AC 2009-2001: A SOFTWARE PROCESS ENGINEERING COURSE

J. Scott Hawker, Rochester Institute of Technology

Dr. Hawker graduated with a B.S. and M.S. in Electrical Engineering from Texas Tech University in Lubbock, Texas, in 1981 and 1982, respectively. He graduated with a Ph.D. in Electrical Engineering from Lehigh University in Bethlehem, Pennsylvania, in 1990. He has over 15 years of industry experience developing large-scale, multi-agent information and control systems for diverse applications including manufacturing, combat pilot decision support and mission management, robotics, and surveillance. In these areas, he developed and applied technologies including distributed, component-based software architectures, software and systems engineering process models, intelligent control, the semantic web, and real-time artificial intelligence. In 1999, Dr. Hawker joined the Computer Science Department at the University of Alabama as an Assistant Professor focusing on software engineering, and in 2004 he moved to the Software Engineering Department at RIT. Dr. Hawker is also co-director of the Laboratory for Environmental Computing and Decision Making, which focuses on modeling and understanding the impact of freight transportation and automotive industry activities and public policies. Dr. Hawker is a member of the ASEE, IEEE, IEEE Computer Society, and the International Environmental Modeling and Software Society.

A Software Process Engineering Course

Introduction

As software product organizations evolve, the roles within the organizations evolve. From the generic "programmer" or "developer" role there evolves distinct roles for the separate but related specializations and practices of requirements development, solution design, implementation, test, etc. Even if these distinct roles are carried out by the same person in a given product development team, the activities, techniques, and artifacts for the roles are separately identified and addressed. As the organization matures, additional roles and specializations are separated out and addressed, such as project management, architecture design, and quality assurance.

We are now seeing an additional, distinct role in maturing software development organizations: Software Process Engineer. In some organizations, the task of defining the software process (process engineering) and tracking its execution is the responsibility of the project manager. In other organizations, a quality assurance organization defines a process for a project, and the project manager tracks its execution. Regardless of the organization or person assigned the role, though, it has become useful to separate the activities of process engineering from the other activities of project managers and quality assurance. For example, project managers are responsible for defining and tracking project milestones, schedules, budgets, risks, etc. Quality assurance is responsible for product verification and validation. The responsibilities of defining, assessing, and improving the software engineering process activities performed by all roles (managers, assurance, developers, etc.) is distinct enough from other project management and quality assurance tasks that is should be separately and clearly identified as a specific set of activities and deliverables. Even if a given individual performs all or part of multiple roles that include both product and process engineering, they should have concepts, methods, and guidance for how to perform the process engineering activities effectively.

As software engineering educators, we need to provide opportunities for our students to learn and practice this software process engineering role and to understand its relationship to other project roles. This paper describes the development of a course on software process engineering as part of a graduate software engineering curriculum, plus our initial experiences teaching the course. We just completed teaching the course the second time, and have identified some successes and opportunities for improvement. This paper describes the course and possible improvements.

In the following sections, we first define the software process engineering role and describe how we model software engineering processes. We then describe the course objectives and the overall approach for meeting those objectives. We then outline the course structure and content and provide observations on the issues and opportunities of various elements of the course. We conclude with an overall assessment and suggestions for on-going improvement.

The Software Process Engineering Role: A Process Designer

A software process engineer defines, for one or more software development projects,

- Method Content: Descriptions of the software development roles, activities, and work products of the software development process, along with specific techniques, tools, guidance, checklists, templates, examples, and other supporting information for performing the activities of the various roles in a project team,
- Delivery Process: The software product lifecycle, which is the ordering of and dependencies between the development activities and their associated work products.

A software process engineer designs a software development process. As Osterweil recognized in his seminal paper, "Software Processes are Software Too," (recognized as the most influential paper of the International Conference on Software Engineering of 1987, and reflected on at ICSE 97), there is value in recognizing the parallel between software engineering processes and software processes. [1,2] Both have requirements, should be carefully designed, have execution behavior, and should be tested and improved. So, based on project requirements, a process engineer should explore design alternatives (selection and composition of activities, techniques, work products, activity sequences, etc.) and assemble and deliver a system of process elements that best suit the needs of the customer—the software development team. Just as with software product design, principles of encapsulation, low coupling, high cohesion, abstraction, reuse, patterns, and other concepts are valuable to a process engineer doing process design.

The Object Management Group has developed a standard for representing software engineering processes, called the Software and Systems Process Engineering Metamodel (SPEM).[3] SPEM includes UML-based diagrams and stereotypes to model software engineering processes. In this paper and in the Process Engineering course, we use the terminology, concepts, and UML stereotype notations in the SPEM standard. This gives us a level of consistency and precision so, as engineers, we can describe, analyze, design, and compare various software engineering process models.

As a concrete example of a process modeled using SPEM, we use the Open Unified Process (OpenUP).[4] OpenUP is an open source software development process framework based on the Rational Unified Process[5] and available as part of the Eclipse Process Framework (EPF) project.[6] The EPF project also provides a tool, called the EPF Composer, plus a library of reusable and composable process components so that process engineers can assemble and tailor processes and deliver those processes as web sites.[7]

SPEM emphasizes a separation of method content from delivery processes. The method content is a set of methods and practices for software development. The delivery process defines the workflow and work breakdown structure capturing the flow of work products from activity to activity and when in the software development lifecycle each activity is performed. The method content can be used in a variety of life-cycle delivery processes. The separation of method content from delivery processes allows the creation of a reusable body of knowledge about software development methods. A given development process is a selection and tailoring of appropriate methods, assembled into a particular process flow. Content and process elements can be combined into larger-grained process patterns and reusable process components that address specific concerns or embody specific practices such as iterative and incremental development, team collaboration, or development methods for specific domains or technology choices.

The IEEE Standard for Developing a Software Project Life Cycle Process[8] also encourage a separation of the process lifecycle of activities from the methods, tools, guidance, and other process content for how to perform the activities (the method content is called Organizational Process Assets in the standard). The course includes this IEEE standard as an alternative and complementary method to SPEM for describing and designing software engineering processes.

Tools such as EPF Composer, which uses SPEM as its underlying tool metamodel, can help manage the libraries of reusable method content and support the assembly of the content into a specific delivery process that is published as a web site that is readily available to the project teams. Using such tools, process engineers and project managers can rapidly select, tailor, and assemble processes specific to the needs of a given project. They can also grow and evolve the method content as new and improved methods become available.

Making the Process Explicit: Software Process Models

The students in our software engineering graduate program have some professional experience executing a software process, usually in the role of developers. They have an intuitive, but sometimes vague, understanding of the various roles in a project and the lifecycle flow of activities in a project. Some have taken a course on software process or project management, so they may have some understanding of the various lifecycle process models (delivery processes). In order to provide a common understanding for all students and to enable concrete discussions and comparisons of process designs, the first part of the course is devoted to software process modeling concepts. This section describes and gives example models of that portion of the course content. The teaching approach leverages the student's understanding of using UML to model systems, in general, and in particular makes parallels between software product models in UML and software process models in UML.

Figure 1 is a UML diagram that illustrates the relationships between roles, activities, and work products in a software engineering process, and Figure 2 is the same diagram, using the SPEM UML stereotype notation.

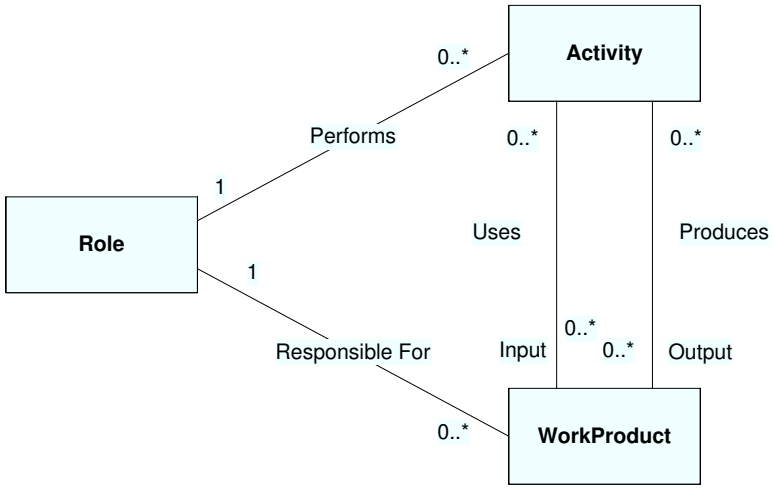

Figure 1. The relationships between roles, activities, and work products

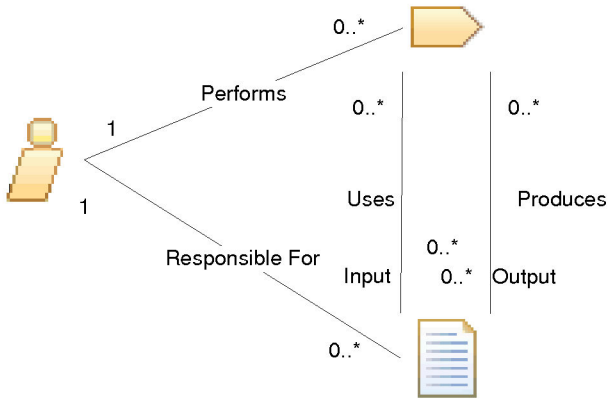

Figure 2. The same information as Figure 1, using the SPEM UML stereotype notation

Roles, activities, and work products can and should have structural relationships. Activities can be decomposed into tasks and steps (generically called Work Definitions in SPEM), and all of them can be described using the semantics and notation of UML behavior models. Collections of activities and work products can be associated with project phases with specific project management decision milestones. This provides for the definition of work breakdown structures and activity sequences. Work products can also have composition structure (such as a requirements specification being a composition of multiple use-case specifications), and work product evolution can be captured with UML state machines (with states such as specified, implemented, reviewed, tested), where work definitions (activities, etc.) progress the work products from state to state.

Figures 3 through 6 are sample diagrams illustrating the detail and structure of the processes modeled using SPEM. Seeing UML diagrams for process models that are similar to the UML diagrams for product models helps the students become comfortable with process modeling and helps them apply product design concepts (separation of concerns, encapsulation, etc.) to process designs. They also see that process models may have alternate visualization notations familiar to project managers and others (such as the work breakdown structure), which reflect underlying UML composition structures. These diagrams are from the OpenUP version 1.5.[4,9]

Figure 3 shows the phases of the Unified Process, modeled as a UML Activity Diagram. Each phase is detailed with a work breakdown structure such as an iteration in the Unified Process Elaboration Phase shown in Figure 4.

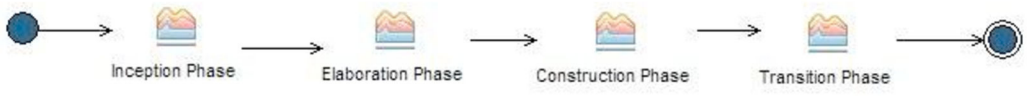

Figure 3. Phases in the Unified Process, modeled as a UML Activity Diagram

Breakdown Element	Steps
Elaboration Iteration [1..n]	
Plan and Manage Iteration	
Plan Iteration	●●●●● ●
Manage Iteration	●●●●●
Assess Results	●●●●●
Identify and Refine Requirements	
Identify and Outline Requirements	●●●●● ●●
Detail Use-Case Scenarios	●●●●
Detail System-Wide Requirements	●●●
Create Test Cases	●●●●●
Develop the Architecture	
Develop Solution Increment	
Design the Solution	●●●●● ●●●
Implement Developer Tests	●●●●● ●●
Implement Solution	●●●●● ●
Run Developer Tests	●●●●
Integrate and Create Build	●●●●●
Test Solution	
Ongoing Tasks	
Lifecycle Architecture Milestone	

Figure 4. Work breakdown structure for an iteration in a Unified Process Elaboration phase

Figures 5 and 6 show some model views of a reusable process component named "Develop Solution Increment." In an incremental development process, this component would be instantiated multiple times, but with different input and output work products, or with work products in different states of completion. Figure 5 captures the collection of activities and their associated input and output work products. Figure 6 uses a UML activity diagram to capture the flow of activities from task to task.

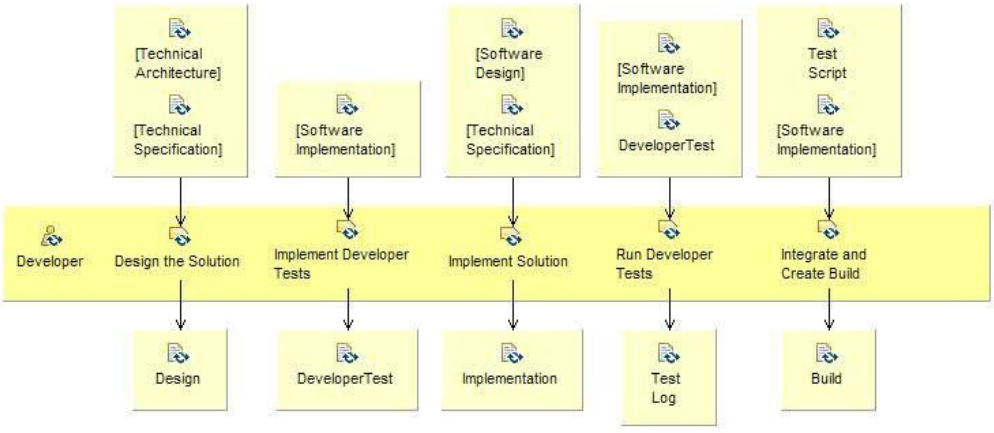

Figure 5. A reusable process component: "Develop Solution Increment"

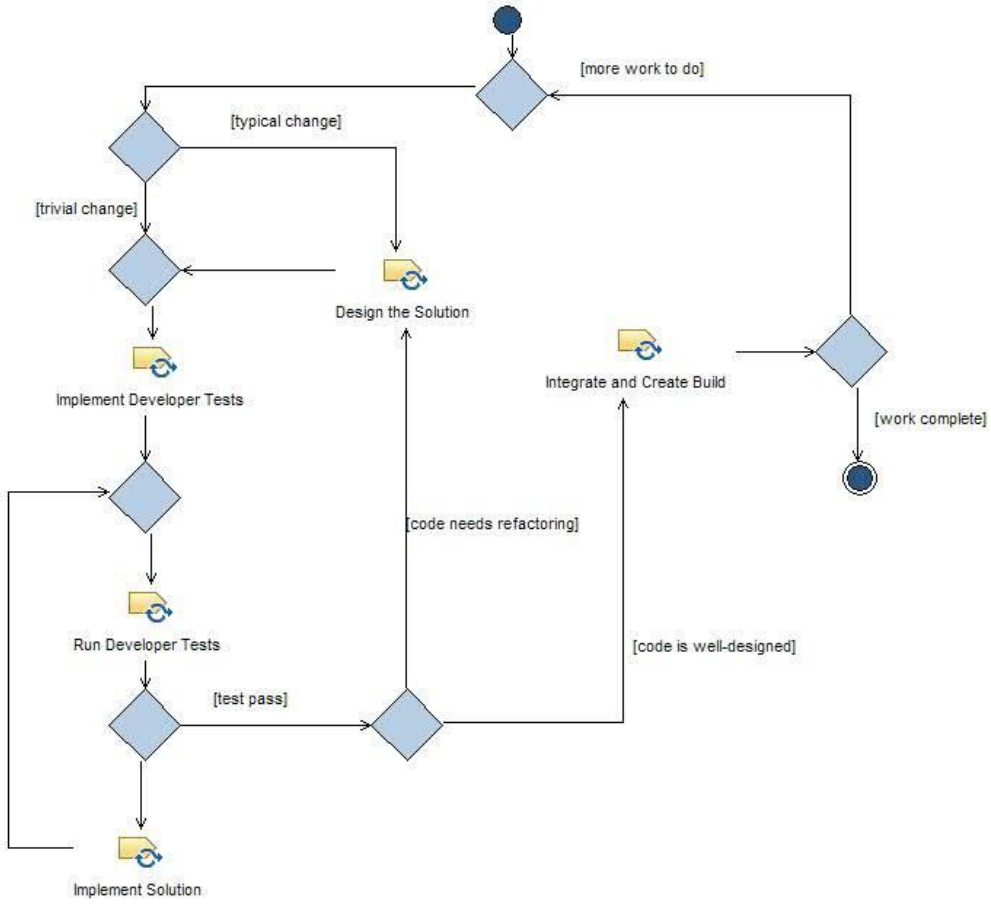

Figure 6. An activity diagram showing the flow of activities for "Develop Solution Increment"

Software Process Engineering Course Description

Our overall goal of the Software Process Engineering course is to equip students with the concepts and skills to be able to design processes to meet the specific needs of a given project or projects in a software development organization. They need to have the abilities to begin to design processes with the scope and depth of OpenUP and similar processes. We seek to give them a survey-level familiarity with various software lifecycle models, software engineering principles, and specific practices. We emphasize over-arching process principles such as incremental development, a balance of formality and agility, and the importance of addressing quality and risk throughout the process. We choose a specific process (OpenUP) that gives a concrete process that meets the balance of discipline and agility that is common in today's software engineering processes. We provide them some basic methods to assess the appropriateness and maturity of a given process and to select process improvements that address specific process deficiencies. Through team projects, the students use a process engineering toolset and library (EPF Composer and OpenUP, Scrum, and other EPF libraries) to design a process for a specific need.

Our specific course objectives are that the successful student will be able to:

- Model, analyze, and compare specific software engineering processes,
- Define and analyze the process needs of various types of software development organizations and projects,
- Use process engineering frameworks to assemble and configure a process from reusable process assets into a process for a specific organization and project,
- Identify software engineering tools and methods and software process assessment and improvement practices for a given software process.

The course textbook is Agility and Discipline Made Easy: Practices from OpenUP and RUP, by Per Kroll and Bruce MacIsaac.[10] This book provides a good summary of current "best" practices in industry that combine disciplined engineering approaches with agile practices. We also make significant use of software engineering standards including SPEM, the IEEE 12207 family of standards on software lifecycle processes, and the IEEE 1074 standard for developing software processes. We also refer to and discuss the SEI Capability Maturity Model family of process assessment models.

The beginning content of the course provides a survey of software engineering process needs, lifecycle models, and common practices. Since many of the students are not software engineers by training (many are computer scientists or from other engineering disciplines, but have been doing software development in industry) and have diverse educational and professional experience, this provides a common basis of understanding. To give the students a feel for the dynamics of incremental processes, we have a process simulation activity. This involves the students in tasks to prioritize work based on risk and incremental delivery, identify project staffing and skills, react to events by re-planning, and perform increment reflection and process improvement.

After establishing this baseline in process models and process execution, we then study SPEM and the instantiation of OpenUP in SPEM. This includes the concepts and content in the previous section of this paper. This course content gives us a common language of process engineering and a concrete process instance to reference. It also gives students experience and practice with how to make process model designs explicit. We then go through the practices described in the chapters of the course textbook. This gives the students familiarity with the practices and principles common in today's software development projects. This begins to fill the student's toolbox of reusable practices that they can combine and tailor for their own projects. We conclude the course with discussions of software process assessment based on the SEI Capability Maturity Model – Integrated and other quality assessment techniques.

To give the students ownership of part of the course content and to leverage peer learning (where students often learn better from each other, complemented by the "expert" knowledge and guidance from the instructor), the practices in the course text are presented in lecture by a pair of students (one pair per chapter). The instructor facilitates discussion during the lecture by asking questions, adding clarity, and providing background and personal experience. The student pair also provides possible exam questions and suggested answers.

There is a term-long team project to design and model a small process or process component. The students select a project from a number of choices, including

- A process defined for our curriculum's practicum course sequence that parallels the process engineering course,
- A process defined for our undergraduate introduction to software engineering course,
- An integration of the SEI Personal Software Process and Team Software Process with agile techniques from Scrum and Extreme Programming,
- A process improvement need from their professional work environment.

In the project, the students define the organizational needs and drivers of the project, define specific process requirements, survey and select appropriate process activities, work products, guidance, etc., and create a portion of the process using EPF Composer. They then present and demonstrate their process to the class and write a report describing the process engineering process and assessing their results and learning experiences.

There is also a research survey component in the course. To encourage students to dig more deeply into process engineering topics that interest them, they select a topic and find three recent conference or journal papers on that topic. They then write and present a short survey paper on that topic.

Course Evaluation

We just completed the second offering of the course. There have been a total of 17 students in the two course offerings, so we can begin to make some observations and identify anecdotes about the course.

There are a number of ways we obtain student feedback on course effectiveness, including the following:

- There are questions in exams that ask the students to identify the new process engineering tools and concepts that they have learned that are relevant to their current or planned career paths and why those concepts are important,
- The project report requires a section on project reflection: what went well, what would be done differently if done again, and key lessons learned,
- Traditional anonymous course evaluations administered for all university courses.

The overall course evaluations are positive, with a clear satisfaction with the course, content, and resources. Grades and assessments of exam, project, text chapter presentation, and research topic paper and presentation indicate that the students are mastering the basics of the course concepts and are demonstrating an ability to apply those concepts in course work. The students are also demonstrating a growing capability in some advanced course concepts, but they are not mastering some of these concepts; mastery is not to be expected from this course. The student's in-class presentations and written work indicates that they are using the concepts and associated terminology correctly and are using that knowledge as part of good reasoning and problem-solving. The process design projects seem to be appropriately sized to be challenging yet still having a manageable process scope. The students felt that the catalog of practices and principles they learned was the most important contribution of the course to their knowledge base. The

students became somewhat adept at designing and analyzing processes, and they were able to assess the processes (or lack of processes) used in their work and academic project experiences and identify and justify process changes. Indeed, some students were eager to incorporate their learning into their work practices, and one expressed an interest in seeking a job as a process engineer.

There are a number of opportunities for course improvement. Some deficiencies and suggested changes that can inform course improvement are summarized below.

Process concepts, in general, are abstract to some students who are more focused on and experienced with product concepts (requirements, design, implementation, test). Process engineering (process design, as opposed to participating in a role in process execution) is one further step removed in abstraction from the hands-on product development that the students are most familiar with. It is important to provide concrete, practical examples of and hands-on experience applying the process and process engineering concepts. In particular, it is difficult for the students to make concrete the requirements of processes and to create and defend process designs. The students are able to model and explain existing processes, but they have difficulty specifying the needs of and designing processes. This is consistent with the difficulty of many software engineering students to perform product requirements and design activities compared to their strengths in product implementation and test activities.

The project work uses Eclipse Process Framework (EPF) Composer and some of the associated content libraries to build the process models and deliver them as a web site to guide process execution by product engineers. EPF Composer is somewhat immature and unstable, and there is not a good deal of documentation on the tool. The IBM Rational Method Composer (RMC) product, which is based on an earlier version of EPF Composer, is also available. However content from RMC and EPF are not interoperable. Further, EPF does not work well for concurrent development and integration of multiple team members working on the same project.

Since EPF Composer is based on the Eclipse platform, the students underestimated the scope and complexity of the tool—they thought it would be easy because they "knew Eclipse," but more as a Java development environment than as a process engineering environment. The course may need to spend more time allowing the students to become familiar with the tools rather than expecting the students to read and follow the available tutorials and help systems. At the end of the project, though, the students did recognize the need and value in process modeling tools for capturing and presenting the scope and detail of a software process.

Now that there are process libraries available from the EPF open source project, there is an opportunity to do a comparison and contrast between different process models, including comparison of process lifecycles and understanding the range of methods for performing software engineering activities. Adding this comparison and contrast activity to the course would help to provide concrete examples of process models and how they are represented in EPF Composer, and it would help the students evaluate alternate process designs, hopefully helping them to become better process designers.

The students sometimes felt that their process models were somewhat disconnected from the real world. They did not have sufficient experience in performing the various activities of product development to be able to know how to assess a process and identify process improvement opportunities. Case studies of process execution using various process models, plus case studies of process assessment and improvement, would help make the process engineering more relevant.

Conclusion

Overall, the software process engineering course has been a moderate success. It gives the students a foundation in process concepts and gives them tools to model and analyze process designs. It gives them a catalog of process content—practices, principles, methods, etc.—that can be applied in a wide range of process life-cycle models and project settings. The course does not give the students a complete knowledge and experience that would enable them to be a lead process engineer for an organization, but it does give them the knowledge and experience that they can be more adept at performing product engineering roles within a number of different process styles, and it gives them some abilities to identify and implement process improvement efforts. The course also gives them some fundamental knowledge and skills in process design that, with further experience, could set them on a career path that includes a role as a process engineer.

Bibliography

1. Leon. J. Osterweil. "Software Processes are Software Too," Proceedings of the Ninth International Conference of Software Engineering, pages 2-13, Monterey, CA, March 1987.
2. Leon. J. Osterweil. "Software Processes are Software Too, Revisited: An Invited Talk on the Most Influential Paper of ICSE 9," Proceedings of the Ninth International Conference of Software Engineering, pages 540-548, Boston, MA, 1997.
3. Object Management Group, Software and Systems Process Engineering Meta-Model Version 2.0, OMG Document Number: formal/2008-04-01, April 2008, http://www.omg.org/spec/SPEM/2.0/PDF, accessed 2009-02-06.
4. Ricardo Balduino, "Introduction to OpenUP (Open Unified Process)," August, 2007, http://www.eclipse.org/epf/general/OpenUP.pdf, accessed 2009-02-06.
5. Philippe Kruchten, The Rational Unified Process: An Introduction, Addison Wesley, 2003.
6. Eclipse Foundation, "Eclipse Process Framework Project," http://www.eclipse.org/epf/, accessed 2009-02-06.
7. Peter Haumer, "Eclipse Process Framework Composer," April 2007, http://www.eclipse.org/epf/general/EPFComposerOverviewPart1.pdf and http://www.eclipse.org/epf/general/EPFComposerOverviewPart2.pdf, accessed 2009-02-06.
8. IEEE Computer Society Software Engineering Standards Committee, Standard for Developing a Software Project Life Cycle Process (IEEE-STD-1074-2006), March, 2006.
9. The materials are used under the Eclipse Public License V1.0. OpenUP downloads are available at http://www.eclipse.org/epf/downloads/openup/openup_downloads.php, accessed 2009-02-06.
10. Per Kroll and Bruce MacIsaac, Agility and Discipline Made Easy: Practices from OpenUP and RUP, Addison-Wesley, 2006.