Agility with the RUP

by **Philippe Kruchten**
Director of Process Development
Rational Software

What characterizes an agile *software development process? Is agility about being fast to deliver? "Faster, better, cheaper" is a laudable goal, but faster in itself is not necessarily the right thing to achieve if it is detrimental to quality. Is agility about minimizing the size of the process used to develop software, the size of its description? Not really, or by that measure the null process would be the best.*

Agility, for a software development organization, is the ability to adapt and react expeditiously and appropriately to changes in its environment and to the demands imposed by this environment.

An agile *process is one that readily embraces and supports this degree of adaptability. So, it is not simply about the size of the process or the speed of delivery; it is mainly about flexibility. In this article, I will explain how the Rational Unified Process® (RUP®) is a process framework that allows this flexibility.*[1]

What Is the Rational Unified Process?

The RUP has several facets:

- It is a software development process.
- It is a catalog of excellent and field-proven software development practices.
- It is developed and commercialized like a software product.
- But it is first and foremost a software process *framework*.

The RUP was intentionally designed with a wide scope of applicability to accommodate variations in:

- Project size
- Application domain (business system, technical system)
- Technology used (language, platforms)
- Business context (in-house development, product development, independent software vendor, contractual development)

As a process framework, the RUP provides a systematic way to capture, organize, and deliver software engineering know-how. It relies on a simple and rigorous underlying process model to organize the know-how: It is more than just a collection of texts or books.

But this process framework is not just an empty shell. It comes prepopulated with a large amount of process know-how already captured over the last fifteen years by Rational folks (and people working for companies later acquired by Rational, and some of our partners, such as IBM, HP, and BEA). The RUP is not a closed, finite process, published once and for all, and it does not answer all the questions or solve all the problems of software development. The RUP framework is an *open* framework, which continues to evolve and is updated twice a year: Its structure is refined, the associated tool support is developed, and its content is expanded.

On one hand, the Rational process development group continues to add to and update the RUP's content as technology evolves and based on feedback from users of its installed base. On the other hand, many partner companies and individuals have adopted RUP to capture and organize their own process know-how, which they use for their own purposes, resell, or integrate with the framework delivered by Rational.

The RUP Framework

The RUP framework is organized around several simple, interconnected concepts:[2]

- It defines process *roles*, which capture the competence, skills, and responsibilities that individuals taking part in the development may be called on to use.
- It describes the work that each role performs in *activities*, decomposed into *steps*.
- These activities operate on concrete *artifacts*: models, code, documents, and reports.
- There is a wealth of associated *guidance* for the roles, activities, and artifacts, in the form of guidelines, templates, examples, and tool mentors, which explain how to perform a certain activity with a given tool.

- All of these process definition elements are organized into *disciplines*.

Presently the basic RUP defines nine disciplines, more than forty roles, and a hundred artifacts, and contains more than a thousand guidance pages. You can also find process "plug-ins" that add even more functionality and content. Some of its detractors call RUP a "heavyweight" process and depict it as a behemoth that forces you to do zillions of useless and unnatural things. We see it more as a rich palette of knowledge from which to choose what you need.

Let me give an analogy. With a well-chosen vocabulary of, say, 800 words, you can get along in many circumstances in many parts of the world. But if you are given a full-blown dictionary (like *Webster's*), first, nobody forces you to use each and every word it contains; second, you can adjust your level of speech to adapt to many other situations beyond the basic ones; and third, you can gradually enhance your vocabulary. Hopefully, the 800 words are a subset of the dictionary.

Adapting the RUP

The RUP is neither a dogma nor a religion. It does not define a rigid, "one-size-fits-all" recipe for software development. No, you do not need a minimum of forty people to fulfill the forty roles, and you do not need to develop more than a hundred different artifacts. You could quickly get into trouble if you were to try this. These process elements are defined in the RUP and provided in *electronic form* to provide the flexibility you need to adapt the process to the demands of your specific development environment.

The RUP embodies many proven software development practices. Rational has given high visibility to six of these:

- Develop iteratively.
- Model visually.
- Manage requirements.
- Control changes.
- Continuously verify quality.
- Use component-based architectures.

The RUP is also based on other key principles that are less visible and more easily forgotten. To mention only a few:

- Develop only what is necessary.
- Focus on valuable results, not on how the results are achieved.
- Minimize the production of paperwork.
- Be flexible.

- Learn from your mistakes.
- Revisit your risks regularly.
- Establish objective, measurable criteria for progress.
- Automate what is human intensive, tedious, and error prone.
- Use small, empowered teams.
- Have a plan.

A key concept in adopting and adapting the RUP is the *development case*. A development case is a tailored, project-specific instance of the RUP. By reference to the RUP framework, this process description defines and identifies the following:

- What will be developed.
- Which artifacts are really needed.
- Which templates should be used.
- Which artifacts already exist.
- Which roles will be needed.
- Which activities will be performed.
- Which guidelines, project standards, and tools will be used.

Producing a Development Case

The development case is usually produced by:

1. Selecting relevant elements in the RUP framework.
2. Eliminating unnecessary process elements.
3. Adding any missing company-, domain-, or technology-specific elements.
4. Tailoring some elements to the precise project context.

As the project unfolds, the development case captures some of the lessons learned by evolving itself in parallel with the software product being developed. Often guidelines or checkpoints for reviews are added to avoid repeating mistakes. The project evolves not only the software it produces, but also its own ability to develop that software.

In many cases, the development case is produced by referring to the generic RUP, but you can also create and deploy an instance of the RUP -- that is, a configuration of the RUP that is tailored to your own needs. A development case is not necessarily a big document.

If starting from a large process framework is an intimidating task, you might want to use a bottom-up approach and start with a minimal set of process elements -- the RUP essentials, which are the elements generally

found in ninety-five percent of software projects.[3] The RUP essentials include:

- A vision for what you want to achieve.
- A plan.
- Some objective success criteria, such as a business case.
- A design.
- A list of risks.
- A list of defects or other kinds of change requests, and so on.

Iterative Development

Of the six best practices embraced by the RUP, the one that has the greatest impact on process agility is *iterative development*.

The RUP describes a versatile development lifecycle with four phases (Inception, Elaboration, Construction, and Transition), each one with specific objectives. Inside these phases, development proceeds in small iterations: Design a little, build a little, and test a little. This is a spiral process, as described by Barry Boehm.[4]

Building the software product incrementally allows not only for early risk mitigation, making tactical changes, and managing scope, but also for fine-tuning the process itself: that is, adjusting the development process, guidance, activities, and artifacts as you learn from previous iterations. So an initial development case is likely to evolve during the development cycle. This is done after each iteration by reflecting on what has worked, what has not worked, and how the organization and the process can be improved. Iterative development provides an ideal setting for a software process improvement process. Jim Highsmith describes an iteration as: Speculate, Collaborate, *Learn*.[5]

Iterative development requires some careful planning. The development case is only one facet of that planning, which must be done not only for the whole cycle, but also for each iteration. The project plan, revisited at each iteration, must reflect the instantiated process and explicitly define everyone's roles and activities. The mapping of roles to individuals embodied in the plan means that not everyone has to know everything about the RUP.

Process Engineering

The RUP framework achieves process agility, the capacity to adapt the development process itself, through one of its disciplines that covers the *process engineering* aspects. In other words, the RUP framework contains the role, activities, artifacts, guidance, and tools to evolve the RUP framework, to build a development case and evolve it throughout the project, and, if you wish, to create your own configuration of the RUP.

The role is the *process engineer*. The *process* itself and the *development case* are the artifacts being produced or modified.

Having an explicitly defined process rather than a vague reference to the literature and a handful of valuable principles is not an attempt to minimize the importance of people. The process does not make people replaceable pawns and does not justify employing less-qualified or less-experienced developers. A defined process also does not, by itself, make the products better along any dimension of quality.

Having a defined process:

- Allows people to communicate better, especially in larger, more distributed organizations, or across multiple organizations.
- Helps teams avoid reinventing process on the fly, arguing about the definition of what needs to be done and how, and debating about the criteria used to evaluate the quality of the result.
- Helps achieve greater consistency in the artifacts produced.
- And finally, allows the process to be used as an objective, concrete object of study and reflection in order to take the explicit step of improving it. When things are not working, having a defined process promotes objective discussion about the failures and weaknesses of the team rather than finger pointing.

In this process engineering effort, which is easier?

1. Starting from a blank slate, with a few key principles, and then building the process from the bottom up?
2. Or starting from a rich knowledge base and choosing, shrinking, modifying, and evolving existing recipes to fit the problem at hand?

With proper guidance (in the process itself) and some tool support, many organizations have found the second approach more efficient and scalable, because it allows them to benefit from the experience of many others, without excluding the integration of their own.

Agility does not always equate to "small." If your environment imposes a high-ceremony, over-the-fence approach, or requires that your organization be certified at CMM (Capability Maturity Model) Level 3,[6] or demands that you deliver design documentation, agility means being able to rapidly accommodate these constraints, which is hard to do when starting from a very small process base. Waving your arms and saying that these are stupid or unfair constraints may not win you the contract.

Project, Process, Organization

The project comes first, with its constraints and environment driving your choice in the most adequate process configuration. The actual process you will use is subordinated to the needs of the project, not the other way around. Development cases can be reused from one project to another,

allowing a faster project start.

In larger organizations that have a method and tools department, an SEPG (software engineering process group), or a QA (quality assurance) department, some more ambitious process engineering can take place. Process know-how can be harvested across multiple projects and departments and then consolidated. To speed up project inception, a predefined, partly instantiated development case can be defined, containing company-specific templates and guidelines. This predefined development case is then fully instantiated by each project (or division, or department) to further tailor it to suit their needs.

The proper order is (1) project needs, which drive (2) process definition, which can be (3) supported and amplified by the organization. It is not, as we often see, first an organization-level process pushed down to the project level, which is then left to try to make the best of it on a given development cycle -- and usually ends up only paying lip service to this process that has fallen from the sky.

Toward Greater Process Agility

Creating the very first development case is often a hurdle, especially when an organization is small or newly created, and has not been exposed to much of the RUP philosophy. There are many choices to make, and as the scope of the RUP framework widens, there will be even more. Also, the process description elements in the RUP are not just standalone pieces; they are tightly integrated with many hyperlinks. So eliminating an arbitrary element is often like pulling one spaghetti noodle out of your plate when you have added too much Emmenthal cheese: It takes a lot of other noodles with it.

This is where process agility can be further improved by helping the project manager or process engineer make the right set of choices for the initial development case. There are several ways to attack this issue:

1. **Predefined configurations.** Provide predefined instances of the RUP framework, in which some of the choices have already been made for a given type of software development environment.
2. **Componentized RUP.** Organize the RUP framework to manipulate smaller chunks of process know-how: process components, for which the amount of coupling is reduced and well defined.
3. **Tools for RUP configuration.** Provide tool support for the construction of a RUP configuration.
4. **Tools for process authoring.** Provide tool support for process authoring, to allow the users to expand the RUP framework and create their own process components.

Predefined Configurations

An example of the predefined configuration is a variant *RUP for Small Projects*, in which many choices have been made to eliminate elements

(artifacts and the corresponding activities and roles) that are not likely to be found in small, "five-people-for-five-weeks" projects, or in projects whose artifacts have been merged or reorganized. Note that a RUP for Small Projects is indeed smaller than the generic RUP framework, but it is not a process reduced to a few high-level recommendations for beginners. This process still aims at providing full guidance on how to achieve quality results, and therefore still contains a lot of guidance. Remember also that software development is not just about programming; there are other important aspects, such as deployment, requirements management, and technical project management. Because of the wide range of factors that affect the project choices, only a few such configurations can be ready-made out of the box.

Componentized RUP

We recently reorganized the RUP framework to fully support the notion of process components, implemented in the form of process plug-ins (see Figure 1 as well as "The RUP: An Industry-wide Platform for Best Practices," in the December 2001 *Rational Edge*).

The framework contains a base RUP, or the very elementary process guidance that will figure in most configurations you can imagine: general definitions, concepts, and principles, as well as the definition of key best practices -- in particular, the definition of the iterative lifecycle or the use of key technology, such as the Unified Modeling Language (UML).

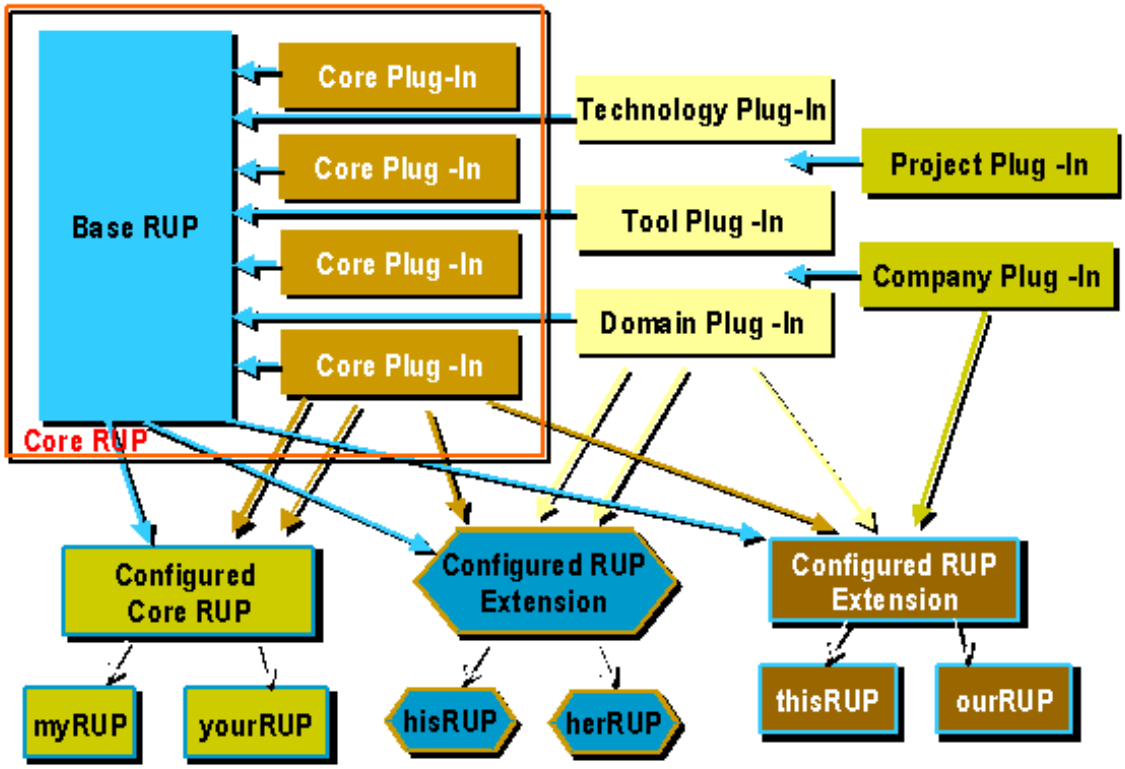

Figure 1: Componentized RUP: Base RUP with Process Plug-Ins

Process components can be added to this base RUP. A process component contains new process definition elements that are added to the base, but it may also redefine and specialize elements already existing in the base. For

example, the base may contain a simple project management discipline, and you might replace it with a more sophisticated project management discipline geared toward the management of a large project under government contract. Extensions are done by "dropping" a plug-in into a compatible base.

Beyond Rational Plug-Ins that provide the core software engineering guidance, very specialized Rational Plug-Ins bring know-how targeted at a given technology (e.g., J2EE), domain (e.g., real-time embedded systems, data warehousing), or tool (e.g., code generator).

The result is still a fully integrated, hyperlinked process, since each of these Rational Plug-Ins "knows" how to relate its own elements to the base. This approach is more like adding spaghetti noodles and cheese and ground pepper to a small initial serving.

Tools for RUP Configuration and Process Authoring

Additional plug-ins can be developed by process engineers using the Rational Process Workbench™, a process modeling tool built on top of Rational Rose®, using UML to model the process. The practitioner -- a project manager, for example -- can configure an instance of the RUP using the RUP Builder, which allows the selection of appropriate and compatible plug-ins for a base RUP (see Figure 2).

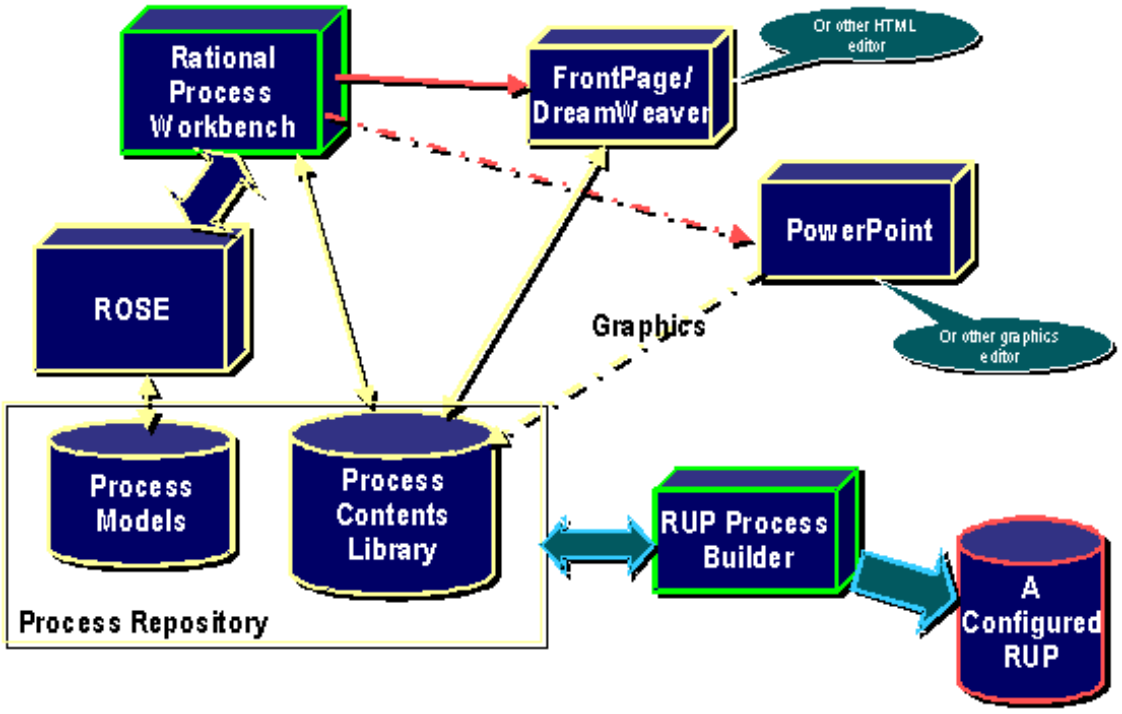

Figure 2: Tools for RUP Configuration and Process Authoring

Toward A Process Marketplace

The concept of process components in the form of plug-ins opens the possibility of a process marketplace. Not all plug-ins have to be developed and marketed by Rational. By making available the very tools with which Rational developed the RUP, and by pushing toward a standardization of

the underlying process metamodel.[7] Rational is opening up this possibility to others. The OO technology it relies on is powerful enough to allow a third party to replace parts of the RUP guidance with its own idea, further opening this marketplace.

Conclusion

Agility is the ability of an organization to adapt and react expeditiously and appropriately to changes in its environment and to demands imposed by the environment. An agile process is one that readily embraces and supports this kind of adaptability.

The Rational Unified Process contains the guidance necessary to adapt its framework to the initial project environment and to evolve it as a project unfolds. The RUP framework provides this agility by offering a wealth of process guidance to choose from, which prevents having to reinvent process and accommodates a larger scope of situations. A great part of the RUP's agility is derived from its iterative approach, which provides many feedback loops and opportunities to make ongoing tactical changes to evolve the process.

By introducing the concept of a base RUP that can be augmented and enhanced with process components and plug-ins and making available the tools to develop them, Rational provides greater flexibility in tailoring the RUP and opens a potential process marketplace.

References

Barry W. Boehm, "A Spiral Model of Software Development and Enhancement." *IEEE Computer*, Vol. 21, No. 5 (May 1988), pp. 61-72.

Tom Gilb, *Principles of Software Engineering Management*. Addison-Wesley, 1988.

James Highsmith, *Adaptive Software Development*. Dorset House, 2000.

Philippe Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed. Addison-Wesley, 2000.

Object Management Group (OMG), *Software Process Engineering Metamodel (SPEM)*, doc ad/01-03-08, 2 April 2001 (http://cgi.omg.org/cgi-bin/doc?ad/01-03-08).

Leslee Probasco, "Ten Essentials of RUP." *The Rational Edge*, December 2000 (http://www.therationaledge.com/content/dec_00/f_rup.html).

Rational Unified Process, version 2001. Rational Software, 2001.

Walker Royce, *Software Project Management: A Unified Framework*. Addison-Wesley, 1999.

Note: This article appeared as Philippe Kruchten, "Agility with the RUP."

Cutter IT Journal, Vol. 14, No.12, December 2001, pp. 27-33.

Notes

[1] See Philippe Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed., Addison-Wesley, 2000 and *Rational Unified Process*, version 2001.
[2] Object Management Group (OMG), *Software Process Engineering Metamodel (SPEM)*, doc ad/01-03-08, 2 April 2001 (http://cgi.omg.org/cgi-bin/doc?ad/01-03-08).
[3] See Leslee Probasco, "Ten Essentials of RUP." *The Rational Edge*, December 2000 (http://www.therationaledge.com/content/dec_00/f_rup.html).
[4] Barry W. Boehm, "A Spiral Model of Software Development and Enhancement." *IEEE Computer*, Vol. 21, No. 5 (May 1988), pp. 61-72.
[5] James Highsmith, *Adaptive Software Development*. Dorset House, 2000.
[6] See http://www.sei.cmu.edu/cmm/cmm.html
[7] Object Management Group (OMG), *Op.Cit*.