A Review of RUP (Rational Unified Process)

ASHRAF ANWAR, PhD *prof1aa@yahoo.com*
College of Arts & Sciences
Department of Computer Science & MIS
University of Atlanta
Atlanta, GA, USA

Abstract

RUP (Rational Unified Process) is an iterative process for software development; originally proposed in 1988 as the Rational Objectory Process. Wennerlund then proposed the Unified Process [1]. The Rational Unified Process can also be regarded as a software engineering process, delivered through a web-enabled, searchable knowledge base [2] & [3]. This paper represents an overview of Rational Unified Process; its history, and practices involved; stressing its advantages and disadvantages. A comparison with other approaches is mentioned in context; whenever appropriate.

Keywords: EUP, OOSP, RUP, SDLC, Software Engineering, UML.

1. INTRODUCTION

The RUP; Rational Unified Process, is an iterative software development process framework; created by the Rational Software Corporation, a division of IBM [4] & [5] & [6].

RUP is an Object-Oriented, and Web-Enabled system development methodology. According to Rational; developers of Rational Rose and the Unified Modeling Language, RUP is like an online mentor that provides guidelines, templates, and examples for all aspects and stages of program development. RUP and similar products -- such as Object-Oriented Software Process (OOSP), and the OPEN Process -- are comprehensive software engineering tools that combine the procedural aspects of development (such as defined stages, techniques, and practices) with other components of development (such as documents, diagrams, models, manuals, help files, code samples, final source code, etc…) within a unifying framework.

EUP (Enterprise Unified Process) is a variant of RUP tailored for enterprise-level disciplines [7] & [8]. EUP extends the SDLC (Systems Development Life Cycle) classic methodologies [9]; by addressing enterprise-level factors. Such factors are especially important when dealing with huge projects in big enterprises.

2. RUP DEFINITION & HISTORY

2.1 Definition of RUP

The Rational Unified Process® (RUP) is a Software Engineering Process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget. The Rational Unified Process is a process product, developed and maintained by Rational® Software.

The development team for the RUP is working closely with customers, partners, Rational product groups, as well as Rational consultant organization; to ensure that the process is continuously updated and improved upon; to reflect recent experiences, evolving practices, and proven best practices. RUP enhances team productivity, by providing every team member with easy access to

a knowledge base with guidelines, templates, and tool mentors for all critical development activities. By having all team members accessing the same knowledge base, no matter if you work with requirements, design, test, project management, or configuration management; they ensure that all team members use *Unified Process* activities to create and maintain models. Rather than focusing on the production of large amount of paper documents, the Unified Process emphasizes the development and maintenance of models; semantically rich representations of the software system under development.

The Rational Unified Process is a guide for how to effectively use the Unified Modeling Language (UML). The UML is an industry-standard language that allows us to clearly communicate requirements, architectures and designs. The UML was originally created by Rational Software, and is now maintained by the standards organization Object Management Group (OMG). The Rational Unified Process is supported by tools, which automate large parts of the process. Those tools are used to create and maintain the various artifacts---models in particular---of the software engineering process: visual modeling, programming, testing, etc. They are invaluable in supporting all the bookkeeping associated with change management, and configuration management; that accompanies iterations in RUP.

RUP is a configurable process. No single process is suitable for all sorts of software development. The Unified Process fits small development teams; as well as large development organizations. The Unified Process is founded on a simple and clear process architecture that provides commonality across a family of processes. Yet, it can be varied to accommodate different situations. It contains a Development Kit, providing support for configuring the process to suit the needs of a given organization. The RUP captures many of the best practices in modern software development in a form that is suitable for a wide range of projects and organizations.

2.2 History of RUP

RUP has matured over many years and reflects the collective experience of the many people and companies that make up Rational Software's rich heritage today. See *Figures 1 and 2* below.

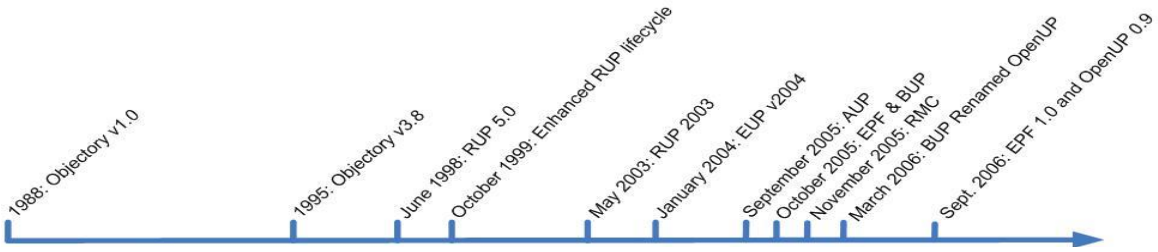

FIGURE 1: Genealogy of RUP; Redrawn From [10].

Having a quick look at the process ancestry, as illustrated in *Figure 2* below, "RUP Evolution", we can note some facts:

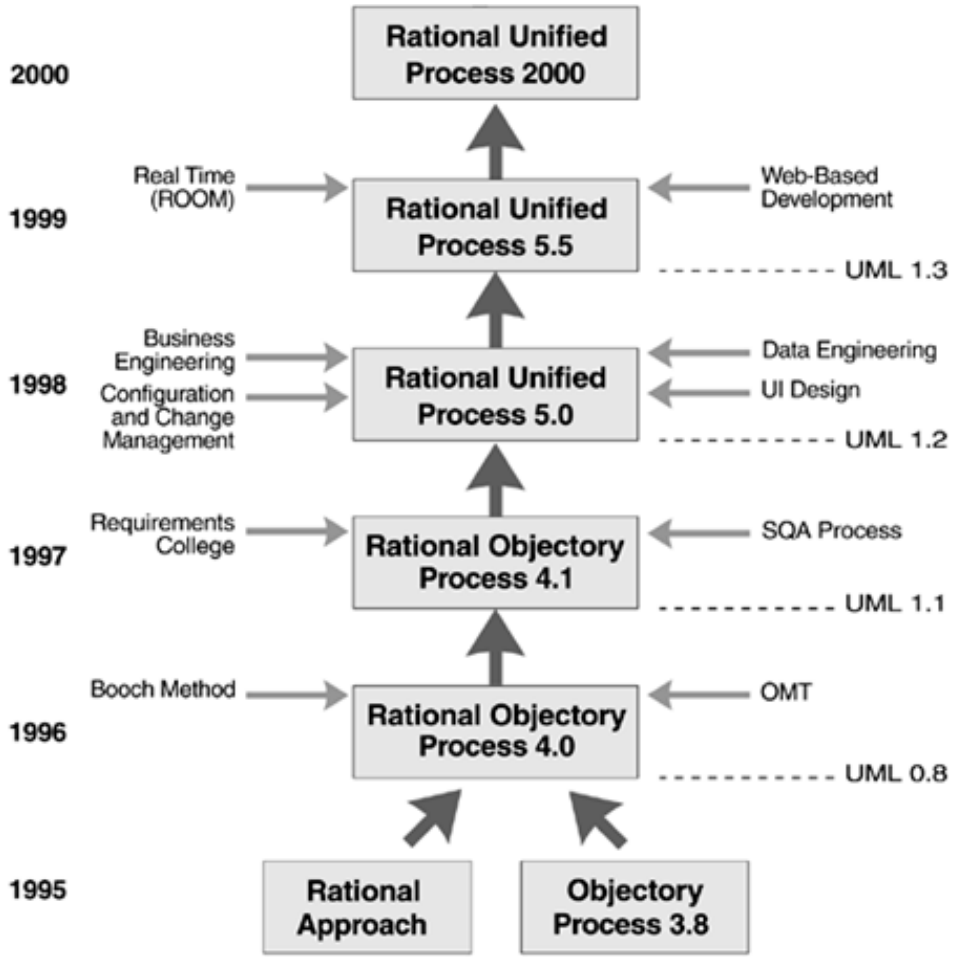

FIGURE 2: RUP Evolution; 1995-Present.

- For Rational Unified Process: Best Practices for Software Development Teams, and going backwards in time; the Rational Unified Process is the direct successor to the Rational Objectory Process, version 4; released in 1996 [10].
- RUP incorporates more material in the areas of data engineering, business modeling, project management, and configuration management (as a result of the merger with PureAtria).
- It also brings a tighter integration to the Rational Software suite of tools.
- The Rational Objectory Process was the result of the integration of the "Rational Approach" and the Objectory Process (version 3), after the merger of Rational Software Corporation and Objectory AB in 1995.
- From its Objectory ancestry, the process has inherited its process structure and the central concept of use cases.
- From its Rational background, the process gained the current formulation of iterative development and architecture.
- This version also incorporated material on requirements management from Requisite Inc. and a detailed test process inherited from SQA Inc., which also merged with Rational Software.
- RUP was the first to use the then newly created Unified Modeling Language (UML 0.8).
- The Objectory process was created in Sweden in 1987 by Ivar Jacobson as the result of his experience with Ericsson.
- This process became a product at Jacobson's company; Objectory AB.
- Centered around the concept of use cases and an object-oriented design methodology, RUP rapidly gained recognition in the software industry and has been adopted and integrated by many companies worldwide.

- A simplified version of the Objectory process was published as a text book in 1992.
- RUP is a specific and detailed instance of a more generic process described by Ivar Jacobson and others [11].

3. RUP PERSPECTIVES

There are three perspectives of the Rational Unified Process. The *dynamic* perspective shows the RUP phases over time. The processes shown in this perspective are dynamic i.e. they are constantly changing. The *static* perspective shows the statics aspects of the RUP phrases. This perspective is made of things that do not change themselves but work to change the dynamic processes. The *practice* perspective is made of the good practices used during the process. Those are practices well known of their effectiveness and usefulness through previous work and experience.

3.1 Dynamic Perspective & Lifecycle Phases

RUP – as shown in *Figure 3* below-; is described along two dimensions:

a) The horizontal axis represents time and shows the dynamic aspects of the process. It is expressed in terms of cycles, phases, iterations and milestones.
b) The vertical axis shows the static aspects of the process. It is expressed in terms of activities, artifacts, workers and workflows.

A RUP project has four phases: *inception, elaboration, construction, and transition;* as in *Figure 4* below. Each phase has one or more iterations and is completed with a milestone. At that point, the project progress is assessed, and major decisions are made based on that progress. The focus of the iterations in each phase is to produce technical deliverables that will fulfill the phase objectives.

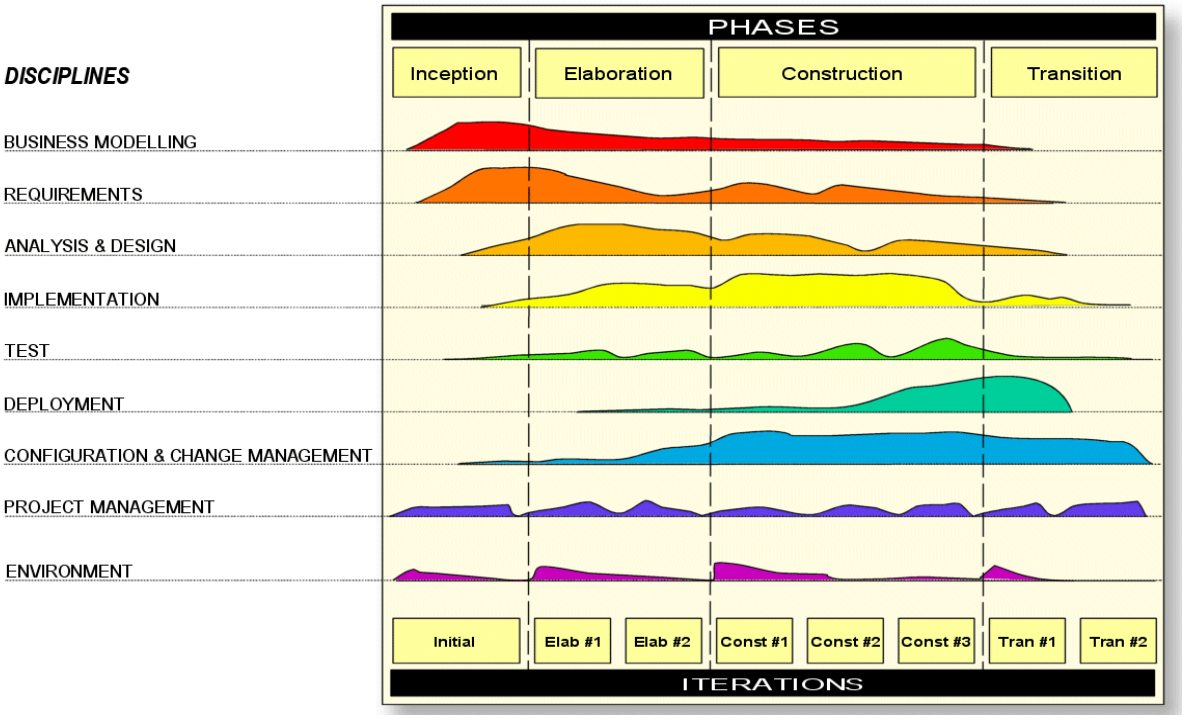

FIGURE 3: A hump chart that illustrates RUP architecture; redrawn from [10].

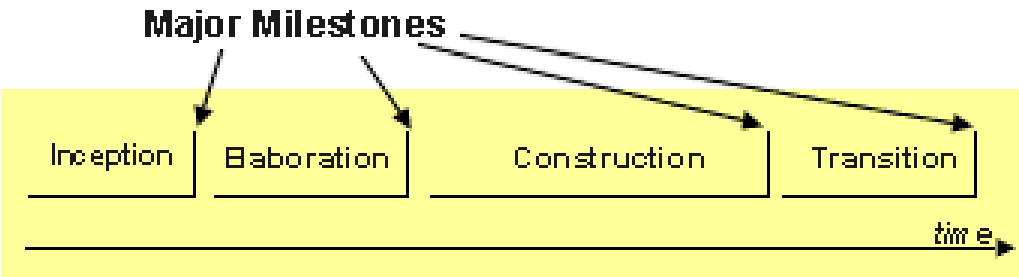

FIGURE 4: Milestones at End of Phases in RUP.

3.1.1 Inception

This phase focuses on a project's launch. The project has to be validated; so that a budget can be approved. A business case for the project is produced which includes the business environment, success factors (such as expected revenue, or return on investment: ROI) and financial forecast [12]. In addition, a project plan, initial risk assessment and project description (core requirements, constraints, key features, cost/schedule estimates) are produced. All the objects that the system will interact with "actors" are identified, and the nature of their interaction is defined. At this stage, full commitment is required from the project's stakeholders. After everything has been prepared, the project team should be able to decide whether to proceed or not. This is somewhat similar to approving a feasibility study [9].

- The stakeholders should agree; at least to a large extent, with the information that is presented to them.
- Requirements should be thoroughly understood.
- Cost and schedule estimates, and risks should be credible.
- A vision document should be prepared i.e. a document with the project's requirements, key features and main constraints. This is similar to system requirements and specifications (SRS) document [9].
- An initial (and very basic) use-case model should be prepared (on average 10-20% completed)
- A baseline should be established at this point which can be used to compare actual expenditures to planned expenditures.

This is the first milestone in the project, and is known as the "Lifecycle Objective Milestone". If the project does not pass that milestone, it can either be cancelled or it can be redesigned so that it can fulfill the constraints and criteria missing.

3.1.2 Elaboration

This phase focuses on addressing the project's major technical risks. The purpose of this phase is to create a basic system (the architecture of the system) which will answer major technical questions; i.e. this stage is where the project starts to take shape. At the end of this phase, the project team should be able to decide whether they can build a successfully working system. By the end of this phase, the following should be complete:

- A use-case model (to show the functionality of the system). Use-cases and actors should be identified and the model itself should be approximately 80% complete.
- A description of the software architecture.
- The business case and risks associated should be revised.
- A development plan for the overall project.
- Prototypes that lessen the technical risks that were identified in the previous stage.

This is the second milestone, and is known as the Lifecycle Architecture Milestone. If the project is unable to pass this milestone, it can either be cancelled or redesigned. This phase is probably the most critical because this is where a major decision is taken (whether to commit to the project

or not). A decision to proceed should be taken carefully because after this stage, the project become a high-risk operation where changes are harder to make, and could have a damaging effect.

3.1.3 Construction

This phase focuses on making an operational system by using the executable architecture that was created in the previous phase. The main focus is on developing components and other features. Most of the coding is done in this phase; as well as extensive testing. In large projects, there may be several construction phases; this is helpful as it allows the use cases to be divided into smaller, more manageable segments. At the end of this stage, the project team should have user manuals and a beta version of the system ready to be evaluated. The transition stage can be delayed if the product is not stable enough to be tested by end users, or if the actual resource expenditures are not acceptable when compared to planned expenditures.

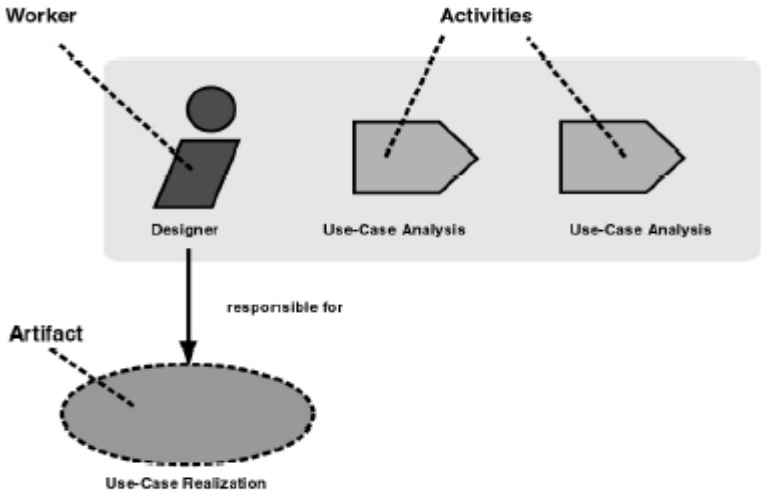

FIGURE 5: RUP Activities, Artifacts, and Workers.

3.1.4 Transition

This phase focuses on transitioning the system from the development stage to actual production; somewhat similar to deployment in traditional SDLC [9]. The system must fulfill the software requirements and needs of its users; so in this stage, the system is evaluated and refined based on feedback from the users. This is only to polish the system because it should already meet the users' needs. This phase also includes training end users. The product is also checked against the quality standards that were set in the inception phase. At the end of this phase, the system is released and the project is evaluated.

This is the last milestone and is known as the Product Release Milestone. This phase concludes the RUP software development cycle.

Each phase will include some or all of the development workflows (disciplines). However, the amount of work in each workflow will differ in each phase. For example, in the inception phase, most of the work will focus on business modeling and obtaining requirements. But in the construction phase, the focal point will be implementation.

3.2 Static Perspective

The Rational Unified Process model is built on three entities; with respect to its static perspective: roles (workers), activities, and artifacts; as in *Figure 5*. Workflows relate activities, artifacts, and roles in sequences that produce valuable results.

A process describes *who* is doing *wha*t, *ho*w, and *when*. The Rational Unified Process is represented using four primary modeling elements:

- Workers, the 'who'
- Activities, the 'how'
- Artifacts, the 'what'
- Workflows, the 'when'

3.2.1 Workers

A ***worker*** defines the behavior and responsibilities of an individual, or a group of individuals working together as a team. We could regard a worker as a "hat" an individual can wear in the project. One individual may wear many different hats. This is an important distinction; because it is natural to think of a worker as the individual or team itself. But in the Unified Process the worker is more the role defining how an individual should carry out the work. The responsibilities we assign to a worker include both to perform a certain set of activities; as well as being owner of a set of artifacts.

3.2.2 Activities

An ***activity*** of a specific worker is a unit of work that an individual in that role may be asked to perform. The activity has a clear purpose, usually expressed in terms of creating or updating some artifacts, such as a model, class, or plan.

Every activity is assigned to a specific worker. The granularity of an activity is generally a few hours to a few days; it usually involves one worker, and affects one or only a small number of artifacts. An activity should be usable as an element of planning and progress; if it is too small, it will be neglected, and if it is too large, progress would have to be expressed in terms of an activity's parts.

Examples of activities include:

- Plan an iteration; for the Worker: Project Manager
- Find use-cases and actors; for the Worker: System Analyst
- Review the design; for the Worker: Design Reviewer
- Execute performance test; for the Worker: Performance Tester

Figure 6 below shows examples of activities performed by certain individuals in a real-life project. Notice that Mary; the second actor, is the Use-Case creator and worker. Her activity is to fill in the details of Use-Cases.

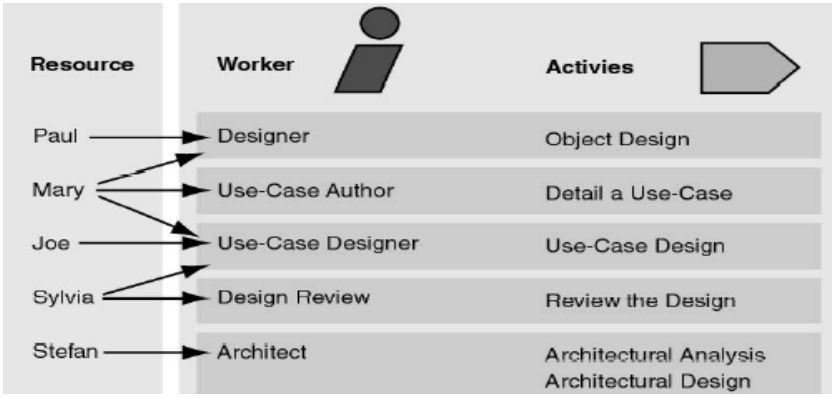

FIGURE 6: Example of People and Workers in RUP.

3.2.3 Artifacts

An ***artifact*** is a piece of information that is produced, modified, or used by a process. Artifacts are typically the tangible products of the project; the things the project produces or uses while working towards the final product. Artifacts are used as input by workers to perform an activity, and are the result or output of such activities. In object-oriented design terms; as activities are operations on/by an active object (the worker), artifacts are the parameters of these activities.

Artifacts may take various shapes or forms; examples of artifacts are:

- A model, such as the Use-Case Model or the Design Model
- A model element, i.e. an element within a model such as a class, a use case or a subsystem
- A document, such as Business Case or Software Architecture Document
- Source code
- Executable

3.2.4 Workflows

A mere enumeration of all workers, activities and artifacts does not quite constitute a process. We need a way to describe meaningful sequences of activities that produce some valuable results, and to show interactions between workers.

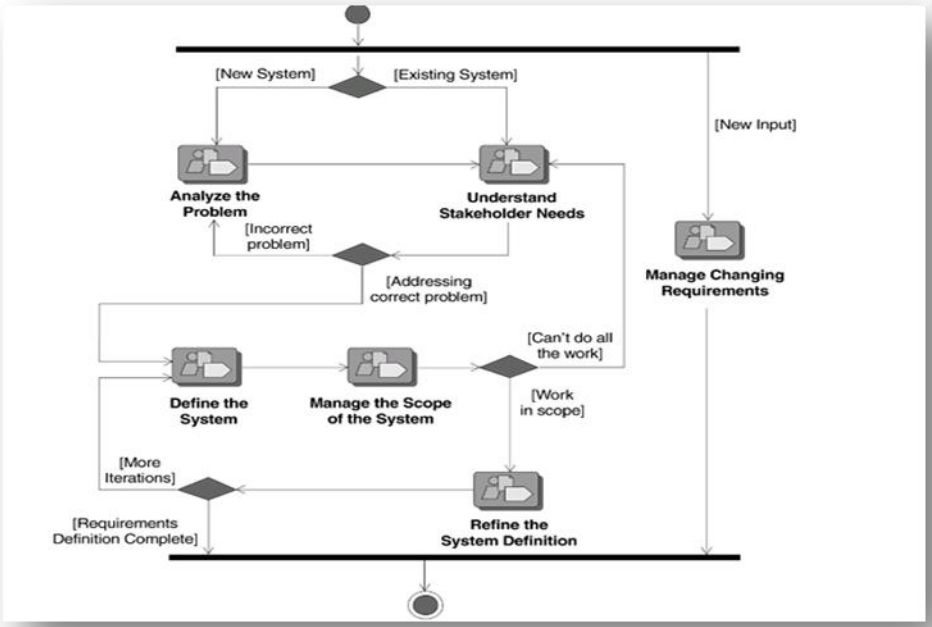

FIGURE 7: Requirements Workflow; Redrawn From [12].

A workflow is a sequence of activities that produces a result of observable value; i.e. a useful outcome, or deliverable [13]. In UML terms, a workflow can be expressed as a sequence diagram, a collaboration diagram, or an activity diagram. The diagram below shows a typical Requirements Workflow for RUP.

3.3 Practice Perspective & the Core Process

Although the names of the six core engineering workflows may evoke the sequential phases in a traditional waterfall process, we should keep in mind that the phases of an iterative process are different and that these workflows are revisited again and again throughout the lifecycle. The

actual complete workflow of a project interleaves these nine core workflows, and repeats them with various emphasis and intensity in each iteration.

The core process workflows are divided into six core “engineering” workflows:

1. Business modeling workflow
2. Requirements workflow; *Figure 7* above
3. Analysis & Design workflow
4. Implementation workflow
5. Testing workflow
6. Deployment workflow

Figure 8 below shows an example of workflow with actors involved.

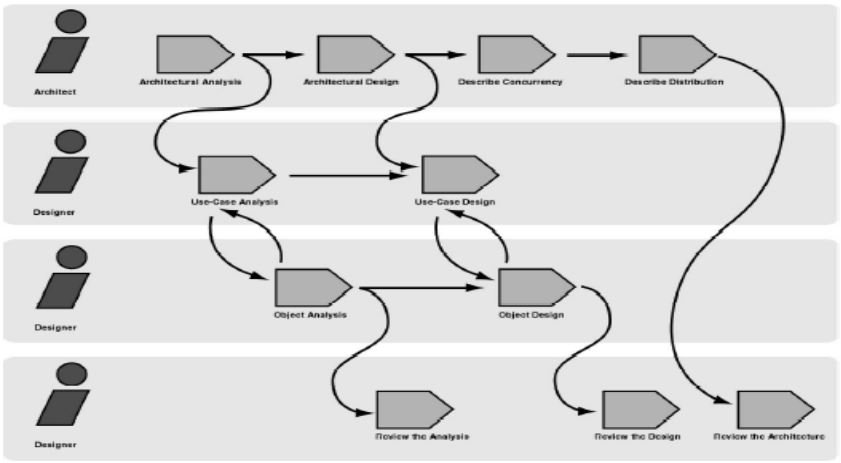

FIGURE 8: Example of a Typical Workflow.

In addition, we have three core “supporting” disciplines or workflows:

1. Environment workflow
2. Configuration and Change Management workflow
3. Project Management workflow

3.3.1 Business Modeling Workflow

Business modeling explains how to describe a vision of the organization in which the system will be deployed and how to then use this vision as a basis to outline the process, roles and responsibilities.

Organizations are becoming more dependent on Information Systems (IS), and Information Technology (IT); making it imperative that information system engineers know how the applications they will develop fit into the organization business needs [13] & [9]. Businesses invest in IT when they understand the competitive advantage and value added by the technology. The aim of business modeling is to first establish a better understanding and communication channel between business engineering and software engineering. Understanding the business means that software engineers must understand the structure and the dynamics of the target organization (the client), the current problems in the organization and possible improvements [13]. They must also ensure a common understanding of the target organization between customers, end users and developers.

One of the major problems with most business engineering efforts is that the software engineering and the business engineering community do not communicate properly with each other. This leads to the output from business engineering not being used properly as input to the software development effort, and vice-versa.

The Rational Unified Process addresses this by providing a common language and process for both communities, as well as showing how to create and maintain direct traceability between business and software models [14].

In business modelling, we document business processes using so called business use cases. This assures a common understanding among all stakeholders of what business processes that need to be supported in the organization. The Rational Unified Process business use cases are analyzed to understand how the business should support the business processes; partly through using Best Practices for Software Development Teams. This support is documented in a business object-model. However, many projects may choose not to do business modeling; though.

3.3.2 Requirements Workflow

This discipline explains how to elicit stakeholder needs and requests to transform them into a set of requirements-depicting-work products that scope the system to be built and provide detailed requirements for what the system must do.

The goal of the *Requirements* workflow is to describe what the system should do and allows the developers and the customers to agree on that description. To achieve this, we elicit, organize, and document required functionality and constraints; track and document tradeoffs and decisions.

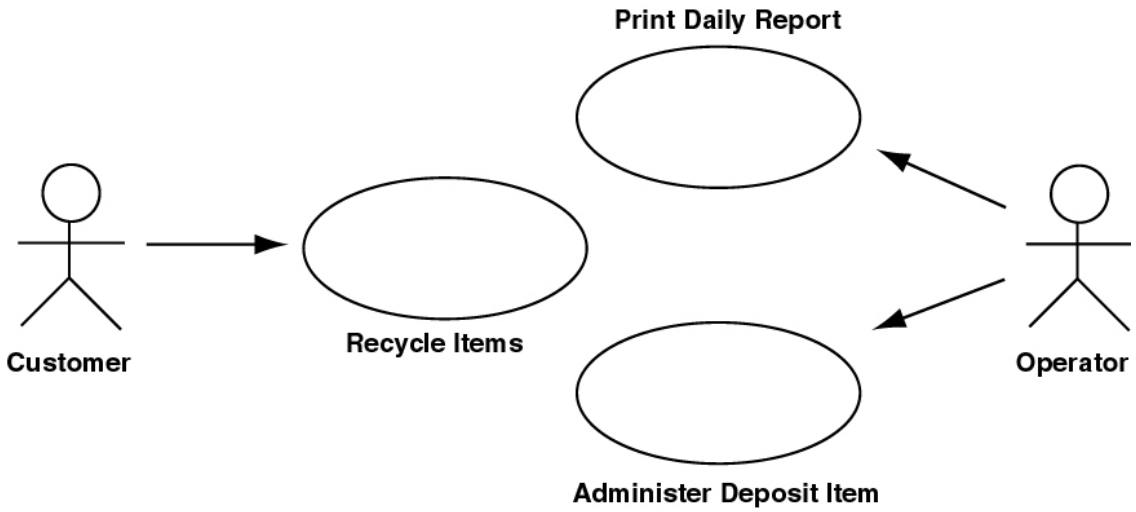

FIGURE 9: An example of use-case model with actors and use cases.

A Vision document is created, and stakeholder needs are elicited. Actors are identified, representing the various users, and any other system that may interact with the system being developed. Use-Cases are identified; representing the behavior of the system. Because use cases are developed according to the actor's needs, the system is more likely to be relevant to the users. *Figure 9* shows an example of a use-case model for a recycling-machine system.

3.3.3 Analysis and Design Workflow

The goal of analysis and design is to show how the system will be realized based on requirements [9] & [16]. The aim is to build a system that:

- Performs — in a specific implementation environment — the tasks and functions specified in the use-case descriptions.
- Fulfills all its requirements; functional and non-functional.
- Is easy to change when requirements change; i.e. agility observed [13].

Design phase results in a design model, and analysis phase optionally results in an analysis model. The design model serves as an abstraction of the source code; that is, the design model acts as a 'blueprint' of how the source code will be structured and written. The design model consists of design classes structured into packages and subsystems with well-defined interfaces, representing what will become components in the implementation. It also contains descriptions of how objects of these design classes collaborate and interact to perform use cases.

3.3.4 Implementation Workflow

The purposes of implementation are:

- To define the organization of the code, in terms of implementation subsystems organized in layers.
- To implement classes and objects in terms of components (source files, binaries, executables, and others).
- To test the developed components as units. An approach as Extreme Programming may well suit this [13].
- To integrate the results produced by individual implementers (or teams) into an executable system. Integration and interfacing tests and used here.

Systems are realized through implementation of components. The process describes how to reuse existing components, or implement new components with well-defined responsibilities, making the system easier to maintain, and increasing the possibilities of their reuse.
Components are structured into Implementation Subsystems. Subsystems could be regarded as, or take the form of subdirectories, with additional structural or management information in them. For example, a subsystem can be created as a subdirectory or a folder in a file system, a subsystem in Rational/Apex for C++, or a package for Java.

3.3.5 Testing Workflow

The purposes of the Testing Workflow are:

- To verify the interaction between objects.
- To verify the proper integration of all components of the software.
- To verify that all requirements have been correctly implemented.
- To identify and ensure that defects are addressed prior to the deployment of the software.
- To ensure that all the defects are fixed, retested, and closed.

Testing typically encompasses both validation and verification [13] & [17]. Validation mainly deals with making sure the system is valid; i.e. performing according to actual intended requirements. Verification mainly deals with making sure that the behaviors of the system are correct; i.e. no errors.

The Rational Unified Process proposes an iterative approach, which means that you test throughout the project. This allows you to find defects as early as possible, which radically reduces the cost of fixing the defects. Tests are carried out along four quality dimensions: *reliability, functionality, application performance, and system performance*. For each of these quality dimensions, the process describes how you go through the test in particular phases; i.e. planning, design, implementation, execution, and evaluation. This is particularly needed; as well as useful, in agile environments [17].

3.3.6 Deployment Workflow

The purpose of deployment is to successfully produce product release(s), and to deliver the software to its end users. It covers a wide range of activities including:

- Producing external releases of the software
- Packaging the software and business application
- Distributing the software
- Installing the software
- Providing help and assistance to users
- Planning and conducting of beta tests
- Migration of existing software or data
- Formal acceptance

Although deployment activities are mostly centered around the transition phase, many of the activities need to be considered for inclusion in earlier phases to prepare for deployment at the end of the construction phase. The *Deployment and Environment* workflows of the Rational Unified Process contain less detail than other workflows [15].

3.3.7 Supporting Disciplines

3.3.7.1 Environment Discipline

The Environment discipline focuses on the activities necessary to configure the RUP process for a specific project. It describes the activities required to develop the guidelines in support of a project. The purpose of the environment activities is to provide the software development organization with the software development environment - both processes and tools - that will support the development team(s). If the users of RUP do not understand that RUP is a process framework; not just a methodology for development, they may perceive it as a weighty and expensive process.

A step-by-step procedure is provided describing how you implement a process in a certain environment or organization. It also contains a Development Kit providing software engineers with the guidelines, templates and tools necessary to customize the process. The Development Kit is described in more detail in the section “Development Kit for Process Customization" found later in this paper. However, certain aspects of the Environment discipline are not covered in the process such as selecting, acquiring, and making the tools work, and maintaining the development environment.

A key concept within RUP is that the RUP process framework could and often should be refined. This was initially done manually, i.e. by writing a "Development Case" document that specified the refined process to be used. Later the *IBM Rational Method Composer* product was created to help make this step simpler. So process engineers and project managers could more easily customize the RUP for their particular project needs. Many of the later variants of RUP, including OpenUP; formerly known as OpenUP/Basic [18], the lightweight and open source version of RUP, are now presented as separate processes in their own right, and cater for different types and sizes of projects, trends and technologies in software development. *Figure 10* below shows the OpenUP lifecycle.

Historically, as the RUP is often customized for each project by a RUP process expert, the project's overall success can be somewhat dependent on the abilities of that one person.

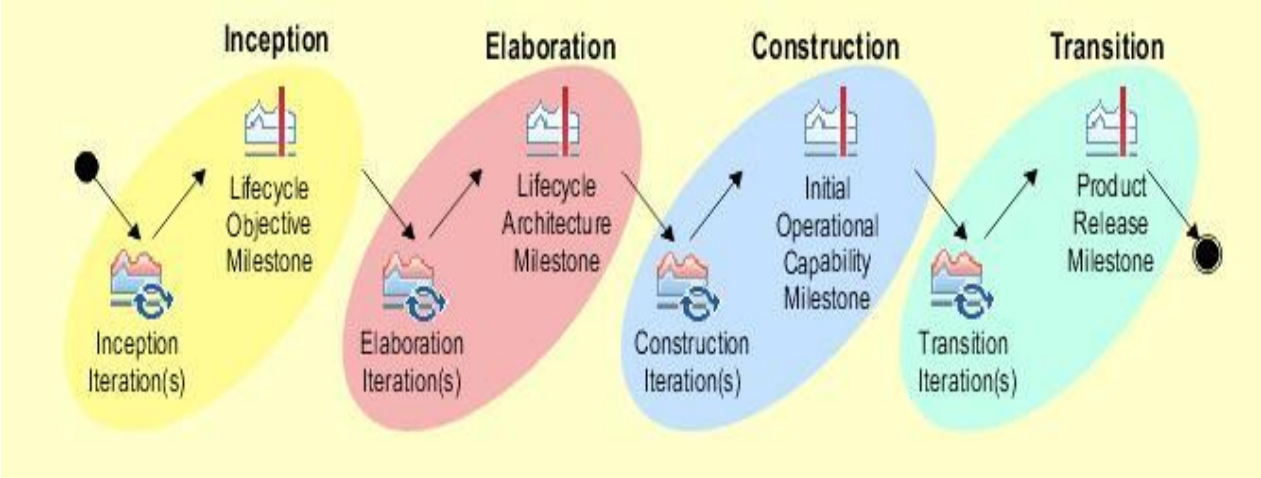

FIGURE 10: OpenUP Lifecycle; Redrawn from [18].

3.3.7.2 Configuration and Change Management Discipline

The Change Management discipline in RUP deals with three specific areas: Configuration management, Change Request management, and Status and Measurement management.

- *Configuration management* is responsible for the systematic structuring of the products. Artifacts such as documents and models need to be under version control and these changes must be visible. It also keeps track of dependencies between artifacts so all related articles are updated when changes are made.

- *Change Request management*: During the system development process many artifacts with several versions exist. CRM keeps track of the proposals for change.

- *Status and Measurement management*: Change requests have states such as new, logged, approved, assigned and complete. A change request also has attributes such as root cause, or nature (like defect and enhancement), priority, etc... These states and attributes are stored in a database so useful reports about the progress of the project can be produced. Rational also has a product to maintain change requests called ClearQuest.

In this discipline we describe how to control the numerous artifacts produced by the many people who work on a common project. Control helps avoid costly confusion, and ensures that resultant artifacts are not in conflict due to one of the following types of problem:

- Simultaneous Update: When two or more workers work separately on the same artifact, the last one to make changes destroys the work of the former.

- Limited Notification: When a problem is fixed in artifacts shared by several developers, and some of them are not notified of the change.

- Multiple Versions: Most large programs are developed in evolutionary releases [9]. One release could be at customer site being used, while another is in test, and the third is still in design. If problems are found in any one of the versions, fixes need to be propagated between them. Confusion can arise leading to costly fixes and re-work unless changes are carefully controlled and monitored [13].

3.3.7.3 Project Management Discipline

The Project management discipline and project planning in the RUP occur at two levels. There is a coarse-grained or **Phase** plan which describes the entire project, and a series of fine-grained

or **Iteration** plans which describe the iterations. This discipline focuses mainly on the important aspects of an iterative development process: risk management, planning an iterative project through the lifecycle and for a particular iteration, monitoring progress of an iterative project, and metrics for management and progress. However, this discipline of the RUP does not attempt to cover all aspects of project management. For example, it does not cover issues such as:

- Managing people: hiring, training, etc…
- Managing budget: defining, allocating, etc…
- Managing contracts: with suppliers, with customers, etc…

Software Project Management is the art of balancing competing objectives, managing risk, and overcoming constraints to deliver; successfully, a product which meets the needs of both customers (the payers of bills) and other users; i.e. all user stakeholders. The fact that so few projects are unarguably successful is enough comment on the difficulty of the task. This discipline focuses mainly on the specific aspect of an iterative development process. Our goal here is to make the task of project management easier by providing:

- A framework for managing software-intensive projects
- Practical guidelines for planning, staffing, executing, and monitoring projects
- A framework for managing risk

4. RUP SIX BEST PRACTICES

Six Best Practices in RUP are software engineering techniques or approaches that comprise ideas to follow when producing software [15]. The goal of the six practices is to minimize faults and maximize productivity. The six best practices are:

a. **Develop iteratively:** Sometimes it's possible to gather all requirements at the beginning of a project. However, this is usually not the case. It's very difficult to define the entire problem thoroughly, design a solution, build the software and then test it in sequential order. Therefore, software must be developed iteratively; i.e. you can go back as many times as you need to in order to change and upgrade the software. Agility deals with this very specific issue [13]. The need for iterative development emerges naturally as you gain an increasing understanding of the problem. RUP addresses the highest risk items at each stage of the lifecycle. This helps decrease the project overall risk. The software development team focuses on producing running results because each iteration ends with an executable release. RAD (Rapid Application Development); specially phased or versioned methodolgy, is typically used for such scenarios [9].

b. **Manage requirements:** RUP places great emphasis on eliciting, organizing and documenting requirements because it is imperative to keep users' requirements in mind when developing software. It is pointless to develop software that does not fulfill requirements because it will be useless; or at least not really beneficial, to the end users.

c. **Use components:** It is unavoidable to break down a large project into smaller, more manageable components. Individual components are not only easier to develop but also easier to test before they are integrated to form a complete system. Another advantage is that code developed for one component might be usable in another component; i.e. reusability [13]. This will help save time and money when developing future projects. However, a minor disadvantage is there; need for integration of components later on. But this disadvantage is typically far outweighed by advantages.

d. **Model visually:** Diagrams should be used to represent all major components, users and their interaction. Clear diagrams make it easier to understand the system and individual processes by hiding technical details. Visual models help you see how the system elements

fit together, help maintain a consistency between design and implementation, and help promote clear-cut communication.

e. **Verify quality:** Poor performance and reliability are problems that hinder the acceptance of a new software product. Therefore, in order to ensure quality, testing should be a key component of the project at any point of time. Although testing becomes more difficult and time-consuming as the project progresses, it is necessary and should be constantly carried out throughout all stages. RUP places great emphasis on quality and that is why quality assessment is a main part of each process. It is incorporated in all activities and involves all team members. It is not treated as something unimportant to be done by a separate group at a lter time.

f. **Control changes:** A project can be made by several teams that work in different locations. Therefore, it is important to ensure that changes made to a system are coordinated and verified constantly to prevent misunderstandings or unneeded work because the RUP environment encourages constant changes [15].

5. ADVANTAGES OF RUP

The RUP has become an industry standard for software processes. It is based on best practices that have been refined over many years of experience in numerous projects. It has been successfully applied by many organizations in many different domains. People are increasingly becoming proficient with the RUP. This makes it easier to find a professional with RUP experience. It does not really make sense to develop your own processes for software development; when such a vast amount of proven practices and information is available. The RUP can be tailored to meet almost any user requirements. Organizations can choose the parts they need; i.e. the parts that pertain to their environment.

The RUP iterative approach has several advantages over regular sequential processes:

a. **Improved management:** high quality software that fulfills users' requirements is delivered regularly and on time. RUP teams regularly produce working software. It is easy to determine if a RUP team is on the right track and redirect them if they are not.

b. **Regular feedback to stakeholders:** stakeholders can quickly see portions of the system coming together; similar to the RAD methodologies [9]. This reassures them that they will eventually receive what they require. If they do not like what they see, they can request changes until the construction phase. After that, changes will become more expensive.

c. **Improved risk management:** working iteratively allows higher risks to be found and addressed early in the process. If a requirement is questionable or a technical challenge or requirement is predicted to be too difficult to overcome, these issues can be addressed in an early iteration. If certain issues cannot be solved or can only be solved in a way that does not meet the stakeholder's requirements, then the project can either be altered or simply cancelled [13].

d. **You implement the actual requirements:** change is unavoidable. It is unrealistic to expect that requirements can be defined once at the start of a project and then not be changed. Hence, agility is desirable [13]. It is also not possible to fully design a whole system before writing any code. Therefore, by developing systems in iterations, it is easy to cater to changes and build software that actually meets the stakeholders' present requirements rather than those requirements that were documented months or even years ago. Phased RAD methodology is quite similar to RUP approach in such [9].

   Changes in requirements that will affect later iterations do not have an impact on the work being done on the current iteration. Changes to requirements in the current iteration are easy to deal with because an individual iteration has a small scope of requirements compared to a

whole project. Changes to previous iterations are scheduled as new requirements for future iterations.

e. **You discover what really works early:** the goal of the elaboration phase is to make sure that your architecture works; because in theory, all architectures work; but in practice, many do not work. By developing a system skeleton during the elaboration phase, it greatly lessens the project technical risks. Through iteration, it is easy to see; as you go along, whether your architecture satisfies requirements or not.

f. **Developers focus on what actually matters:** when the RUP is implemented correctly, developers spend more time on actual software development; unlike other more traditional approaches where teams will often spend months preparing models, writing documentation, reviewing their work, waiting for accepting, developing detailed plans before they write even a single line of code. The focus of software development should be on developing software, not on numerous processes that will eventually lead to developing software. Implementing RUP properly greatly improves information systems development productivity.

6. CONCLUSION

The RUP is being increasingly used in the software development industry because it is based on practices that have been tested repeatedly in many different projects and domains; and more importantly, proven successful. It is a configurable software development process platform that delivers practices and a configurable architecture. It enables the developer to select and deploy only the process components needed for each stage of a project. By using an iterative approach with constant testing, continuous feedback from stakeholders, and focusing on software development, the RUP greatly increases the quality of software produced.

7. REFERENCES

[1] Y. Wennerlund (1998). "The Unified Process*".* Sweden: EuroSPI, 1998.

[2] Suniwe (2008). "RUP Summary". WWW: http://projects.staffs.ac.uk/suniwe/project/rup.html

[3] SearchSoftwareQuality.com (2010). "Rational Unified Process". WWW: http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci810206,00.html

[4] TechTerms.com (2010). "RUP". WWW: http://www.techterms.com/definition/rup.

[5] Webopedia (2003). "RUP". WWW: http://www.webopedia.com/TERM/R/RUP.html.

[6] Wikipedia (2010). "IBM Rational Unified Process". WWW: http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process.

[7] S. Ambler (2008). "The Agile System Development Life Cycle (SDLC)". USA: Ambysoft & WWW: http://www.ambysoft.com/essays/agileLifecycle.html.

[8] Enterprise Unified Process (2009). "History of the Unified Process". WWW: http://www.enterpriseunifiedprocess.com/essays/history.html.

[9] A. Dennis, B. Wixom, and R. Roth (2006). Systems Analysis & Design, 3rd ed. USA: Wiley.

[10] S. Ambler (2010). "History of the Unified Process". WWW: http://www.enterpriseunifiedprocess.com/essays/history.html.

[11] I. Jacobson, G. Booch, and J. Rumbaugh (1999). The Unified Software Development Process. USA: Addison-Wesley.

[12] S. Ambler (2005). “A Manager’s Introduction to the Rational Unified Process (RUP)”.USA: Ambysoft & WWW: http://www.ambysoft.com/downloads/managersIntroToRUP.pdf.

[13] I. Sommerville (2010). Software Engineering, 9th ed. USA: Addison Wesley.

[14] P. Kruchten (2003). The Rational Unified Process, an Introduction. USA: Addison-Wesley.

[15] Rational (2000). “Rational Unified Process: Best Practices for Software Development Teams”. USA: IBM & WWW: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

[16] IBM (2005). “The Analysis Model”. USA: IBM & WWW: http://publib.boulder.ibm.com/infocenter/rtnlhelp/v6r0m0/index.jsp?topic=/com.ibm.rsa.nav.doc/topics/canalysismodel.html

[17] uTest (2011). “Agile Software Testing, Ten Tips for Launching and Testing High Quality Apps in an Agile Environment”. USA: uTest, Inc.

[18] Wikipedia (2012). “OpenUP/Basic”. WWW: http://en.wikipedia.org/wiki/OpenUP/Basic.