A Rational Unified Process (RUP) Plug-in To Support Requirements Quality Assurance

Andreas Altmannsberger
Schulstraße 6
63128 Dietzenbach, Germany
A.Altmannsberger@freenet.de

Prof. Dr. Frank Bühler
Dept. of Computer Science / FB Informatik
University of Applied Sciences / FH Darmstadt
Haardtring 100
64295 Darmstadt, Germany
f.buehler@fbi.fh-darmstadt.de

Michael Rowe, Ph.D.
Computer Science and Software Engineering Department
University of Wisconsin – Platteville
215 Ullrich Hall
Platteville, Wisconsin 53818
rowemi@uwplatt.edu

Abstract

There are many general-purpose, off-the-shelf software development processes that require tailoring to better fit the needs of an enterprise and/or a specific project. Process activities, roles and/or artifacts may be streamlined or deleted when a less formal, heavy weight process is appropriate; process activities may be modified to support continuous software process improvement. Contrariwise, new process activities, roles or artifacts may be added to support missing features. This paper studies the process and feasibility of tailoring the Rational Unified Process (RUP) by means of a plug-in to add Requirements Quality Assurance (RQA) functionality. Although this is a specific example, the study of the tailoring mechanism is a good educational exercise in both the motivation for and practice of customizing a given software engineering process. The paper discusses the coupling and cohesion problem of the different RUP process elements. As a result students will learn to take a critical standpoint with respect to "predefined" process models.

1. Background and Motivation

In the Standish Group's CHAOS report [TSG-1994], a survey of projects indicated that 31.1 percent of projects were canceled before they could be completed and another 52.7 percent were completed but were over-budget, over time estimate and offered fewer features than originally specified. Of the canceled projects, the two most commonly reported reasons were "Lack of User Input", 12.8 percent, and "Incomplete Requirements and Specifications", 12.3 percent.

In another survey of 3,800 software industry professionals, Leffingwell and Widrig [LW-2003] reported that 50 percent of the respondents indicated that the two largest software engineering problems were related to requirements specification and management of requirements. Finally, Lefflingwell and Widrig concluded that requirements errors were likely to consume 25 to 40% of the total project budget [LW-2003].

The above two studies support a fundamental need for requirements quality assurance in a software engineering process. Though one of these studies was conducted more than 10 years ago [YSG-1994], the newer study [LW-2003] indicates that requirements quality assurance has been and remains a chronic problem in software engineering. The question arises to what extent commonly known process models such as IBM's Rational Unified Process (RUP) [RUP] take this into account and offer an improved treatment of the requirements engineering discipline.

The Rational Unified Process is an iterative software development process that describes how to deploy software effectively using commercially proven techniques. It is not a rigid process but a process framework. It encompasses a large number of different activities, and is designed to be tailored, in the sense of selecting only the needed features suited for a particular software project, considering its size and type. After analyzing RUP's process activities it becomes apparent that each project has to implement its own software development process in order to meet its needs.

Referring to [BERGS-2004] and [PEP] a *process implementation* is the effort of putting a process to use within an organization or a project. This process may be pre-customized before the implementation. If the process is customized while implementing it within an organization or a project, Bergström coins this *adoption of a process*. The *adoption* is carried out as a four step approach:

"Assessment + Planning + Customization + Implementation (Execution/Evaluation)".

In this paper, the following definition of the term "process customization" and "process tailoring" is used: "process customization/tailoring is the act of refining, adjusting and adapting a given process model, to suit the special requirements of a particular enterprise, business unit or project, by modifying, expanding or removing process elements of the process model with appropriate approval".

With respect to requirements analysis the RUP specifies a so-called requirements workflow. Use Cases are the core artifacts for capturing functional requirements in the requirements discipline and are defined as the basis for the entire development process. In section 3, we will describe the requirements workflow in more detail.

2. Related Work and Research Questions

Requirements Specification and Management have been the focus in the Software Engineering field for many years. Frameworks such as VOLERE [ROB-1999] address these problems. In addition, various requirements analysis techniques, which all focus on different aspects, have been developed over the last years. To name a few, Inspection [IEEE-1983], Simulation [IESE-2005], Natural Language Requirements Measurement, [FGLM-2002], [ARM-1996], and Petri Nets [SILVA-2003] have been investigated. In addition, many research efforts have concentrated on the improvement of specific areas such as value-based requirements engineering for e-Commerce information systems [GORDIJN-2002], or work based on linguistic and ontological approaches for the creation of a normative expert language [OS-1996]. However, an evaluation of these techniques is beyond the scope of this paper.

Furthermore, various software process models which include a requirements process have been established over the years. Among so called heavy-weight models, the RUP is a well known and widely used software developed process both in the academic and commercial world. An interesting research question is to what extend the various research results on requirements engineering have been adopted and included within the RUP.

RUP's Environment Discipline introduces the Process Engineering Process, for short PEP, which itself focuses on the improvement of the development process [PEP]. The development of RUP plug-ins helps extend or change the RUP with respect to missing features or not needed activities and artifacts. Though the development of the plug-ins is supported by specific tools, this is not a trivial task. This brings up the questions, how easily is the RUP changed and what does this tailoring process look like?

3. Overview of the RUP

3.1 Introduction

The IBM Rational Unified Process [RUP] is one of the more formal, all encompassing software engineering processes being used in current software development projects. Like other process frameworks, the RUP defines three components for a process. As depicted in figure 1, these include *Roles*, *Activities* and *Artifacts*.

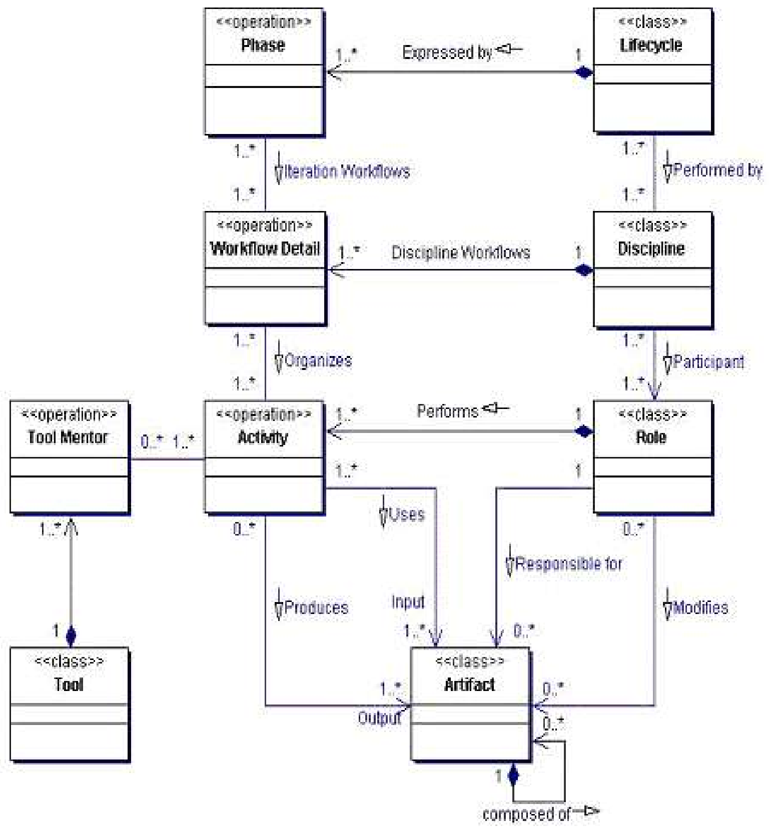


Figure 1: RUP's Meta Model.

Roles define responsibilities of individuals in the process. The individuals are assigned to their roles to perform *Activities*. The outputs of *Activities* are *Artifacts*. Artifacts are also the inputs that drive other Activities and become transformed by an Activity into new Artifacts. The Artifacts are the key pieces of information produced during the project to describe or visualize specific parts of the system or the project. The most important Artifact for a successful software project is the compiled, executable code. Other Artifacts include requirements, use cases, UML diagrams, test cases, etc. The purpose of other Artifacts, like models, documentation and plans, are for supporting the project progress. In addition to Roles, Activities and Artifacts, the RUP also provides Workflows. Workflows Details provide the sequence of when to perform the Activities.

Despite the RUP being considered one of the most complete software engineering processes for object-oriented development, it provides only minimal support for ensuring the quality of requirements. The following figure 2 illustrates the *Requirements Discipline Workflow*.

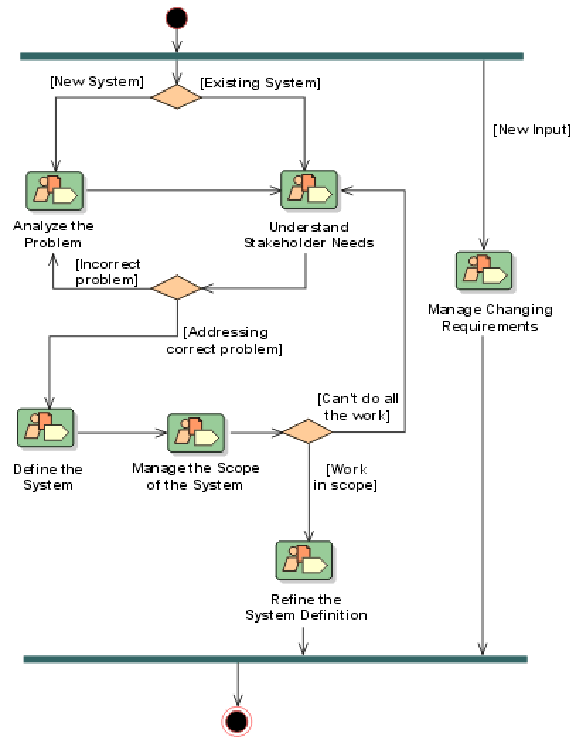

[New System]
[Existing System]
[New Input]
Analyze the Problem
Understand Stakeholder Needs
[Incorrect problem]
Manage Changing Requirements
[Addressing correct problem]
[Can't do all the work]
Define the System
Manage the Scope of the System
[Work in scope]
Refine the System Definition

Figure 2: RUP's Requirements Discipline Workflow.

There is only one activity named *Review Requirements* in the whole requirements discipline, which deals with the evaluation of the requirements' quality. However, this activity is only carried out when requirements are changed. There is no mention of an activity that specifically ensures the quality of the set of originally requirements. Referring to the RUP, the purpose of the *Review Requirements* activity is *to formally verify that the results of Requirements conform to the customer's view of the system* [RUP]. Even though this is a very important quality criterion, it is only one of many quality criteria that high quality requirements must satisfy. Therefore, in the authors' view the RUP does not provide a sufficient way to assure high quality requirements.

As illustrated in figure 3 we propose a hierarchy of artifacts which handle the evolution of manageable requirements which is not just based on checklists as defined within the RUP. In particular, the analysis and management levels will be regarded by the new proposed RUP plug-in. To support this, new activities and artifacts will be defined.

We have argued for the need to augment the RUP with a Requirements Quality Assurance (RQA) plug-in. This plug-in has been successfully developed and will be discussed in section 4. In addition, this paper describes the process for and feasibility of tailoring the RUP.

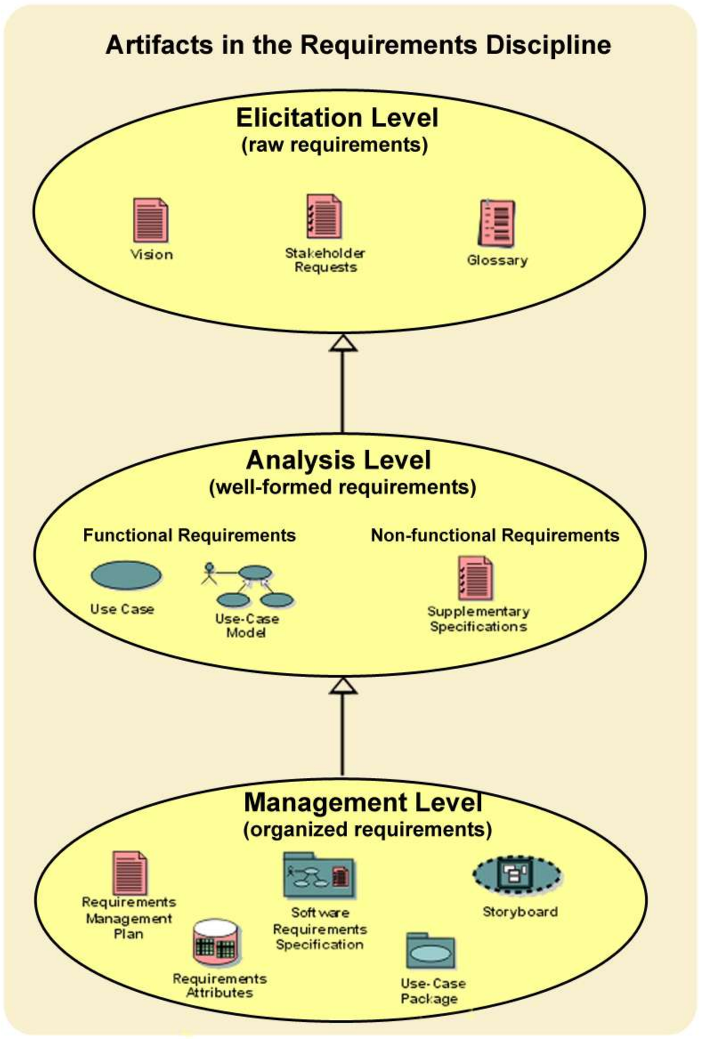


Figure 3: The Artifacts Hierarchy in the Requirements Discipline.

3.2 RUP Tools and Plug-In's

The RUP comes with a few tools which help customize the RUP. The RUP Builder [RUP-TOOLS] provides an environment for tailoring or customizing the RUP. This is done through the support of *Builder plug-ins* that employs three mechanisms for tailoring a process. These mechanisms include: deleting, adding, and modifying process elements.

Over the past years RUP plug-ins have been developed for different purposes. Plug-ins are commercially available for a fee and through open source channels free of charge. On IBM's website many plug-ins are downloadable for free, from many different categories: domain specific plug-ins (e.g. System Engineering), tool specific plug-ins (e.g. Rational Rapid Developer), platform specific plug-ins (e.g. J2EE), technology specific plug-ins (e.g. IBM WebSphere Application Server) and resource specific plug-ins (e.g. Wylie College Resources). A market analysis revealed that there is no specific plug-in available which focuses on the points we raised with respect to the Requirements Discipline.

The RUP Builder and the Rational Process Workbench (RPW) [RUP-TOOLS], which are bundled with the RUP, facilitates the creation of new plug-ins and modification of existing plug-ins. The RPW contains two individual tailoring tools, the RUP Modeler (an add-in to Rational Rose XDE) and the RUP Organizer. The Modeler supports activities involved in developing and managing *process models* and is used to create completely new process models or to modify existing ones. After a process model is developed with the Modeler, it is loaded with the Organizer to create and assign new process content (html files, icons, images, text documents) to the process elements in the process model. At the end, the process model and the process content can be compiled into a RUP plug-in, which then can be loaded by the RUP Builder to publish the process website. Figure 4 shows the tools provided by IBM for customizing the RUP. In particular, it clarifies the development of plug-ins and tools involved in this process (see also next section 4).

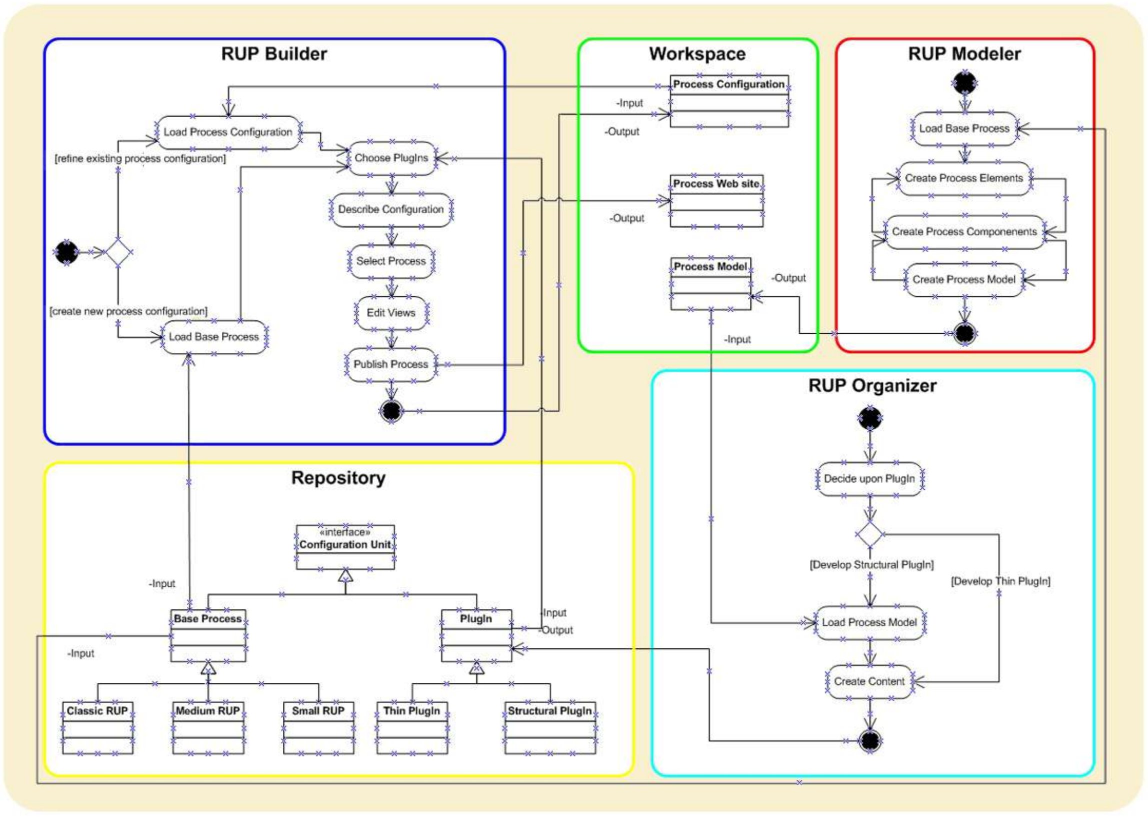


Figure 4: RUP's Toolset for process customization.

4 Description of the Requirements Quality Assurance (RQA) Plug-in

In this section, we will describe and discuss the new plug-in, Requirements Quality Assurance (RAQ), which has been developed in order to improve the quality of requirements defined during a software development project. The development of a specific plug-in is regarded as a good educational exercise. First, it enforces to study a given process model in some depth. Second, students learn to handle the different instruments needed to develop a plug-in and realize that the adoption process is by far not easy. Third, students are set in the position to argument for further improvements or amendments of the process model.

The assessment of RUP's requirements discipline indicates that actually no role exists, which is responsible for the quality assurance of requirements. Therefore, a new role, namely *Requirements Quality Controller*, is required. The assessment of the solution domain came up with new quality assurance techniques, which are currently not supported by RUP and would enrich the requirements discipline by helping to assure high quality requirements. The new role (*Requirements Quality Controller*) must perform four activities to carry out the techniques and one additional activity to create and establish a specification of quality criteria, which shall be fulfilled by requirements. These five new activities produce six new artifacts.

The *Requirements Quality Controller* is responsible for the quality assurance of all new developed, modified and changed functional and non-functional requirements. The five new activities are defined as follows:

- *Specify Quality Criteria* – to establish a catalog of quality criteria, which every requirements, functional and non-functional must satisfy.
- *Simulate Requirements* – to evaluate the requirements in a more formal and precise way to check whether they fulfill the specified quality criteria.
  For a simulation it is necessary to rewrite the natural language requirements in a formal requirements specification language, in order that the requirement can be executed by a simulator.
- *Inspect Requirements* – together with other development team members, formally verify that the requirements fulfill the specified quality criteria. In addition to the Requirements Quality Controller, also the *Customer*, *System Analyst*, *System Architects*, *Tester* and *Requirements Specifier* should be part of the inspection team.
- *Measure Requirements Quality* – to quantitatively measure the quality of the requirements. Sufficient requirements quality metrics and measurement techniques must be identified.
- *Execute Use Cases* – to determine the quality of Use Cases from a behavioral perspective during runtime (e.g. to find deadlocks) by using Petri net analysis.

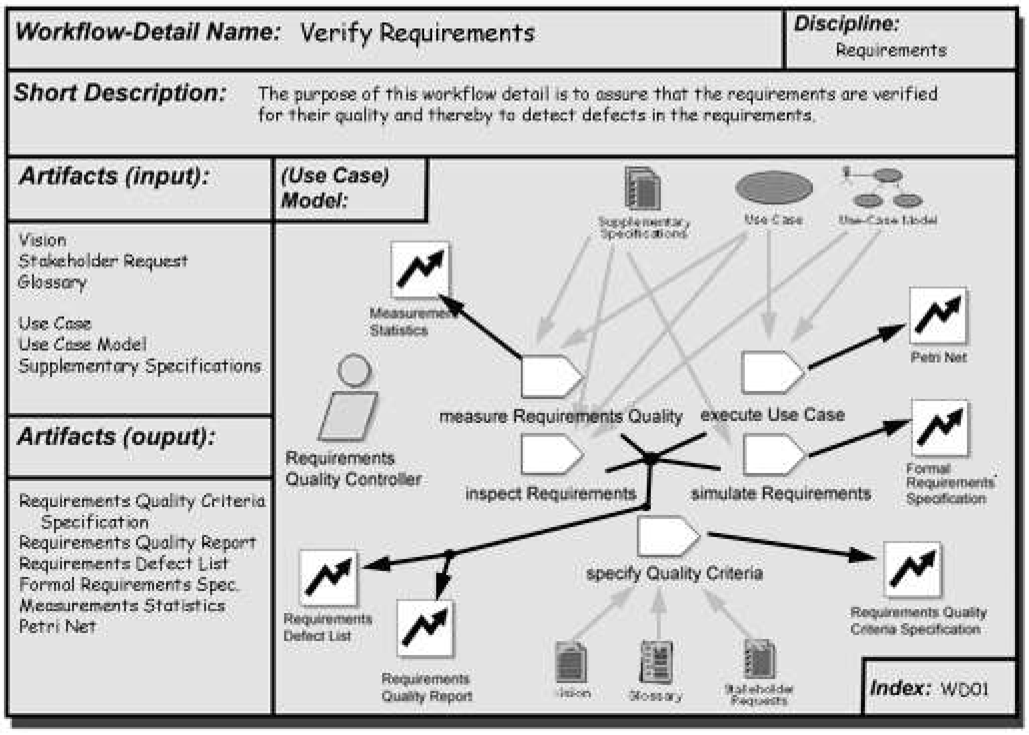
These activities produce six new artifacts, which include:

- *Requirements Quality Criteria Specification* – specifies and defines all quality criteria, which must be fulfilled by all requirements documents.

- *Requirements Quality Report* – describes the approach and the results of the performed quality assurance activity.
- *Requirements Defect List* – maintains the list of defects discovered in all checked requirements documents.
- *Formal Requirements Specification* – contains the translation of natural language based requirements in a formal requirements specification language.
- *Measurement Statistics* – lists the quantitative quality assessment of requirements document.
- *Petri Net* – a Petri net, which is created on the basis of a Use Case.

All these new process elements need to be integrated into the process. This is done by a new Workflow-Detail, which encapsulates the new process elements and enhances the Requirements Discipline.

To specify and visualize new process elements and their correlation in the process, the authors of this paper devised a card based visualization technique called *Process Specification and Visualization Cards* (PSVC), which were derived from the concept of CRC cards used by Alistair Cockburn [COCKB-2005] to collect the responsibilities and collaborations of classes. Figure 5 shows an instance of a PSVC - the *Workflow-Detail Specification and Visualization Card* - of the new workflow-detail *Verify Requirements.*


Workflow-Detail Name: Verify Requirements

Discipline: Requirements

Short Description: The purpose of this workflow detail is to assure that the requirements are verified for their quality and thereby to detect defects in the requirements.

Artifacts (input):

Vision
Stakeholder Request
Glossary

Use Case
Use Case Model
Supplementary Specifications

Artifacts (ouput):

Requirements Quality Criteria Specification
Requirements Quality Report
Requirements Defect List
Formal Requirements Spec.
Measurements Statistics
Petri Net

(Use Case) Model:

Index: WD01

Figure 5: Workflow-Detail Specification and Visualization Card (WSVC).

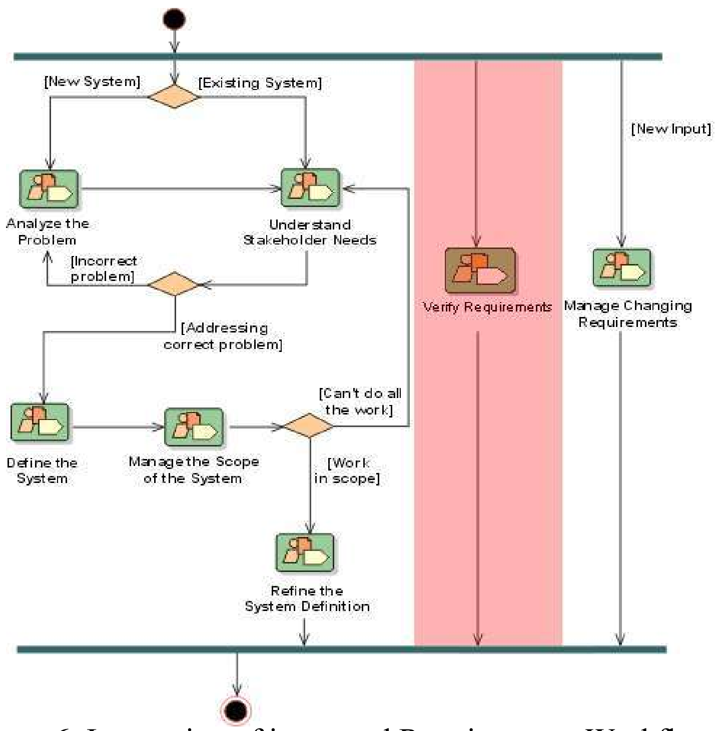


Figure 6: Integration of improved Requirements Workflow.

Figure 6 depicts the integration of the new *Verify Requirements* workflow-detail. The RQA plug-in is on an upper intermediate level of complexity when compared to other RUP plug-ins. The RQA plug-in is a structural plug-in, which requires the use of Rational XDE (resp. RUP Modeler) and the RUP Organizer. It adds completely new process elements to the RUP by linking them to the right elements.

The development of the plug-in took approximately 10 hours for a novice user to create the process model and around 20 hours to author the plug-in. Typically the authoring time is higher than the modeling time. Yet, the Rational's Process Engineering Process [PEP] mentions that plug-in developers generally spend more time in RUP Organizer than in Rational XDE (resp. RUP Modeler). This would lead us to estimate that probably another 20 hours would be required to provide the practitioners of the RQA plug-in with examples, templates and a full and detailed description of all activities, roles and artifacts.

As one can image it takes a lot of time to edit all the images of the diagrams. This leads to the question of whether it is necessary to keep all diagrams consistent to the process model. The workflow detail and role diagrams can be easily visualized with tables, therefore images are not really required. Only the flow within the discipline and phase diagrams are not be easily visualized with tables due to the high degree of variability. From the authors' point of view the discipline and phase diagrams provide a good overview for inexperienced practitioners of the RUP, but trained and experienced practitioners know how the RUP works and, therefore, do not really need the diagrams. From this perspective the diagrams are nice to have but are not really necessary and due to the significant effort associated with keeping them consistent with the process model, the authors judge the maintenance of the diagrams as infeasible.

5 Process and Discussion of RUP Tailoring

The development of a RUP plug-in is done in three steps: the *process modeling* to define the architecture of process elements in a process model, the *process authoring* to create process content (html files, images, diagrams) and to link this content to the model, and finally the *process publishing* to apply the Plug-In. For a more detailed step-by-step tutorial on developing a sample RUP plug-in we refer the reader to [BENCOMO-2005a] and [BENCOMO-2005b].

5.1 Process Modeling

The process model is the principal artifact used while developing a new RUP plug-in. It combines all other process components together with its individual set of process elements to define the whole process. Figure 7 shows the modeling result of the new role *Requirements Quality Controller* of the Requirements Quality Assurance (RQA) plug-in, together with its new activities and artifacts.

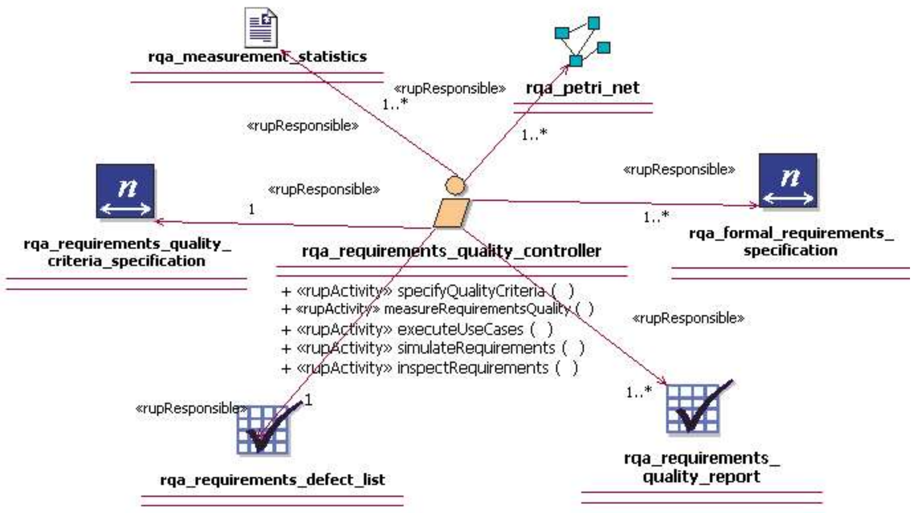

rqa_measurement_statistics
«rupResponsible»
1..*
rqa_petri_net
«rupResponsible»
1..*
«rupResponsible»
«rupResponsible»
1
1..*
rqa_requirements_quality_criteria_specification
rqa_requirements_quality_controller
rqa_formal_requirements_specification
+ «rupActivity» specifyQualityCriteria ( )
+ «rupActivity» measureRequirementsQuality ( )
+ «rupActivity» executeUseCases ( )
+ «rupActivity» simulateRequirements ( )
+ «rupActivity» inspectRequirements ( )
«rupResponsible»
«rupResponsible»
1
1..*
rqa_requirements_defect_list
rqa_requirements_quality_report

Figure 7: Role-Activity-Artifact model of the Requirements Quality Assurance plug-in.

In addition to the modeling of the new role, activities and artifacts it is necessary to model a new process component, which contains these process elements and the dependencies between the process component and the RUP. It is also possible in the case of the RQA plug-in to create a new workflow-detail and to link this workflow-detail to phases and disciplines.

5.2 Process Authoring

The process model only defines the structure or architecture of a plug-in, but the final result after the process publishing is a process website with html pages, images, workflow diagrams and additional PDF files. This content is mapped to individual process elements of the process model, during the authoring process. Thus, a plug-in is made out of a model and content. This mapping can be done by dragging content files and dropping them on process elements in the RUP Organizer. A process engineer probably generally spends more time in the preparation of content than in the modeling of the process elements. Additional tools like image editing tools, word processing tools and html editors are required to produce the actual content.

5.3 Process Publishing

In this final step the new developed RUP plug-in is loaded into the RUP Builder and linked to the RUP. After the process website is generated (published) the new process elements can be observed in the website, see figure below.

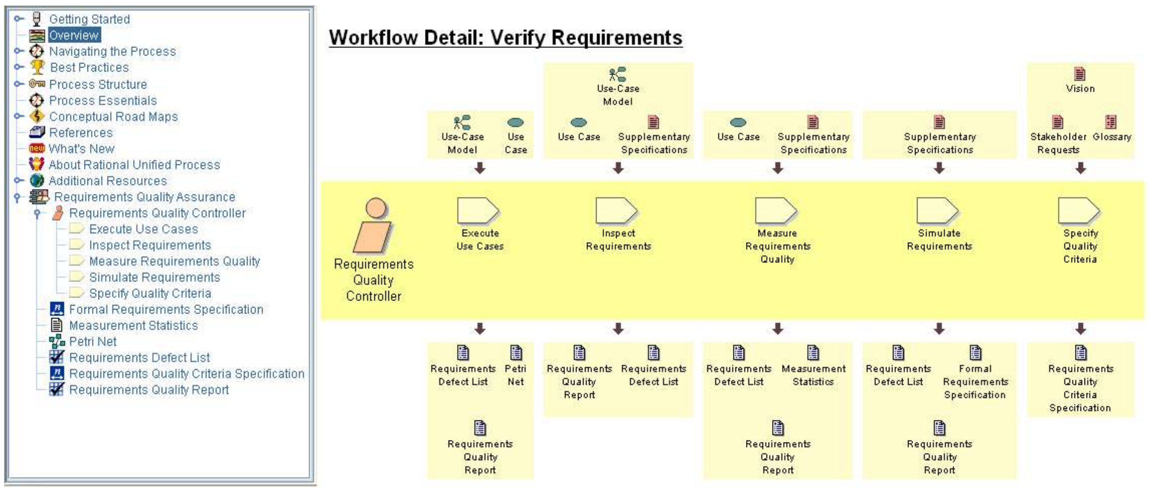


Figure 8: Applied Requirements Quality Assurance plug-in in the RUP website.

5.4 Restrictions and Problems while developing a RUP Plug-In

The previously described steps seam to be easy and smooth, but in fact there are some pitfalls, which need to be carefully considered. It was mentioned above that it is possible to delete process elements by plug-ins. This is not quite true, it is possible to delete *phases*, *workflow-details*, *activities* and *tool mentors*, but it is not possible to directly delete *roles* and *artifacts.* Because of this limitation in RUP's plug-in technology in practice a workaround is used. If a role or an artifact needs to be deleted, the whole process component in which the role or the artifact is defined is deleted in the RUP Builder. Afterwards a RUP plug-in is applied to the RUP, which redefines the deleted process component, but without the process elements, which actually needed to be deleted. This workaround works fine, but it is not sufficient or even safe.

Even if the deletion of activities is possible by RUP plug-ins, it is hard to determine which other process elements may be affected by the deletion. This is due to the fact that the output of one activity may affect the input of another activity. Let's assume that activity_1 produces artifact_A and that activity_2 uses artifact_A as input. In case activity_1 is deleted, then artifact_A will never be produced and activity_2 can not be performed because artifact_A is missing. That means that there is a strong dependency between these two activities, and the deletion of one of them must be considered very carefully. In addition to this simple example, the dependencies between entire process components must be examined.

We have investigated the degree of coupling between different process elements. As figure 9 below indicates, the degree of coupling of RUP's process components and consequently the degree of coupling of individual process elements is very high. The RUP defines 11 core process components with zero or more subcomponents, which sums up to 32 process components, which in turn define hundreds individual process elements. The authors have defined a coupling metric as the percentage of the total number of all other process components to which a process component is dependent, thus 100% indicates very high coupling (a process component is dependent to all other process components). The figure below shows, that there is a very high coupling for approximately one third of all process components. 21 process components have a coupling less than 25%, 8 process components have a coupling of more than 25% and 3 process components more than 40%. This abnormal high level of coupling leads thereto that the correct deletion or replacement will result in cascading problems with the consistency of other process elements and in most cases results in an enormous obstacle with respect to producing a consistent process.

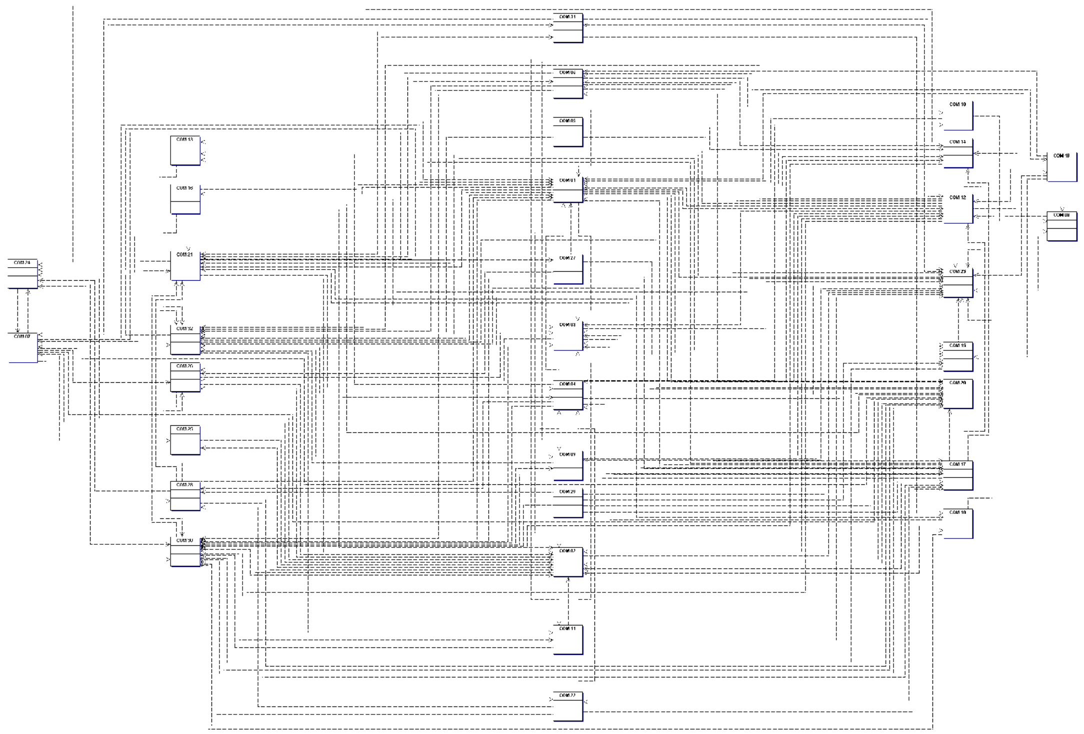


Figure 9: A dependency diagram of RUP's process components.

6 Conclusions and Future Work

IBM Rational promotes the Rational Unified Process as a highly customizable framework, which can be adjusted to all specific needs, with a wide range of supporting customizing tools. This paper investigated the development and integration of a RAQ plug-in. It was shown that the RUP is customizable, but with significant limitations. The most significant limitation described is that the removal of roles and artifacts is nearly impossible due to the many dependencies among process elements.

We plan to extend and evaluate the RAQ plug-in as part of a case study. In addition, we will consider other requirements processes and compare them with our model. As the development of RUP plug-ins is not an easy task and the dependencies of RUP process elements are very high we will develop a helper tool to tackle this problem.

References

[BENCOMO -2005a] Bencomo, Alfredo: *Extending the RUP, Part 1: Process Modelling*, www.therationaledge.com, 2005, last visited 14th June 2005.

[BENCOMO-2005b] Bencomo, Alfredo: *Extending the RUP, Part 2: Process Description*, www.therationaledge.com, 2005, last visited 14th June 2005

[BERGS-2004] Bergström, Stefan, Råberg, Lotta: *Adopting the Rational Unified Process: Success with the RUP*, Addison Wesley, 2004

[COCKB-2005] Cockburn, Alistair: *Crystal: A family of software development methodologies*, http://alistair.cockburn.us/crystal/crystal.html, last visited 4th October 2005.

[FGLM-2002] Fantechi A., Gnesi S., Lami G., Maccari A.: *Application of Linguistic Techniques for Use Case Analysis*, http://fmt.isti.cnr.it/WEBPAPER/REJfin2.pdf, 2002, last visited 1st August 2005.

[GORDIJN-2002] Jaap Gordijn: *Value-based Requirements Engineering - Exploring Innovative e-Commerce Ideas*, SIKS Dissertation, Vrije University, Netherlands, 2002.

[IEEE-1983] IEEE STD 729-1983: *IEEE Standard Glossary of Software Engineering* Terminology, http://www.ieee.org/, 1983.

[IESE-2005] Choi, Yunja: *Formal and Informal Verification Techniques for High Quality Requirements*, Fraunhofer Institute for Experimental Software Engineering, http://www.iese.fraunhofer.de, 2005.

[LW-2003] Leffingwell, D., Widrig, D.: *Managing Software Requirements - A Use Case Approach*, 2nd edition, Addison Wesley, 2003.

[OS-1996] E.Ortner, B. Schienmann: *Normative Language Approach - A Framework for Understanding*, Proceedings of the Entity Relationship Approach, ER'96, Addison-Wesley, 1995.

[PEP] Process Engineering Process: A RUP Adoption Process, Part of the Rational Process Workbench (RPW), version 2003.06.01.06.

[ROB-1999] Suzanne and James Robertson: *Mastering the Requirements Process*, Addison-Wesley, 1999.

[RUP] IBM Rational Unified Process (RUP), The browser-based product, version 2003.06.13, http://www-306.ibm.com/software/awdtools/rup/, last visited 20-January-2006.

[RUP-TOOLS] IBM Rational Unified Process (RUP), http://www-306.ibm.com/software/awdtools/resources/rmc.html, last visited 20-January-2006.

[SILVA-2003] Silva, Jose Reinaldo; Santos, Eston Almança dos: *Applying Petri Nets for Requirements Validation*, http://www.pmr.poli.usp.br/d-lab/artigos/COB03-1825.pdf, 2003, last visited 4th October 2005.

[TSG-1994] The Standish Group, *The CHAOS report*, http://www.standishgroup.com/sample_research/chaos_1994_1.php, 1994, last visited 20-January-2006.

[WRLH- 1996] Wilson, William M., Rosenberg, Linda H., Hyatt, Lawrence E.: *Automated Quality Analysis of natural Language Requirements Specifications*, http://satc.gsfc.nasa.gov/support/, 1996, last visited 1st August 2005.