A Manager's Introduction to The Rational Unified Process (RUP)

Scott W. Ambler

This version: December 4, 2005

Executive Summary

Software development is a complex endeavor, one which is fraught with peril if you do not follow a proven software method. The Rational Unified Process (RUP) is one such method. The RUP takes an evolutionary approach to development which has been shown in practice to be far more effective than the traditional, serial "waterfall" approach which is prevalent in many organizations.

This white paper overviews the lifecycle, phases, disciplines, and best practices of the RUP. It also provides suggested web-based resources which you may find useful if you are looking for more detailed information. Although written for managers, IT practitioners will also find this paper a valuable introductory resource to the RUP.

Acknowledgements

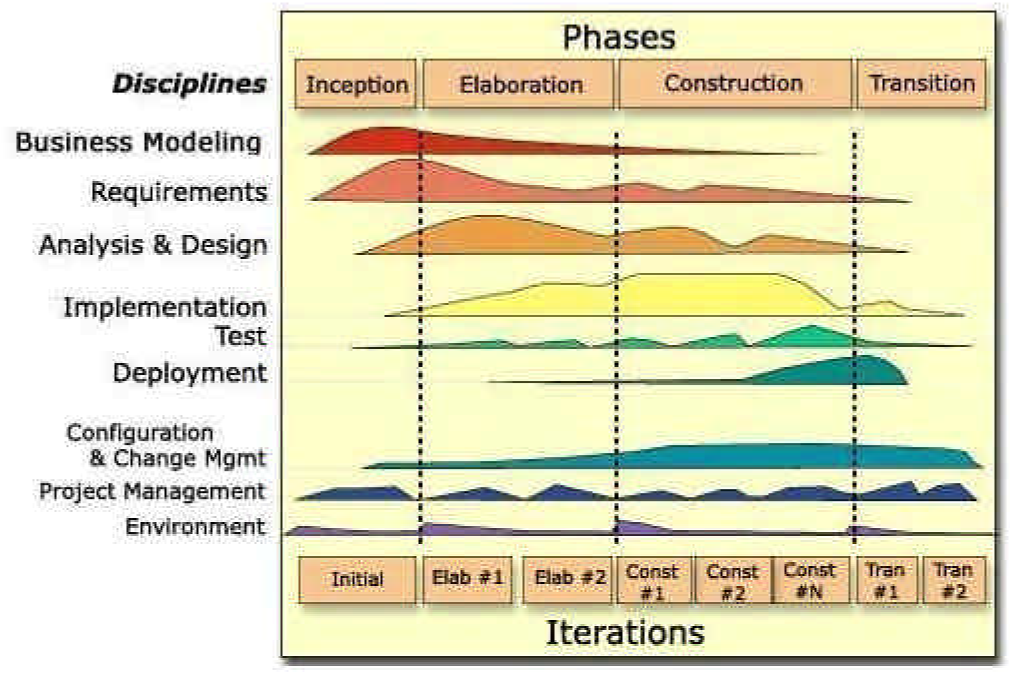
I would like to thank Peter Alfvin, Rob Exley, Mark Lines, and Frank Schophuizen for their insightful feedback which I have incorporated into this paper.

Table of Contents

The IBM Rational Unified Process (RUP) is a prescriptive, well-defined system development process, often used to develop systems based on object and/or component-based technologies. It is based on sound software engineering principles such as taking an iterative, requirements-driven, and architecture-centric approach to software development (Kruchten 2004). It provides several mechanisms, such as relatively short-term iterations with well-defined goals and go/no-go decision points at the end of each phase, to provide management visibility into the development process.

Figure 1 depicts the RUP lifecycle, commonly referred to as the “hump chart” diagram. The horizontal “humps” for each discipline give a rough estimate of the relative effort for each throughout the four phases. For example you can see that a large portion of Business Modeling takes place in Inception, although it does continue through to early Transition. Work on deployment usually does not start until Elaboration and doesn’t really kick into high gear until the middle of Construction. The hump chart is a visual mechanism to provide a rough overview of how much each discipline is executed in each phase.

Figure 1. The RUP v2003 lifecycle[1].



[1] This diagram is Copyright 1999-2005 IBM.

IBM Rational has made, and continues to make, a significant investment in the RUP. In November 2005 IBM Rational announced that the RUP product (IBM 2004), along with Summit Ascendant (another IBM-owned software process), would evolve into Rational Method Composer (RMC). RMC is an Eclipse-based tool which enables you to define, maintain, and deploy software process related material (Kroll 2005). Although RMC now forms the software process product offering from IBM Rational going forward, RUP as a process framework will still continue.

I like to characterize the RUP lifecycle as:

1. Serial in the large
2. Iterative in the small
3. Delivering incremental releases over time
4. Following proven best practices.

1 Serial in the Large

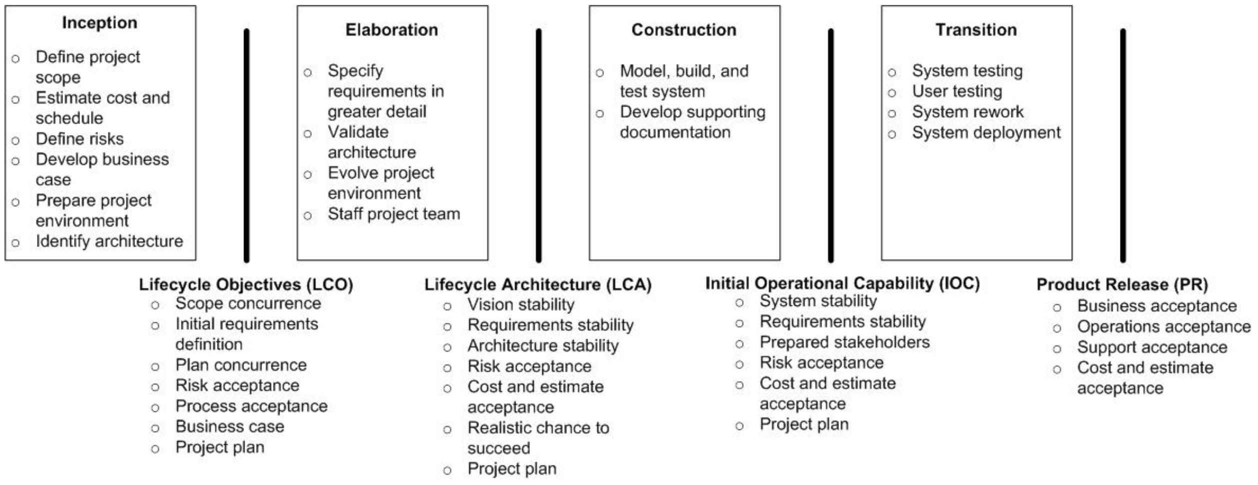
The RUP is structured in two dimensions: phases, which represents the four major stages that a project goes through over time, and disciplines, which represent the logical activities that take place throughout the project. The serial aspect of the RUP is captured in its phases and the iterative nature of the RUP by its disciplines. You see in Figure 1 that the four phases – Inception, Elaboration, Construction, and Transition – are listed across the top of the lifecycle.

Phases are the "Seasons" of a Project

Gary Evans, of Evanetics (www.evanetics.com), likes to compare the phases of a RUP project to the seasons of the year. For example, during Inception you'll do the same sorts of activities that you do during Construction, but the extent that you do them, and the order in which you do them may change. During Inception a project team will spend a lot of time writing initial, point-form use cases in order to understand the scope of their system, but if they write any code at all it is likely for user interface (UI) prototyping. During Construction any use case work will likely be to finalize the definition of a use case before writing the detailed source code which implements it. During both phases you work on use cases and source code, but in different ways.

Each phase ends with a well-defined milestone. At these points, the stakeholders assess the project, including what has been done and the plans for moving forward. A go/no-go decision is made about whether to proceed with the project. Each phase has a specific set of goals, which are addressed within the iterations of the phase, so that the phase milestone may be met. Figure 2 overviews the primary activities and milestones of each phase; each phase is explored in detail within the following sections. In many ways phases provide the glue which holds the RUP together, but the disciplines of the RUP (Section 2) capture its heart.

Figure 2. The RUP phases and their milestones.



1.1 The Inception Phase

The primary goals of the Inception phase are to achieve stakeholder consensus regarding the objectives for the project and to obtain funding. To do this you will develop a high-level requirements model which will delimit the scope of the project, and potentially start the development of a user interface (UI) prototype. You will start to install the work environment and tailor the process for the team. You will also develop a high-level plan for how the project will proceed. At the end of this phase you hold the Lifecycle Objectives (LCO) milestone where your stakeholders assess the state of the project and must agree:

1. On the scope of the project
2. That the initial requirements have been identified (albeit without much detail)
3. That your software development plan is realistic
4. That the risks have been identified and are being managed appropriately
5. That the business case for the project makes sense
6. That the development process has been tailored appropriately

1.2 The Elaboration Phase

During the Elaboration phase you specify requirements in greater detail and prove the architecture for the system. The requirements are detailed only enough to understand architectural risks and to ensure that there is an understanding of the scope of each requirement for subsequent planning. To prove the architecture you will implement and test an “end-to-end skeleton” of working code which supports the high-risk use cases for your system. At the end of this phase you must hold the Lifecycle Architecture (LCA) milestone review where your stakeholders assess the state of the project and must agree that the:

1. Project vision has stabilized and is realistic
2. Requirements for the project (although they will still evolve)

3. Architecture is stable and sufficient to satisfy the requirements
4. Risks are continuing to be managed
5. Current expenditures are acceptable and reasonable estimates have been made for future costs and schedules
6. Project team has a realistic chance to succeed
7. Detailed iteration plans for the next few Construction iterations, as well as a high-level project plan, are in place

1.3 The Construction Phase

The focus of the Construction phase is to develop the system to the point where it is ready for deployment. Emphasis shifts now to prioritizing requirements and completing their specification, analyzing them, designing a solution to satisfy them, and coding and testing the software. If necessary, early releases of the system are deployed, either internally or externally, to obtain user feedback. At the end of this phase you must hold the Initial Operational Capability (IOC) review where your stakeholders assess the state of the project and must agree that the:

1. Software and supporting documentation are acceptable to deploy
2. Stakeholders (and the business) are ready for the system to be deployed
3. Risks are continuing to be managed effectively
4. Current expenditures are acceptable and reasonable estimates have been made for future costs and schedules
5. Detailed iteration plans for the next few Transition iterations, as well as a high-level project plan, are in place

1.4 The Transition Phase

The Transition phase focuses on delivering the system into production. There will be testing by both system testers and end-users, and corresponding rework and fine tuning. Training of end users, support, and operations staff is done. At the end of this phase you must hold the Product Release (PR) milestone review where your stakeholders assess the state of the project and must agree that the:

1. System, including supporting documentation and training, is ready for deployment
2. Current expenditures are acceptable and reasonable estimates have been made for future costs
3. System can be operated once it is in production
4. System can be supported appropriately once it is in production

1.5 Examining the Four Phases

There are five critical observations to be made concerning the RUP phases:

1. **Activities continue in each phase**. Traditionalists will often mistakenly think that the Inception phase corresponds to the traditional requirements phase, that Elaboration corresponds to design, that Construction corresponds to coding, and that Transition corresponds to testing. Nothing could be further from the truth. Look at Figure 1 again – the nine disciplines (Section 2) cross into each phase. You'll be designing, coding, and testing in an iterative manner during all four phases.
2. **Work products evolve during each phase**. Work products – models, plans, source code, documents – evolve throughout the life of your project. Work products aren't finished until you release your system into production, in fact you may find that you'll be doing requirements modeling the day before you release.
3. **The project is planned in a rolling wave**. With a rolling wave approach (Githens 1998) detailed planning is done for things closer to you (like the relative height of the crest of a wave) and less detailed for things further away (the lack of height of the back side of a wave). As your project progresses and tasks get closer detailed planning for them is done.
4. **Risk management is crucial to your success**. IT professionals who state "we know the risks, there's no need to document them" or "there's so few risks, it's not worth our time to record them" are asking for trouble. Risks that are assumed to be "known" by everyone are rarely understood to be the same by everyone. Risks that are few in number should entail a small amount of time to document so why not invest the miniscule effort. Simply going through the effort of thinking about risks and strategies to deal with them in order to document them is time well spent.
5. **Each phase ends with a go/no-go decision**. Your stakeholders must agree to move forward into the next phase. This may entail reworking your strategy for running the project, or they must agree to cancel the project. A project may be cancelled because it's not acceptable due to quality concerns or lack of appropriate documentation, it is deemed too expensive to deploy and/or support, or the strategic direction for the company may have shifted. Although it is rare, it is possible that a project may even be cancelled at the end of Transition.

Typical RUP projects spend approximately 10% of time in Inception, 25% in Elaboration, 55% in Construction and 10% in Transition, although these figures vary from organization to organization and even from project to project. For example, a "green field" project in a new technical area may spend more time in Elaboration, for example, as the team addresses new architectural issues. A project that is heavily dependent upon business processing may spend more time in Inception figuring out new business processes and how (or if) technology can support them. A project consisting of minor enhancements and fixes to an existing system may spend very little time in Inception and Elaboration. The bottom line is that every project is different and the process that it follows must be tailored to meet its needs.

2 Iterative in the Small

RUP phases are divided into one or more iterations, although the term “increment” would likely have been more appropriate. Iterations address only a portion of the entire system being developed (unlike the waterfall approach, which attempts to do it all at once). Each iteration has a fine-grained plan with a specific goal. Iterations build upon the work done by previous iterations and assemble the final system incrementally. Iterations are indicated along the bottom of Figure 1 with each phase subdivided into one or more iterations. When an iteration ends, in particular during the Construction phase, a small subset of the system has been completed which could conceivably be deployed to users as a release, even if only as an alpha or beta version.

The iterative nature of the RUP is reflected in how you approach its disciplines, which are logical grouping of activities that take place over the lifetime of a project. The heart of the RUP is truly in its disciplines, not its phases. During each iteration you will alternate back and forth between the activities of the disciplines, performing each task to the extent needed at the time, to achieve the goals of that iteration. In other words, during an iteration a portion of the requirements are selected, analyzed, designed, coded, tested and integrated with the products from earlier iterations. The nine disciplines of the RUP are:

1. **Business Modeling**
2. **Requirements**
3. **Analysis and Design**
4. **Implementation**
5. **Test**
6. **Deployment**
7. **Configuration and Change Management**
8. **Project Management**
9. **Environment**

2.1 *The Business Modeling Discipline*

The goal is to understand the business of the organization, usually confined to the scope of the business that is relevant to the system being developed. Working closely with project stakeholders, you will:

1. Assess the current status of the organization, including your ability to support a new system
2. Explore the current business processes, roles, and responsibilities
3. Identify and evaluate potential strategies for reengineering the business processes
4. Develop a domain model which reflects that subset of your business

Suggested Resources
Extending the RUP with the Enterprise Business Modeling Discipline
www.enterpriseunifiedprocess.com/essays/enterpriseBusinessModeling.html

2.2 The Requirements Discipline

The goal is to elicit, document, and agree upon the scope of what is and what is not to be built. This information is used by analysts, designers, and programmers to build the system, by testers to verify the system, and by the project manager to plan and manage the project. Activities of the Requirements discipline include:

1. Working closely with project stakeholders to understand their needs
2. Defining the scope of the system
3. Exploring usage, business rules, the user interface, and technical (non-functional) requirements via appropriate modeling techniques
4. Identifying and prioritizing new or changed requirements as they are identified throughout a project

Suggested Resources
Agile Requirements Modeling www.agilemodeling.com/essays/agileRequirements.htm
Agile Requirements Best Practices
www.agilemodeling.com/essays/agileRequirementsBestPractices.htm

2.3 The Analysis and Design Discipline

The goal is to analyze the requirements for the system and to design a solution to be implemented, taking into consideration the requirements, constraints and all applicable standards and guidelines. Critical activities of this discipline include:

1. Formulating, and then defining, a candidate architecture for a system
2. Constructing a proof-of-concept, or spike, to validate a candidate architecture
3. Understanding (analyzing) the requirements for the system
4. Design of components, services, and/or modules
5. Network, user interface, and database design

Suggested Resources
Agile Model Driven Development (AMDD) www.agilemodeling.com/essays/amdd.htm
Agile Architectural Modeling www.agilemodeling.com/essays/agileArchitecture.htm
Extending the RUP with an Enterprise Architecture Discipline
www.enterpriseunifiedprocess.com/essays/enterpriseArchitecture.html

2.4 The Implementation Discipline

The goal is to transform the design into executable code and to perform a basic level of testing, in particular unit testing. Primary activities include:

1. Understanding and evolving the design model
2. Writing program source code
3. Implementing components, services, and/or modules
4. Unit testing source code
5. Integrating the code into subsystems and/or a deployable build

Suggested Resources
Introduction to Test Driven Development (TDD) www.agiledata.org/essays/tdd.html
Refactoring Home Page www.refactoring.com
The Process of Database Refactoring www.agiledata.org/essays/databaseRefactoring.html
Pair Programming Home Page pairprogramming.com

2.5 The Test Discipline

The goal is to perform an objective evaluation to ensure quality. This includes finding defects, validating that system works as designed, and verifying that the requirements are met. Critical activities include:

1. Defining and planning testing efforts
2. Developing test cases
3. Organizing test suites
4. Running tests
5. Reporting defects

Suggested Resources
The Full Lifecycle Object-Oriented Test (FLOOT) Method
www.ambysoft.com/essays/floot.html
Agile Testing www.testing.com/agile/

2.6 The Deployment Discipline

The goal is to plan for the delivery of the system and to execute the plan to make the system available to end users. Activities within this discipline include:

1. Planning the deployment strategy
2. Developing support and operations material
3. Creating deployment packages
4. Organizing alpha/beta/pilot testing efforts
5. Deploying software to installation sites
6. Training end users
7. Managing acceptance testing efforts

Suggested Resources
Development Sandboxes: An Agile Best Practice
www.agiledata.org/essays/sandboxes.html
System Deployment Tips and Techniques www.ambysoft.com/essays/deploymentTips.html
Strategies for Effective Training and Education in IT
www.ambysoft.com/essays/trainingAndEducation.html

2.7 The Configuration and Change Management Discipline

The goal is to manage access to the project's work products. This includes not only tracking versions over time but also controlling and managing changes to them. Critical activities of this discipline include:

1. Managing change requests
2. Planning configuration control
3. Setting up the CM environment
4. Monitoring and reporting configuration status
5. Changing and delivering configuration items
6. Managing baselines and releases

Suggested Resources
Agile Requirements Change Management
www.agilemodeling.com/essays/changeManagement.htm
*SCM Patterns for Agility*www.scmpatterns.com

2.8 The Project Management Discipline

The goal is to direct the activities that take place on the project. This includes managing risks, directing people (assigning tasks, tracking progress, etc.), and coordinating with people and systems outside the scope of the project to be sure that it is delivered on time and within budget. Critical activities include:

1. Initiating a new project
2. Managing project staff
3. Enhancing the relationship with external teams and resources
4. Risk management
5. Estimating, scheduling, and planning
6. Managing an iteration
7. Closing out a phase or project

Suggested Resources
Agile Project Planning Tips www.ambysoft.com/essays/agileProjectPlanning.html
Extending the RUP with the People Management Discipline
www.enterpriseunifiedprocess.com/essays/peopleManagement.html
Scrum Home Page www.controlchaos.com

2.9 The Environment Discipline

The goal is to support the rest of the effort in terms in ensuring that the proper process, guidance (standards and guidelines), and tools (hardware, software, etc.) are available for the team as needed. The critical activities of this discipline are:

1. Tailoring the process materials for an individual project team
2. Identifying and evaluating tools
3. Installing and setting up tools for the project team
4. Supporting the tools and process throughout a project

Suggested Resources
Extending the RUP with the Software Process Improvement Discipline
www.enterpriseunifiedprocess.com/essays/softwareProcessImprovement.html

2.10 Making Iterations Work in Practice

An important concept when working within an iteration is that work does not proceed strictly serially, as it does in a waterfall approach. Yes, you still start with requirements, perform analysis, design and code but you do not have to finalize one before you can move on the next. A more normal approach is to address some subset of the requirements, do some analysis, go back and re-work some of the requirements, move to design, re-work some requirements and/or analysis, start coding, re-work the design. In order for this to work, you need a team that works together on projects. Requirements people need to work with designers who work with coders who work with testers. Of course this works best if the same people can perform requirements analysis, design, coding and test but it's not strictly necessary. The important point is that work proceeds in a serial nature in the broader sense but that work products are not finished and handed off but revised as needed by latter phases.

Iterations are planned according to risks. Higher priority risks are addressed in earlier iterations while lower priority risks are addressed later. This is at the heart of a risk-management approach to developing software.

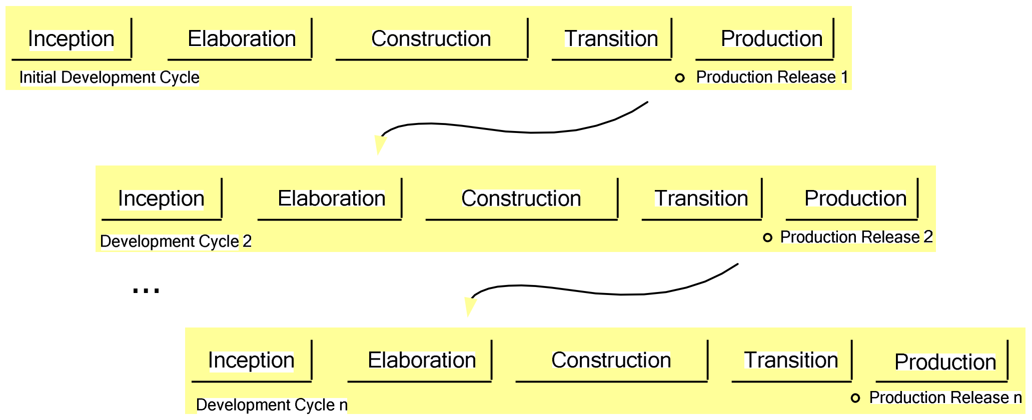
Iterations that are near-term are planned in greater detail than longer-term iterations. The longer-term iterations may change in scope or goal depending on what happens in earlier iterations. Planning for iterations further down the line should be assumed to be of lower accuracy with a higher degree of variance based on current information; information that may change and cause changes to iteration planning. That does not mean that planning for later iterations should be taken lightly. It just means that the best planning based on current data should be done but that things may change before the iteration is realized. That is the reason that longer-term iteration plans are coarse-grained; the situation may very well change so it is often not productive to attempt to plan in detail. It can, in fact, be misleading to stakeholders and others if you attempt to plan work that is far in the future with any specificity.

3 Delivering Incremental Releases Over Time

The first version of a system is usually not the final version. If that were so, every system would remain version 1.0. Instead, systems evolve over time. New functionality is added, user-requested enhancements are added, new standards are adopted and supported, etc. In the RUP a pass through the four RUP phases creates a single version of a system called a production release.

During or after the Transition phase a new project may be started to address any outstanding requirements. The new project may begin at the Inception phase again although some teams may decide to start in the Elaboration phase or even the Construction phase if appropriate. As Figure 3 depicts this can continue indefinitely, as long as new requirements are identified that the stakeholders agree are worthy of a new version of the software.

Figure 3. The incremental release of a system into production.



4 Following Proven Best Practices

In the past the original best practices (develop iteratively, manage requirements, use component architecture, model visually (UML), continuously verify quality, and manage change) reflected software development issues. In October 2005 IBM announced a revised list of best practices for the RUP which takes into account the overall "business picture" of continuously evolving systems (Kroll and Royce, 2005). These best practices are:

1. Adapt the process.
2. Balance competing stakeholder priorities.
3. Collaborate across teams.
4. Demonstrate value iteratively.
5. Elevate the level of abstraction.
6. Focus continuously on quality.

4.1 Adapt the Process

This practice reflects one of my fundamental philosophies, that you need to follow the right process for the job. Every person, project team, and organization is different as is the environment in which they work. One process size does not fit all, instead you need to tailor your software process to meet your exact needs. If you try to adopt RUP "straight out of the box" you are very likely going to fail – there is far more material in the RUP than any project team needs, and you will often discover that the RUP will be missing a few things which are pertinent to your situation. You must adapt the RUP appropriately.

Suggested Resources
Choose the Right Software Method for the Job
www.agiledata.org/essays/differentStrategies.html
Extending the RUP with a Software Process Improvement Discipline
www.enterpriseunifiedprocess.com/essays/softwareProcessImprovement.html

4.2 Balance Competing Stakeholder Priorities

Any development project will have a variety of stakeholders – end users, business management, operations staff, enterprise architects, external customers, and so on – and they will have different needs and priorities. One of the hardest parts of software development is to balance those priorities, particularly when they often change as your project progresses. By taking an evolutionary approach to development where your stakeholders are active participants you are much more likely to achieve your goal of building high-quality, working software which meets your stakeholders needs.

Suggested Resources
Active Stakeholder Participation
www.agilemodeling.com/essays/activeStakeholderParticipation.htm
Agile Requirements Change Management
www.agilemodeling.com/essays/changeManagement.htm

4.3 Collaborate Across Teams

Software is developed by talented, motivated people who communicate and collaborate effectively. To achieve this successfully, you need to motivate people to not only do their best, but also to actively learn new skills from their co-workers and other sources (such as training classes, books, and web sites). My experience is that the best developers are generalizing specialists, people with one or more specialties (such as writing Java code or design modeling), a broad understanding of software development in general, and an understanding of the domain in which they work. With this broad understanding of development and the domain, generalizing specialists are better suited than specialists to collaborate effectively with other people, thereby not only getting the job done but usually learning new skills in the process.

To foster effective collaboration, your organization must provide effective tools and working environment. Although nice offices and private cubicles are nice, they effective erect a barrier to communication between people and hamper their ability to collaborate effectively. A better strategy is to co-locate both developers and stakeholders together, or at least in very close proximity, to foster communication. You should also adopt tools and practices which foster sharing of information, including effective configuration management tools and the practice of shared work products across the team (the team "owns" a document, not an individual).

Suggested Resources
Communication on Agile Software Projects
www.agilemodeling.com/essays/activeStakeholderParticipation.htm
Generalizing Specialists
www.agilemodeling.com/essays/generalizingSpecialists.htm
Collective Ownership www.agilemodeling.com/practices.htm#CollectiveOwnership

4.4 Demonstrate Value Iteratively

The fundamental idea is that you want to reduce the feedback cycle by delivering working software early and regularly. You do this by organizing your project into iterations, see Section 2.10, where you develop a small portion of the system at a time. During the Elaboration phase you drive out critical technical risks by developing a working skeleton early in the lifecycle.

During the Construction phase you deliver, at least internally, working software which completely fulfills a portion of your stakeholder's requirements: software which they can evaluate and provide feedback on. Some of this feedback may take the form of new or changed requirements: It's common to hear "Oh, I forgot about…" or "That really isn't what I meant, instead…" from your stakeholders. Traditionalists balk at this, decrying the danger of "scope creep". Effective development teams embrace and manage change, they understand that a changed requirement late in the lifecycle provides a competitive advantage if you're able to act on it. Isn't it better to seek feedback early, and then adjust your plans accordingly to ensure that you deliver what stakeholders actually need?

4.5 Elevate the Level of Abstraction

Software development is hard, and growing harder all the time. Effective development teams raise the level of abstraction by adopting modeling tools, by reusing existing work products, and by focusing on architecture to think through the big issues early in the project.

Suggested Resources
Agile Architectural Modeling www.agilemodeling.com/essays/agileArchitecture.htm
Extending the RUP with the Strategic Reuse Discipline
www.enterpriseunifiedprocess.com/essays/strategicReuse.html
Types of Reuse in Information Technology www.ambysoft.com/essays/typesOfReuse.html

4.6 Focus Continuously on Quality

The entire team is responsible for quality, not just the testing team. This is one of the primary reasons why testing and validation is so prevalent throughout the RUP – every discipline includes reviews, either formal or informal, of the generated work products and testing activities are critical to both the Implementation and Test disciplines. Quality is so important to agile developers that we take a Test Driven Development (TDD) approach to implementation where we write a unit test before we write enough production code to fulfill that test, and then we refactor our code as needed to ensure that it remains the highest quality possible.

Suggested Resources
The Full Lifecycle Object-Oriented Test (FLOOT) Method
www.ambysoft.com/essays/floot.html
Introduction to Test Driven Development (TDD) www.agiledata.org/essays/tdd.html
The Process of Database Refactoring www.agiledata.org/essays/databaseRefactoring.html
Refactoring Home Page *www.refactoring.com*

5 Why the RUP?

Fundamentally, the RUP has become a de facto industry standard for prescriptive software processes. The RUP works, it's here to stay, and Rational Method Composer (RMC) contains most and very likely more information than you require to define your own software process. The RUP is based on best practices gleaned from many years of experience on many projects. It has been successfully applied by many organizations in many domains. A growing number of professionals are proficient with the RUP, making it easier to find people with RUP expertise. It doesn't make sense to develop your own process when such a wealth of proven material already exists.

The RUP as a system development process framework which can be tailored to meet your specific needs. Using the RUP as a base, organizations can pick and choose the portions they wish to use and add new items particular to their environment. Augmentations to the RUP (called "plug-ins") are developed and shared by many organizations to speed the adoption of a particular aspect of software development or technology.

The RUP's iterative and incremental approach has several advantages over serial strategies:

1. **Improved governance**. True earned value, the delivery of high quality working software which meets the actual needs of stakeholders, is delivered early and regularly following the RUP. RUP teams produce tangible results, working software, on a regular basis. You can easily determine if a RUP team is on track or not, and redirect them if required.
2. **Regular feedback to stakeholders.** Stakeholders can see portions of the system sooner and receive assurances that they will receive what they want. If they don't like what they see they can provide course corrections at a much earlier point than with the waterfall approach.
3. **Improved risk management**. Working incrementally allows higher risks to be addressed early. If there is a question about whether or not a requirement can be met or a technical challenge can be overcome, it can be addressed in an early iteration. If it cannot be implemented or can be implemented but in a manner which does not meet the stakeholders' needs, the project can be refocused or cancelled outright.
4. **You implement the actual requirements**. Change is inevitable. Expecting to define requirements once at the beginning of a project and to not have to change them is unrealistic, as is expecting to design an entire system before writing a line of code. By developing systems in smaller iterations you can react to any changes and thereby build software which meets the actual needs of your stakeholders instead of their perceived needs which were documented months or years earlier. Changes in requirements that impact later iterations do not impact the work being done on the current iteration. In addition, changes to requirements within the current iterations are easier to deal with because the scope of requirements for each iteration is smaller. Changes to previous iterations are simply scheduled as new requirements in future iterations.
5. **You discover what works early**. The goal of the Elaboration phase is to ensure that your architecture works early in your project. Every architecture works on paper, but many don't work in practice. By developing the skeleton for your system during Elaboration you mitigate much of the technical risk on your project. By developing the

system in iterations you regularly discover that your architecture and design continues to meet the changing needs of your stakeholders.

6. **Developers focus on what matters**. When the RUP is instantiated effectively developers discover that they spend more time doing things that matter, actual software development. With traditional approaches a team will often spend months modeling, writing documentation, reviewing their work, waiting for acceptance, reworking things, and developing detailed plans which inevitably change before they get to write a single line of code. Shouldn't the focus of software development be software development, not bureaucracy? When you instantiate the RUP intelligently – remember that there's no guarantees, you could still do something stupid – then your overall IT productivity improves dramatically.

6 The RUP as a Process Framework

The RUP is not only a system development process but also a system development process framework. That is to say, it is a structure from which a process can be created. It was never intended to be a silver bullet that organizations should apply "as is". Indeed, the RUP calls for tailoring of the process to be done for each project to address its particular needs. IBM Rational clearly advocates that organizations customize it to create a process that is specific to meets their particular needs.

IBM Rational has made, and continues to make, a significant investment in the RUP and in RMC. The RUP is a framework from which you can tailor a process which meets your organization's needs. The RUP defines a comprehensive description of the roles, activities, procedures, guidelines, templates, and so on which are consistent and interrelated – in short it is an incredible resource which your organization should consider taking advantage of. A side benefit of the RUP is that because of its popularity within the IT industry many people are familiar with, if not expert at it. The implication is that hiring and training new employees can become easier. As you would expect, IBM Rational also supports tools for tailoring the RUP, including RMC and the Eclipse Process Framework (EPF) (Eclipse 2005).

This customization of the RUP entails the modifications that an organization makes to address their specific needs[2]. While the main concepts remain the same, the specifics of the process vary somewhat from customization to customization. This allows organizations to tailor the process to meet their specific business priorities, personnel skillsets, and philosophical – or cultural – approach. For example, if an organization develops shrink-wrapped software, it would tailor the deployment aspects to reflect the manufacturing of software and documentation; an IT organization that only develops for internal use doesn't worry about that aspect. An organization

[2] See http://www.enterpriseunifiedprocess.com/essays/softwareProcessImprovement.html for a detailed discussion of approaches to tailoring the RUP.

may also, for example, decide to do more (or less) business modeling than is typical for a RUP implementation. The RUP also allows for the addition (or deletion) of phases and/or disciplines.

Each organization must decide how much of the process to implement and which roles, activities and work products that they will use. It is not unusual for organizations to have several tailorings of the RUP for use in the different types of projects that they normally execute. This reflects the reality that not only do process needs differ across organizations but also across projects as well.

Two publicly available tailorings that I am responsible for are:

1. **The Agile Unified Process (AUP)**. Visit www.ambysoft.com/unifiedprocess/agileUP.html for details.
2. **The Enterprise Unified Process (EUP)**. This is a tailoring of the RUP that is best for organizations with multiple systems and multiple development teams. The EUP adds new disciplines and phases to the standard RUP, making it more effective for these types of organizations. Visit www.enterpriseunifiedprocess.com for details.

7 Concluding Remarks

The Rational Unified Process is a product offered by IBM Rational. It is a system development process which is serial in the large, iterative in the small, delivering incremental releases over time, while following proven best practices. It consists of four phases: Inception, Elaboration, Construction and Transition, which execute sequentially. Work is grouped into logical activities called disciplines, which are performed iteratively throughout the four phases. The RUP is not only a system development process but also a system development process framework that can be tailored to meet organizations individual needs.

References and Suggested Reading

Ambler, S.W. (1999b). *Enhancing the Unified Process*. Software Development, October 1999. www.sdmagazine.com/documents/s=753/sdm9910b/9910b.htm
Ambler, S.W. (2002). *Agile Modeling: Best Practices for the Unified Process and Extreme Programming*. New York: John Wiley & Sons. www.ambysoft.com/agileModeling.html
Ambler, S.W. (2003b). *Agile Database Techniques: Effective Strategies for the Agile Software Developer*. New York: John Wiley & Sons. www.ambysoft.com/agileDatabaseTechniques.html
Ambler, S.W. (2004). *The Object Primer 3rd Edition: Agile Model Driven Development with UML 2*. New York: Cambridge University Press. www.ambysoft.com/theObjectPrimer.html.
Ambler, S.W. (2005). *The Agile Unified Process*. www.ambysoft.com/unifiedprocess/agileUP.html
Ambler, S.W., Nalbone, J., and Vizdos, M. (2005). *The Enterprise Unified Process: Extending the Rational Unified Process*. Upper Saddle River, NJ: Prentice Hall PTR. www.ambysoft.com/enterpriseUnifiedProcess.html
Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley Longman, Inc.
Beck, K. (2003). *Test Driven Development: By Example*. Boston, MA: Addison-Wesley.
Evans, G. (2001). *Palm-Sized Process*. Software Development, September 2001.
Githens, G.D. (1998). *Rolling Wave Project Planning*. www.catalystpm.com/NP02.pdf
IBM (2004). *Rational Unified Process Home Page*. www-306.ibm.com/software/awdtools/rup/support/
Eclipse (2005). *The Process Framework (EPF) Project <Beacon>*. www.eclipse.org/proposals/beacon/
ITIL (2003). *The ITIL and ITSM Directory*. www.itil-itsm-world.com
Jacobson, I., Booch, G., and Rumbaugh, J. (1999). The *Unified Software Development Process*. Reading, MA: Addison-Wesley Longman, Inc.
Kroll, P. (2005). *Introducing Rational Method Composer*. http://www-128.ibm.com/developerworks/rational/library/nov05/kroll/index.html
Kroll, P. and Royce, W. (2005). *Key Principles for Business-Driven Development*. www-128.ibm.com/developerworks/rational/library/oct05/kroll/index.html
Kruchten, P. (2004). *The Rational Unified Process 3rd Edition: An Introduction*. Reading, MA: Addison-Wesley Longman, Inc.
Royce, W. (1998). *Software Project Management: A Unified Framework*. Reading, MA: Addison-Wesley Longman, Inc.
Williams, L., and Kessler, R. (2002). *Pair Programming Illuminated*. Boston, MA: Addison-Wesley.

About the Author

Scott W. Ambler is software process improvement (SPI) consultant living just north of Toronto. He helps organizations adopt and tailor software processes to meet their exact needs as well as provides training and mentoring in various techniques. He is the founder and practice leader of the following methods:
- Agile Data (AD) (www.agiledata.org)
- Agile Modeling (AM) (www.agilemodeling.com)
- Agile Unified Process (AUP) (www.ambysoft.com/unifiedprocess)
- Enterprise Unified Process (EUP) (www.enterpriseunifiedprocess.com)

Scott is the (co-)author of several books, including *Agile Modeling* (John Wiley & Sons), *Agile Database Techniques* (John Wiley & Sons), *The Object Primer 3rd Edition* (Cambridge University Press), *The Enterprise Unified Process* (Prentice Hall), and *The Elements of UML 2.0 Style* (Cambridge University Press). Scott is a contributing editor with *Software Development* magazine (www.sdmagazine.com) and a full list of his books is presented at www.ambysoft.com/booksAmbler.html.

Scott's personal page is www.ambysoft.com/scottAmbler.html and his writings are linked to at www.ambysoft.com/onlineWritings.html, many of which are published in full on the web. In his spare time Scott studies the Goju Ryu and Kobudo styles of karate. Scott has spoken and keynoted at a wide variety of international conferences including Software Development, UML World, Object Expo, Java Expo, and Application Development. Scott graduated from the University of Toronto with a Master of Information Science.