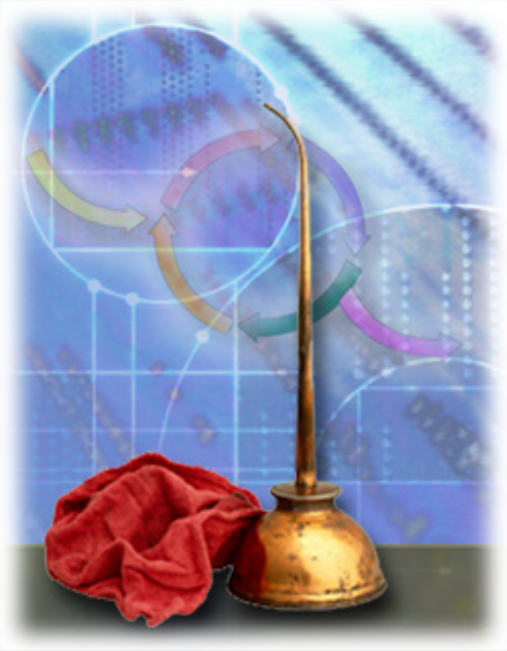
Software Maintenance Cycles with the RUP

by **Philippe Kruchten**
Rational Fellow
Rational Software Canada



The Rational Unified Process®(RUP®) has no concept of a "maintenance phase." Some people claim that this is a major deficiency, and are proposing to add a production phase to cover issues like maintenance, operations, and support.[1] In my view, this would not be a useful addition. First, maintenance, operations, and support are three very distinct processes; although they may overlap in time, they involve different people and different activities, and have different objectives. Operations and support are clearly outside the scope of the RUP. Maintenance, however, is not; yet there is no need to add another phase to the RUP's sequence of four lifecycle phases: Inception, Elaboration, Construction, and Transition. The RUP already contains everything that is needed in terms of roles, activities, artifacts, and guidelines to cover the maintenance of a software application. And because of the RUP's essentially iterative nature, the ability to evolve, correct, or refine existing artifacts is inherent to most of its activities.

Software Maintenance

The IEEE defines software maintenance as the "process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment."[2] Software maintenance is the process that allows existing products to continue to fulfill their mission, to continue to be sold, deployed, and used, and to provide revenue for the development organization. Generally speaking, *maintenance* refers to all the activities that take place after an initial product is released. However, in common usage, people apply the term *maintenance* not so much to major *evolutions* of a product, but

regarding efforts to:

- Fix bugs (*corrective* maintenance).
- Add small improvements, or keep up with the state-of-the-art (*perfective* maintenance).
- Keep the software up-to-date with its environment: the operating system, hardware, major components such as DBMS, GUI systems, and communication systems (*adaptive* maintenance).

The RUP applies well to all these circumstances, mostly because of its iterative nature. Indeed, the evolution or maintenance of a system is almost indistinguishable from the process of building that system in the first place -- so much so that the IEEE standard on maintenance[3] looks like a recipe for development, covering problem and modification identification, analysis, design, implementation, regression and system testing, acceptance testing, and delivery!

For a look at the RUP's role in more significant evolutions for existing systems, see my article in the May issue of *The Rational Edge*: Using the RUP to Evolve a Legacy System.

Software Development Cycles

The RUP defines a software development *cycle*, which is always composed of a sequence of four phases:[4]

- *Inception* phase: specifying the end-product vision and its business case, defining the scope of the project.
- *Elaboration* phase: planning the necessary activities and required resources; specifying the features, and designing and baselining the architecture.
- *Construction* phase: building the product, evolving the vision, the architecture, and the plans until the product -- the completed vision -- is ready for a first delivery to its user community.
- *Transition* phase: finishing the transition of the product to its users, which includes manufacturing, delivering, training, supporting, and maintaining the product until the users are satisfied.

What is really important about the phases is not so much what you do in them, or how long they last, *but what you have to achieve*. A phase is judged by the *milestone* that concludes it, and each of these major milestones has some clear exit criteria attached to it, expressed in terms of artifacts that must be produced or evolved, and measurable objectives to be attained.

Then, within each phase, software development proceeds by iteration, repeating a similar set of activities and gradually refining the system to the point where the product can be delivered.

The phases are not optional, or skipped; in some cases they may be reduced to almost no work, but never really to nothing at all. Don't be fooled by the RUP "Hump Chart" (Figure 1); it does not imply that you have to spend time and money doing busy work. One of the RUP's key principles always applies: *Do nothing without a purpose*.

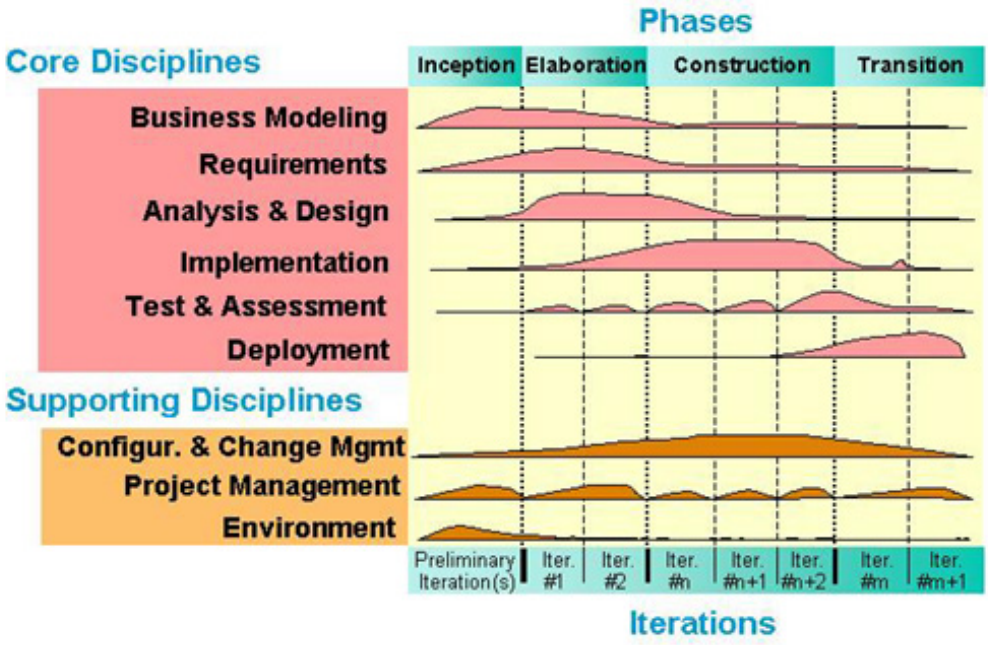


Figure 1: RUP "Hump Chart"

For example, suppose that just after exiting the inception phase, you realize that you already have all the elements in place for exiting the elaboration phase:

- The requirements are understood.
- The architecture will not change.
- The plan for the first iteration of the construction phase is in place.

Then you have probably done all the work you need to do for the elaboration phase. It is still worthwhile, however, to take a very good look -- just to be sure that this is really the case before hastily jumping to the next phase. At a minimum, you will need an end-of-phase review.

It is very rarely the case that you can collapse a phase to almost nothing for a new development project (greenfield development), but you may be able to do so for an existing system. Figure 2 shows a typical resource profile for an initial development cycle.

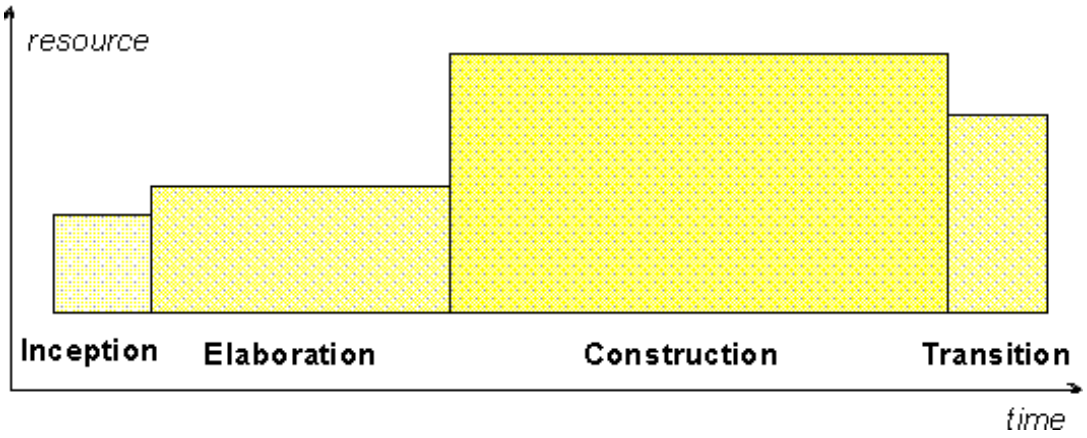


Figure 2: Version 1.0: Initial Development Cycle

Evolution Cycles

So now let us assume that the system exists, and it has gone through a RUP initial development cycle. What happens next is an *evolution cycle*. It has the same overall sequence of phases: Inception, Elaboration, Construction, and Transition, and will result in a new product release. Yet, since the system already exists, the ratios of effort required for the phases, and the actual activities performed in each phase, will be different from an initial development cycle.

In an initial development cycle, there is a lot of discovery and invention, and artifacts are often created *from scratch*; this is done mostly in the iterations of the Inception and Elaboration phases. In contrast, in an evolution cycle we proceed mostly by *refinements* of a body of existing artifacts. Is this new? No. This is exactly what we were already doing in the trailing iterations of the Construction phase and during the whole Transition phase of the initial development cycle.

Version 2.0: A Simple Extension

Let's assume we have released an initial product: Version 1.0. The evolution cycle for going to Version 2.0 could look like the one shown in Figure 3.

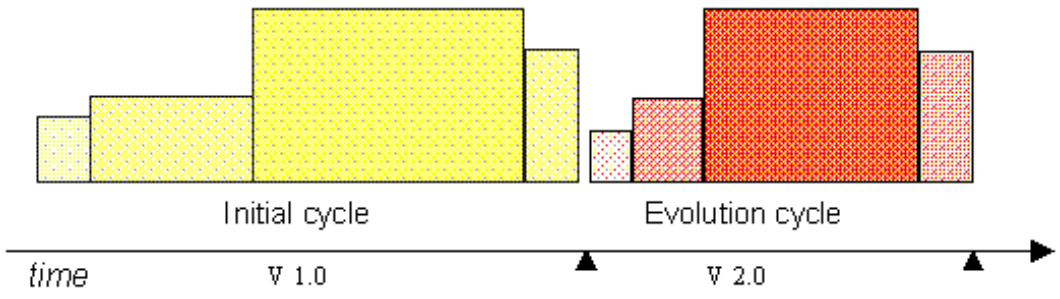


Figure 3: Adding an Evolution Cycle for a Simple Extension

To start, we have a business case for going to Version 2.0. The scope of this project is to:

- Complete all the requirements that were "scoped out" (but already captured) during the initial cycle;
- Add one or two features that were discovered on the way and captured, but which were left out of scope to avoid disrupting the schedule of the initial cycle;
- Fix a handful of bugs in the defect database.

If none of these requirements affects the overall architecture, and if there is no major risk to mitigate, then the Elaboration phase is reduced to almost nothing -- maybe a day or two of work. Construction (up to a beta version) and transition proceed iteratively as in an initial development cycle, with the same type of staff and allocation of roles. All artifacts are updated to reflect the evolution. The development cycle would then look like the one shown in Figure 4.

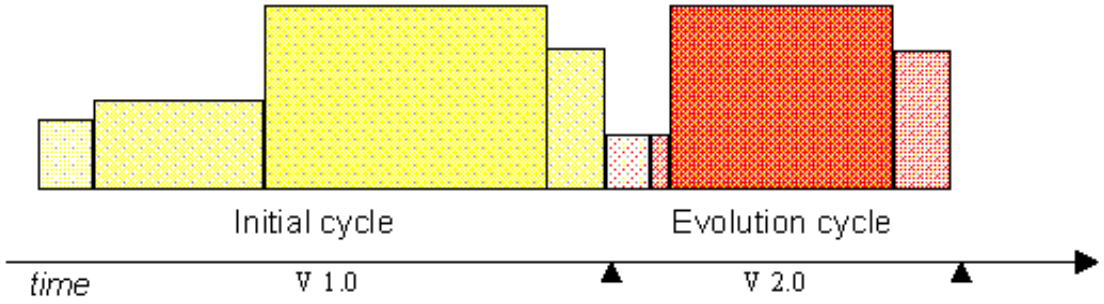


Figure 4: Adding an Evolution Cycle with a Minimal Elaboration Phase

Version 3.0: A Major Addition

Not all evolution cycles may be that simple. Let us assume that, based on the success of Version 2.0 (a single-user, single-processor system), the requirements for Version 3.0 include going to a distributed system supporting multiple users. We then need a serious Elaboration phase to evolve and validate the architecture of the system, since the evolution is riskier. Now the cycle would look more like the original profile of an initial development cycle (see Figure 2), with a non-trivial effort in inception and elaboration.

For more discussion of perfective maintenance, and how to tackle system evolution when the original system was not developed with the RUP, see my May *Rational Edge* article, "Using the RUP to Evolve a Legacy System."

Maintenance Cycles

Now let us look at the more typical cases of corrective and adaptive maintenance.

The "Bug Fix" Release: Corrective Maintenance

Suppose we need a new release of the system that fixes some annoying problems discovered by users. The evolution cycle would include:

- *Inception* phase: As for any project, we need a scope (What must really be fixed?), a plan (What is the commitment of effort and time?), and a business case (Why should we be doing this?).
- *Elaboration* phase: This should be minimal; hopefully, most of our bug fixing will not require a change to the requirements or the architecture.
- *Construction* phase: One iteration is needed to do the fixes, test the fixes, do regression testing, and prepare a release.
- *Transition* phase: If we are very lucky, and the end of construction showed no regression, then this phase might not need much work.

The key point is this: *All the activities we run through are already in the RUP*.

Alternative: Extending Your Transition Phase with More Iterations

If the corrective maintenance entails only a minimal amount of change, then you may consider it simply as an additional iteration in your Transition phase (see Figure 5).

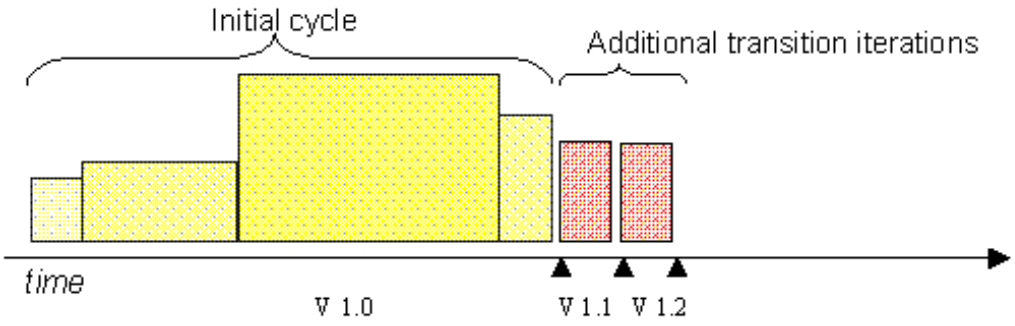


Figure 5: Adding Small Corrective Maintenance Iterations to the Transition Phase

If we take this to the extreme, we have a pattern of incremental delivery, as described by Tom Gilb:[5] after some good, solid work up-front, the system can be delivered incrementally, bringing additional functionality at each step.

Example: Activities for Simple Corrective Maintenance

Here is a list of activities that you may go through for a simple corrective maintenance iteration:

Activity: develop iteration plan

Objective is defined by a selection of the change requests to be done.

Activity: plan test

Identify specific test to create to validate the correction and all regression tests to run on the release.

Activity: schedule and assign work

Activity: create development workspace

Activity: create integration workspace

Activity: fix defect

This is repeated for each defect.

Activity: execute unit test

Activity: integrate system

Activity: execute system test

This includes all tests to validate the new release.

Activity: create baseline

Activity: write release note

Activity: update change request

Activity: conduct iteration acceptance review

Activity: release product

The "Compatibility" Release: Adaptive Maintenance

Often, we need a maintenance release because part of the system has evolved to a new release number. There could be changes to a system component, such as the database, or to some elements of the system environment, say the operating system, platform, or communication interface. In order to remain compatible, we must rebuild (sometimes) and retest (always) the system against the new elements. But the system itself does not need to be extended.

This type of maintenance cycle will also have a streamlined shape. In the simple case, few artifacts will have to change, and most of the activities will be in regenerating the system and testing it. If an interface has changed, then there may be some design and code to change. All the activities to run through are already defined in the RUP.

However, it is wise to plan two iterations, because of the inherent risks:

- An iteration to do the port or the conversion, and do thorough regression testing: *to actually confront the risk*.
- An iteration to do whatever corrections are identified from the first iteration: *to resolve any issues that arose*.

Comparing the Initial Cycle and the Maintenance Cycle

What is different in a maintenance cycle, compared to the initial development cycle, then? Mostly, we have to adjust the level of formality to:

- The size of the organization.

- The risks at stake.

If we are employing a very small staff, with maybe just *one* person to do these jobs, then that person will have to play many roles defined in the RUP, performing all the activities involved in updating the various artifacts, changing the code, testing the system, and releasing it. The artifacts are neither new nor different; they are simply updated, just as we did in the trailing iterations of the initial development cycle. The better the job we have done in the earlier development cycles, the easier the task of the persons who carry out these maintenance cycles.

If there is a high level of risk, technical or otherwise, then the development must proceed more carefully, making sure that the risks are properly addressed and mitigated, early in the cycle.

Configuration and change management get a higher profile in a maintenance cycle, especially with regard to parallel maintenance of many product variants and older releases.

Overlapping Cycles

Can a maintenance cycle start before the previous cycle is complete? Yes; it is feasible to overlap the cycles slightly, as shown in Figure 6. However, the project manager should keep the following in mind.

- There are often frantic efforts to complete a cycle, especially an initial cycle, and the best people may be needed to meet a scheduled delivery date. Consequently, there may be few people readily available to do a good job in the Inception and Elaboration phases of the next cycle.
- As you increase the overlap, there comes a point at which modifications to project artifacts (source code or others) for the next cycle will result in the need for an onerous reconciliation (merge) with the output of the current cycle.
- The more the cycles overlap, the higher the risks (e.g., miscommunications, regression, rework, competition for resources).

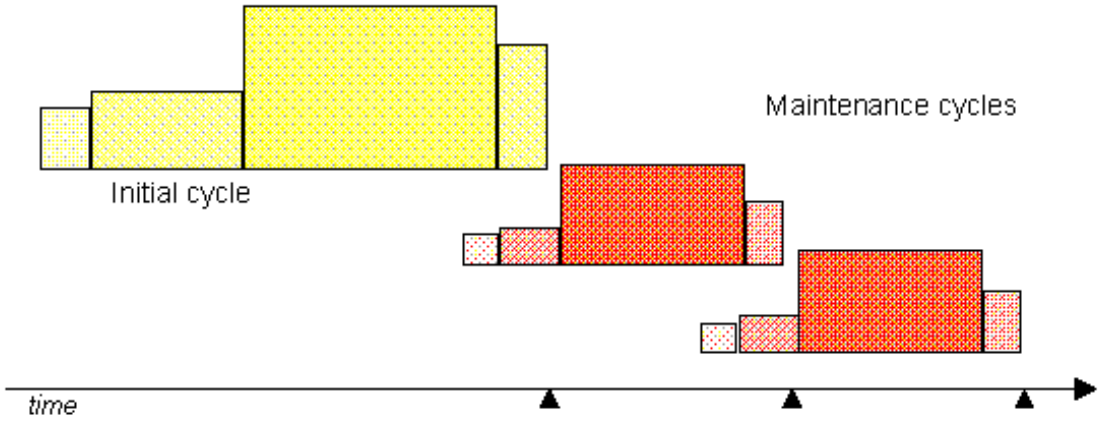


Figure 6: Overlapping Cycles: Feasible but Tricky

Conclusion

The RUP has no notion of a "maintenance phase" following its Transition phase because it does not need one. It has *evolution cycles* and *maintenance cycles*, which follow the very same pattern of phases: Inception, Elaboration, Construction, and Transition. Each phase has to satisfy the same exit criteria as for an initial development cycle. Depending on the nature of the evolution or maintenance, the relative effort spent in each phase will vary greatly, compared to that of an initial development cycle. All the artifacts, activities, roles, and guidelines of the RUP still apply, but with an emphasis on correction and refinements of the existing body of artifacts rather than invention and creation of a new one. What happens in a maintenance cycle is not at all different from what is done in the later iterations of an initial development cycle. As the size of the team is often reduced, sometimes to a handful of staff or fewer, the persons executing a maintenance cycle must be more polyvalent or versatile in terms of skills and competencies, since they will play more of the roles defined in the RUP.

Acknowledgments

Thank you to my friends and colleagues for shaping this article: John Smith, Grady Booch, Craig Larman, and the many participants in our internal process forum.

Footnotes

[1]Scott Ambler, "Enhancing the Unified Process." *SD Magazine*, October 1999.
[2]IEEE Standard 610.12:1990, *Glossary of Software Engineering Terminology*.
[3]IEEE Standard 1219-1998, *Software Maintenance*.
[4]Rational Software Corporation, *Rational Unified Process*. Cupertino California, 2001.
[5]Tom Gilb, *Principles of Software Engineering Management*. Addison Wesley, 1988.