences of a large number of object practitioners. These process patterns reflect the lessons learned on a multitude of small, medium, and large projects in a wide range of industries, including telecommunications, finance, internet, system outsourcing, government, military, and manufacturing. Like the OPEN Process, the OOSP is the fusion of real-world experience with leading software engineering theories.

There are several strengths to the OOSP. First, it is a comprehensive approach to the software process, taking the entire lifecycle into account. Second, it honestly portrays the fact that large-scale, mission-critical software is actually serial in the large and iterative in the small – contrary to popular belief, real-world development isn’t purely iterative. Third, like the OPEN Process it explicitly supports programme/infrastructure processes, enabling large-scale reuse and effective management of your portfolio of software projects. Fourth, significant work is being done in process patterns and organizational patterns, with a large body of knowledge being posted online almost daily (http://www.ambysoft.com/processPatternsPage.html for links). Fifth, the OOSP lifecycle explicitly includes a maintain and support phase to constantly remind developers that they need to take maintenance and support considerations into account as they create software. Finally, the OOSP explicitly includes things that have a tendency to fall through the cracks, such as the need to generalize your work if you intend to make it reusable, the need to perform risk management, the need to perform quality assurance, and the need to assess your project and project team once your software has been delivered.

There are also several weaknesses. First, because the OOSP is comprehensive it can be overwhelming to most software professionals that prefer to focus on their small part of the software process. Second, like the OPEN Process the OOSP also suffers from similar marketing challenges.

5.4 Comparing the Leading Processes

The easiest way to compare the three process is to put them up against the requirements for the software process and see how they fare. Earlier I stated that the Capability Maturity Model (CMM) effectively forms the requirements for a development process, so let’s start there. Table 3 lists the key process areas (KPAs) of the CMM (described above in Table 2) and then indicates which workflows, stages, activities, and so on from each of the above processes that support the KPA. The “(I)” notation indicates that support for the KPA is implied in the definition of the stage/workflow or activity, which for the most part means that the general idea is there although details need to be fleshed out some more.

The OOSP supports the CMM KPAs the most thoroughly, which should be expected because the CMM was in fact used as the requirements model for that process. The OPEN Process also fairs well because of its strong ties to the CMM and the process community in general. You see that the Unified Process appears lacking in several areas, in most part, I suspect, due to the fact that it was aggregated from the processes of several tool vendors instead of being developed from a set of industry-recognized standards.
<table>
<thead>
<tr>
<th>Key Process Area</th>
<th>Unified Process</th>
<th>OOSP</th>
<th>OPEN Process</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Defect prevention</strong></td>
<td>• Test</td>
<td>• Define and Validate Initial Requirements</td>
<td>• Evaluate Quality task</td>
</tr>
<tr>
<td></td>
<td>• Implementation</td>
<td>• Test in the Small</td>
<td>• Analyze Metrics subtask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Test in the Large</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Quality Assurance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Metrics Management</td>
<td></td>
</tr>
<tr>
<td><strong>Integrated software management.</strong></td>
<td>• Environment</td>
<td>• Define Infrastructure</td>
<td>• Tailor Lifecycle Process subtask</td>
</tr>
<tr>
<td></td>
<td>• Project management</td>
<td>• Project Management</td>
<td>• Project Initiation activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Programme Planning activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Resource Planning activity</td>
</tr>
<tr>
<td><strong>Intergroup coordination</strong></td>
<td>• Project management (I)</td>
<td>• Initiate phase</td>
<td>• Project Initiation activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Project Management</td>
<td>• Programme Planning activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Resource Planning activity</td>
</tr>
<tr>
<td><strong>Organization process definition</strong></td>
<td>• Environment workflow</td>
<td>• Process patterns</td>
<td>• Tailor Lifecycle Process subtask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Define Infrastructure</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Infrastructure Management</td>
<td></td>
</tr>
<tr>
<td><strong>Organization process focus</strong></td>
<td>• Environment</td>
<td>• Infrastructure Management</td>
<td>• Tailor Lifecycle Process subtask</td>
</tr>
<tr>
<td></td>
<td>• Phase assessments</td>
<td>• Assess stage</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Peer reviews</strong></td>
<td>• Business Modeling</td>
<td>• Define and Validate Initial Requirements</td>
<td>• Review Process task</td>
</tr>
<tr>
<td></td>
<td>• Requirements</td>
<td>• Test in the Small</td>
<td>• Inspections task</td>
</tr>
<tr>
<td></td>
<td>• Analysis &amp; Design</td>
<td>• Test in the Large</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Implementation</td>
<td>• Quality Assurance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Test</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Process change management</strong></td>
<td>• Environment (I)</td>
<td>• Define Infrastructure</td>
<td>• Review Process task (I)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Initiate</td>
<td>• Post-Implementation Review subtask (I)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Construct</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Test in the large</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Assess</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Support</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Quality assurance</td>
<td></td>
</tr>
<tr>
<td><strong>Quantitative process management</strong></td>
<td>• Project management (I)</td>
<td>• Assess</td>
<td>• Post-implementation review (I)</td>
</tr>
<tr>
<td></td>
<td>• End of phase assessments</td>
<td>• Infrastructure Management</td>
<td>• Resource planning (I)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Develop Education and Training Plan task</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Specify Individual Goals task</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Analyze Metrics Data subtask</td>
</tr>
<tr>
<td><strong>Requirements Management</strong></td>
<td>• Requirements</td>
<td>• Define and Validate Initial Requirements</td>
<td>• Requirements Engineering activity</td>
</tr>
<tr>
<td></td>
<td>• C&amp;CM</td>
<td>• Rework</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Identify Defects and Enhancements</td>
<td></td>
</tr>
<tr>
<td><strong>Software Configuration</strong></td>
<td>• C&amp;CM</td>
<td>• Deliverables Management</td>
<td>• Configuration Management task</td>
</tr>
</tbody>
</table>
Table 3. Mapping the CMM key process areas (KPAs) to the Leading Processes.

Table 4 takes another approach, comparing each process against the significant features of the other two. Once again the OOSP and OPEN Process fairs better than the Unified Process, for the most part because their scope is that of the entire software process and not just that of the development process. Also, the OOSP and OPEN Process explicitly include support for programme/infrastructure activities whereas the focus of the Unified Process is that of a single project. The Unified Process, however, fairs better in its explicit support for iterative development.
<table>
<thead>
<tr>
<th>Process Feature</th>
<th>Unified Process</th>
<th>OOSP</th>
<th>OPEN Process</th>
</tr>
</thead>
</table>
| **Architecture-driven development.** Does the process include architectural modeling and architecture-related activities? | • Elaboration phase  
• Analysis & Design workflow | • Model stage  
• Infrastructure Management task | • Build activity |
| **Domain Modeling.** Does the process include the concept of modeling the business? | • Business Modeling workflow | • Model Stage | • Domain Modeling activity |
| **Environment definition.** Does the process include explicit activities to define/select the tools, processes, standards, and guidelines that it will use and/or follow? | • Environment workflow | • Define Infrastructure stage | • Project Initiation activity |
| **Implementation Planning.** Does the process include the concept of planning for the deployment of your software? | • Deployment workflow | • Project Management | • Implementation Planning activity |
| **Interim checkpoints.** Does the process explicitly include go/no-go checkpoints? | • End of Phase assessments | • Project Management (weak) | • User Review activity |
| **Iterative development.** Does the lifecycle include explicit support for iterative development. | • Yes. | • Yes. | • Yes. |
| **Production.** Does the process explicitly include the concept of a maintenance phase? | • No. Currently out of scope. | • Maintain and Support Phase | • Use of System activity |
| **Metrics management.** Does the process include processes for defining, collecting, and acting on metrics? | • Project Management workflow (weak but being improved) | • Metrics Management task | • Analyze Data subtask  
• Setup Metrics Collection Program stubtask |
| **Operations/Support Phase.** Does the process explicitly include the concept of a operations and support phase? | • No. Currently out of scope. | • Maintain and Support Phase | • Use of System activity |
Programme/Infrastructure management. Does the process explicitly include the concept of multi-project management and support?

- Barely. Hints at the concept with architecture focus but the management mechanisms are not in place.

- Infrastructure Management task
- Reuse Management task
- Risk Management task
- Define Infrastructure stage
- Assess stage
- Model stage

- Programme Management activities

Reuse management and planning. Does the software process include explicit support for reuse management?

- Project Management workflow (weak)
- Reuse Management task
- Generalize stage
- Construct phase

- Domain Modeling activity
- Optimize Reuse task
- Create New Reusable Components task
- Manage Library of Reusable Components task

Use-case driven development. Does the process support use-cases in a meaningful way?

- Requirements workflow
- Analysis & Design workflow
- Testing workflow
- Define and Validate Initial Requirements stage
- Model stage
- Test in the Large stage

- Requirements Elicitation activity
- Evaluation activity

Table 4. Comparing the Processes on Specific Process Features.

You’ve seen how they compare, but which should you choose? I believe the answer is the Unified Process. Although I definitely have skin in the game with the Object-Oriented Software Process (OOSP), I have to recognize the cold, hard fact that for better or worse the Unified Process is the market leader and will likely remain so despite its flaws. Unfortunately the Unified Process isn’t perfect, none of them are, but it’s the one that we need to make work as an industry if we want organizations to be successful at software.

6. Enhancing the Unified Process

So how do you enhance the Unified Process? Well, the first place to start is to expand the scope of the Unified Process to include the entire software process, not just the development process. This implies that processes for operations, support, and maintenance efforts need to be added to the Unified Process. Second, to be sufficient for today’s organizations the Unified Process also needs to add support for the management of a portfolio of projects, something other processes have called programme, multi-project, or infrastructure management. These first two steps result in an enhanced version of the Unified Process lifecycle (Ambler 2000a; Ambler 2000b; Ambler, 2000c), presented in Figure 5. Finally, the Unified Process needs to be fleshed out with material from both the OPEN Process and the OOSP, as indicated in Tables 3 and 4, to fill the remaining gaps.
The enhanced lifecycle for the Unified Process includes a fifth phase, Production, representing the portion of the software lifecycle after a system has been deployed. As the name of the phase implies, its purpose is to keep your software in production until it is either replaced with a new version, from a minor release such as a bug fix to a major new release, or it is retired and removed from production. Note that there are no iterations during this phase, or there is only one iteration depending on how you wish to look at it, because this phase applies to the lifetime of a single release of your software. To develop and deploy a new release of your software you need to run through the four development phases again.

Figure 5 also shows that there are two new workflows, a core workflow called Operations & Support and a supporting workflow called Infrastructure Management. The purpose of the Operations & Support workflow is exactly as the name implies, to operate and support your software. Operations and support are both complex endeavors, endeavors which need processes defined for them. This workflow spans several phases, as do all the others. During the Construction phase, and perhaps as early as the Elaboration phase, you will need to develop operations and support plans, documents, and training manuals. During the Transition phase you will continue to develop these artifacts, reworking them based on the results of testing and you will train your operations and support staff to effectively work with your software. Finally, during the Production phase your operations staff will keep your software running, performing necessary backups and batch jobs as needed, and your support staff will work with your user community in working with your software. This workflow basically encompasses portions of the OOSP’s Release stage and Support stage as well as the OPEN Process’s Implementation Planning activity and Use of System activity. In the Internet economy where you do 7/24 operations you quickly discover that high-quality and high-availability are crucial to success – you need an Operations and Support workflow.

The Infrastructure Management workflow focuses on the activities required to develop, evolve, and support your organization’s infrastructure artifacts such as your organization-wide models, your software processes, standards, guidelines, and your reusable artifacts. Your software portfolio management efforts are also performed in this workflow. Infrastructure Management occurs during all phases, the blip during the Elaboration phase represents architectural support efforts to ensure that a project’s architecture...
appropriately reflects your organization’s overall architecture. In *Process Patterns* (Ambler, 1998b) I present a modeling process which includes infrastructure modeling activities such as the development of an enterprise requirements/business model, a domain architecture model, and a technical architecture model. These three core models form your infrastructure models that describe your organization’s long-term software goals and shared/reusable infrastructure.

The processes followed by your Software Engineering Process Group (SEPG) – which is responsible for supporting and evolving your software processes, standards, and guidelines – are also included in this workflow. Your reuse processes are included too. Practice shows that to be effective reuse management is a cross-project endeavor. For you to achieve economies of scale developing software, to increase the consistency and quality of the software that you develop, and to increase reuse between projects, you need to manage your common infrastructure effectively – you need the Infrastructure Management workflow.

Comparing the enhanced lifecycle for the Unified Process of Figure 5 with the initial lifecycle for the Unified Process of Figure 2 you will notice that several of the existing workflows have also been updated. First, the Test workflow has been expanded to include activity during the Inception phase. You develop your initial, high-level requirements during this phase, requirements that you can validate using techniques such as walkthroughs, inspections, and scenario testing. Two of the underlying philosophies of the OOSP are that you should test often and early and that if something is worth developing then it is worth testing. Therefore testing should be moved forward in the lifecycle. Also, the Test workflow needs to be enhanced with the techniques of the OOSP’s *Test In The Small* (Ambler, 1998b) and *Test In The Large* stages (Ambler, 1999).

The second modification is to the Deployment workflow, extending it into the Inception and Elaboration phases. This modification reflects the fact that deployment, at least of business applications, is a daunting task. Data conversion efforts of legacy data sources are often a project in their own right, a task that requires significant planning, analysis, and considerable effort to accomplish. Furthermore, my belief is that deployment modeling should be part of the Deployment workflow, and not the Analysis & Design workflow as it currently is, due to the fact that deployment modeling and deployment planning go hand in hand. Deployment planning can and should start as early as the Inception phase and continue into the Elaboration and Construction phases in parallel with deployment modeling.

The Environment workflow has been updated to include the work necessary to define the Production environment, work that would typically occur during the Transition phase. The existing Environment workflow processes effectively remain the same, the only difference being that they now need to expand their scope from being focused simply on a development environment to also include operations and support environments. Your operations and support staff need their own processes, standards, guidelines, and tools, the same as your developers. Therefore you may have some tailoring, developing, or purchasing to perform to reflect this need.

The Configuration & Change Management workflow is extended into the new Production phase to include the change control processes needed to assess the impact of a proposed change to your deployed software and to allocate that change to a future release of your system. This change control process is often more formal during this phase than what you do during development due to the increased effort to update and re-release existing software. Similarly, the Project Management workflow is also extended into the new Production phase to include the processes needed to manage your software once it has been released.

The Project Management workflow is expanded in the enhanced lifecycle for the Unified Process. As mentioned earlier it is light on metric management activities. It also needs processes for subcontractor management, a CMM level 2 key process area, a key need of any organization that outsources portions of its development activities or hires consultants and contractors. People management issues, including training and education as well as career management, are barely covered by the Unified Process. There is far more to project management than the technical tasks of creating and evolving project plans, you also need to manage your staffs and mediate the interactions between them and other people.
7. Software Process Improvement (SPI)

There is far more to defining a software process, or improving an existing one, than reading a white paper, reading a couple of books, or purchasing a large collection of HTML pages. The following tips should help you to successfully define/improve a software process within your organization.

1. **Start slowly.** The reality of process improvement is that you cannot make all of the changes that you want to immediately; it is simply too great a change for your organization to absorb at once. This is why we have efforts such as the Software Engineering Institute’s (SEI’s) Capability Maturity Model (CMM) efforts and the Software Process Improvement Capability Determination (SPICE) efforts of the International Standards Organization (ISO). Both of these organizations suggest that you prioritize the process improvements that your organization needs to make, expect that it will take several years to make the needed changes, and expect that you will experience difficulties while doing so. Experience shows that organizations that try to make immediate, large-scale process changes are likely to fail doing so. The reality is that it takes time, often several years, to permanently improve the productivity of your software development efforts. There is not a simple, quick fix to your problems.

2. **Remember the KISS rule.** KISS stands for “keep it simple, silly” – a concept that is directly applicable to your process improvement efforts. A common mistake that organizations make is to overspecify the processes that they intend to follow. Never forget that your goal is to produce software that meets the needs of your user community.

3. **Keep the real goal in mind.** My experience has been that software processes, when applied intelligently, increase the productivity of developers. My experience has also been that when processes are applied less than intelligently, or when the paper pushers have too much influence within an organization, processes can also decrease your productivity. Organizations that keep the end goal in mind – that of developing, maintaining, and supporting software that fulfills the needs of their user community – will be successful with implementing software processes. Those that follow processes simply for the sake of doing so are likely to fail.

4. **Be aware of your organization’s current culture.** As you can see in Figure 1 one of the forces that affect the success of your process improvement efforts is the culture of your organization. Organizations with cultures that are positive toward process improvement are likely to want to supply a quality product with reasonable business returns, have middle managers that are willing to set and work toward targets of meeting your organization’s needs and business goals, and have senior management leadership that is willing to launch and sustain a long-term change effort.

5. **Recognize that the fundamentals remain the same, the details vary.** Contrary to popular belief, the fundamentals of software development have been known for many years. You need to perform requirements engineering. You need to model. You need to write code. You need to test. You need to perform change control. You get the picture. Every successful software organization will have a similar set of processes, hence the similarities of the three sets of processes discussed in this paper, but the way that your organization brings them in and how they implement them will differ. Your requirements process may be slightly different than your competitors, but you will both have one that will generally do the same sort of thing.

6. **Treat process improvement like a project.** Have an experienced project manager, ideally someone with experience in both process-oriented and object-oriented development. Define the requirements for your processes, model them, implement them, test them with a trailblazer project, and then improve the processes.

Copyright 1999-2000 Scott W. Ambler, Ronin International
Visit [http://www.ronin-intl.com](http://www.ronin-intl.com) for more development white papers
7. **Improve your processes in order.** There are five maturity levels in the CMM for a reason: you need to progress in order from level one to level two to level three and so on. The implication is that by knowing which aspects of a software process map to which CMM maturity levels, as indicated in Table 3, you have a rough idea of the order in which you should introduce those processes to your staff.

8. **Accept that the big picture is overwhelming.** Because of the complex nature of software development most developers specialize in one aspect of it and focus solely on that. This becomes a problem for organizations that wish to tailor a software process for their exact needs because when they put the individual pieces together the overall process becomes very large. For example, the Rational Unified Process (http://www.rational.com) version 5.1.1 is over 2,500 pages in size, and it is only a development process. Once developers see how large your organization’s software process is they often go into denial, claiming that you can not possibly achieve your goals. Yet when you ask an individual which part of their process they can simplify they’ll often balk at the idea.

9. **Democracies do not always work, nor do dictatorships.** Organizations that wish to reach consensus regarding their software process tend to flounder. You’ll never achieve complete agreement on how things should be done, although organizations that dictate processes from above tend to fail as well. Effective process improvement efforts seek consensus at some points and dictate things at other points.

10. **Identify the consumers and suppliers for each process.** Every process has inputs and outputs, and you need to ensure that there is a supplier for each input and a consumer for each output. Fundamentally, if nobody is going to use an artifact that is produced by a given process then why bother producing it? You also need to look at collections of processes to see if the artifacts that they produce add value in combination. I once worked in an organization whose logical data modeling group produced models for their client’s logical data modeling group. Although each group was busy producing models, nobody outside of those two groups ever used them for anything (other than for interesting wall decorations). You could have gotten rid of both groups and nobody would have noticed the difference.

11. **Expect to change your organization’s structure and culture.** Process, organizational structure, and corporate culture all go hand-in-hand – change one and you will affect the others.

12. **Defining a process is the easy part.** Many organizations are very successful at defining a software process, often producing binders of documentation (oops, I mean web pages), but visit them a few months after introducing a new process and it will be all but forgotten. Getting people to accept your new process, and making the changes that go along with it, will take significant time and effort to accomplish. Writing a process is the easy part, following it is the hard part.

13. **Take the best of breed.** The three leading object software processes – The Unified Process, OOSP’s process patterns, and the OPEN process – have been summarized and presented in this paper. None of them are perfect but all three are great starting points from which you can start. Choose one process as your base and then fill in the missing gaps using material from the other two processes. There is no need to reinvent the process wheel, instead you can reuse it.

14. **Run a trailblazer project to validate your new processes.** Regardless of how well you define a process, no process is perfect. Test your new software process using a trailblazer project, one that is given the extra resources required to try new techniques and to update them appropriately.

15. **Introduce a Software Engineering Process Group (SEPG) to your organization.** The sole responsibility of your SEPG is to support the definition and improvement of your organization’s software process. The SEPG should be kept small – as a rule of thumb, I suggest one SEPG member for every one hundred developers in your organization – and should be a full-time job.
16. **Document processes to the appropriate level.** Because all people are not created equal, regardless of the feel-good pap currently being foisted on society, not all processes need to be documented equally. My experience is that you should describe your architectural modeling efforts lightly and your project development efforts in greater detail. The reason for this is simple: your architecture group is likely to be a small team of highly experienced people, the type of people who do not need to refer to a documented process to do their jobs. Your project teams, however, are likely to be composed of larger groups of people who may not have worked together before and who are likely to be less experienced.

17. **Avoid “fire hazard processes.”** A common mistake is to produce volumes of documentation describing your processes. Your goal is simply to describe your process patterns to such a level that they can be given to a professional skilled in the techniques of that process so they can work the processes appropriately.

18. **Adopt processes because they make sense.** If a process makes sense to you, and you believe it will add value to your effort, then adopt it. Otherwise, do not.

19. **Hold everyone responsible for process improvement.** Senior management must be willing to actively support and sustain process improvement, project managers must be held responsible for ensuring that their teams follow the defined processes, and developers must be held responsible for learning and then following the processes. This is often a difficult task because senior management often demands immediate results, whereas process improvement often takes years. Project managers resent diverting scarce resources from their projects, and developers often resent being told how to do their jobs.

20. **Bring in an expert to advise you.** Process improvement is a complex and difficult endeavor, one for which you are likely to need help to accomplish. You can increase your chance of success by bringing in a consultant who has both a process background and an OO development background – someone who has been actively involved in a process improvement program and who has worked on large-scale, mission-critical software development projects using OO technology. Ronin International specializes in exactly this kind of effort.

21. **Do not think that everyone is on board.** There is likely to be a small core of people within your organization who do not want to use object technology for large, mission-critical projects, and these people will actively undermine your efforts. You need to identify these dissenters and work together with them to help them see the advantages of working with object technology and of following a set of defined process patterns to help in the development of OO software.

22. **A fool with a process is still a fool.** For your organization to be successful with a software process your software professionals will need to understand the processes, the concepts, the techniques, and the problem domain. Implementing the OOSP in your organization involves more than going out and purchasing a couple of new books and development tools.

23. **Develop a user guide for your process.** You can make it easy for your staff to learn your chosen processes by providing a well-written handbook (I have seen good ones written in less than ten pages) that provides an overview of your process as it is to be implemented in your organization.

24. **Concentrate on construction.** From the point of view of the OOSP, it is in the Construct phase that most of the differences between object-oriented and structured techniques exist. Yes, the Initiate, Delivery, and Maintain and Support phases are all affected by the OO paradigm, but none nearly as much as the Construct phase.

25. **Have patience.** Progress will be slow at first, slower than you hoped or expected. Introducing a software process into an organization takes time – the required culture shift often takes years to complete.
26. **Don’t flounder in bureaucratic requirements.** Too many process efforts run aground because of preconceptions forced on them by senior management, an overly burdensome documentation and review process, or unrealistic requirements to achieve consensus.

27. **Align your software process with business goals and objectives.** You could have the best process in the world, but if it doesn’t meet your organization’s goals then it doesn’t matter. Do you intend to build a portfolio of applications that integrate with, and build on, one another? If so then infrastructure management is important. Do you intend to sell shrink-wrapped software to be used by millions of users? If so then architecture may be less important to you in favor of getting a product to market quickly.

28. **Define your process early.** The longer you leave process definition the bigger the mess you will have to clean up. Without direction, developers will typically go and do what they think is right, the only problem being that each person has their own idea of what “right” is.

8. In Closing

We have known the fundamentals of the software process for years, one has only to read classic texts such as Fred Brook’s Mythical Man Month, originally published in the mid-1970s, to see that this is true. Unfortunately, as an industry we have generally ignored these fundamentals in favor of flashy new technologies promising to do away with all our complexities, resulting in a consistent failure rate of roughly 85%. It’s the cold, hard truth that seven out of eight projects fail. This failure rate, in addition to embarrassments such as the Y2K crisis, are clear signs that we need to change our ways. It is time for organizations to choose to be successful, to choose to follow techniques and approaches proven to work in practice, to choose to follow a mature software process.

In this paper you have seen that the requirements for a mature software process do exist, and that three viable processes are available on the market for you to select from. The Unified Process, OPEN Process, and Object-Oriented Software Process (OOSP) each have their strengths and weaknesses. Alone it is unlikely that any of them meet your exact needs, although when the best features of each are combined you can tailor a process that is right for your organization. You were also introduced to the enhanced lifecycle for the Unified Process, an improvement to the Unified Process, tailored to meet the mission-critical needs of large-scale development.

Software development, maintenance, and support are a complex endeavors, ones that require good people, good tools, good architectures, and good processes to be successful. The software process is a significant part of the solution to the software crisis, something that your organization has likely ignored to its peril. We are at the cusp of a new millennium. Now is the time to learn from our past mistakes; the time to choose to succeed. A failure rate of roughly 85% implies a success rate of only 15%. Think about it.
9. References and Suggested Reading

9.1 Web-Based Resources


9.2 Printed Resources


10. About the Author

# 11. Index

**A**  
Analysis patterns .......................................................... 1  
Architectural patterns ...................................................... 1  
Author contacting ............................................................. 25  

**B**  
BON ............................................................................. 9  

**C**  
Capability Maturity Model (CMM) .................. 13  
Capability Maturity Modeling (CMM) ............ 3  
Career management ....................................................... 20  
Catalysis ........................................................................ 7  
Change control process .................................................... 3  
Configuration & Change Management workflow .................................................... 19  
Construction phase ......................................................... 8  
Contract-driven life cycle ................................................... 9  
CORBA ........................................................................ 1  

**D**  
DCOM ......................................................................... 1  
Deployment workflow ....................................................... 19  
Design patterns ................................................................ 1  
Domain architecture model .............................................. 19  
Dynamic System Development Method (DSDM) ...... 7  

**E**  
Elaboration phase ............................................................ 8  
Enterprise application integration (EAI) ............ 1  
Enterprise JavaBeans (EJB) ............................................ 1  
Enterprise management .................................................... 10  
Enterprise requirements model .................................... 19  
Environment workflow ..................................................... 19  
Extreme Programming (XP) ........................................... 7  

**F**  
Failure rate .................................................................... 23  
Fire-hazard processes ...................................................... 22  
Firesmith ....................................................................... 9  

**G**  
Go/no-go decision point .................................................. 9  

**I**  
Inception phase ............................................................... 8  
Incremental development .................................................. 1  

**K**  
Keep it simple, silly (KISS) ........................................... 20  
Key process area (KPA) .................................................... 3, 13  

**M**  
Maintenance ..................................................................... 2  
Maturity levels ................................................................. 3  
MOSES ........................................................................ 9  

**O**  
Object Modeling Language (OML) .................... 9  
Object-Oriented Software Process (OOSP) ....... 2, 11  
phases ........................................................................... 12  
stages ............................................................................ 12  
strengths ........................................................................ 13  
tasks ............................................................................. 12  
weaknesses ..................................................................... 13  
Objectory Process ......................................................... 7  
OOram .......................................................................... 9  
OPEN Consortium ......................................................... 9  
OPEN Process ............................................................. 2, 9  
strengths ........................................................................ 10  
weaknesses ..................................................................... 11  
Operations & Support workflow ............................. 18  
Organizational culture .................................................... 20  

**P**  
P2K ............................................................................ 1  
Patterns ......................................................................... 1  
People management ......................................................... 20  
Process improvement ....................................................... 20  
Process patterns .............................................................. 11  
Production phase ............................................................. 11  
Programme management ................................................ 10  
Project Management workflow .................................... 19  

**S**  
Skills shortage ................................................................. 1  
Software Engineering Institute (SEI) ................. 3  
Software Engineering Process Group (SEPG) .... 19  
Software maturity ............................................................ 6  
Software process definition .......................................... 1  

Infrastructure management ........................................... 10  
Infrastructure Management workflow .................. 18  
Infrastructure modeling ............................................... 19  
Iterations ........................................................................ 7  
Iterative development ..................................................... 1  

Copyright 1999-2000 Scott W. Ambler, Ronin International  
Visit [http://www.ronin-intl.com](http://www.ronin-intl.com) for more development white papers
Software Process Improvement Capability
  Determination (SPICE) ............................................. 20
  Software process mentor ....................................... 22
SOMA ........................................................................ 9
Supplementary Specification .................................. 8
Support ..................................................................... 2
Synthesis .................................................................. 9

T
  Technical architecture model .................................. 19
  Test workflow ......................................................... 19

Trailblazer project .................................................. 21
Training and education ........................................... 20
Transition phase ....................................................... 8

U
  Unified Modeling Language (UML) ....................... 1, 7
  Unified Process ..................................................... 2, 7
enhanced lifecycle .................................................. 17
  strengths ................................................................ 9
  weaknesses .......................................................... 9

Y
  Y2K crisis ............................................................... 2
The Rational Unified Process (RUP) is a comprehensive software development process framework emphasizing use-cases, architecture focus and an iterative approach. RUP is widely known and many organizations have tried to adopt it. Being a framework, RUP has to, in some way, be tailored to the specific context of use, no software development project is alike. During the research presented in this paper, data was gathered from the work of two groups. Group 1 consisted of 13 senior computer science students who worked in five teams. The students were asked to trace and analyze the process by which they retrieved information from UML diagrams of a given system. The Unified Process facilitates reuse for a single system, but falls short handling multiple similar products. In this paper we present an enhanced Unified Process, called UPEPL, integrating the product line technology in order to alleviate this problem. In UPEPL, the product line related activities are added and could be conducted side by side with other classical UP activities. In this way both the advantages of Unified Process and software product lines could co-exist in UPEPL. We show how to use UPEPL with an industrial mobile device product line in our case study. Keywords: Product Line C True/False 1. The primary objective of the systems analyst is to create a wonderful system. Ans: False Response: See page 2. 2. The planning phase is the fundamental process of understanding how an information system should be built and determining who on the project team will build it. Ans: False Response: See page 4. 3. During the analysis phase of the SDLC the systems analyst will decide how the hardware, software and network infrastructure, user interface, forms and reports will be used. Full file at Ans: False Response: See pages 4-5. 4. The new information system is purchased or built during