



University of  
Stavanger

Faculty of Science and Technology

## MASTER'S THESIS

Study program/ Specialization:  Master of Science Computer Science	Spring semester, 2011  Open
Writer: Ole Foss Johannesen	..... (Writer's signature)
Faculty supervisor: Kåre Auglænd	
Titel of thesis: Model based analysis of IBM's tools for Collaborative Application Lifecycle Management	
Credits (ECTS): 30	
Key words: Rational Team Concert Rational Requirements Composer Rational Quality Manager IBM Collaborative Application Lifecycle Management (C/ALM) Modeling Analysis	Pages: 59, including front page, excluding Appendix + enclosure: DVD-ROM.  Stavanger, 14.06.2011

## Summary

This thesis is an analysis of three of IBM's latest tools for Collaborative Application Lifecycle Management (C/ALM), Rational Team Concert, Rational Requirements Composer and Rational Quality Manager. Each tool focuses on a different phase in the development. RTC was first released at 25<sup>th</sup> of June 2008 <sup>1</sup>, RQM followed with its first release at 16<sup>th</sup> of March <sup>2</sup>, and last released was RRC at 25<sup>th</sup> of May 2009 <sup>3</sup>.

To hopefully make the analysis a little bit easier to understand, figures have been made of different parts of each of the programs. The general models shows a brief overview of everything within the program, while the other models take a little bit more detailed approach. The figures in question is has been created with using another tool from IBM, Rational Software Architect.

Rational Requirements Composer is probably the smallest of the three tools, and focuses on the planning phase. It is used to create the conditions which a new program need to fulfill, or the conditions which a new version of the program needs to be able to do.

Rational Team Concert is the program which has been most extensively tried in this thesis, it focuses on the development phase of a project. People work in teams to develop, source code is written locally, but is shared when the user think its time to share it.

Rational Quality Manager is probably the most comprehensive of the three tools, and focuses on the stage of testing applications. It includes a lab management tool to keep track of all machines, which one is available and which one is not at all times.

While all of the tools can be used standalone, using them together has its benefits, and it could make the transition from phase to phase easier. It also makes it easier for users of each tool to work simultaneously, to increase efficiency.

## Foreword

There was some challenges with an assignment like this, mainly because when testing and analyzing tools for collaboration between a team, it probably be more ideal to be more than one person doing this. The ideal would probably be two or three people, to make it easier to see all of the capabilities which are available in the tools. The way I worked was to create different accounts to see how changes done from one account would affect the view from another account. Particularly in one of the tools, Rational Requirements Composer it was a bit difficult to use multiple accounts, because it was not possible to create separate workspaces. Changing from one account to another required to change the name of the particular workspace. Instead of creating your own workspaces, the tool always used a workspace named "Workspace", so to change account, I would have to change name of the workspace (and thus account which is tied to that exact workspace) to "Workspace" so that the tool would load it. This was very heavy handed, as changing accounts is something which happens a lot while working this way.

Other than it was a little bit troubling to work around these things at occasions, i have learned a lot about these products, and also about the way an application is created. It requires a lot of thought and planning, where I personally mostly have concentrated on the development part of creating a an application, it has shown me how much work goes into the planning and testing of an application. Especially the testing part, seems very extensive, and Rational Quality Manager is a very comprehensive and deep tool.

In the end i would really like to thank my teaching supervisor Kåre Auglænd, for a lot of hours of discussions revolving this thesis, and how to get it best possible. Also a thank to Sverre Nymo from IBM for some good ideas for a master thesis, although, the final subject strayed a little bit away from the original ideas.

Stavanger, June 13th 2011

-----  
Ole Foss Johannesen

# Table of Contents

Summary.....	page 2
Foreword.....	page 3
Table of Contents.....	page 4
<b>1 Introduction.....</b>	<b>page 6 - 9</b>
<b>1.1 Background.....</b>	<b>page 6</b>
<b>1.2 Description and goal of the assignment.....</b>	<b>page 6</b>
<b>1.3 Tools.....</b>	<b>page 6</b>
<b>1.4 Report.....</b>	<b>page 6</b>
<b>1.5 Analytical method and uncertainties.....</b>	<b>page 7</b>
<b>1.6 Jazz.....</b>	<b>page 7</b>
<b>1.7 Application Lifecycle Management (ALM).....</b>	<b>page 8</b>
<b>2 Rational Team Concert.....</b>	<b>page 9 - 30</b>
<b>2.1 Setup of the Jazz Team Server.....</b>	<b>page 9</b>
<b>2.2 Administrator Web Interface for Rational Team Concert.....</b>	<b>page 10</b>
<b>2.3 Getting started with a Rational Team Concert project.....</b>	<b>page 11</b>
<b>2.3.1 Creating a new project and choosing a process for it.....</b>	<b>page 12</b>
<b>2.3.2 Creating Users.....</b>	<b>page 12</b>
<b>2.3.3 Creating a Team.....</b>	<b>page 13</b>
<b>2.3.4 Inviting a user to a project, and accepting it.....</b>	<b>page 13</b>
<b>2.3.5 Creation of work items, plans and iterations.....</b>	<b>page 14</b>
<b>2.4 Architecture of Rational Team Concert.....</b>	<b>page 15</b>
<b>2.5 Licenses and Permissions in Rational Team Concert.....</b>	<b>page 16</b>
<b>2.6 Rational Team Concert with Simple Team Process.....</b>	<b>page 18</b>
<b>2.7 Rational Team Concert with Scrum.....</b>	<b>page 20</b>
<b>2.7.1 Scrum, focus on the different Work Items.....</b>	<b>page 20</b>
<b>2.7.2 Scrum, focus on Role and Members.....</b>	<b>page 21</b>
<b>2.7.3 Scrum, focus on Plan and Iteration.....</b>	<b>page 22</b>
<b>2.7.4 Complete overview model of RTC with Scrum.....</b>	<b>page 24</b>
<b>2.8 Source Control.....</b>	<b>page 25</b>
<b>2.8.1 Creating Repository Workspace.....</b>	<b>page 25</b>
<b>2.8.2 Delivering code to a stream.....</b>	<b>page 25</b>
<b>2.8.3 Checking changes on another account.....</b>	<b>page 27</b>
<b>2.9 Generalization model of RTC and corresponding object model.....</b>	<b>page 27</b>
<b>3 Rational Requirements Composer.....</b>	<b>page 30 - 40</b>
<b>3.1 Configuring the server.....</b>	<b>page 30</b>
<b>3.2.1 Artifacts in Requirements Composer.....</b>	<b>page 30</b>
<b>3.2.2 Reviews in Requirements Composer.....</b>	<b>page 33</b>
<b>3.3 Permissions and Licenses in Requirements Composer.....</b>	<b>page 33</b>
<b>3.4 Architecture of Rational Requirements Composer.....</b>	<b>page 35</b>
<b>3.5 General model og Requirements Composer.....</b>	<b>page 36</b>
<b>3.6 Requirements Composer review.....</b>	<b>page 38</b>
<b>3.7 Requirements Composer artifacts.....</b>	<b>page 39</b>

<b>4 Rational Quality Manager</b> .....	page 41 - 52
<b>4.1</b> Configuring the server.....	page 41
<b>4.2</b> Architecture of Rational Quality Manager.....	page 41
<b>4.3</b> Getting started.....	page 42
<b>4.4</b> Licenses & Repository Permissions in Rational Quality Manager.....	page 42
<b>4.5</b> General model of Rational Quality Manager.....	page 44
<b>4.6</b> Work Items and Defects in Rational Quality Manager.....	page 44
<b>4.7</b> Lab Management in Rational Quality Manager.....	page 46
<b>4.8</b> Planning in Rational Quality Manager.....	page 47
<b>4.9</b> Construction in Rational Quality Manager.....	page 49
<b>4.10</b> Execution in Rational Quality Manager.....	page 51
<b>4.11</b> Reporting in Rational Quality Manager.....	page 52
<b>5 Collaboration with the three tools</b> .....	page 53 - 56
<b>5.1</b> Configuration of the three servers.....	page 53
<b>5.2</b> Linking projects and objects within the different projects.....	page 53
<b>6 Conclusion &amp; Future Work</b> .....	page 56 - 57
<b>6.1</b> Conclusion.....	page 56
<b>6.2</b> Future Work.....	page 56
<b>7 Sources &amp; References</b> .....	page 58
<b>8 Enclosure</b> .....	page 59
<b>9 Appendix</b> .....	page 59 - 63

# 1 Introduction

## 1.1 Background

Based on the knowledge from speaking to employees of IBM, it seems that the knowledge surrounding the tools developed by the company may be a bit lacking in Norway. Much of this is of course due to the fact that these tools are not developed in Norway, but rather in other countries. Therefore an assignment based around getting more knowledge for some of these tools seemed like a good idea.

## 1.2 Description and goal of the assignment

The assignment is to investigate and analyze three different tools from IBM which are created for Collaborative Application Lifecycle Management. Find out how each of the tools is built up and what they are used for. How do they work standalone, and how do they work when being used together, what possibilities opens up compared to using them alone? There will be used models to hopefully illustrate how each of these tools work and what is available in each of them. In addition to these models there will of course also be explanations of each tool.

## 1.3 Tools

There is two other tool used other than the three which the thesis is centered around, and that is Rational Software Architect. RSA <sup>4</sup> is a comprehensive modeling and development environment which is developed by IBM. In this project this tool was used to create the models which describe the different parts of the three tools which has been analyzed. The other one is Microsoft Paint, to draw one of last figures in the report. (Figure 33)

## 1.4 Report

The report is divided into four main chapters. One for each of the tools, and also one for the collaboration possibilities between the three of them, since that is a very important part of this project. The tool which has been most thoroughly analyzed is Rational Team Concert, and because of this, the part on this tool is also a bit longer than the parts concerning the two other tools. In this report, there has been taken a little bit different approach than usual UML, by using boxes to represent some objects in some figures. There is a couple of reasons for doing it this way, as these two figures will show.

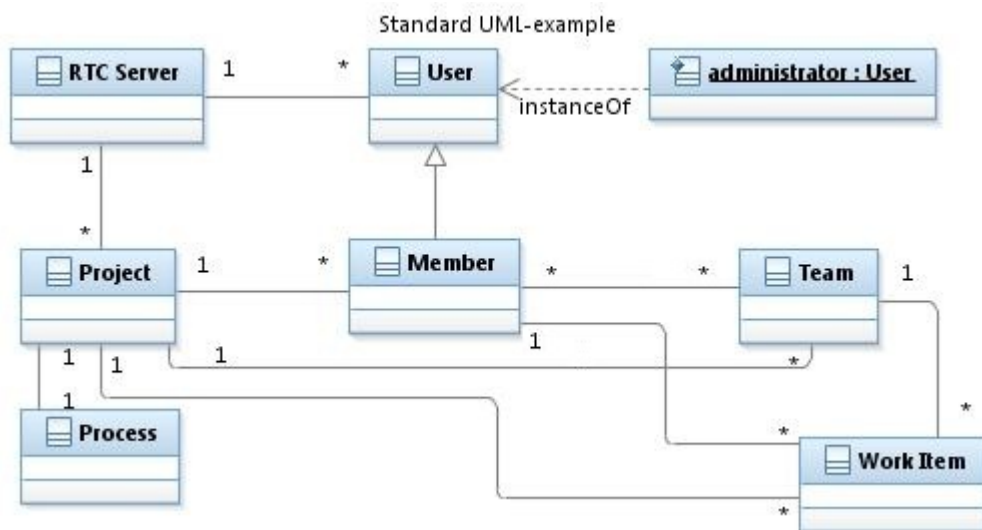


Figure 1: Example of standard UML.

This first figure (Figure 1) shows a scenario created in UML, this is a rather simple scenario, but still there is a lot of associations and it is already starting to get a little bit chaotic. You will also have to look closely at each figure to find out what is inside the project, and what is not.

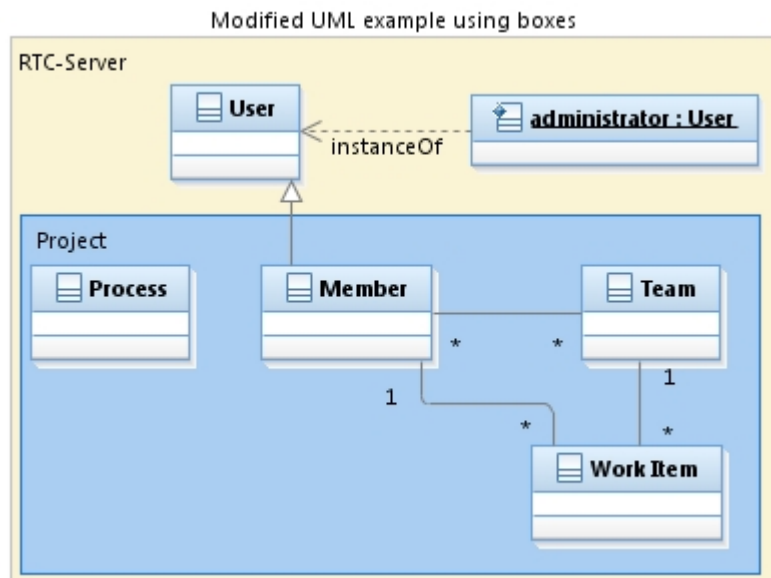


Figure 2: Example of modified UML, which is used in this thesis.

This model (Figure 2) shows the exact same scenario as the UML model above, and it is a lot easier to see what is inside and what is outside the project. It also gives a lot better view of the user-member scenario which all of these three tools uses.

When models become more complex than this example, it will be easier to look at the models and distinguish what is what. The boxes is used when there is other objects inside one object, a project is a good example. Instead of having everything which is within a project associated with it (using association arrows), a box is used to represent all these associations. In this example, process, member, team and work item all are associated with the project, by putting it inside the box.

## 1.5 Analytical method and uncertainties

When doing this analysis there was of course some certain methods used. The main method to was to try out the tools, test different aspects of how they worked and explore the possibilities which is given through them. Secondary knowledge came from reading, and when reading it was usually the help center which all three tools has one of. When using the help center to form an opinion of how something worked, or how to get something to work, there is sources given in the appropriate places throughout the report. Most of this report is however based on how I see each of the tools from the outside, and what the possibilities in them can be used for.

In such a scenario there will always be uncertainties of how this actually is and it has to be considered that it is possible that these figures does not represent everything entirely correctly, since I have no information of the actual structure of these tools.

## 1.6 Jazz

Jazz is a technology platform for developing software collaboratively from IBM. It is designed to allow globally distributed teams working in an environment which makes it easy to exchange data and synchronize people and processes.

Jazz is built on three main principles <sup>5</sup>:

- ✓ An architecture for lifecycle integration.
- ✓ A portfolio of products designed to put the team first.
- ✓ A community of stakeholders.

The platform consists of several products, each designed for being an essential tool for employees with different roles. The three main programs on the Jazz platform is:

- ✓ Rational Team Concert
- ✓ Rational Requirements Composer
- ✓ Rational Quality Manager

Rational Team Concert is tool for developers, while Jazz offers Rational Requirements Composer for analysts and for quality professionals there is Rational Quality Manager.

## 1.7 Application Lifecycle Management (ALM)

First, to avoid confusion, C/ALM is an expression which IBM uses for ALM, but it has not been adopted as a common expression yet, so in this part the more common term, ALM will be used. When talking about Application Lifecycle Management, there is a lot of different categories which can be included. Requirements Analysis, Requirements Management, Feature management, Modeling, Design, Build management, Software Testing, Release Management, and there is a lot of other categories which is possible to add to these.

However, with the products from IBM in mind, it might be more fitting to create a triangle of phases. (Figure 3) Each of these phases represent a specific stage in the development, and with it, an IBM program associated with it.

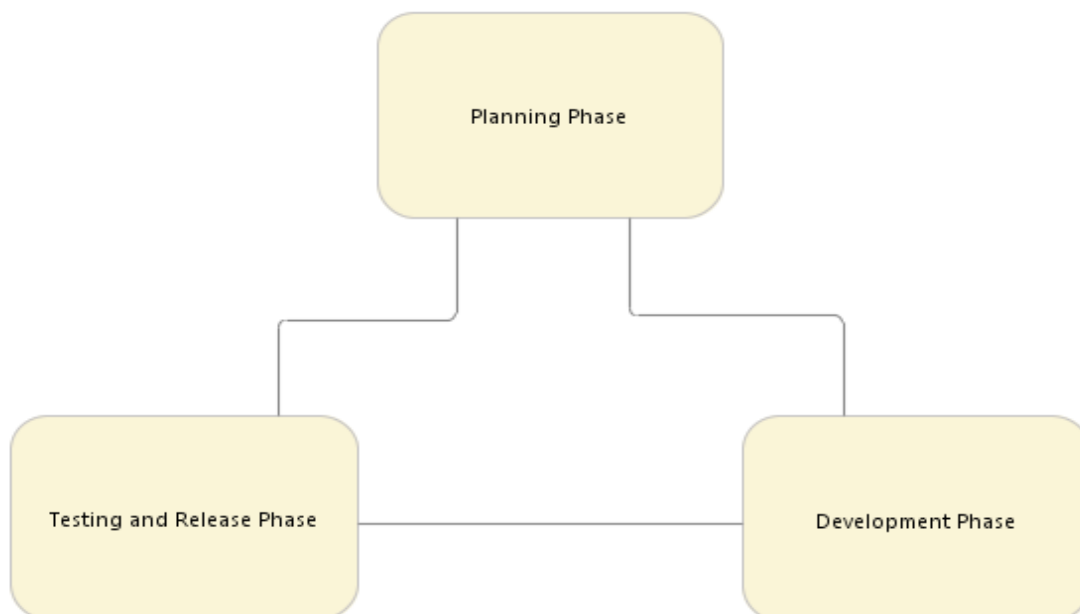


Figure 3: The three important phases in a development scenario.



The planning phase naturally is the first one to take place. Categories which might be found in this phase is Requirement Analysis and Management, in addition to Feature Management. What does this mean? A feature management is planning and executing what the finished product is supposed to be able to perform, this is however something which is continuing throughout the project.

Requirement Analysis <sup>6</sup> is the main category in the planning phase, this means communicating with the different stakeholders, and find out what the needs and conditions with the program (or update of an existing program). In addition to this, in the planning phase it is important to identify and find out if it is possible to reuse earlier resources. This is where Rational Requirement Composer comes into play, and why it is connected with the planning phase.

The next phase is to start the actual efforts to create the program. In the early parts of this phase its natural that we find designing and modeling of the program, or the refinement of an existing program. When the design and modeling (if such an approach is used) is completed, the development of source code is the next step. The final part of the development phase is the building of the source code, where the source code is converted into an executable file, which may run independent outside of the development environment. IBM represents this phase with the program Rational Team Concert, as this is where the development is in focus.

The final phase is the phase of testing and releasing the program. Obviously in the testing phase there might be flaws that are discovered which causes a setback to previous phase, and having to redo the building phase.

Through the first two phases, there are usually put a lot of emphasis that everything is correct when a project reaches to this point. When thinking consequences of discovering faults, one defect at this point might effect a lot of other parts of the program, making it very costly. Is the defect discovered early, chances are that the consequences is a lot smaller.

The final step is the release of the program, which is the distribution of this particular version of a program. Before the release, a program is often associated with the term "Release Candidate" <sup>7</sup>, which is supposed to be a potential release, pending that there are not any last minute critical errors. When a version is released, it is back to square one, to keep the program updated and comparative. For this phase, the program Rational Quality Manager is very well suited.

## 2 Rational Team Concert

### 2.1 Setup of the Jazz Team Server

The very first thing to do when starting with any product based on Jazz, is to set up the corresponding server for the program in question.

This is done by logging into the Jazz Team Server (Picture 1, in the Appendix) for the very first time, open th URL in a web browser: <https://localhost:9443/jazz/setup>. (This is of course a local URL, shall it be used through the Internet, a different address is needed)

Since this is the first visit with the Team Server, there will be a need to use the default values for login and password. UserID: ADMIN and Password: ADMIN, due to the fact that there will not yet have been created an administrative profile. Login and password are case-sensitive, so be sure to use upper case letters. <sup>8</sup>

Logging in for the first time, you will be faced with a setup wizard to run through to set up the server. In the introduction screen there will be two options given, **Fast Path Setup** and **Custom Setup**. In this case we use the Fast Path Setup, since that does not include configuring a database, but instead using the preconfigured Derby database. Derby is a lightweight database, which can be

used when there is ten or less users.

As this is done, one will see that the wizard jumps from step one to five, as step two, three and four only needs to be done if using the Custom setup option.

In this screen one will be able to choose between two different ways to handle user management. Using the default, Tomcat User Database is the easiest option, as you will be able to manage users from the Team Server. The other available option is Non-LDAP External Registry, which means that the connection between every single user in the Jazz Repository and the external registry will have to be done manually. In this project Tomcat User Database has been used.

In addition to choosing how the registry of users will be, one will also have to register an administration account. This account can be used to log in on the team server after this wizard is complete. The administration account will have a JazzAdmins role, and will be able to modify everything which is stored on the server, from projects, to user accounts and to users roles etc. There is also a choice to disable the default ADMIN user after creation of this account, as well as a choice if the administrator account should have one of the ten free licenses which follows with Rational Team Concert Express-C.

When this is complete it is time to press the next button to the final screen, which is just an overview of what the settings of the server will be, overview it and click finish.

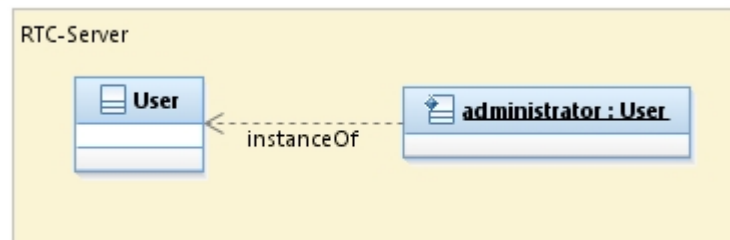


Figure 4: First figure in step by step model. What exists so far.

This is the start of a step by step model (Figure 4) for what is needed to get Rational Team Concert up and running, and most essential objects in the opening phase. It will be filled in over the couple of sub chapters. This figure shows what is existing at this point. At this point the Team Concert is set up and configured, in addition there is created an administrative user, with a name of your choice, thus it is only called administrator in this figure.

## 2.2 Administrator Web Interface for Rational Team Concert

After the configuration is finished, you will be directed to the web interface of the team server. An administrator can use this web interface to administrate projects and users. The interface itself is very user friendly and it is easier to administrate from it, than to do it directly from the Rational Team Concert client. The administrative web interface has five main tabs (Picture 2 in Appendix):

- ✓ Project Areas
- ✓ Server
- ✓ User Management
- ✓ Project Area Management
- ✓ Process Template Management

The "Project Areas" tab will show you all existing projects, with the possibility to open them for further information and details. The details within a project will be discussed at a later stage, so that will be it for now.

The "Server" tab contains two underlying tabs, "Status" and "Configuration", under the Status tab

one can find details of the server, version of the server, version of Team Concert, server uptime and different kinds of statistics. Under the configuration tab there is naturally different kind of configuration options, like License Key Management (As earlier mentioned, there is ten licenses in Rational Team Concert Express-C), E-Mail Settings, Database Connection to name a few. Under the "User Management" tab possibilities to create users, import users, assign licenses to users, in addition to archiving users.

**Note:** In Rational Team Concert it is not possible to delete users or projects, instead the term archive is used. Archiving a project is done either because it is finished or that it is currently inactive. An archived user automatically loses its Developer License, which will be available to assign to other users. If it should be necessary to bring back an archived user, that is possible too.

The "Project Area Management" tab is for the configurations of the projects, here it is possible to create or edit teams, assign roles to users, and different options to customize each project. The most important possibilities will be discussed later on. The last tab is the "Process Template Management" tab, which allows the administrator to either customize existing processes, or create their own. Processes will also be discussed later on.

It is worth mentioning that the layout of the administrative interface is identical in all three tools, so this could also be used to describe the administrative interfaces in either Requirement Composer or Quality Manager.

### 2.3 Getting started with a Rational Team Concert project

To set up the project, the tutorial in the help center has been used.<sup>9</sup>

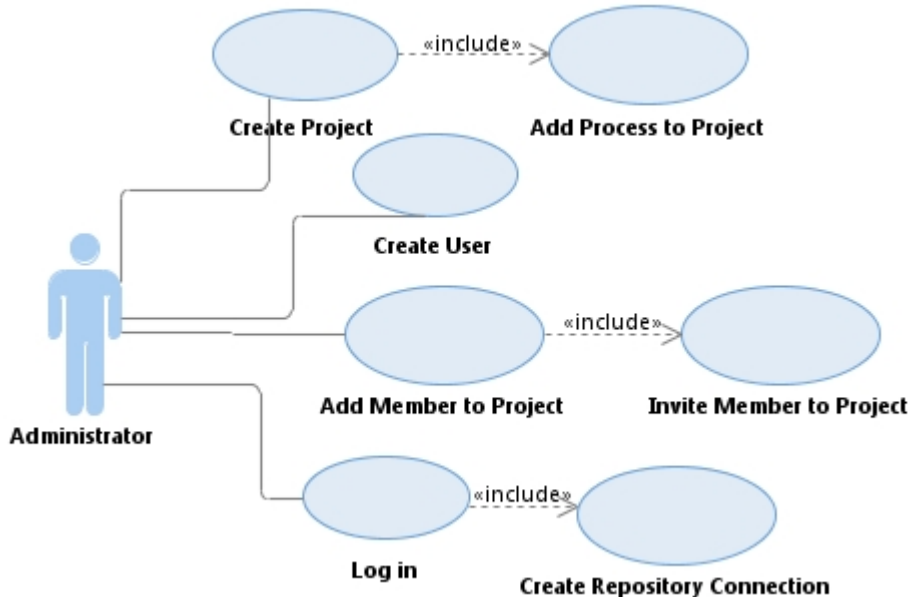


Figure 5: Use Case diagram showing the possibilities for the administrator, this is relevant when getting started with RTC.

This figure (Figure 5) shows the use cases for the administrator, creating project, users, and also adding them to projects. The invite part is to send an e-mail to the user in question so that he can identify himself and be verified as a member of the project. These use cases will be discussed along the next pages.

These are the most important use cases, although an administrator can operate as a normal user also, and do all the things which they can do. However in most cases an administrator is only administrating the server and projects.

The very first thing to be done after starting the client is to choose where to place the initial workspace, this workspace shall belong to the administrative user.

Workspaces generally should be kept separate for each user, so when connecting with another user, create a new folder as the workspace.

Now on to create a project and a couple of users which can be a part of a project.

To start out one will need to use the administrative account created when configuring the Jazz Team Server. The first thing which needs to be done is to create a repository connection for the administrative account. The repository connection you could say is to create the login mechanism from the client in an operating system, to verify an account against the server. (See appendix for picture of the creating of a repository connection, picture 3). To create a repository connection, the server address, user name and password is needed.

### **2.3.1 Creating a new project and choosing a process for it**

The next thing on the list to do is to create a brand new project, there is not much to say about the actual project, it is given a name to tell what it is a project about, other than that it uses a process which determines many of the properties within it.

A process is a description of how the project is managed. What process is chosen affects what roles is available in the project, and what type of permissions which is assigned to each role. The process is used to determine how the flow of work in the project is managed. To use an example, two of the default processes available in Rational Team Concert is Scrum, and the Simple Team Process. Simple Team Process is the lightest process of the two, with few roles, and few types of work items. It is designed for small teams and beginners which will start of quickly. Scrum is a little bit more sophisticated and deep, with both more roles, and more types of work items. It is also possible to create and define your own processes, that is however advised for more experienced users. This subject will be discussed further later on.

### **2.3.2 Creating Users**

Once the project is created, the project overview will appear, and a lot of new things will show, including the time line for the plans, and the first work items. However they should be ignored as of now. The next thing to do is to create some actual project users.

When creating users, the users will be created outside of the project, the reasoning for this is of course if there exists multiple projects on a server, odds a that some users will be involved in more than one project. In such a situation it is handy that users lies on the outside of the projects, and can be imported in (Picture 4 in the Appendix). Members within a project will be given a role within the project. These roles will be discussed later on.

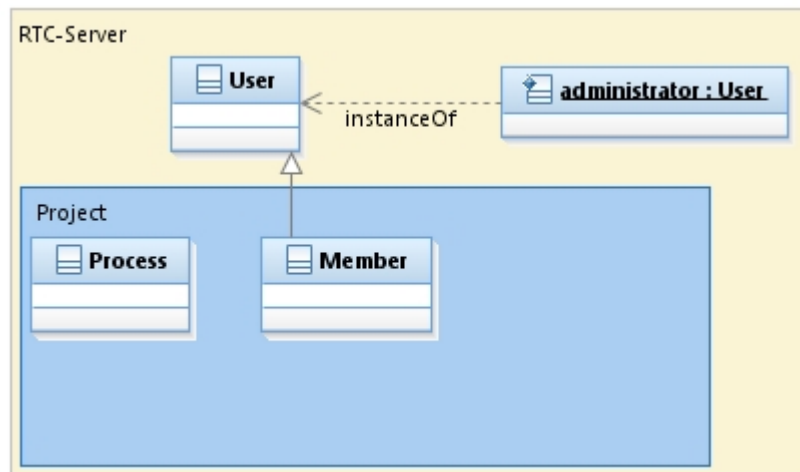


Figure 6: Second of the step by step figures. Project, process and members added.

Since the last figure there is now three changes. First of a project is created, and with a project one is forced to choose a process which affects some of the properties. In addition users has been created, and added as members of the project.

### 2.3.3 Creating a Team

Teams and projects might be a little confusing at first, the easiest way to think of it, is that a large project will most likely need to be cut down into smaller teams which takes care of one part each within the project. However if the project is small, and consists of few members, there might not be a need for teams within the project. Team and project within Rational Team Concert is more or less the same thing, except that teams is within the project, and that a project can contain many teams. The team area works in the same way as the project, giving members roles within the team. By default the roles within the team inherit from the project, so roles in a team will have the same properties as they do in the parent project.

### 2.3.4 Inviting a user to a project, and accepting it

The users are created, but the persons which shall use the profiles still need to be invited (Picture 5 in the Appendix).

If an e-mail system is available, there will be auto composed a standard e-mail, which the administrator can send to e-mail which was assigned to the user-profile.

The new user will usually sit on his own machine, but in this case, since all in this thesis is done locally, it is important not to use the same workspace for the new user as another user. If it had been on one machine in a big network, there would not be an issue. (Since workspaces would have been on different machines then.)

When a user accept an invitation, the repository connection will be created for the user (and not have to create this manually as the administrator did), and the user will be able to work inside Rational Team Concert once this process is done.

### 2.3.5 Creation of work items, plans and iterations

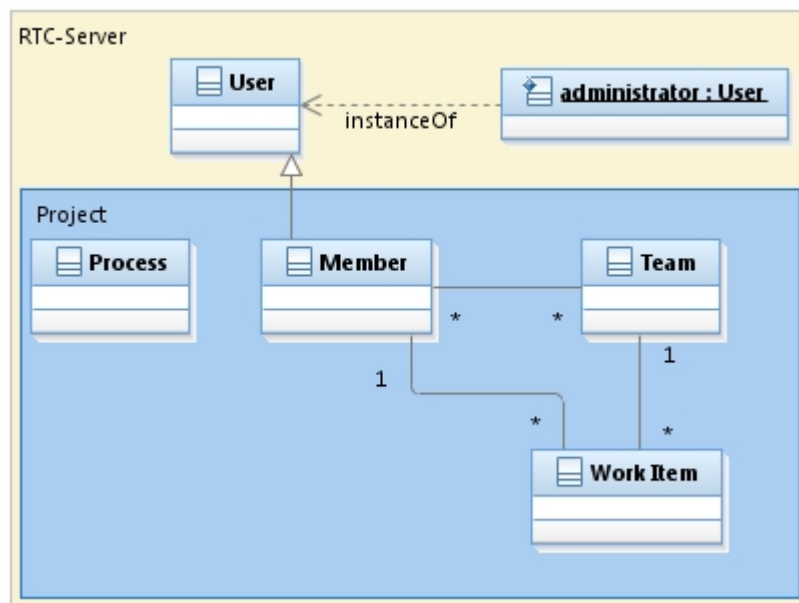


Figure 7: Third of the step by step figures. Team and Work Item now added.

Since last figure, there is now created a team which consists of members within the project. After that Work Items which consists of the various assignments to be completed is created. Everything which shall be done within a project will have some kind of work item tied to it, so that is why it is important to mention it as one of the essential parts.

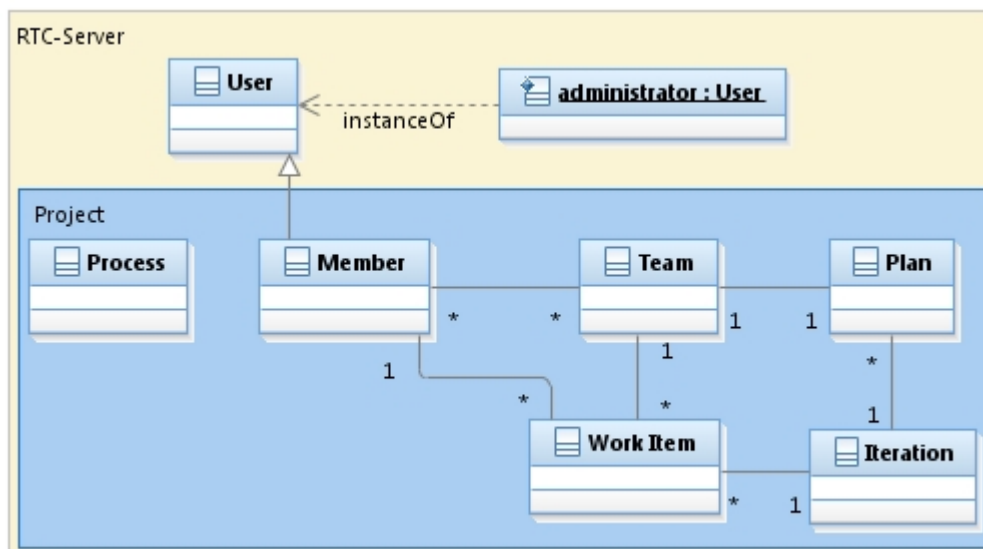
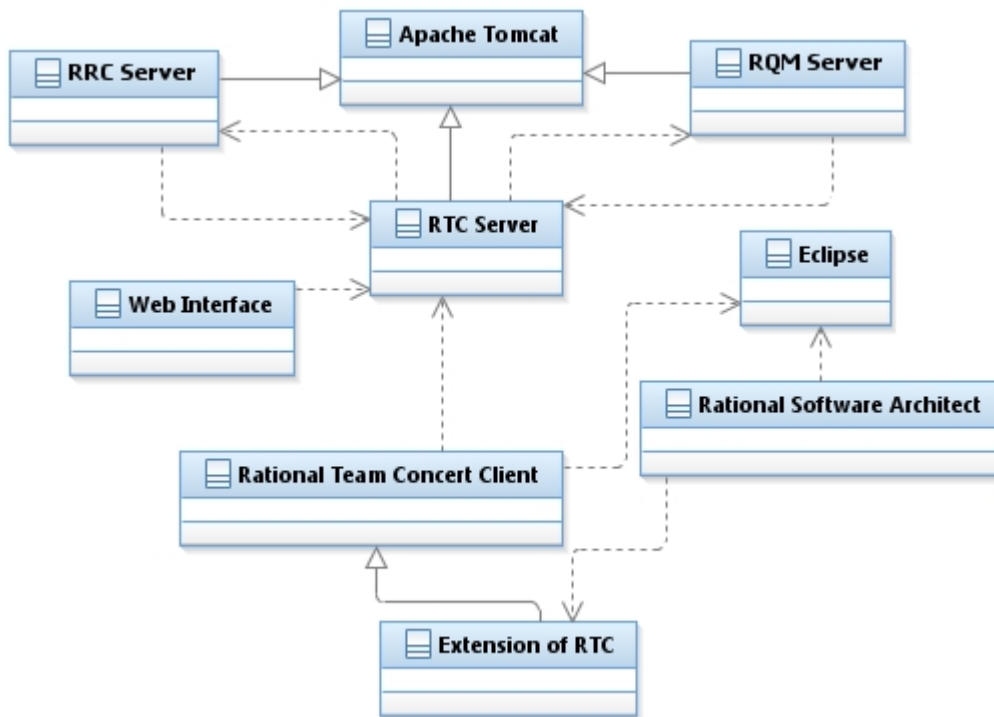


Figure 8: Fourth and final of the step by step figures. The last step is to add iteration and plan.

In the final figure in this step by step series, there is added plan and iteration. The plan is created for the team to give a good overview of who does what in a team. There is also created an iteration

for the start of the project.

## 2.4 Architecture of Rational Team Concert



*Figure 9: The architecture of Rational Team Concert, including when integrated with Rational Software Architect.*

This figure shows the architecture of Rational Team Concert, in addition to how Rational Software Architect fits into the picture. Both Rational Team Concert and Rational Software Architect is built upon the Eclipse client, which is an integrated development environment (IDE). Originally it is used to create Java applications, but since Eclipse has a broad variety of plug-ins, it is possible to use number of other languages today. It is however also possible to get a version of Rational Team Concert which is built upon Microsoft Visual Studio IDE.<sup>10</sup>

Rational Team Concert consists of two different parts, the client (built on Eclipse), and a server. This server is either an Apache Tomcat, or a Websphere server.

Naturally, the client is dependent on the server, as all projects will be saved on the server. The reasoning for including Rational Software Architect in this figure, is that Rational Team Concert does not provide any sort of modeling options as a standalone application, and since it is possible to integrate Team Concert inside of RSA with the help of an extension.

When the extension of Rational Team Concert is installed, it will be a part of Rational Software Architect. But since Team Concert is also using a server outside of instance of the client, so Rational Software Architect is also required to establish a connection to the server of RTC. If this is not is not done, the extension of Rational Team Concert installed within RSA is utterly useless. The profiles of the users is required to log in to the server, if not logged in, the user will not be able to use anything from RTC within RSA. By installing this extension inside of the RSA-client, every possibility from RTC is available in RSA, alongside the possibilities that RSA itself brings.

When the installation is finished, RSA is dependent on both Eclipse, and the extension of RTC. In the figure the RTC-client is a generalization of the extension of RTC.

There is also some dependencies between the RTC-server, RQM-server and RRC-server, however, this is of such importance, that it will be discussed in a later chapter.

## 2.5 Licenses and Permissions in Rational Team Concert

Within the server, there is created users which is associated with the program, but on the outside of the projects. As there are team-roles which gives exclusive rights within a project, there is also permissions and licenses which states which type of user a specific user is and what rights it has.

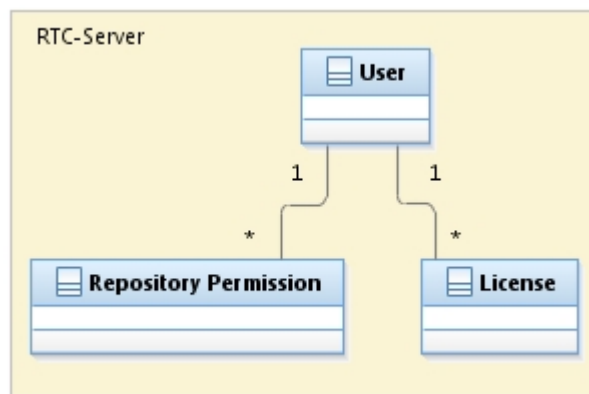


Figure 10: First part of the license & repository permissions model.

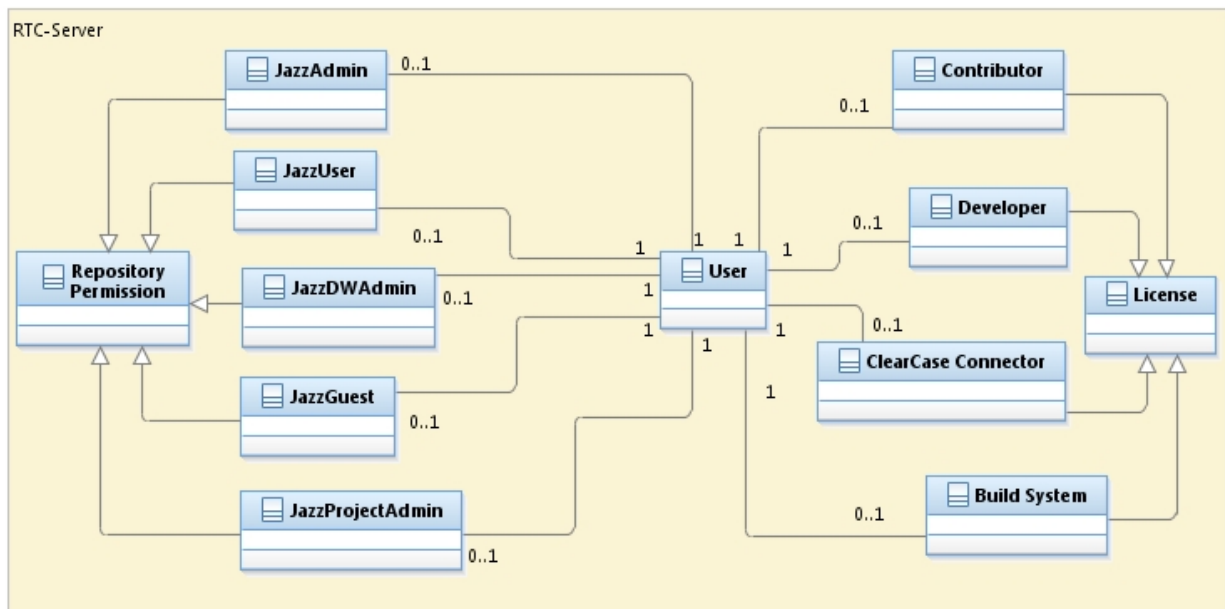


Figure 11: Second part of the licenses & repository permissions model.

To explain the licenses and repository permissions, there is two figures needed. The reason for this, is that one user can have all the available licenses, thus a 1-\* cardinality. But at the same time, the



user is not allowed to have more than one of each. Thus the 1-0..1 cardinalities in the second figure. It might have been possible to show this in one single figure, but it would have been very unreadable with all these associations in one figure, therefore it is better to split it up in two halves.

The first one does not need much explaining, users can have more than one license and repository permission. The second one shows that one user can not have more than one of each type of license or repository permission. At the one side we have the repository permissions, which is related to the rights towards the server (the repository) where all data is restored. There is five types of different repository permissions, as can be seen on the figure. <sup>11</sup>

- ✓ JazzAdmins
- ✓ JazzDWAdmins
- ✓ JazzGuests
- ✓ JazzProjectAdmins
- ✓ JazzUsers

The first one is the JazzAdmins group, users which has this type of permission has full read and write access to the server where all data is stored.

The second one is a special administrators group, while this group does not have full read and write everywhere, they have administrators rights to control special data warehouses on the Jazz server.

Third we have the JazzGuests group, which naturally are guests which only are allowed sneak peeks on the server, users which belong to this group is not allowed to write anything to the server whatsoever.

JazzProjectAdmins is a group which allows users to have administrator rights at specific projects, and are able to adjust such as project areas, team areas and which type of process template a project shall follow.

The final group is the JazzUsers group, which is by far the most common group, these types of users have what is considered regular read and write access to the repository.

Licenses is the other part of this figure, and is also connected to the user. However, licenses does not specify any relationship between the user and the server. Licenses says something about of which rights the user has against in the client and within projects. There are four different types of licenses within Rational Team Concert <sup>11</sup>:

- ✓ Contributor
- ✓ Developer
- ✓ ClearCase Connector
- ✓ Build System

There is a difference between the two first, and the two last types of licenses. The two first is intended for human users within the organization, while the two latter is intended for users which are non-human.

The Contributor has read-only access, with the exception that they can create work item queries (used for searching existing work items) and work items. This type of license would probably be typical for a stakeholder, which is not going to create anything, but still has interest in the project. The Developer gives a user full read and write access within a project. This is naturally the type of licenses which are given to everyone which works within Rational Team Concert, in one or more projects.

ClearCase Connector licenses is assigned to non-human users which grants the user the ability to extract information from source control streams.

The Build System license is the last license, this type of license is granted to a specialized user which is supposed to be doing the build process within a project.

As a last note, Contributor-, Developer- and Build System licenses will be restricted based on which type of roles (these roles will be addressed later on) they are given within certain projects.

## 2.6 Rational Team Concert with Simple Team Process

This is a specialized version of the items which are affected when choosing to use the Simple Team Process process template. (Figure 12, see next page, due to using a landscape page for this figure.) Simple Team Process is a very simple template which is one of the default processes which can be chosen when creating a new project. This template is mainly included because it makes up for a very good start point for a first project. It is meant to be used by beginners, or projects which are very small, includes few developers and not a lot of source code.

On the very outside of this figure we find the user, which is associated to be a project-member inside of a project. Inside the project, the member is associated with a project role. In Simple Team Process there is only one role available, which is the "Team Member" role. The project role is a generalization of this "Team Member" role. As seen on the drawing, "Team Member" lies within a rectangle named "Simple Team Process", the classes/objects within the rectangle is those who made available because of the use of the process Simple Team Process. A link between the process class and the the Simple Team Process rectangle is also made.

Within the blue rectangle, everything which is associated with the Team class is placed, a Team-Member is associated with the project-member, since a project-member can be part of numerous teams, with being a Team-Member, there is also a role within the team. The roles available in a project and a team is the same types, and relates to the process which is chosen.

In addition to this, "Work Items" and "Plans" is also associated with the team, however, as seen from the figure, these two classes is related to both the Team rectangle, as well as the Iteration rectangle.

The "Work Items" class is a generalization of the three types of work items which is made available by using the "Simple Team Process" process. These three are Task, Enhancement and Defect.

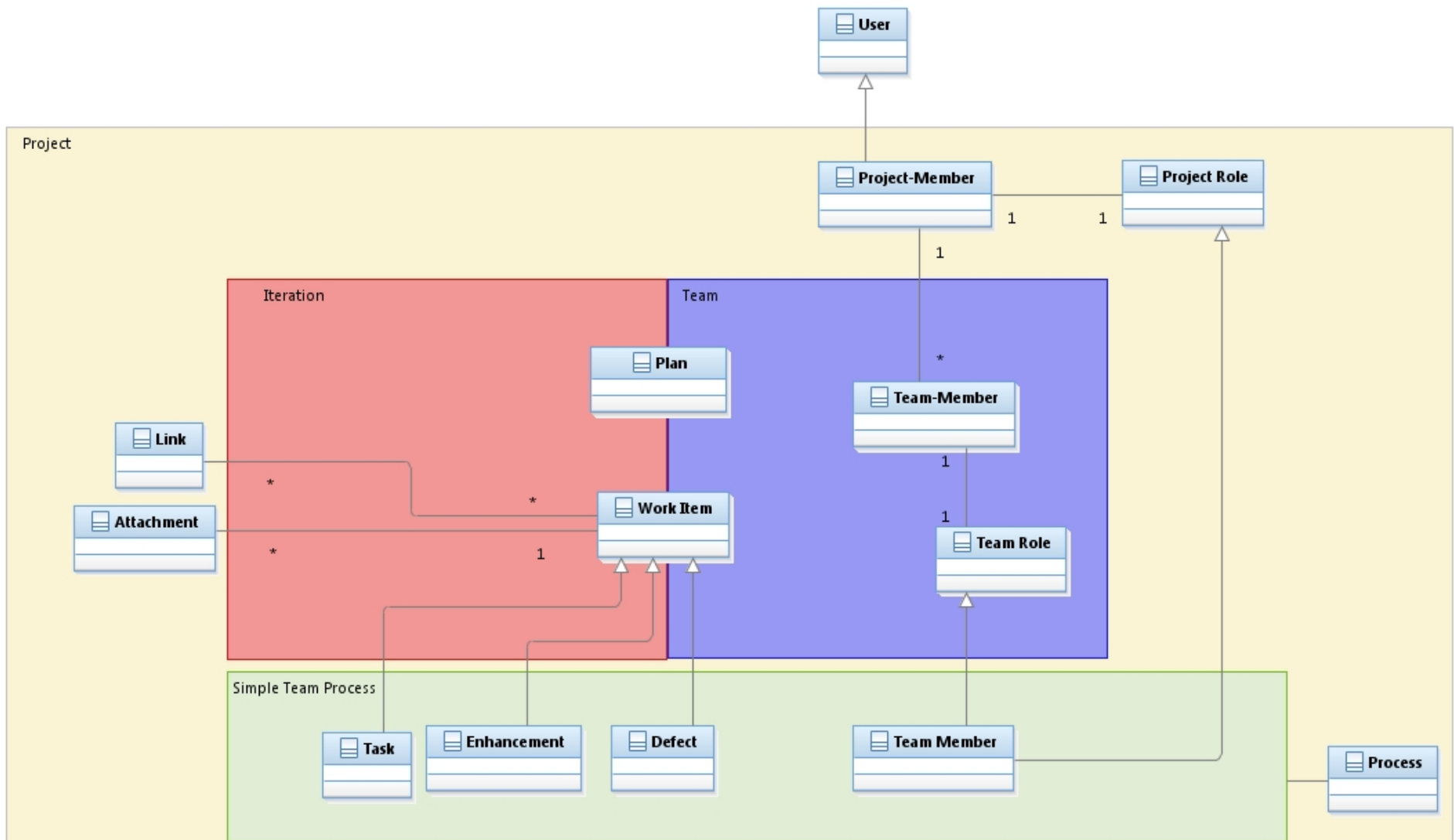


Figure 12: Rational Team Concert project when using Simple Team Process.

## 2.7 Rational Team Concert with Scrum

As a project consisting with a scrum process brings along a lot of possibilities, there will be several figures in this section, first three figures focusing singlehandedly on Work Items, Members and Roles, and the last one will focus on what effects choosing Scrum has on Plans and Iterations. In the end, a figure which includes everything from the three previous will be shown, but it is easier to show it separately at first, and then put it all together in the end.

### 2.7.1 Scrum, focus on the different Work Items

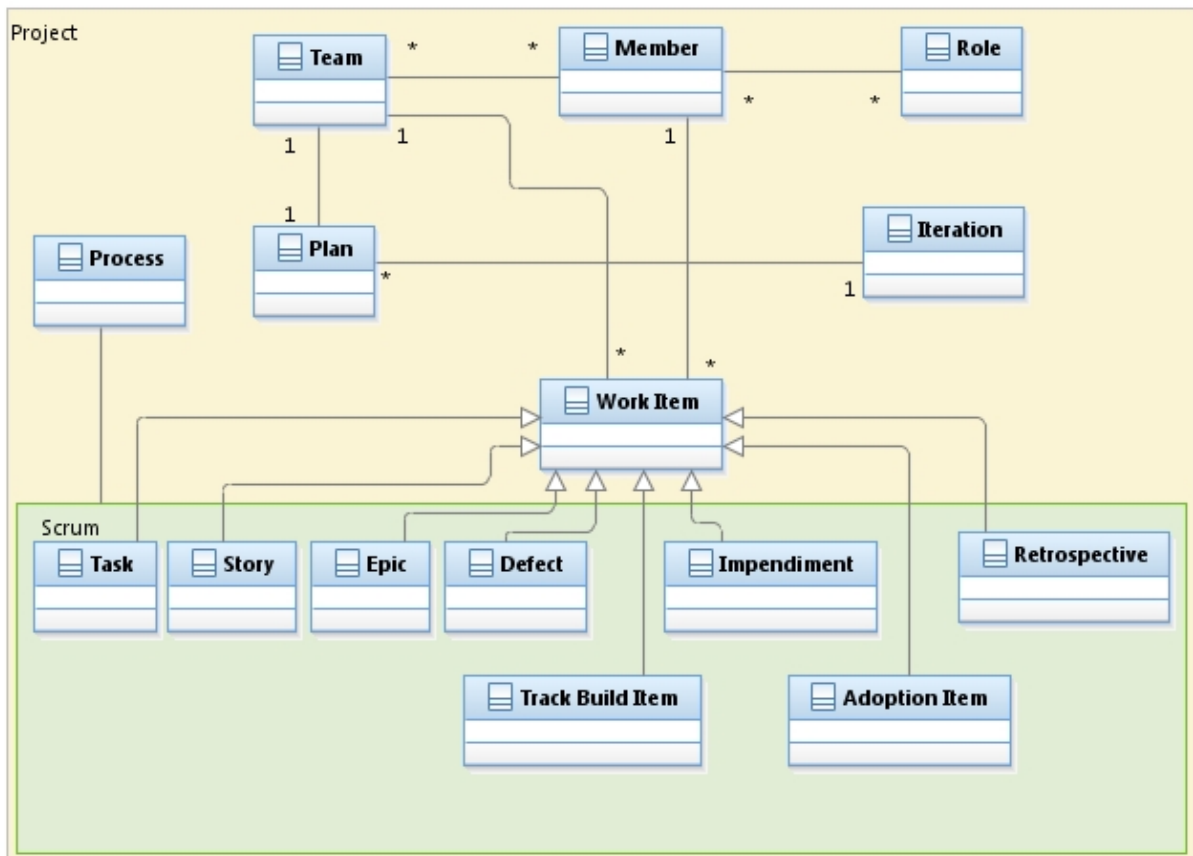


Figure 13: Rational Team Concert project with Scrum, focus on work items.

Watching the figure, one question might be why these other classes are in this figure, Members, Plan, Iteration and Role. The reasoning for having these in here, but not the other classes which exist within a project is that all the classes are directly affected by choosing to use Scrum, which the two next figures will show. This figure shows what type of Work Items which will be possible to create if the project is based on the Scrum process. The Work Item class is a generalization of these eight types, and each of them is a specialization of the Work Item class.

The names of the eight specialized Work Items is <sup>12</sup> :

- ✓ Task
- ✓ Story
- ✓ Epic
- ✓ Defect
- ✓ Track Build Item
- ✓ Impediment

- ✓ Adoption Item
- ✓ Retrospection

These are all types one can choose from when you are about to create a new Work item. All the default work items created in the initialization of the project are of the type task. Task is also the most common work item in Rational Team Concert, with the rest being more specialized cases. Task is the most ordinary work item, and describes something which needs to be done. A story is used to describe a part of the use case. Epic is used when a story is too big to be completed in one iteration/sprint. An epic is formed by two or more stories, and thus can be broken down to several stories.

Adoption item tracks when a team needs to adopt changes from another team to stay updated.

Defect is identifying a bug which needs to be corrected.

Retrospective identifies what was working good, and what was not working good in the previous iteration, seems to be like an evaluation of previous work.

Impediment item is supposed to track things which could hinder progress of development

Track Build Item is created for finding out what is wrong with a build and what needs to be done for it to complete correctly. These could possibly also have been placed as defects, but this is a special case for problems in relation with a build.

These work items do have some different properties tied to each type, for example, a story has story points, which tells a team how hard the specific story is.

### 2.7.2 Scrum, focus on Role and Members

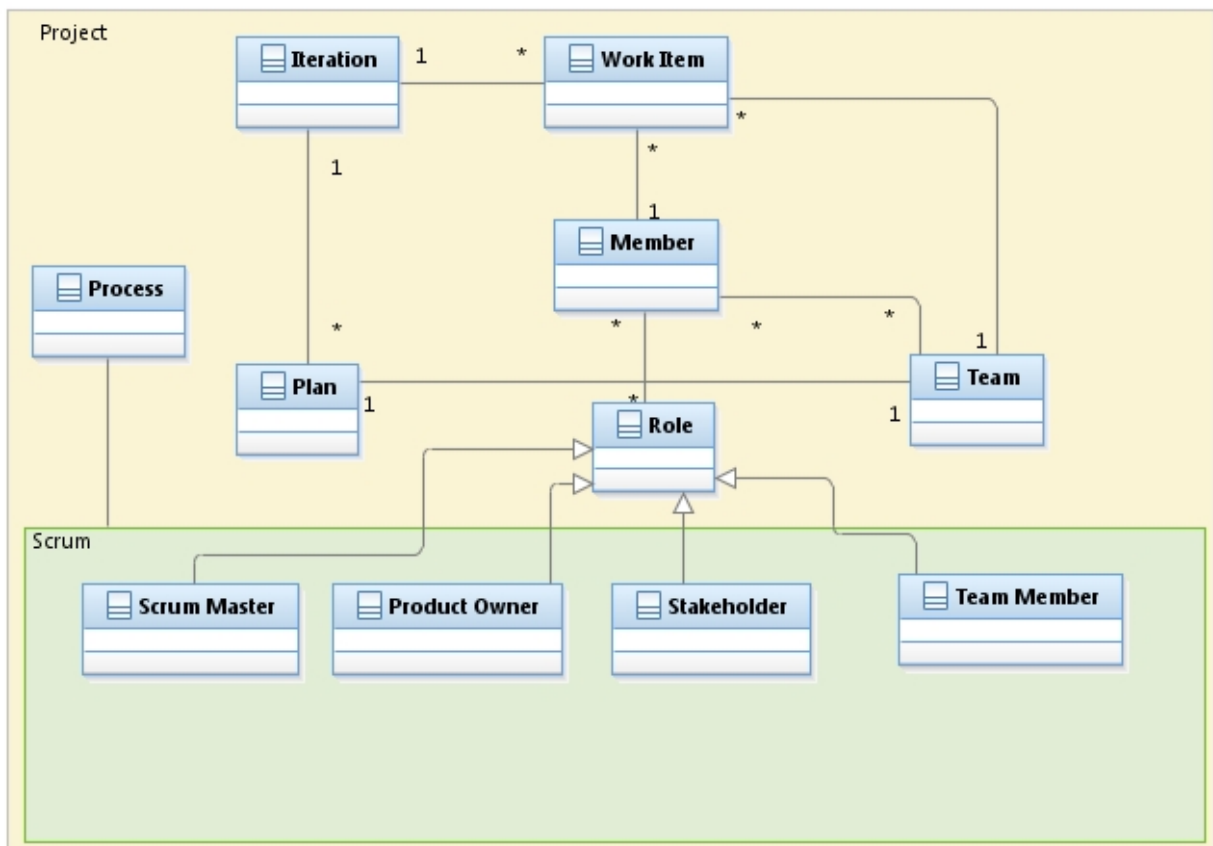


Figure 14: Rational Team Concert project with Scrum, focus on role and members.

In the next figure, it is the same classes which is shown inside the project box, but now the focus is

switched from work items, and onto role and member. In a scrum project, members can be assigned roles both within the project, and also in one or more teams.

Scrum is a process which is a little bit more complex than the Simple Team Process. It has five different types of roles:

- ✓ Default
- ✓ Team Member
- ✓ Product Owner
- ✓ Scrum Master
- ✓ Stakeholder

All these have a specialization-generalization relationship towards the Role class, where these represent the specialization, and the Role class represent the generalization.

There is a default role which is automatically assigned to everyone which joins a project.

There is however a difference from the default role from Simple Team Process, in Scrum the default role has some permissions, There is for example the possibility of saving the personal dashboard, plus saving an attachment to a work item.

The Team Member role in Scrum permissions to do the most of the things available in a project, they can however not save the project area.

Both Product Owner and Scrum Master has every possible permission on all levels, the Product Owner is the person responsible of the Product Backlog, which one could call a "master list" over everything which needs to be done. <sup>13</sup>The Scrum Master is the person which is responsible for the process in the project.

The last role in Scrum is the Stakeholder role, which is a role given to a user which has a stake in the product being produced. The role grants very limited access to the project, other than reading, since the user is on a level of interest, and not a level of being part of producing the product.

Now, to assign roles to each individual, just mark the user, and click the process roles button, and add and/or remove the desired roles from the user.

### **2.7.3 Scrum, focus on Plan and Iteration**

In Scrum there is also automatically generated some plans and iterations unlike in Simple Team Concert, there is not auto generated any plans or iterations. (Figure 15, next page)

Looking at the specific objects which are created while using a Scrum template, Scrum uses a strict setup when it comes to the planning, with three backlogs <sup>14</sup>, which are instances of Plan:

- ✓ Sprint Backlogs
- ✓ Release Backlogs
- ✓ Product Backlogs

Then there is also created four different instances of Iterations;

- ✓ Release 1.0
- ✓ Backlog
- ✓ Spring 1.0
- ✓ Sprint 2.0.

Both Sprint iterations is sub iterations of Release 1.0.

The three plans are all linked to one iteration each, Release Backlog is linked to Release 1.0, Sprint

Backlog is linked with Sprint 1.0, and Product Backlog is linked with the Backlog iteration.

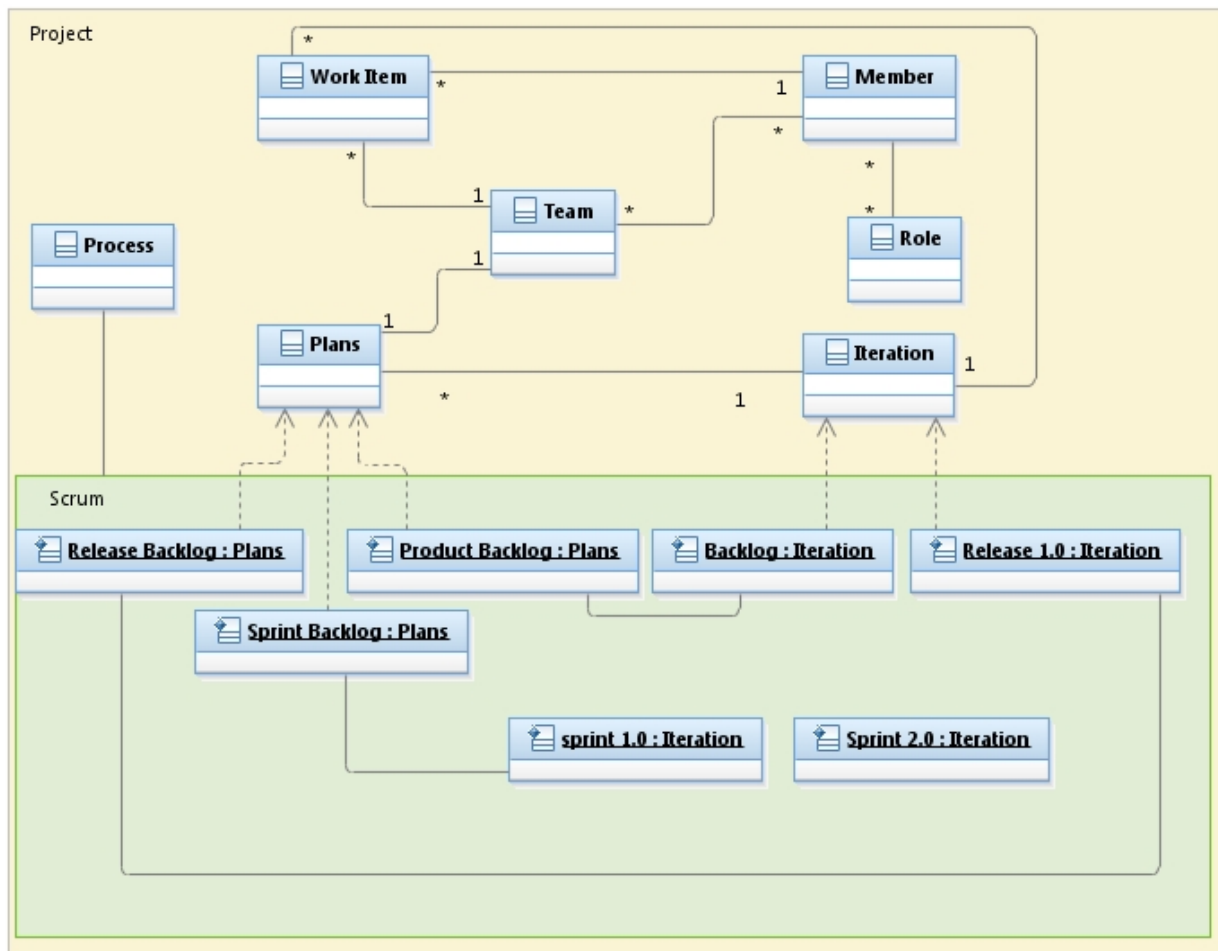


Figure 15: Rational Team Concert project with Scrum, focus on plans and iterations.

Product backlogs usually contains the high level goals and requirements, often provided by the stakeholders, this backlog is maintained through the whole project. The release backlog breaks down the requirements from the product backlog, into low level requirements and more specific problems to be solved.

The sprint backlog is the backlog during a sprint, a sprint can be defined as given time, say for example ten days, which a set of goals shall be completed.

When opening a plan, there is two important tabs (three in total). The first is "Overview", where the goals and summary of the plan will be written. The other tab is the "Planned Items" tab, here will there will be an overview of the different members of the project or team, and it will be possible to assign work items to each of them. This is done by dragging the work items onto the member it will be assigned to. This leads to a simple question, how to show available work items.

## 2.7.4 Complete overview model of RTC with Scrum

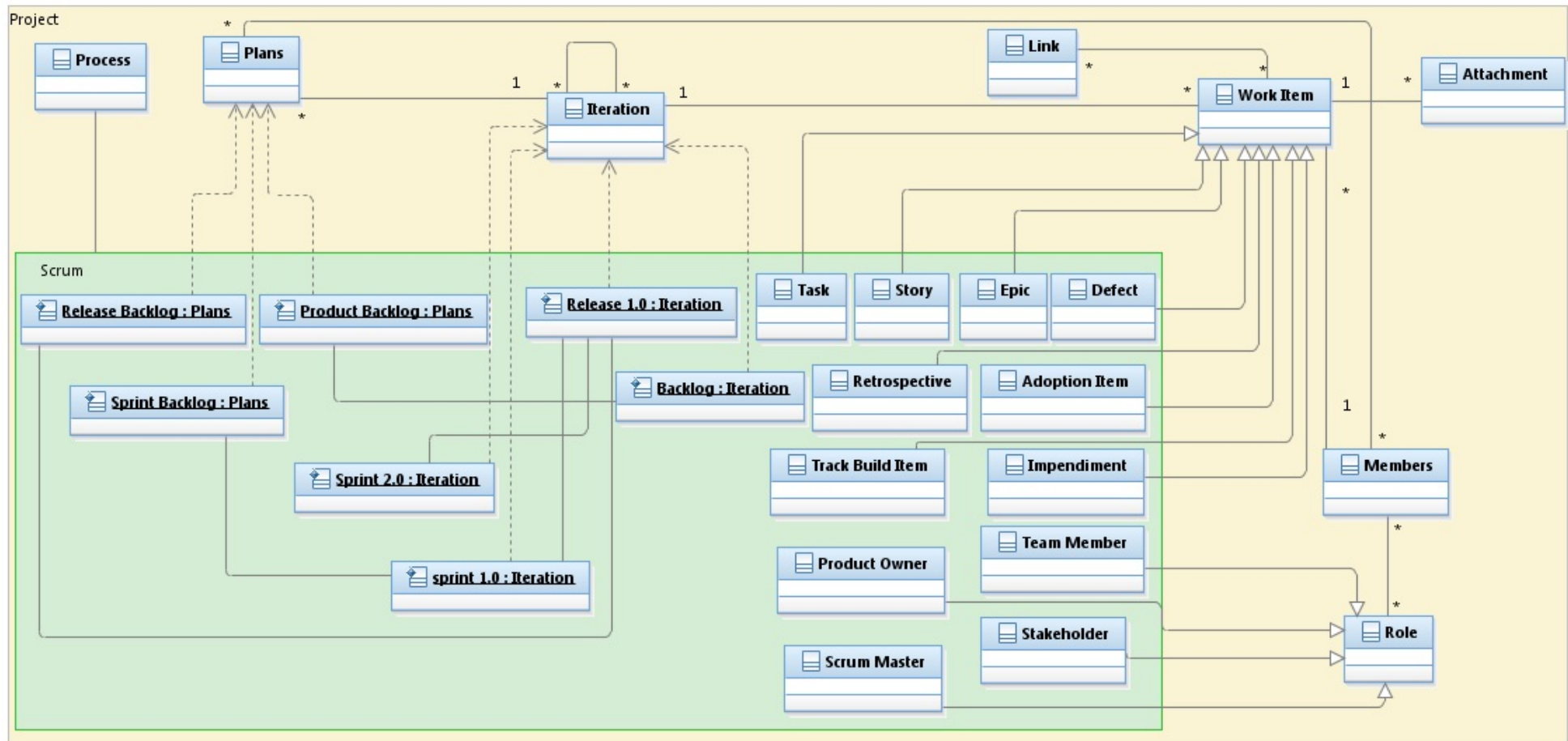


Figure 16: Rational Team Concert project with Scrum, complete model.

As the figure for Simple Team Process, this is a specialized version of a project, where all the items which are affected by choosing to use the Scrum as the process template for a project. Scrum is a more advanced process when comparing it to the very simple "Simple Team Process", as can be seen from this model. It is more suited for users which have some experience with Rational Team Concert, as well as working in team based environments. Everything which is located within the green box, is associated with Scrum, and a result of using Scrum as the process for projects.



## 2.8 Source Control

Jazz Source Control is the heart of Rational Team Concert, without it, everything else is just plain pointless if it was not possible to share code with each other, which is what source control is about. The first thing worth mentioning is that to share work, a stream is used. This stream can either belong to the project (this one is created automatically when the project is created), or to a team. If it belongs to the project, everyone which is set as members of the project can access it. If it belongs to a team, everyone within that team can access it.

**Note:** To specify one thing, even though a team is lying within a project, it is NOT enough to be a member of a team, to listen to a project stream. A member has to be in the specific members list for the project, before being able to get information from the project-stream.

The second thing which is needed, is that a user has a Repository Workspace, these are not created automatically. It is important to not confuse "Workspace" and "Repository Workspace" for the same thing, because it is not.

### 2.8.1 Creating Repository Workspace

A repository workspace is a location on the server which are delimited for a specific user, and connected to a stream, so that the user can see what is currently within the stream. This workspace is essential for being able to share through a stream to all the other members which are listening to the stream which is shared on.

When creating a new repository workspace, you will have to choose a stream to flow with, for example the project stream. (Screenshot, next page) After this you will give the workspace a name, and choose the reading access, it can be private, public or scoped. When, finished, the repository will open another wizard, to state what shall be loaded. There is five different options here, but the obvious one is to load all the eclipse projects. (all work which is done) After that the repository workspace is ready to use.

### 2.8.2 Delivering code to a stream

The next part is very essential, so this will be described with an example with details at a level which the rest of the report has not done.

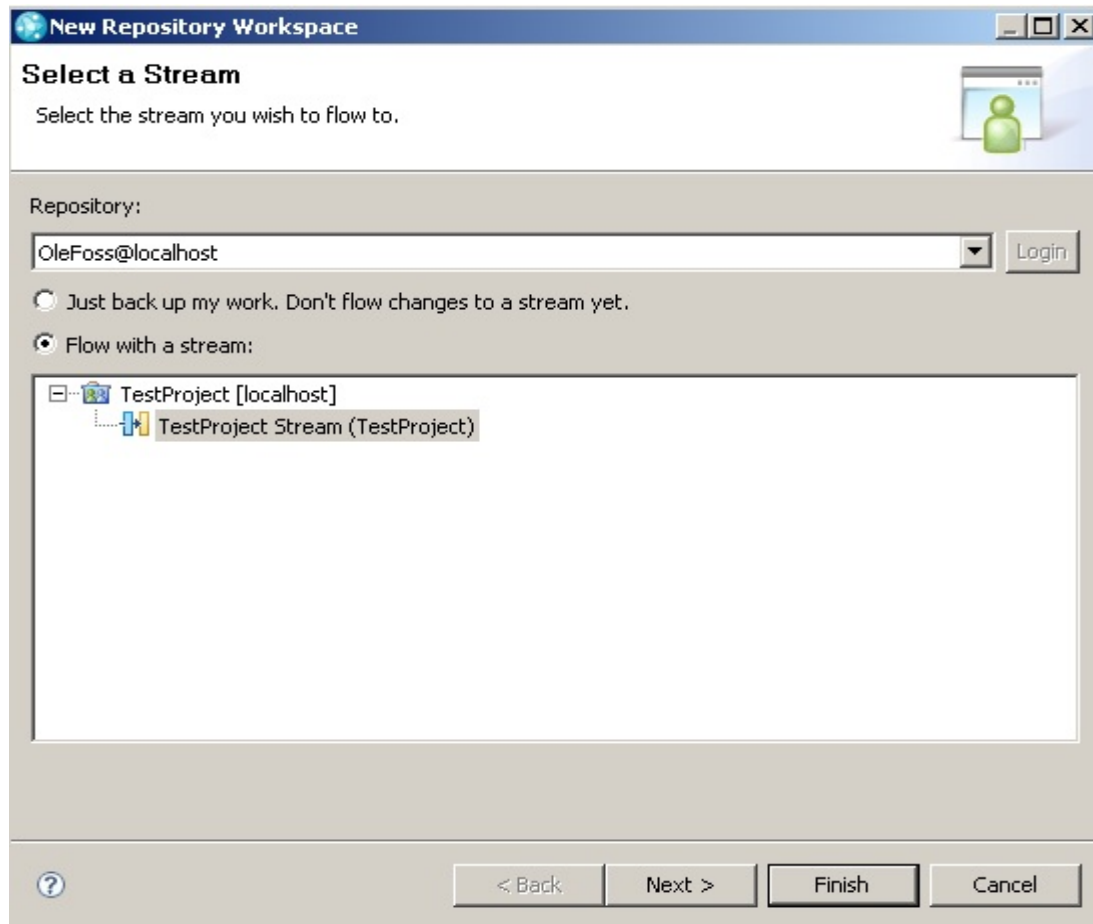
Now everything is ready, so that Java-projects can be shared over a the project stream.

To share a project the first thing need to be done is to maneuver to the Java perspective of Rational Team Concert. Right-click the project to be shared, and select "Team" and then "Share Project".

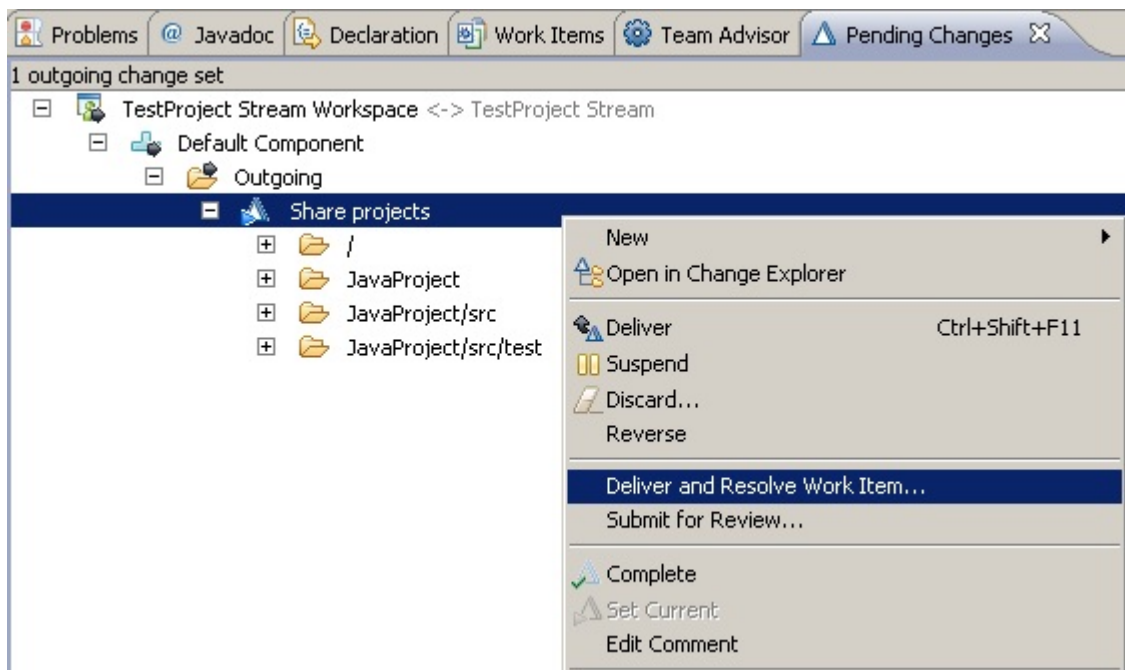
In the wizard appearing, Jazz Source Control shall be selected and then move on to the next screen. Now select the component which is found expanding the repository workspace just recently created, and then finish. Now the project is loaded into the repository workspace, the next step is to deliver it to the project stream.

A tab named "Pending Changes" will appear in the bottom window. Expand the repository workspace until the "Share Projects" option appear. Right-click and choose "Deliver and resolve Work Item" (Screenshot 2, next page). Next up is to associate a work item with the code being shared. The work item "Share code with Jazz Source Control" (This is a work item which is automatically created when a new project is created) is perfect for this.

Search for the work item, and choose it. When delivering something new to the stream, it is needed to add a comment to the delivered work to inform the other members of what this is. It is also possible to associate a member which shall approve it, if that is necessary. When this is finished, it is delivered to the stream. Now all other members of the project can access the shared code through their own repository workspace.



*Screenshot 1: Creating Repository Workspace, and choosing which stream to connect with.*



*Screenshot 2: Pending changes before putting a project into the stream.*

### 2.8.3 Checking changes on another account

To check how the sharing worked, log in with another user which is a member of the project. Select a repository workspace which is connected with the project stream. The "Pending Changes" screen will appear in the bottom window, and tell about the incoming change set from the other member. Right click and accept, and the Java project will be loaded into the Java perspective of the member. It is as simple as that. There is a little thing which is a little bothering, and that is that the member have to load its repository workspace manually to see if there is any incoming changes from other members of the project. In a perfect scenario one would imagine that an auto refresh which updates the state of the repository and notes the user of changes would be very useful.

One last thing about the sharing is the concerns of how multiple users can update the same files, and how this is done. For example, we have a project P in the initial state, and two developers D1 and D2 working with this project. Assume both start working from the initial state at the same time, and then D1 delivers his progressed work. When D2 now delivers his work, without including the changes D1 did, will D2's work overwrite what D1 did?

The answer to this question is no, there will be an error if a user tries to deliver without accepting the latest changes, if this happens the user could either merge the files himself, including the latest changes, or let the program try to auto-merge the files. It is however not always possible to auto merge.

## 2.9 Generalization model of RTC and corresponding object model

This model is probably the most important of all the models in this section, as this shows a brief overview of the possibilities which lies within the Rational Team Concert.

All of this will be within the RTC-server, which is the reason for putting a RTC-server box on the outside. The user which is created is on the server level, and is created with one or more repository permissions and one or more licenses. A user can be part of multiple projects, and that is why it needs to be place on the outside of the project box.

Inside the project box there is a lot of classes, starting with the process, which is not directly tied with anything else then the project.

The member class is set as a specialization of the user, this is done because the connection to the repository workspace is to the user, rather then the member within the project. A repository workspace is possible to reach for a user from all projects he is a member of, not only within the project where the repository workspace was created. A member is also associated with the different sub-projects he is part of, as well as the building of the code. In addition, there is associations with the teams he is part of, and the work items he owns.

A work item has several associations, as which iterations it is planned in, and which team is supposed to solve. It can also be associated sub-projects, as in coding-projects and modeling-projects (Which requires Rational Software Architect).

An iteration is only associated with plans other than work items, a plan is the schedule for a team within one iteration. The length of the iteration can either be time-framed, or decided by the amount of work which shall be done in the iteration. The iteration can be associated with several plans (from different teams), but a plan can only be associated against one iteration. The coding-project also have an association with the builds which is creating the runnable applications.

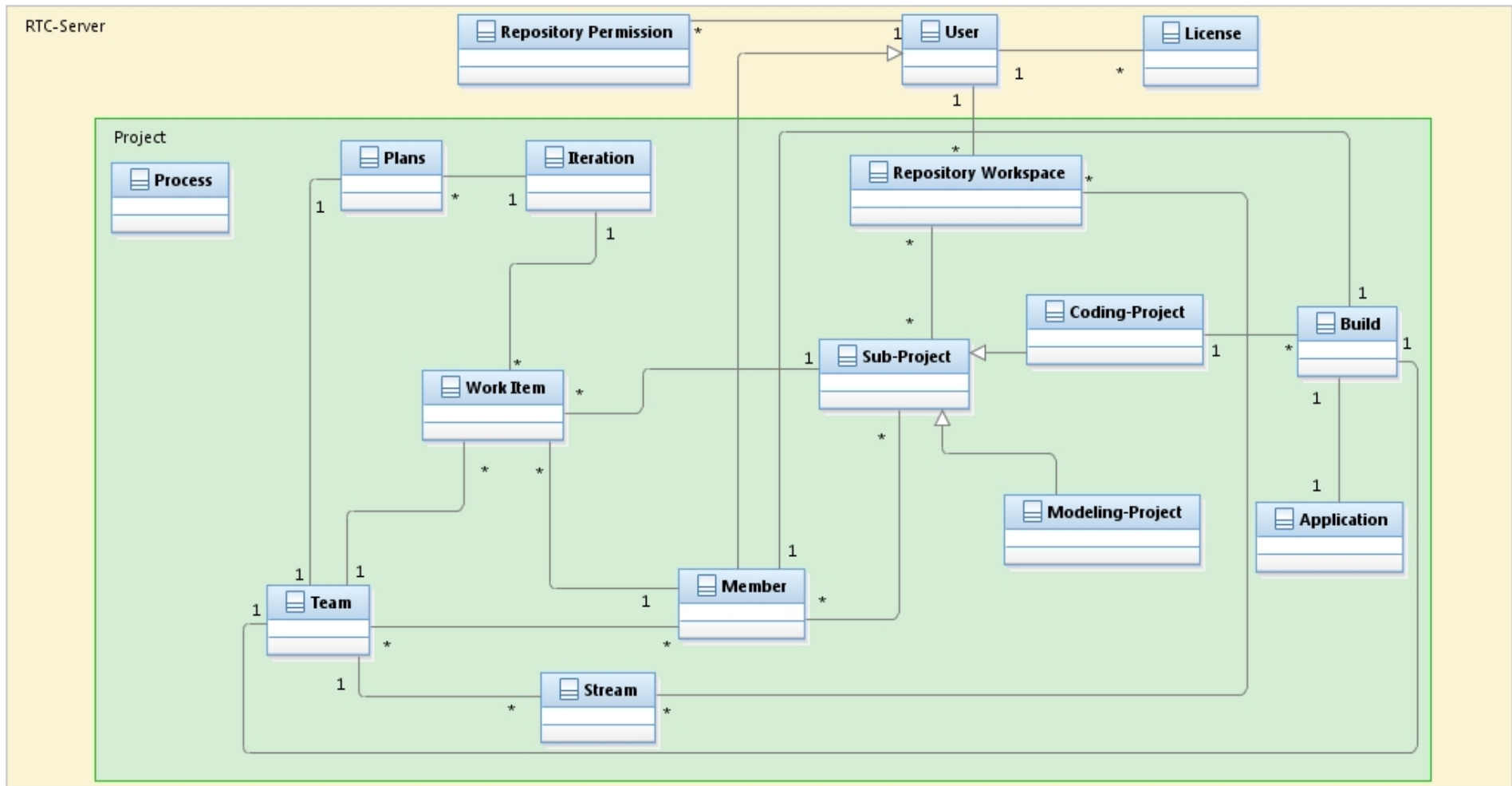


Figure 17: Generalization model of Rational Team Concert.

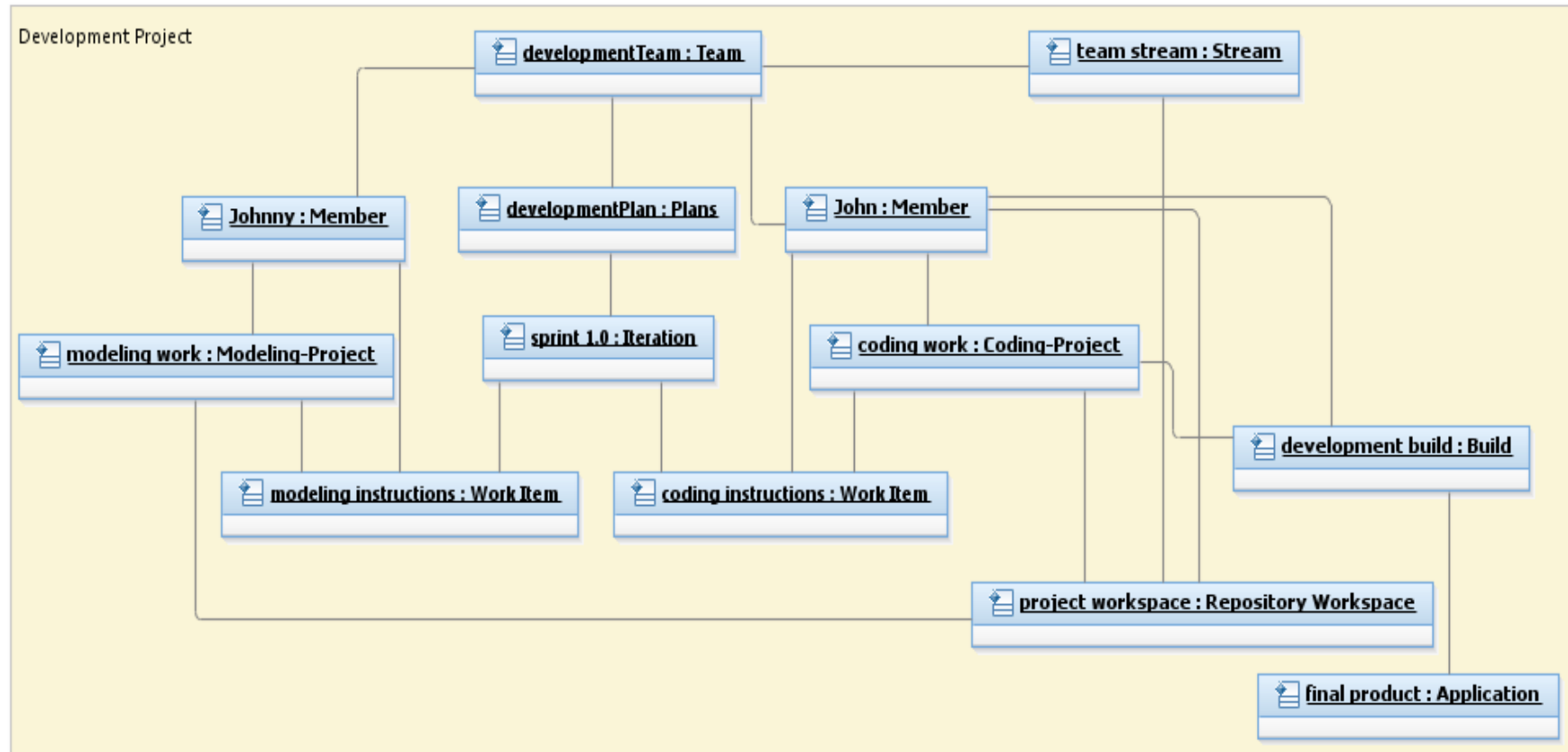


Figure 18: Corresponding object model to the generalization model for Rational Team Concert.

The team has two other associations which has not been mentioned. When a build is completed it will be linked against the team which started the build. The other association is a stream. This stream is crucial for a team to work. All work which members do is done locally, so that the other members of a team shall be able to get hold of this work, it needs to be shared through a stream. A stream can either be linked to a team, or to the whole project itself. For a code project to be shared, it first has to be put in a repository workspace, and then can be put to the stream. This is also the reason why there is no direct connection between coding- and modeling-projects and streams, since it is required to put it in a repository workspace first.

The two final objects is the build, and the application, all the builds associations has been mentioned earlier, except the one to the application, or in other words the final product. The application is of course a result of a series of builds, since builds will be performed very often, probably every time there has been significant changes to the source code.

## 3 Rational Requirements Composer

### 3.1 Configuring the server

When starting up for the very first time, like all Jazz products there is a server which needs to be configured. The setup for the Requirements Composer server is very identical to the setup of the one for Team Concert. This is done by connecting to the server within a browser to the address where the server is placed. In this case the address is on a local machine: <https://localhost:9443/rdm/setup>. Like in Team Concert it is possible to choose a fast path, or a custom path, depending on which type of database the server should be set up with. A server which is meant to be used by a big company would need to use the custom setup path, as the preconfigured path does not support more than ten users. After this the creating of administrative user should be done, and if desired, the default administrative user can be disabled. The only real difference when setting up the servers for RTC and RRC comes in the end. If there is used a previous version of Requirements Composer, it is possible to migrate existing data from the previous version of Requirements Composer, so that it can be used in the new version.

### 3.2 Getting Started

When the server is properly configured it is time to start up the client. When starting up for the first time it is necessary to create a repository connection to the server. To create such a connection, a user name, password and the URL to the server must be provided. When this is done, it is time to create a project, give it a name, and choose a project template. The default templates are given, one is sample project with a lot of predefined artifacts. The second has no artifacts, but creates default folders and attribute groups. The last one only create attribute groups. Folders are used to group the different artifacts and different type of categories.

Attribute groups is the attributes which are attached with each artifact, like creation date, artifact type, which user created the artifact and the name of the artifact.

#### 3.2.1 Artifacts in Requirements Composer

When opening up the personal dashboard, there is one very large viewlet in the middle of the screen, this viewlet focuses on the last artifacts which has been changed. Artifacts is definitely the most important term when talking about Requirements Composer, the whole program revolves around artifacts, and how to use them get a good overview of the tasks and requirements for the an creation or upgrading of for example a program or web page.

Inside the program there is a lot of different main artifacts which can be created, a total of twelve be exact:

- ✓ Requirement
- ✓ Glossary
- ✓ Business Process Diagram
- ✓ Document
- ✓ User Interface Part (UI part)
- ✓ User Interface Sketch
- ✓ Screen Flow
- ✓ Storyboard
- ✓ Actor
- ✓ Use Case
- ✓ Use Case Diagram
- ✓ Collection

Of the twelve main artifacts it is without doubt that requirement artifacts is the most important one of the bunch. A requirement is probably best described as a criteria which the final product is supposed to meet. For example it can be to deliver specific information through to a customer of a web shop. If this person wants to buy a product, a requirement could be that a product which has been added to the shopping cart, actually appears in the cart. All scenarios, big or small is some sort of requirement, and that is why this artifact is so very important in a system like this, for analysis and planning.

A requirement in Requirement Composer can be created in some different ways.<sup>15</sup> The most common and natural way is to go via "Create Artifact" button in the project menu, and thus get a totally new artifact. Another way, if in another requirement (or document), it is possible to mark a part of the text and mark it as an requirement and creating it that way. The last way to create a requirement is to mark a graphical object in artifacts which contains graphical objects (a sketch for example), and the push the "Show Requirements" button, and creating a new requirement in that way. Requirements also have its own tab in the side menu in every artifact, which shows which other requirements are associated with this exact artifact. All in all, this is definitely the most important artifact, and it is not very easy to describe it. The easiest way is probably to say that without requirements this program lose its purpose.

The next artifact is glossary, a glossary is a list with extended explanations of words which needs it. If there is any uncertainty of what a word means in a certain setting, it can be a good idea to add the word to a glossary and write an extended explanation of the meaning. Each word within a glossary is a term, which in itself is a child artifact. It is not listed as one of the twelve artifacts for the reason, that it is not possible to create a term artifact directly form the project menu. A term has to be created within an existing glossary.

The business process diagram is a powerful modeling tool within Requirements Composer which focuses on business modeling of a program or of a system. It uses a business modeling process notation to show the flow through a program, or a part of a program.

The business modeling can be divided into two different parts<sup>16</sup>, either simple processes or "Business-to-Business" processes. The former is the smallest of the two, and might for example show a specific part within a larger system. For example, searching a catalog in a web shop, modeling this exact event would be a simple event.

A business-to-business process would on the other hand probably show the flow of the whole

session, from accessing the site, search for the product, put it in a shopping cart and in the end purchase it.

The document artifact is exactly what it is described, a document, nothing fancy about it. It represents a text editor within Requirements Composer. It has the most important capabilities which a text editor should have, like font size, bolding/italic/underlined, left/center/right-alignment for the text, numbering and bulletins. It is very handy to have a text editor within the program instead of having to open Microsoft Office or Open Office externally. It possible to store these types of documents within the program, but it is not possible to open them within the program.

The user interface part artifact is a graphical tool to create parts to use in a larger sketch, it is very useful if there is need to create a part which shall be used more than once. An example might be a type of menu on web site. Such a menu will be used more than one time, as menus usually appear no matter where you actually are on a website, so it will be shown on all pages of the site. There is a lot of different objects which is available when creating a UI part, labels, buttons, text field, text areas, menus, panels and image placeholders (frame to import a picture into). These are a few, and maybe the most important objects available.

User interface sketch is very much alike the user interface part, except it shows more. Many user interface parts might create one user interface sketch. A typical user interface shows one web page, for example the main page of web site, which probably consists of some headlines and some menus. The objects available in the user interface sketch is the same as the ones available in the user interface part artifact.

Screen flow is an artifact which is used to show the flow between sketches, if we assume an example where one sketch represents one page, the screen flow will depict how the flow through the website is. For example in a web shop, starting at the main page, then go on to search for an item, the next screen shows the possible matches. Then it might be plausible to view some details of the item of interest. Further put the item in the cart and go for checkout. The screen flow shows the flow from one screen to next. This might seem similar to a storyboard, but the screen flow is a model, which the storyboard is not.

Storyboard is the last artifact which is closely related to the user interface artifacts, and the screen flow artifact. It shows step by step using sketches how a system can be utilized, hence the name storyboard, because it tells a story in detail with full screen sketches from beginning to end. Each step in a storyboard is called a frame. The sketches does not necessarily need to change from each frame. The difference from one frame to another might for example be that a user has filled in some information in a text field. An example could be a user which has provided some information when purchasing a product, like a credit card number if that is the payment method.

Use case, actor and use case diagram is so closely tied that they will be explained together. Both the use case artifact and the actor artifact, are documents which explains in detail what the use case or actor represent in an actual use case diagram. A use case id something that can be done with the system, program or website. In a web shop, a use case can for example be to purchase a product from the store, another use case could also be to search the website for the product. The actor on the other hand is the one which you could say performs the use cases. In a scenario of a web shop, there could be a customer which is the actor which purchases a product. Another actor could be a site owner which is responsible for that the products in the shop actually is in stock.

The use case diagram artifact is a visualization of the use cases, actors and the associations between



them. In this artifact within Requirements Composer it is also possible to use placeholders. There is placeholders for both actors and use cases. If an actor placeholder is used in a use case diagram it means that there is no actor artifact tied to that specific actor object in the diagram. The use case placeholder is used in the same way.

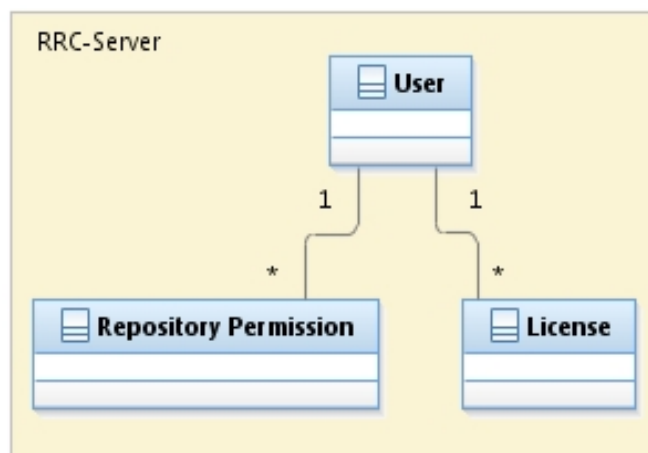
The last artifact is the collections artifact, which basically is an artifact to group other artifacts which the user for whatever reason wants to put in one place. It is possible to either put live versions into a collection, or versions from a project snapshot, if there is a specific version which needs to be used.

### 3.2.2 Reviews in Requirements Composer

The second important part of Requirements Composer is the possibility to create reviews of artifacts. All types of artifacts can be reviewed, and a review can cover more than one artifact at a time. Multiple members can also be part of the review. Each artifact within the review needs to be reviewed separately and is given a status. What status is given depends on which type of reviewer the user is. A reviewer can just set the review as reviewed without specifying if it was acceptable or not, however, that can be written in the mandatory comment. An approver can on the other hand approve it or disapprove it. In addition both type of reviewers can abstain the review. There is one more type, which is an optional reviewer, which is the same as a reviewer, but an optional reviewer does not affect the final result of the review.

The review in itself has four different states, the first one being the draft, which is before any reviewing has started, this is where it is usual to add all artifacts and members which shall participate. The next state is the "in progress" state, which is obviously the state when the reviewing takes place. The third one is the "reviewed" state, which happens when all items have gained a status. The last one is "finalized", which means when it reaches this stage, nothing in this review can be undone, and the review is locked.

## 3.3 Permissions and Licenses in Requirements Composer



*Figure 18: First part of the license & repository permissions model in RRC.*

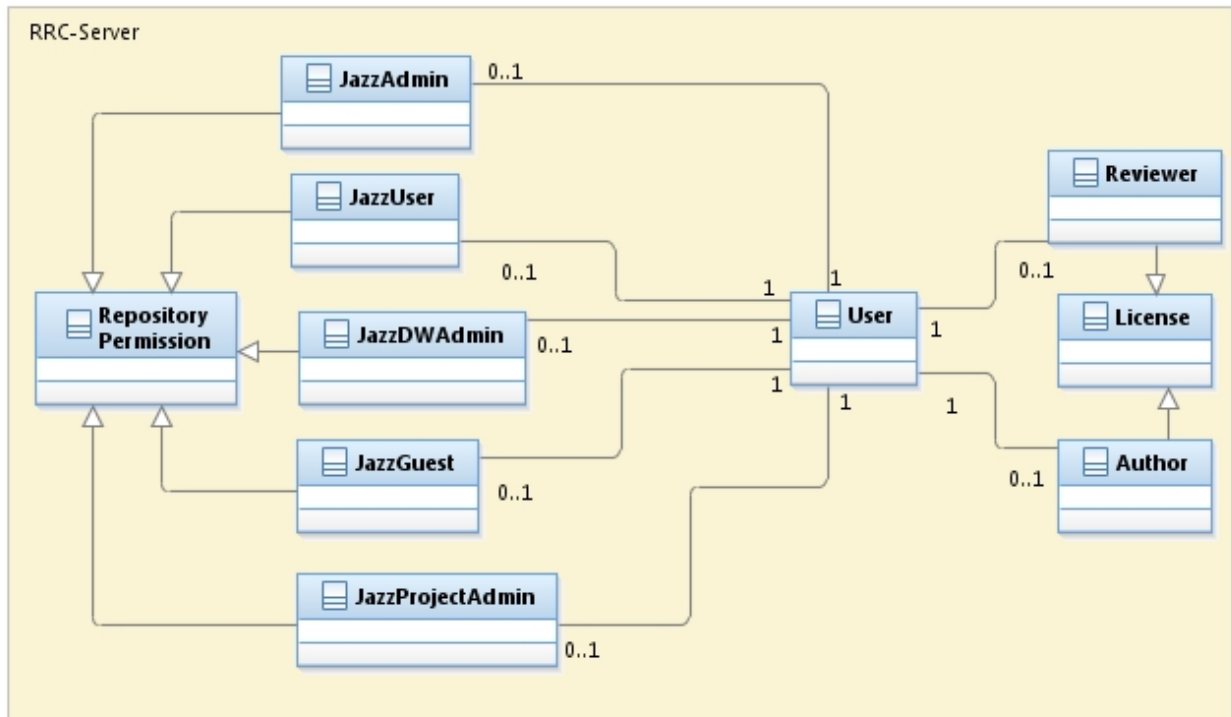


Figure 19: First part of the license & repository permissions model.

A new program, and thus also a new server. Which again means some differences in licenses and repository permissions. The approach in the figures is similar to one which is found for the licenses and repository permissions in RTC. Well, the repository permissions is identical, the server type is the exact same as earlier, built upon Apache Tomcat, so the server side permissions are no different, and consists of the five different types:

- ✓ JazzAdmins
- ✓ JazzDWAdmins
- ✓ JazzGuests
- ✓ JazzProjectAdmins
- ✓ JazzUsers

There is no reason to repeat what each of them means, as this has been thoroughly explained earlier in this report. The big difference though is the type of licenses which is available on the client side of things. In Rational Team Concert there was four different types of licenses, in Rational Requirements Composer however, there is only two different types of licenses which can be provided to the users <sup>17</sup>.

- ✓ Author
- ✓ Reviewer

The difference between these two licenses is pretty simple, if the user is supposed to be able to create new artifacts within a project, it is necessary to assign an author license. This allows the user create all types of artifacts which is within Rational Requirements Composer.

The reviewer license might be comparable to the "Stakeholder" license known from Rational Team Concert. It gives the user read access within a given project, and it is also possible to comment on

existing artifacts, but it does not have the right to create new objects within a project. It is also possible to make personalized tags, so that specific objects is easier grouped, and easier to find and access.

### 3.4 Architecture of Rational Requirements Composer

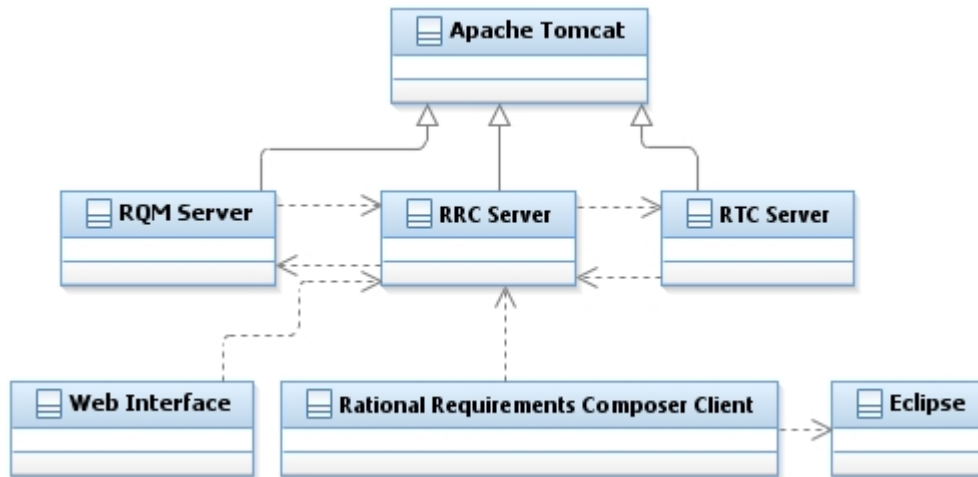


Figure 20: Architectural model of Rational Requirements Composer.

The architecture of Rational Requirements Composer is not overly different from the architecture of Rational Team Concert.

The server is built upon a Apache Tomcat server, in addition there is also the dependencies between the servers of all the three programs, but as mentioned, this will be put to detail in a later chapter.

The client in Rational Requirements Composer is like the one in Team Concert based on the Eclipse integrated development environment. Usually using Eclipse opens for the use of a variety of extensions, but unlike Rational Team Concert it does not seem to be a goal for Rational Requirements Composer, based on what is possible within the client itself.

There are some possibilities to integrate Requirements Composer with other tools<sup>18</sup> which is not been covered in this thesis, the first is Rational DOORS, which is another requirement tool, but this tool is not based upon the Jazz platform, but is rather a tool which IBM acquired when they bought a Swedish company name Telelogic.<sup>19</sup> The other tool which is possible to integrate with outside of the Jazz platform is a tool called Rational RequisitePro. Without knowing too much about this program it is an older requirement tool, and it is possible to transport old work from RequisitePro into Requirements Composer. It might be possible to call RequisitePro the predecessor of Requirements Composer.

One last thing which is a little bit different in architecture is that there is no possibilities to create local workspaces on a local machine with your client. There will automatically be created one single workspace, and should someone wish to switch workspace, locating the automatically created workspace will be necessary. Change the name on the workspace, and there will be created a new one next time the client is started.

This was a quite the problem when trying this program locally on one machine, since it is very handy to be able to log in with different users on a project, a not only one, as these programs are created with collaboration in mind, and it is necessary to see things from different accounts.

### 3.5 General model of Requirements Composer

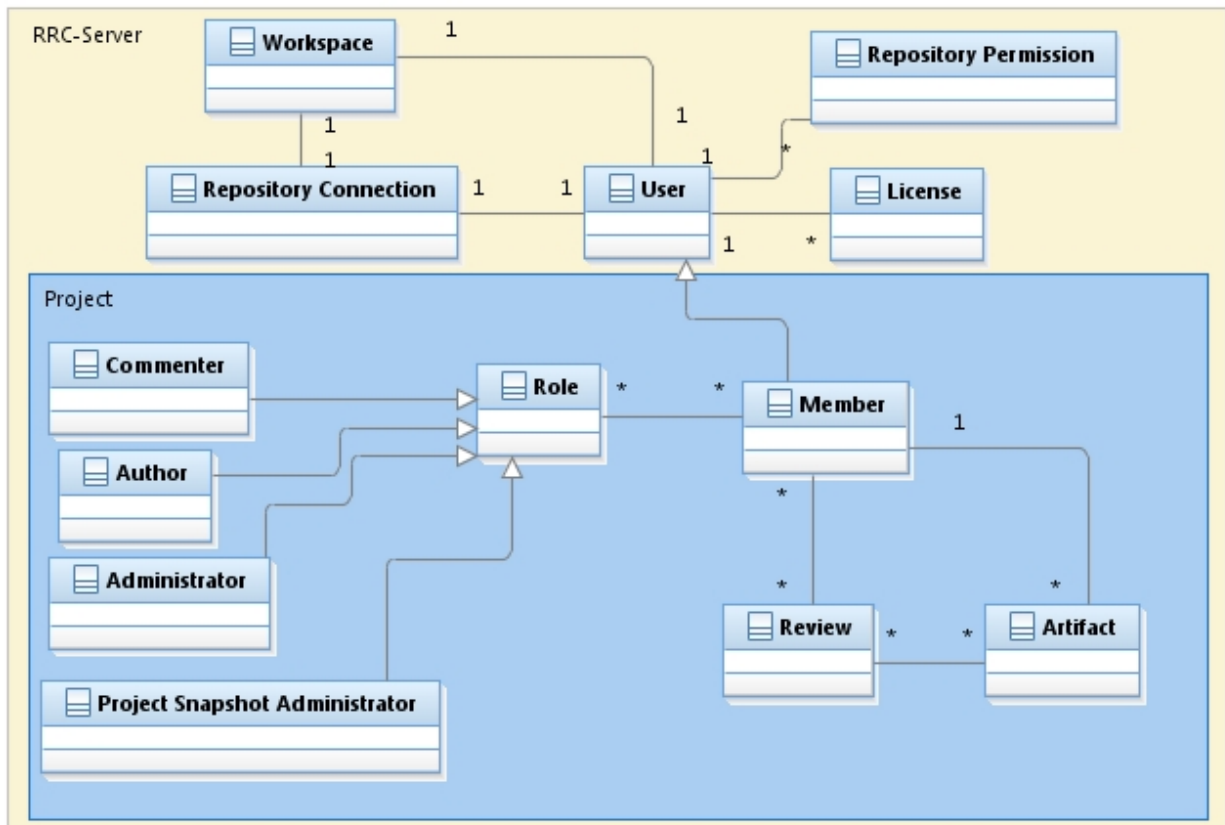


Figure 21: General model of Rational Requirements Composer.

This is the general model of Rational Requirements Composer, and gives a light overview of everything which exists within the program.

First of all, as seen earlier on the outside of the project users are created, these users have at least one license and at least one repository permission. The license says what type of options a user has on the client side of Requirements Composer, while the repository permission states the rights which the users has on the server side of Requirements Composer.

Further, when starting the client, the user is forced to create a repository connection, which consists of the address to the server, the user id, and the password for the account which is provided. When this repository connection is created, there is automatically created a workspace. This address will typically be something similar to this:

```
C:\Users\C:\Documents and Settings\
```

This of course depends what type of operating system there is on the target machine. This workspace is like earlier explained tied automatically to this user, and thus there is a three-way association between the user, repository connection and the workspace. In addition the repository connection has an association with the Requirements Composer server (RRC-server). It is possible to create more than one connection, but this requires different URLs, it is not possible to make two repository connections to a single server for a single user.

Now everything on the outside of the project is explained, so its time to look at what is inside a project in Requirements Composer. As in Team Concert, a member in a project is a specialization of a user in Requirements Composer.

In Requirements Composer there are no teams, well, at least not in client, there is a possibility to create teams in the web interface, but it will not accessible from the client-side. It is just that the administrative side of the server is built the same way in all the three tools, and in the two other teams are needed.

Further it is possible for a member to be invited to review artifacts for other members, the meaning of this is of course to assure the quality of the artifacts. It is possible for more than one member to review the same artifact, and a member can review many reviews (thus the \*,\* cardinality). Reviews will be explained in further detail later. In addition there is as mentioned artifacts, and there is a grand total of twelve different, a figure will be shown a little bit later. A member with sufficient rights can create as many artifacts as he wants, and will be the owner of these artifacts.

The last thing in this domain model is the various roles which is available for a member within a project. A member can have a default role (not in the figure), and after that can have one or multiple roles in addition to that. There is four different roles in Requirements Composer<sup>20</sup> :

- ✓ Administrator
- ✓ Author
- ✓ Commenter
- ✓ Project Snapshot Administrator

The administrator role is rather self-explainable, a user with this role can do everything within a project, add and remove members of the project, such a user can also manage connections to other requirements programs, if those are necessary. (RequisitePro and Doors, as earlier mentioned) An administrator can also change and manage artifact templates. It is also necessary for an administrator to have a an author *license*, NOT to be confused with the author role.

The author role allows the member to create and delete artifacts, in addition to comment on artifacts , be part of reviews and also move artifacts from one folder to another. It is also possible to create shared tags for artifacts as well as personal. The author also needs a author license to be able to do these things.

The commenter role allows the user read access to artifacts, plus the possibility to participate in reviews, comment on artifact and also create personalized tags for the artifacts. For a commenter it is enough to have a Reviewer license.

The project snapshot administrator has the ability to take and manage snapshots of a project, a snapshot could be said to be the version of the project that exists at this very moment.

### 3.6 Requirements Composer review

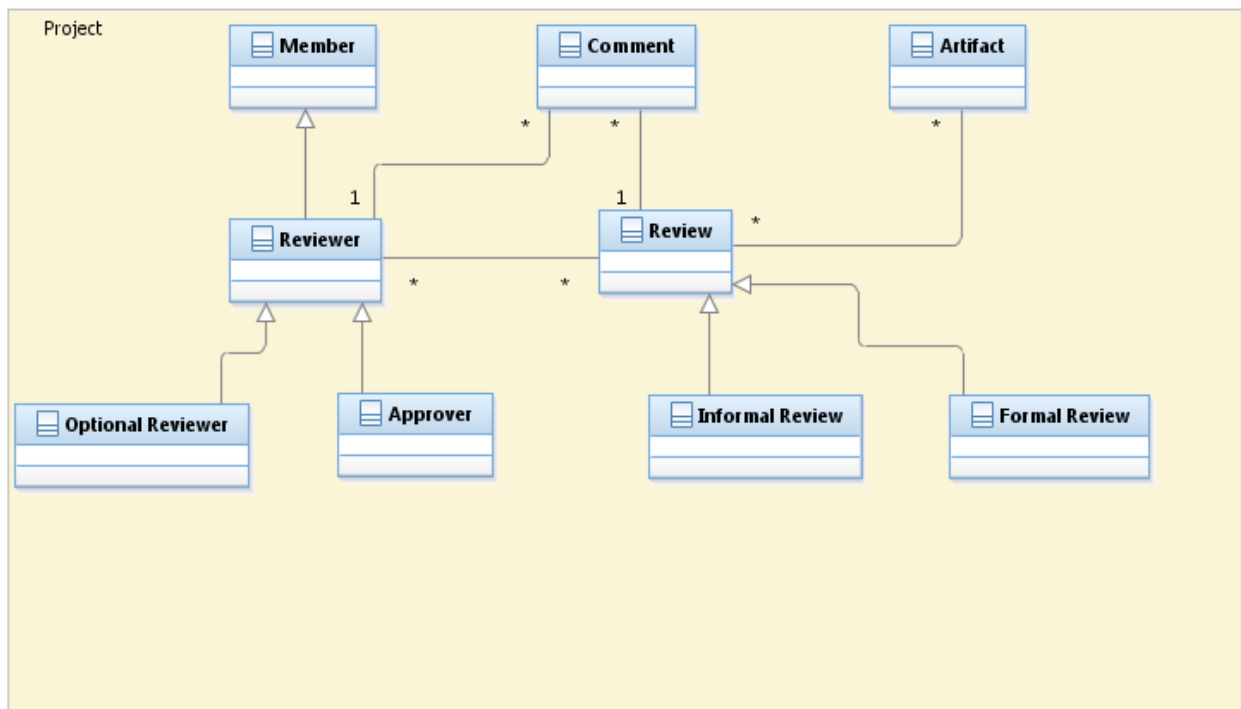


Figure 22: Model of the components tied to reviewing in Rational Requirements Composer.

This is a figure which shows the details of what is tied to the review.

Starting with the review itself it is obviously associated with one or more artifacts, as seen from the figure. In a review multiple artifacts can be reviewed, it is not necessary to create a new review for every artifact.

The review itself is a generalization of two different types of reviews:

- ✓ Formal Review: This is a type of review where there is taken a snapshot of the artifact in question at the time where the review is created. If there is any changes in the artifacts which shall be reviewed, the reviewer will not see these changes.
- ✓ Informal Review: This is the opposite of the formal review, no snapshot is taken of the artifact at the time when the review is created, instead the reviewers will always see a live and updated version of the artifacts.

To review the review, you need a reviewer. A reviewer must be a member of the project, and a reviewer can be one of three different types:

- ✓ Reviewer: A reviewer either reviews or abstains a an artifact, rather than approving and disapproving, and uses the the corresponding comment to say what needs to be said. Artifacts must gain a status before the review can be finished. (reviewed or abstained)
- ✓ Approver: The approver can either approve or disapprove an artifact, but can also abstain as well. All artifacts must gain a status before a review can be finalized (approved, disapproved or abstained).
- ✓ Optional Reviewer: This essentially the same as a reviewer, but in this case the reviewer is invited, and the review of an optional reviewer has nothing to say for the state of a review.

The last association is the comment, a reviewer will have to submit a comment to the review every

time he takes a decision of approving, disapproving, reviewing or abstaining an artifact. So a reviewer will create many comments, and all of these comments will be associated with the review, these comments will show in the comments field in the tab for the review.

### 3.7 Requirements Composer artifacts

This might be the most complicated figure (Figure 23, next page because of landscape page) which is being presented in this report. This is much due to the fact Requirements Composer revolves around Artifacts, this is what the program is about.

An artifact has three different associations, where there is a tab for each tab, links, comments and requirements. Links are either incoming links or outgoing links, in addition links are divided into internal or external link sources. An artifact can have many comments, and the comments is again associated with a member which has posted it.

Artifacts is divided into four categories, where three is main categories, and then there is requirements. The reasoning for putting requirements separately of all the other types of artifacts is as said that requirements has its own tab in every type of artifact. If a Business Process Diagram is linked with an requirement in some way, this link will show up under the requirement tab in the artifact. All artifacts has this tab, and thus the requirement artifact needs to be this high in the hierarchy.

The three other categories is "Artifact without specialized link", "Artifact with specialized link", and "External Artifact". An external artifact is a type of artifact which cannot be created within Requirements Composer, but still plays an important role. Image is such an artifact, as images is very important for the user interface categories (which is graphical artifacts).

Artifacts without specialized links is artifacts which do not have a menu with any specialized buttons to add a link to specific type of artifact. The artifacts which falls under this category is, documents, glossaries, business process diagrams and collections.

The last category is the artifacts which has specialized links, this means that within the program there is buttons to add a specific type of artifact. A very good example of this is the use case diagram. A use case diagram is formed of actors and use cases, and thus in the menu for the use case diagram there is specialized buttons for adding both use cases and actors. Linking to other types of artifacts has to be done through the link tab.

The other four artifacts which is linked in some kind of way is the storyboard, screen flow, UI part, and UI sketch. An UI part is often a single image, and has specialized links to add other UI parts (association to itself in the figure), and images. A UI sketch usually consists of many UI parts, and other types of things (labels, buttons etc.) It has specialized links to images, UI parts, and UI sketches.

A screen flow shows the flow between user interface sketches, and has two specialized links, screen flows (other screen flows) and user interface parts. The storyboard is the last artifact, this artifacts shows a story, where the story consists of multiple user interface sketches. For example a story might be to shop an item in an online store.

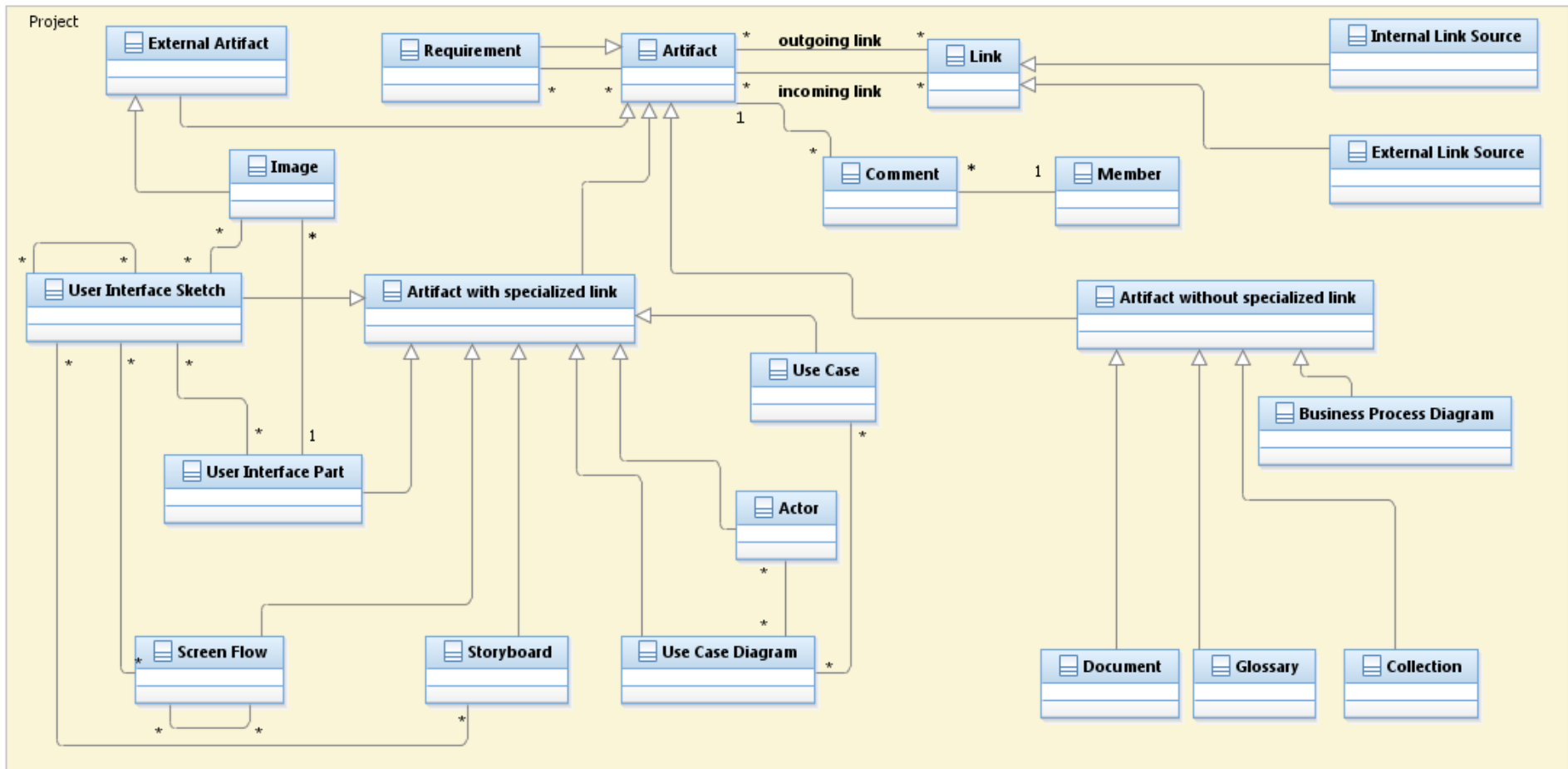


Figure 23: Model of the artifacts in Rational Requirements and the relations between them.



# 4 Rational Quality Manager

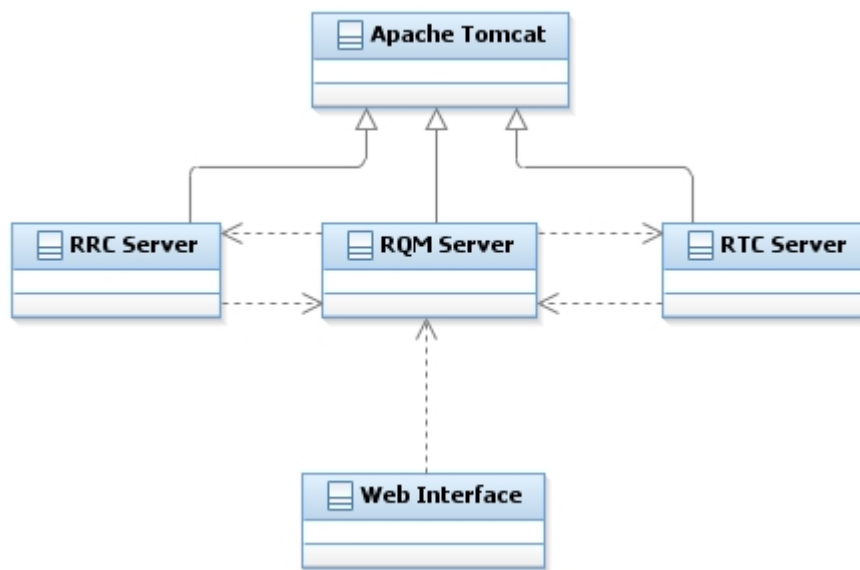
## 4.1 Configuring the server

The configuration of the server in Rational Quality Manager is more or less a mirror image of configuring the Team Concert server, by default it uses the exact same address:

<https://localhost:9443/jazz/setup>.

When having logged in with the predefined administrator user, the steps and possibilities are identical. It is worth mentioning this, but not to write in detail, as this is written in detail earlier under the part of RTC.

## 4.2 Architecture of Rational Quality Manager



*Figure 24: Architectural model of Rational Quality Manager.*

The architecture of Rational Quality Manager is fairly similar to Team Concert and Requirements Composer, but there is one big difference here, which is the complete lack of an actual client which you can work with on your on operating system on your own machine.

The server build is the same as the other two programs, with server being based on an Apache Tomcat server, the server itself has dependencies to the RRC server and the RTC server, if these are enabled.

As said, the big difference is that there is no runnable client for use in an operating system. Instead to get access to Quality Manager it is necessary to use the web interface/web client. The web interface has two different parts, one administrative part, and one which is dedicated to actually using Quality Manager. Overall, it would probably it would always be better to have the choice between using the web interface or an executable client, than be forced to use one of them. This is of course personal preferences from person to person, but I for one like to work in a client, rather than a window in a browser.

### 4.3 Getting started

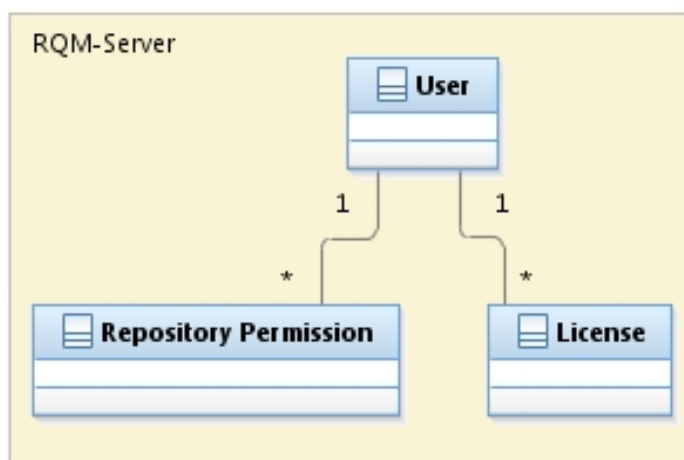
Since there are no client in RQM, there is not a client either to configure to work with the server. Instead the web interface has to be used as a client.

In the starting phase RQM is very much alike RTC since there is projects, teams, users and work items. The roles which users are assigned within the projects are different though and there is a total of five of them:

- ✓ Architect
- ✓ Tester
- ✓ Test Lead
- ✓ Test Manager
- ✓ Lab Manager

In addition to these five there is also a role which is called "Everyone" which is there by default. What is very different in RQM when comparing them to the to other tools when it comes to roles, is that these roles are totally identical. For some reason IBM thinks that each team would be better off to configuring these roles themselves. I can see the benefits of that, but would it really hurt to have a default setup for each of the roles? Leaving them totally identical, without any real rights or permissions just seems unfinished, and this came as a small negative surprise.

### 4.4 Licenses & Repository Permissions in Rational Quality Manager



*Figure 25: First part of the license & repository permissions model in Rational Quality Manager.*

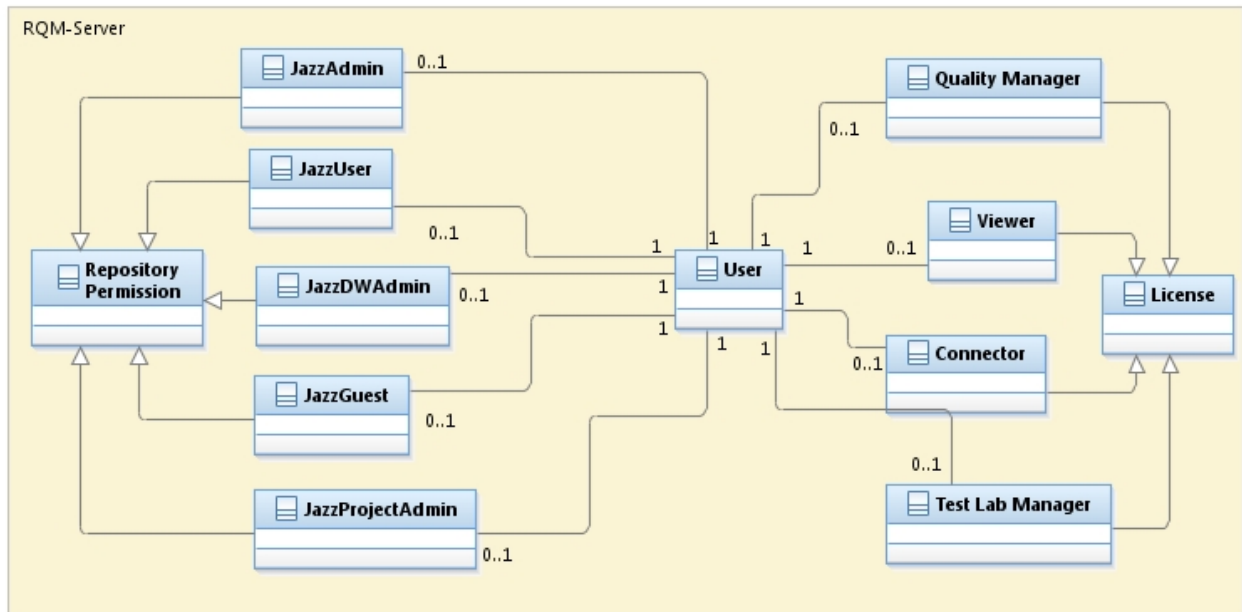


Figure 26: Second part of the license & repository permissions model.

As in the other jazz based programs, Rational Quality Manager uses a Apache Tomcat server. The repository permissions which is available for the server side is the same as the in the two other programs, these five permissions are:

- ✓ JazzAdmins
- ✓ JazzDWAdmins
- ✓ JazzGuests
- ✓ JazzProjectAdmins
- ✓ JazzUsers

No further explanations of those is necessary at this point.

The difference from the other products again lies in which licenses is available for the user on the client side of the program. In Rational Quality Manager there is four different licenses, although one is of the specialized sort, these are <sup>21</sup> :

- ✓ Quality Manager
- ✓ Test Lab Manager
- ✓ Viewer
- ✓ Connector

The quality manager license gives read access to everything in the program, in addition the user is granted write access to everything unless the user is restricted by the role he is given within a project. These type of users are the only one which count against the total user limit which the server can hold.

The test lab manager licenses grants read and write access to everything which is related to lab management. The exception is if the users has a role within a project which restricts this.

The viewer license gives the user read access to everything within Quality Manager, the user also has the right to write to the personal dashboard and work items within a project, unless there is a role-based restriction in a project.

The last license is the connector license which is a specialized license, this type of user is to grant a connector access to import and export with Quality Manager. Since this is not human user, these

types of user is not granted access to the web interface for Rational Quality Manager.

## 4.5 General model of Rational Quality Manager

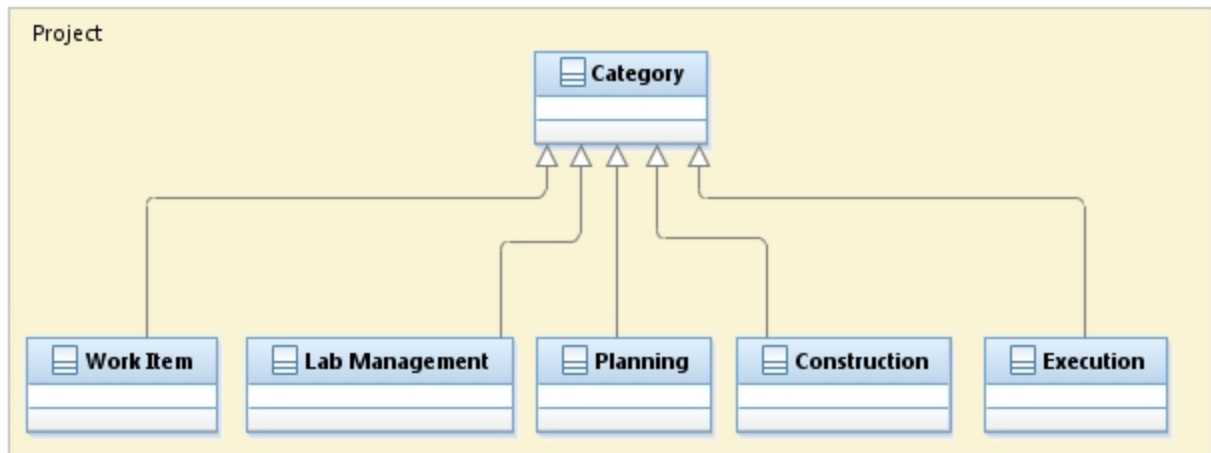


Figure 27: General model of Rational Quality Manager

When modeling the figures in Rational Quality Manager, there has been taken a little bit different approach. Quality Manager divides the program into several different categories inside the program. The categories are not really anything else than headlines to group some objects within Rational Quality Manager.

To use planning as an example, under the planning category is where the test plans are located, you can either look at the existing test plans, or create a new one. The test plan is the one which is associated with other objects from the other categories. But the categories in it self is NOT associated with other categories, even though objects which lies within them are associated.

This is why there is no associations between the different types of categories in this general model, and it might a little bit difficult to understand at this point.

Then one could argue that these categories seems pretty pointless, and in a practical way yes, but they make the structure of Rational Quality Manager much easier to understand. That is also the reason why this approach is used in this report, because the later figures would probably be harder to understand if the objects was not divided into different categories.

## 4.6 Work Items and Defects in Rational Quality Manager

The next figure (Figure 28, next page) may cause a little bit of confusion if you are looking at the menus in Quality Manager, as there is nothing mentioning work items in the menu. However, there is an icon for "Defects", and this is where the work items are stored. The icon on the menu is a little bit misleading that way, because a defect is not the only work item available, although it probably is the most used in a tool like Quality Manager.

A work item in Quality Manager is the same as a work item in Team Concert, but there is other work items to choose from. It is possible to attach files to a work item, if there for example is a document on a local machine which is crucial for resolving the work item it can be attached. In addition to this it is possible to link other work items, if there is anything which they have in common.

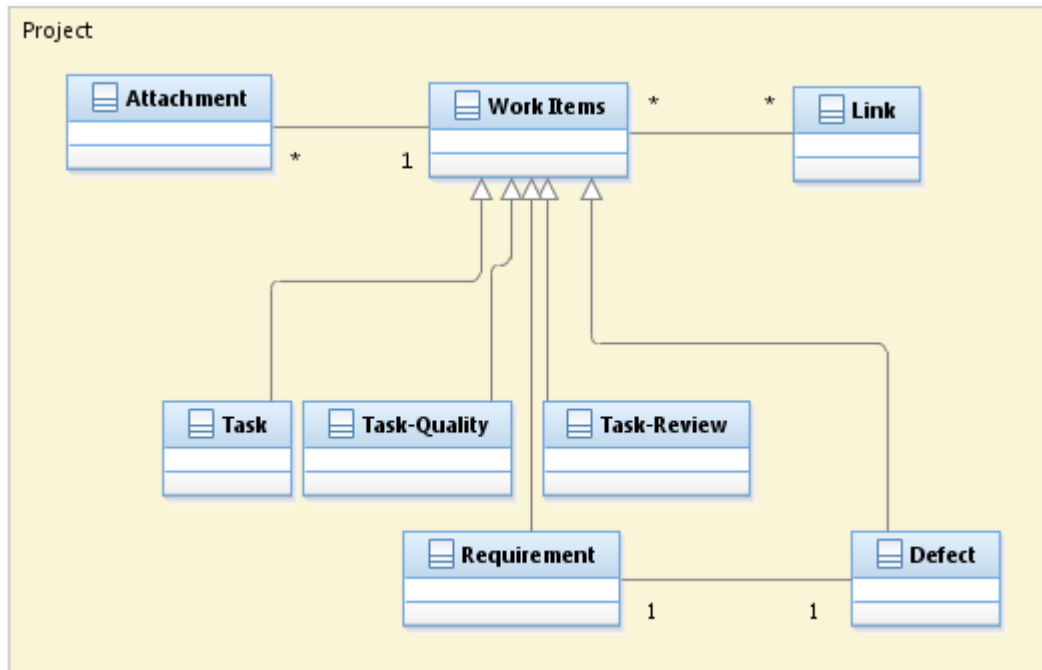


Figure 28: Work items in Rational Quality Manager

There is a total of five different types of work items, these are <sup>22</sup> :

- ✓ Defect
- ✓ Task Review
- ✓ Task Quality
- ✓ Task
- ✓ Requirement

Defect is probably the work item which is used most in RQM (like task in RTC and requirement in RRC), which obviously describes something which does not work as it should with program.

Task review is a work item which is used to assign a user the task of reviewing test plan or a test case to oversee that everything is correct.

Task quality is assigned to a user which is responsible to create or finish a test plan, test case, or a test script. In such a scenario a task quality work item should be assigned.

If there is a task which falls outside of category of the two latter work items, then it should be created as a task work item, these are not as usual in Quality Manager as in Team Concert.

The requirement work item represents the same as in Requirements Composer, a criteria which the finished (and tested) product should meet.

In addition it is important to mention that defect items often are associated with the requirement work item which the program failed to meet the expectations for, so that it is easier to locate for someone which shall try and correct it.

## 4.7 Lab Management in Rational Quality Manager

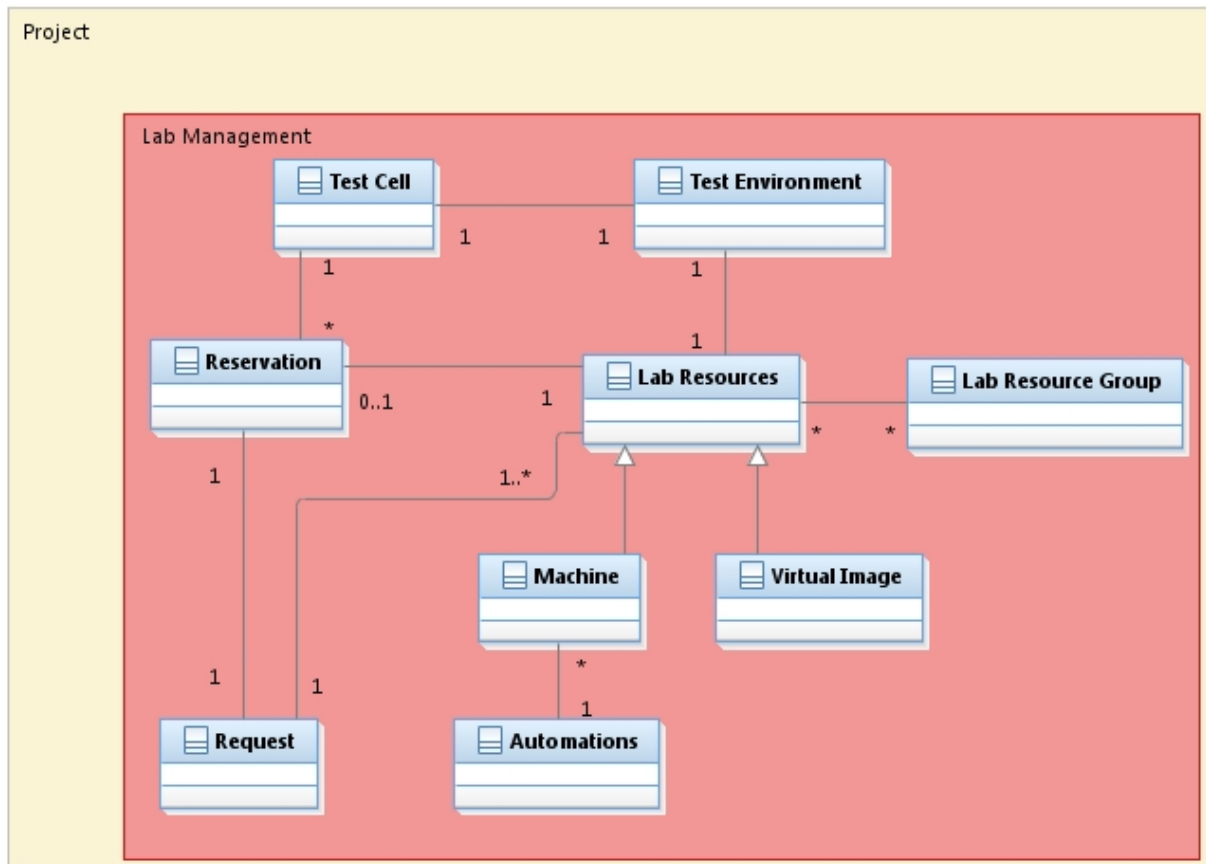


Figure 29: Model of the objects within the Lab Management category in Rational Quality Manager

The Lab Management category of Quality Manager is a tool for managing the lab which is used to testing for a team in Quality Manager.

The main object in the lab management category is lab resources, which is more or less connected with everything within this part of Quality Manager. There is two different types of lab resources:

- Machines
- Virtual Image

Machine is a resource which is a machine in the lab, this lab can either be local, or it can be remote. The local machine which the server is hosted on, will be discovered automatically of program, and the details it can get hold of filled in. To other machines, a machine resource will need to be created. There is a lot of information which can be provided, what type of hardware is in the machine, IP address, what type of software is installed, and a lot more. There is also a list of the recent events which the machine participates in. If the machine is remote, there must also be set up some sort of remote connection. (HTTP, FTP or telnet).

These machines can be part of automations, which requires additional tools (not part of this thesis), to run automated scripts. An example of such a program could be IBM Rational Build Forge.

Virtual Image<sup>23</sup> is the other type of resource which can be created, this is data for a file which can

be used with a physical machine to create a virtual machine. A virtual machine can be used to create a different testing scenery for a program being tested than the actual physical machine can offer.

The lab resource group object works similar to the collection artifact in RRC, it is used to group together resources which has something in common, for example being reserved for the same event for testing.

Test environment is rather self explaining, it is an environment for testing which specifies the attributes for a specific lab resource. These attributes can for example be CPU, operating system, type and amount of memory, and also if there is software which is relevant to the test environment, this can also be added.

A test cell is closely connected to the test environment, as this a test cell is a group of lab resources from your lab which together represents the test environment. The test cell is what is actually used when testing, but the test cell is based upon the specifications which is made in the test environment.

Requests and reservations is the two last objects available in the Lab Management category, and is pretty similar. A reservation can be made for lab resources (or if using a test cell, multiple lab resources), a time frame for the reservation must be made, and it of course have to be free at the dates which it is being reserved for.

A request on the other hand is when a member of the project needs some help to configure one or more resources, and requests this from the lab manager.

## 4.8 Planning in Rational Quality Manager

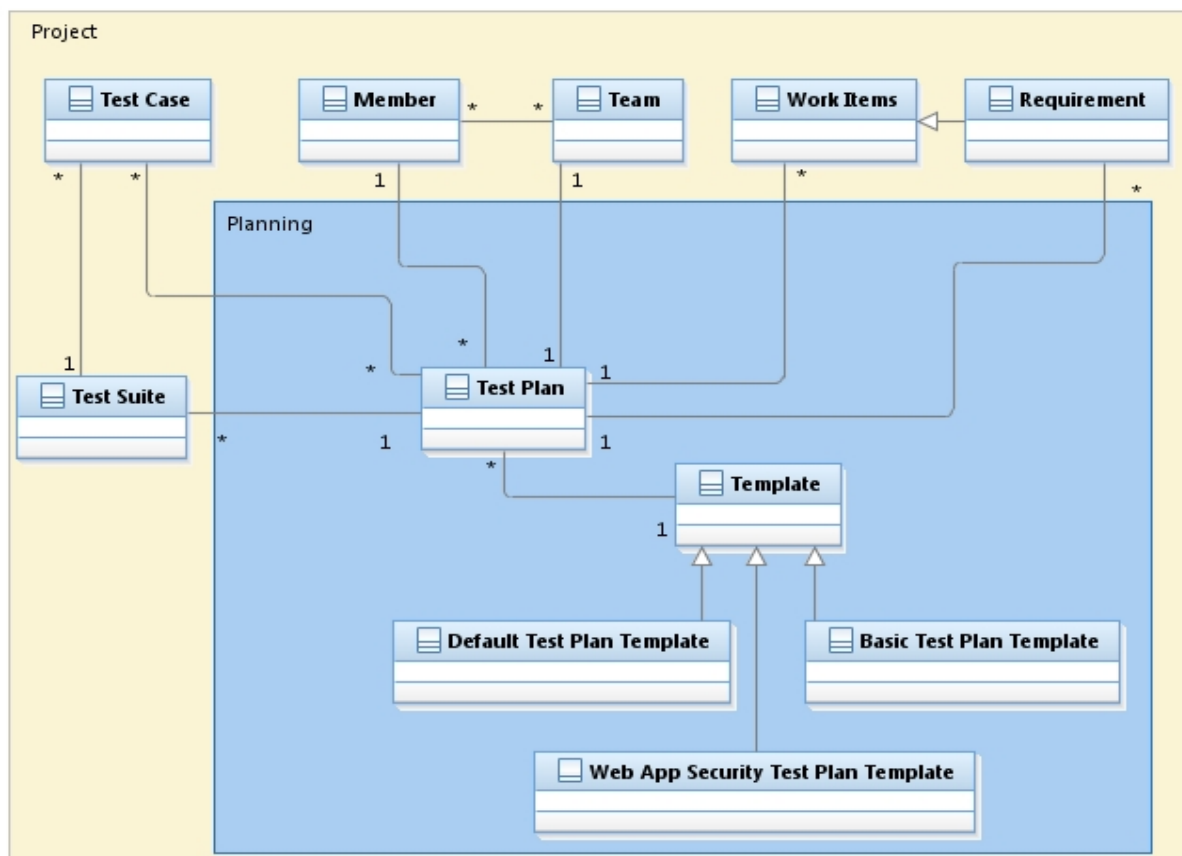


Figure 30: Model of the objects related to the Planning category in Rational Quality Manager.

The planning part of Rational Quality Manager has one object in focus, and that object is probably the most important object in the whole program, the test plan.

When looking at the figure, it is probably most natural to look at the other object which is inside the planning box, the template. A test plan must be created from the basis of the template. In addition there is quite a few associations to other objects which is within the project. A member of the project will be the creator of the test plan, and thus the association between member and test plan. A test plan can also be associated with test suites and also must be associated with at least some kind of test case. A test plan can also be assigned to a specific team, if that should be desired. Work items can be created and associated with the test plan, these work items is used to specify which sections that need work to do the test case completely ready. The last association is to requirements, which specifies the criteria which the program being has to be able to perform.

The test plan is the tool which is used to plan all details which is important in a test scenario, within a test plan there is various *sections*. These sections is used to give the user a better overview of what is available in the test plan. If it was not organized in such a way, it would be very difficult to track all the different information. In all there is a total of nineteen available section types.

- ✓ Summary
- ✓ Business Objectives
- ✓ Test Objectives
- ✓ Formal Review
- ✓ Requirements
- ✓ Requirement Collection Links
- ✓ Risk Assessment
- ✓ Test Schedules
- ✓ Test Estimation
- ✓ Test Environments
- ✓ Test Team
- ✓ Quality Objectives
- ✓ Entry Criteria
- ✓ Exit Criteria
- ✓ Test Cases
- ✓ Resources
- ✓ Attachments
- ✓ Child Test Plans
- ✓ Application Security

A test plan is created based upon a template. There is originally three available types of template, but it is also possible to create your own templates, if none of these fit your exact criteria. The three predefined templates is:

- ✓ Default Test Plan Template
- ✓ Web App Security Test Plan Template
- ✓ Basic Test Plan Template

What the user defines is which of the nineteen types of sections which shall be included in the template. The two only mandatory sections which have to be included is "Summary" and "Test



Cases".

All of these nineteen sections will not be discussed in detail here, instead the most important ones will be mentioned.

The basic test plan template only includes five (whereas the two other templates uses sixteen and seventeen sections respectively), which is the most important ones. These five are; summary, requirements, test cases, test schedules and attachments. Summary is the overview of the test case, what is the product in question, which release, and what type of testing is gonna be performed. Requirements is the associated requirements which the program has to fulfill. Test cases is the test cases which is associated with this particular test plan. Test schedules is the specifies the start and end dates for when the testing will take place. The attachments section is for uploading external files, should there be anything which needs to be added from the outside of Rational Quality Manager.

## 4.9 Construction in Rational Quality Manager

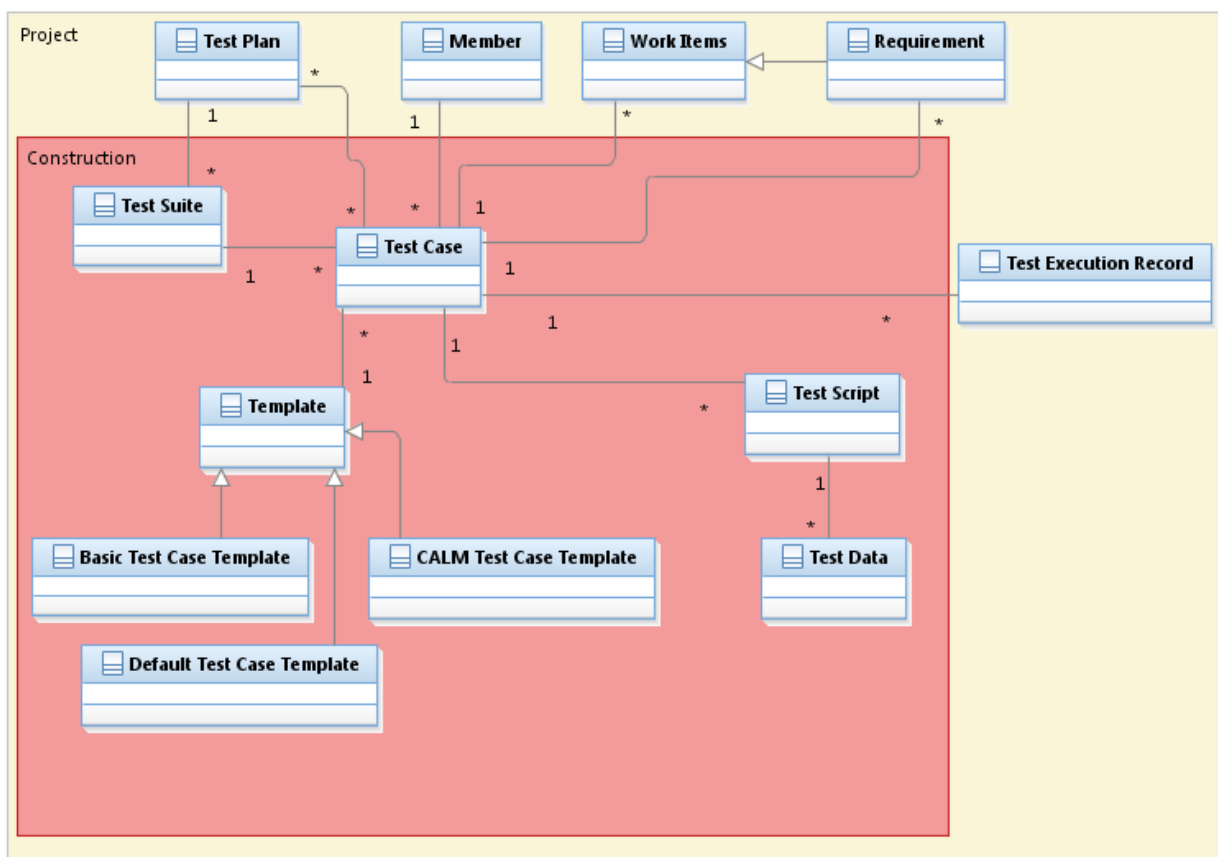


Figure 31: Model of the objects within the Construction category in Rational Quality Manager.

In the Construction category in Quality Manager the most central object is the test case. The test case, like a test plan must be created based upon a template. Other than this, there is a few other associations the two have in common, like that the test case is associated with the member creating it, and that work items can be created for each section of the test case if something is not finished. (There is sections in test cases also). Requirements can also be associated with a test case.

A test case is tested with test scripts, these can either be manual or automated, like mentioned earlier, if automated test script is needed, you will need external tools as this is not possible with RQM by itself. The test scripts can be associated with some sort of test data, but that is not

mandatory.

A test case can be associated with multiple plans, and a test plan can have multiple test cases. If there is two or more test cases which shall use the same test script, they can be grouped together in a test suite, so that more than one test case can be tested at once.

When a test case has been tested with a test case, a test execution record will be created, this type of object will be further discussed in the execution category.

As mentioned, the test case uses a template to create the test cases, and like with the test plans, there are three predefined templates for test cases.

- ✓ Basic Test Case Template
- ✓ CALM Test Case Template
- ✓ Default Test Case Template

When creating a template there is a total of fourteen sections to add to the template:

- ✓ Summary
- ✓ Test Case Design
- ✓ Formal Review
- ✓ Plan Items
- ✓ Requirements
- ✓ Requirement Links
- ✓ Risk Assessment
- ✓ Precondition
- ✓ Post-Condition
- ✓ Expected Results
- ✓ Test Scripts
- ✓ Test Execution Records
- ✓ Attachments
- ✓ Notes

Summary and test scripts are the only sections which are absolutely mandatory to include in all templates for test cases. Using the basic test case template as an example (as the basic test plan was used as an example), the basic test case includes four sections; summary, test scripts, requirements and attachments. The summary says what type of test case it is, what is the theme, and how important it is, weighted in points. Test scripts are the test scripts which the test case is meant to be tested with. Requirements and attachments are the same as in the test plan, requirements are the requirement which this specific part has to fulfill to be considered successful, and the attachments are for adding documents which are not available within RQM.

Test scripts are what the testing is based upon, in this report only manual testing has been tried, since using external programs to create automated test scripts was not part of the thesis.

A manual test script consists of steps, these steps each consist of a description of the step, and also the expected result. There is no limit on how many steps which may be included.

In test scripts it is also possible to add test data, if that is necessary. Test data must be created outside of RQM, and must be a CSV file (Comma Separated Value). This can be useful if it is necessary to provide some realistic data.

## 4.10 Execution in Rational Quality Manager

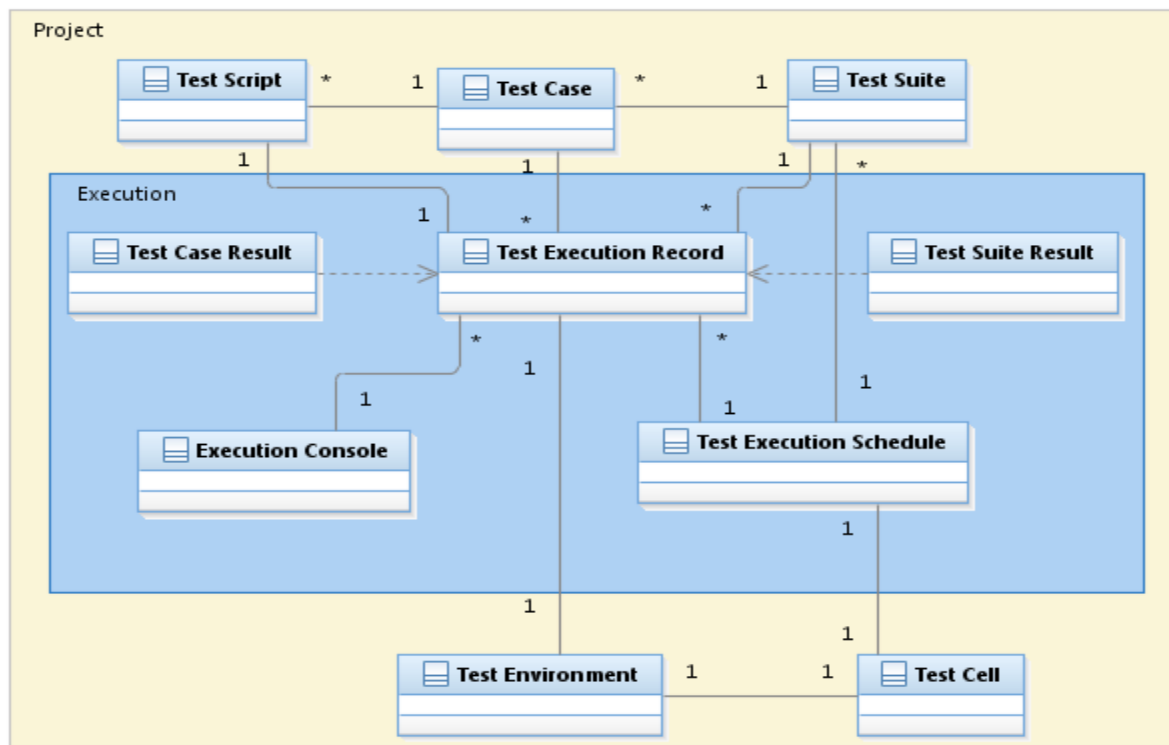


Figure 32: Model of the objects which are related to the Execution category in Rational Quality Manager.

The Execution category has one object which is probably a bit more important than the rest, the test execution record. This is all the actual testings which has been done, one execution of a test case is a test execution record. One test case or test suite can be associated with several different test execution records, since a testing of course can happen more than once.

The test execution record is also the only item which is associated with the execution console. The execution console lists all executions which is currently in progress at this exact time.

A test execution record is also associated with a test environment and a test script, since those are necessary to execute a test.

The test environment says what type of environment the testing is being done in, while the test script describes the steps which needs to be completed for a test execution record to be successful.

When the test execution record from a test case or a test suite (which as mentioned is two or more test cases, tested at once.) is completed, there is generated either a test case result, or a test suite result, which in the figure has been set as dependent on the text execution record.

The results will be filled with details of what went wrong or right, and a total evaluation of the result will be given, this result have a total of nine different results<sup>24</sup>:

- ✓ Passed, everything went as expected.
- ✓ Fail, the testing did not meet the expected standard of quality.
- ✓ Incomplete, for some reason the test is not completed at this point.
- ✓ Error, problems with completing the testing itself.
- ✓ Deferred, postponed because it is not possible to test in this exact release, but is planned to be tested at a later stage.
- ✓ PermFailed, what was supposed to be tested is not supported by the product anymore.

- ✓ Blocked, the execution cannot be performed because the preconditions are not met.
- ✓ Partially Blocked, some of the preconditions are not met.
- ✓ Inconclusive, the actual result is unknown at this point, requires further investigation to come up with the proper result.

The test execution schedule is associated with three other objects, test suite, test execution record and the test cell which is a group lab resources which fits the requirements for this schedule (Correct test environment, and lab resources which is available at the scheduled time). When creating a test execution schedule it consists of different steps, a step is either a test suite, or a test execution record. The schedule will be performed in the sequence which the steps are set up. For example, step 1; test suite object, step 2; test execution record object; step 3; test suite object. The testing will be performed in this exact sequence.

#### 4.11 Reporting in Rational Quality Manager

The report category in RQM is a strange one, well, maybe not strange, but it is easy to be misinterpreted. This is a function which allows the user to search through all data which is within the project by using parameters. It is a querying engine, which uses different parameters to find the information which the user desires, and the end result here, is what is called the report. There is no figure for this, as it is not really much to draw from it, but it is still an important part of RQM. In larger scale projects, it will be essential to navigate to the documents, work items, execution results or whatever type of document you are looking for.

The reports part is sliced up into eight different main themes:

- ✓ Defects
- ✓ Execution
- ✓ Lab Manager
- ✓ Requirements
- ✓ Scorecard
- ✓ Summary
- ✓ System
- ✓ Test Case

Underneath all of these themes there are also reports (36 in total), each sub item has a special set of parameters, so that it is easy for the user to conduct the correct query.

To take a simple example, under Test Case, one of the reports is "Test Case Review". In this "report" there is two parameters to create the query with. The first one is to choose which test plans to include. The other parameter is the test case state. In each parameter it is possible to include as many as the user want, for example, it is possible to choose only one test plan (or zero, but that does not make sense) or include all of the available test plans in the query. This makes the queries very flexible, and easy to find exactly what you are looking for.

# 5 Collaboration with the three tools

## 5.1 Configuration of the three servers

Looking at each of the three architectural figures earlier (page 13, page 33 and page 39), there is always drawn a dependency arrow to each of the two other servers, and the reason for that is what is the possibility to collaborate between the tools.

In order for this to be possible each of the three servers have to be properly configured, so that connections between each server can be established. <sup>25</sup>

These steps needs to be carefully followed, if they are not, the servers will not be able to communicate with one another, and thus there is not an option to use these tools together.

The first step is to set the public URI on all of the servers. The public URI will be the domain name which is used to log into the web interface. In this case, since everything is on a local machine, it is *https://localhost:9443/jazz* for RTC, change to port 9444 for RRC and port 9445 for RQM. These are running at the same address by default (the address used for RTC, thus changing the port name), so a little tweaking in the XML-files for two of the servers has to be done. These changes are made under the advanced properties for the server, in the administrative area of the web interface. This is of course done because all three cannot run on the exact same address at same time.

There is another step in between here, if these servers has been set up for C/ALM earlier, all these connections has to be deleted before configuring new ones, this is not a problem here, as this has not been done earlier.

So the next step after this is to enable the actual communication between servers, as the earlier step only made it possible. As the activating of the public URI, this communication is also activated in the server configuration available in the administrative area of the web interface.

When adding other servers as friends there are three things which is needed, a title, the is just the name of your choosing. The second requirement is to have the URL to the rootservices for the server which this shall connect with. In this example the rootservices address for RTC would be; *https://localhost:9443/jazz/rootservices*.

The last step is an OAuth authentication. <sup>26 27</sup> OAuth (short for Open Authentication) is an open standard for authorization which can be used by one party (in this case one server) to grant access to its private files to a secondary party (in this case, one of the other servers), in a provider-consumer relationship.

The OAuth used on the servers are pretty simple, as it grants the administrator to enter an OAuth secret of his choice, which is then associated with a an auto generated provisional key. After this is done, an administrator on the other server which first server requested access to, needs to authorize it.

When all this is finished, these servers are friends, and it is possible to start collaborating between these tools. To have access between all the three servers, this procedure naturally has to be repeated six times. All servers know about each other, both ways since this is a provider consumer relationship.

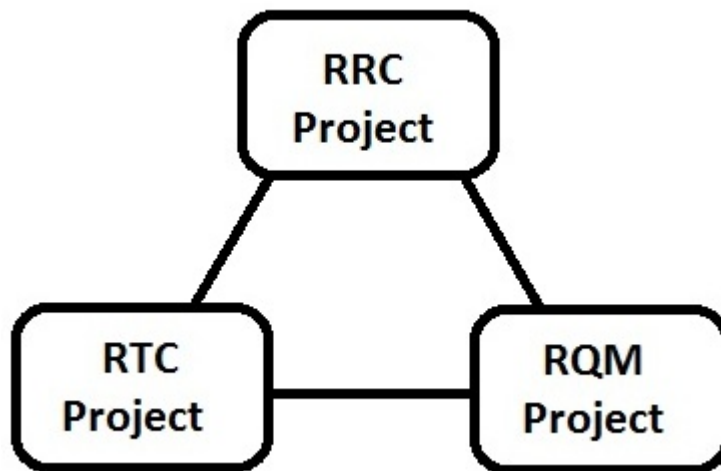
## 5.2 Linking projects and objects within the different projects

When all linking between each of the servers is finished, all is set for linking the projects which you would want to be collaborating. It is very important to specify that even though the servers are friends, it is not like you can connect to everything from another server. To gain access, first a link between two projects in two different tools have to be created. These links are two way links, so creating a link at one server, will create a link at the corresponding server as well. (In that way it is

different from establishing the communication between each server, where the connection had to be created separately at each server.)

To link a project to another project on another server, the project area management part of the administrative area is where you have to be.

There it is possible to add links to another project, when doing so, you will be able to choose from the servers which has been added to the friends list of this server. Choosing a server opens a new window to log into the other server, and after that the user will be able to see the projects on the other server which are available to link up to.



*Figure 33: Drawing showing one project from each tool linked together.*

When projects from all tools has been linked together, the connections should be like this, under all projects there should be two links, one to projects on each of the two other servers.

There are also some different descriptions when two projects have been linked together, depending on which tool they are from

- ✓ RQM project *validates* RRC project
- ✓ RQM project *tests* RTC project
- ✓ RTC project *implements* RRC project

I would assume it is done like this to describe what type of relationship there is between the two tools. For example, in RQM, source code from RTC is tested, while RTC tries to implement all the requirements which is generated in RRC.

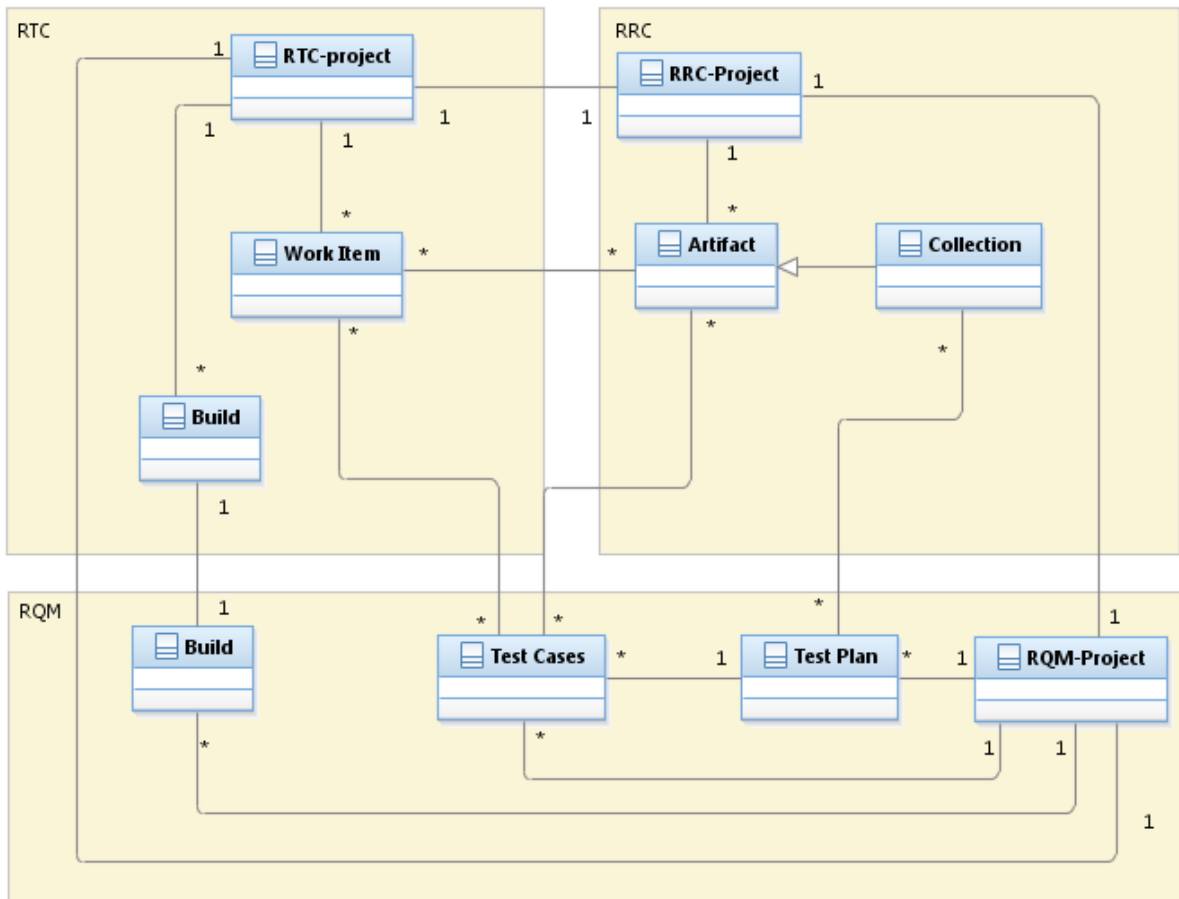


Figure 34: Model showing what from each tool which are associated with each other.

When all this is set up properly, it is time to talk about the reason for doing all this. It all comes down to flow of information between all the tools, that's what it is about. In each of the three tools there are either artifacts, work items or other objects. All these items can be linked other items from another project within another tool. Why is this so great one might ask, it is great because it is very easy to link objects which are related to each other. By doing so make it easily accessible for those who need that exact information, while working in another project and tool.

The best way to describe such a relationship is by an example, so using the web shop example might be fitting again. In RRC a requirement is made for how the shopping cart in the web shop should. This requirement will then be linked up to a task in RTC, a task about developing the shopping cart of the web shop. These two objects are clearly related to each other since the both concerns the same thing. In addition to this, there will always be a test case which is supposed to test if the shopping cart actually do work the way it is supposed to do. Because of this, both the requirements for the shopping cart, as well as the task have to be linked with this new test case which shall be testing the quality of the work.

If this shopping cart does not live up to the requirements which has been set, or has some kind of bug which needs to be fixed so that the testing fails, it is possible for the tester to create a new defect in RTC, and assign it to the user in RTC which was responsible for the original task. If this is a continuing problem which has been, and a defect on this subject already exists, it is possible to just link new results directly to an existing defect instead.

The above example is a good example of the flow information which happens when integrating the three tools together for C/ALM.

There are however some other advantages also. When there has been created a whole lot of requirements within RRC, there is a possibility to create a collection item. A collection of requirements can be used to auto generate work items within RTC, instead of having to manually create all work items one by one.

A similar possibility is found within RQM, where test cases can be created from the same requirements within a collection. To do so, first a test plan has to be linked with a requirement collection, and then create the test cases from the linked collection.

There is a thing which is a little bit odd though, when linking something from RRC to RQM, it is always mentioned as requirement linking. The fact is of course that all types of artifacts from RRC can be linked, not only requirements. But of course, requirements will probably be the most linked artifact type from RRC, just like the task work item will be the most linked artifact from RTC and defect from RQM.

The last thing which will be mentioned here, is that it shall also be possible to connect RQM to RTC and integrate builds into RQM from RTC. This was tried but did not succeed in this project, and with limited time at that point it was decided not to do anything further with it. It is however a very important thing to mention, because it is an important foundation for C/ALM to work with these three tools. If this project had a practical part, where the assignment was to create something with use of these tools, it would be very important to be able to this. (Such an assignment will be discussed in the "Future Work" section).

## 6 Conclusion & Future Work

### 6.1 Conclusion

The result of this assignment is an analysis which covers the essential parts to use three of IBM's tools for Collaborative Application Lifecycle Management. Each tool covers a different phase in the development. All three tools can be used individually, but using them together gives numerous advantages.

This has been an interesting project to work with, because analyzing these tools shows a lot about the process of creating an application, web site or some other feature.

IBM's tools for Collaborative Application Lifecycle Management might not be the most user friendly products if someone without any knowledge about this process tries them. They are however very deep tools, with a lot of functionalities, which probably is the most important thing, since developers will usually have dealt with a similar tool earlier.

Rational Requirements Composer is probably the easiest tool to start with, as this is probably the least deep in terms of features. The client which is used gives a nice overview and is very easy to use.

On the other hand, Quality Manager is probably the tool with most features of the three, and might be a little bit overwhelming when trying it out. Team Concert is the type of tool i am most familiar with, since it is the development tool of the three. It is a great tool when working in teams to write source code. Integrating it with Rational Software Architect also gives it a lot of possibilities in the modeling category.

### 6.2 Future Work

There are definitely possibilities to do some further work within this type of thesis. Considering that especially Rational Quality Manager to an extent needs an actual project to see the full capabilities of the tool, that is one possibility. To test an actual program would be a better way to really grasp all the potential. Such a work would probably need more than one person to complete though, as



learning the programs will be a lot of work in it self, and then also using it to create a program would require much work. I would think two or three people might be the needed amount of people required for such a task.

Another possibility is to take a similar approach like this thesis did, there are a lot of other rivaling companies which got their own tools for Application Lifecycle Management. I know Microsoft has a team system for Visual Studio. Other large companies are Borland and Hewlett Packard.

A master thesis could be to do a similar work like this one, analyzing the opportunities which is within the system.

Another approach could be to use this as a basis, and compare the solution provided with IBM with another company. A problem with such a masterwork would probably be that this version of the tools from IBM might be outdated, and it is impossible to say how much the changes which IBM are making would affect the tools.

## 7 Sources & References

1 <https://jazz.net/downloads/rational-team-concert/>  
2 <https://jazz.net/downloads/rational-quality-manager/>  
3 <https://jazz.net/downloads/rational-requirements-composer/>  
4 [http://en.wikipedia.org/wiki/IBM\\_Rational\\_Software\\_Architect](http://en.wikipedia.org/wiki/IBM_Rational_Software_Architect)  
5 <https://jazz.net/about/about-jazz-vision.jsp>  
6 [http://en.wikipedia.org/wiki/Requirements\\_analysis](http://en.wikipedia.org/wiki/Requirements_analysis)  
7 [http://en.wikipedia.org/wiki/Software\\_release#Release\\_candidate](http://en.wikipedia.org/wiki/Software_release#Release_candidate)  
8 [https://jazz.net/downloads/rational-team-concert/releases/2.0.0.2iFix5?p=install.docs/install\\_express\\_c#dqx1t\\_server\\_installation\\_setup\\_wizard](https://jazz.net/downloads/rational-team-concert/releases/2.0.0.2iFix5?p=install.docs/install_express_c#dqx1t_server_installation_setup_wizard)  
9 [http://publib.boulder.ibm.com/infocenter/rtc/v2r0m0/index.jsp?topic=/com.ibm.team.concert.tutorial.doc/topics/tut\\_rtc\\_abstract.html](http://publib.boulder.ibm.com/infocenter/rtc/v2r0m0/index.jsp?topic=/com.ibm.team.concert.tutorial.doc/topics/tut_rtc_abstract.html)  
10 <https://jazz.net/projects/rational-team-concert/>  
11 Description source is from RTC-server, from where the licenses are applied to users. (See Appendix, picture 6 for an example of this).  
12 [http://publib.boulder.ibm.com/infocenter/rtc/v2r0m0/index.jsp?topic=/com.ibm.team.workitem.doc/topics/c\\_work\\_items.html](http://publib.boulder.ibm.com/infocenter/rtc/v2r0m0/index.jsp?topic=/com.ibm.team.workitem.doc/topics/c_work_items.html)  
13 <http://agilesoftwaredevelopment.com/blog/artem/product-backlog>  
14 <http://www.selectbs.com/ad/pt/process-maturity/what-is-scrum-development>  
15 [http://publib.boulder.ibm.com/infocenter/rpcmpose/v2r0/topic/com.ibm.rational.rrc.help.doc/topics/t\\_create\\_review\\_reqs.html](http://publib.boulder.ibm.com/infocenter/rpcmpose/v2r0/topic/com.ibm.rational.rrc.help.doc/topics/t_create_review_reqs.html)  
16 [http://publib.boulder.ibm.com/infocenter/rpcmpose/v2r0/index.jsp?topic=/com.ibm.rational.rrc.help.doc/topics/c\\_process.html](http://publib.boulder.ibm.com/infocenter/rpcmpose/v2r0/index.jsp?topic=/com.ibm.rational.rrc.help.doc/topics/c_process.html)  
17 Description source is from RRC-server, from where the licenses are applied to users. (See Appendix, picture 6 for an example of this).  
18 <http://publib.boulder.ibm.com/infocenter/rpcmpose/v2r0/index.jsp?nav=/6>  
19 [http://en.wikipedia.org/wiki/Telelogic\\_DOORS](http://en.wikipedia.org/wiki/Telelogic_DOORS)  
20 [http://publib.boulder.ibm.com/infocenter/rpcmpose/v2r0/topic/com.ibm.rational.rrc.help.doc/topics/r\\_permissions\\_roles.html](http://publib.boulder.ibm.com/infocenter/rpcmpose/v2r0/topic/com.ibm.rational.rrc.help.doc/topics/r_permissions_roles.html)  
21 Description source is from RQM-server, from where the licenses are applied to users. (See Appendix, picture 6 for an example of this).  
22 [http://publib.boulder.ibm.com/infocenter/rqmhelp/v2r0/index.jsp?topic=/com.ibm.rational.test.qm.doc/topics/c\\_understand\\_workitems.html](http://publib.boulder.ibm.com/infocenter/rqmhelp/v2r0/index.jsp?topic=/com.ibm.rational.test.qm.doc/topics/c_understand_workitems.html)  
23 [http://publib.boulder.ibm.com/infocenter/rqmhelp/v2r0/index.jsp?topic=/com.ibm.rational.test.lm.doc/topics/t\\_crevirtualimages.html](http://publib.boulder.ibm.com/infocenter/rqmhelp/v2r0/index.jsp?topic=/com.ibm.rational.test.lm.doc/topics/t_crevirtualimages.html)  
24 [http://publib.boulder.ibm.com/infocenter/rqmhelp/v2r0/index.jsp?topic=/com.ibm.rational.test.qm.doc/topics/c\\_analyze\\_exec\\_results.html](http://publib.boulder.ibm.com/infocenter/rqmhelp/v2r0/index.jsp?topic=/com.ibm.rational.test.qm.doc/topics/c_analyze_exec_results.html)  
25 [http://publib.boulder.ibm.com/infocenter/rpcmpose/v2r0/index.jsp?topic=/com.ibm.rational.rrc.help.doc/topics/r\\_calm\\_cfg\\_roadmap.html](http://publib.boulder.ibm.com/infocenter/rpcmpose/v2r0/index.jsp?topic=/com.ibm.rational.rrc.help.doc/topics/r_calm_cfg_roadmap.html)  
26 <http://en.wikipedia.org/wiki/OAuth>  
27 <http://oauth.net/about/>

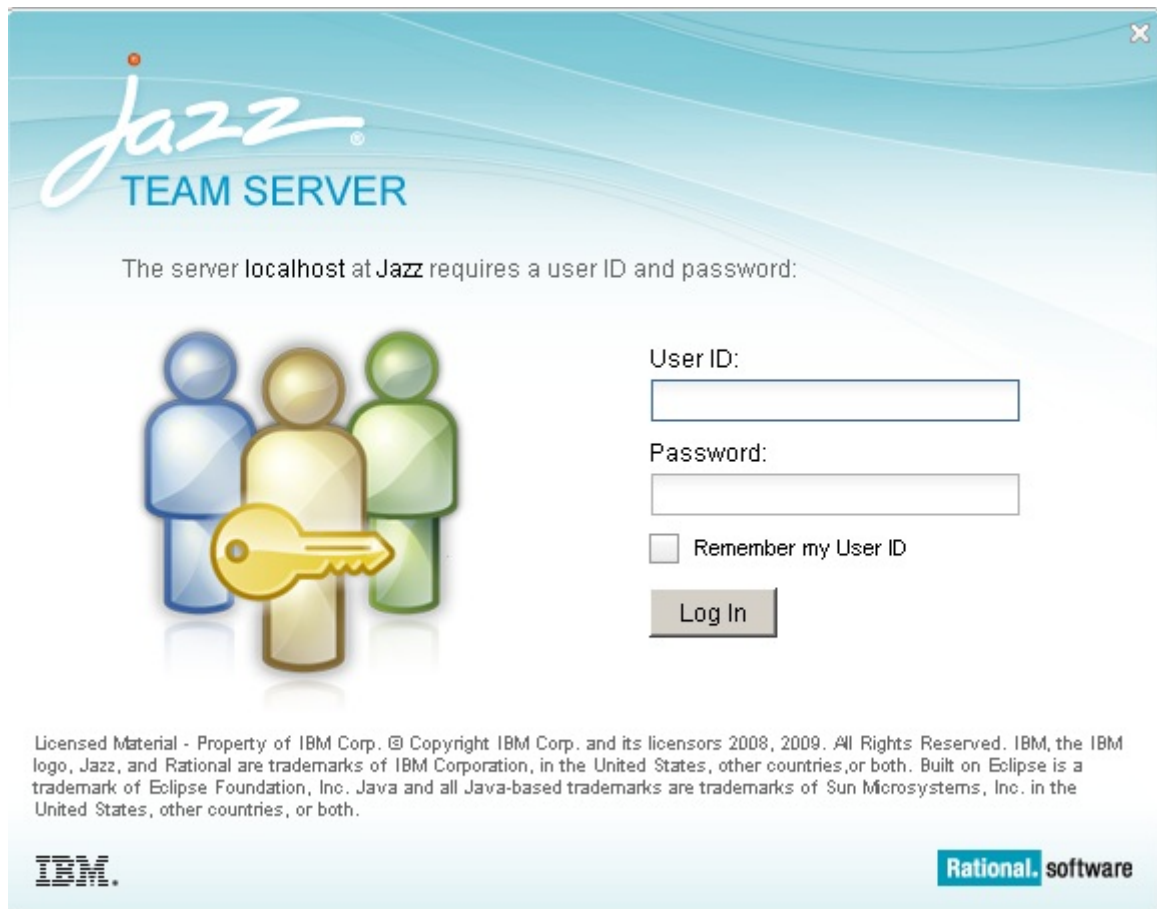
# 8 Enclosure

Summary of what is on the DVD:

- Copy of the report in pdf format.
- Folder with all figures.
- Slides used in the poster session in pdf format.

# 9 Appendix

8.1 Pictures from various situations in Rational Team Concert.



*Picture 1: Picture of loginscreen to the web interface, all login screens look like this, no matter which of the three tools you are trying to log into.*

Rational Jazz Team Server Ole Foss Johannesen | Log Out | ?

Project Areas **Server** User Management Project Area Management Process Template Management **Admin** | Select a Project Area

**Status**

- Status Summary
- Statistics
- Component Status
- Active Services

**Configuration**

- E-mail Settings
- Database Connection
- Feed Settings
- Themes
- License Key Management
- OAuth Consumer Management
- Cross-Server Communication
- Advanced Properties

## Status Summary

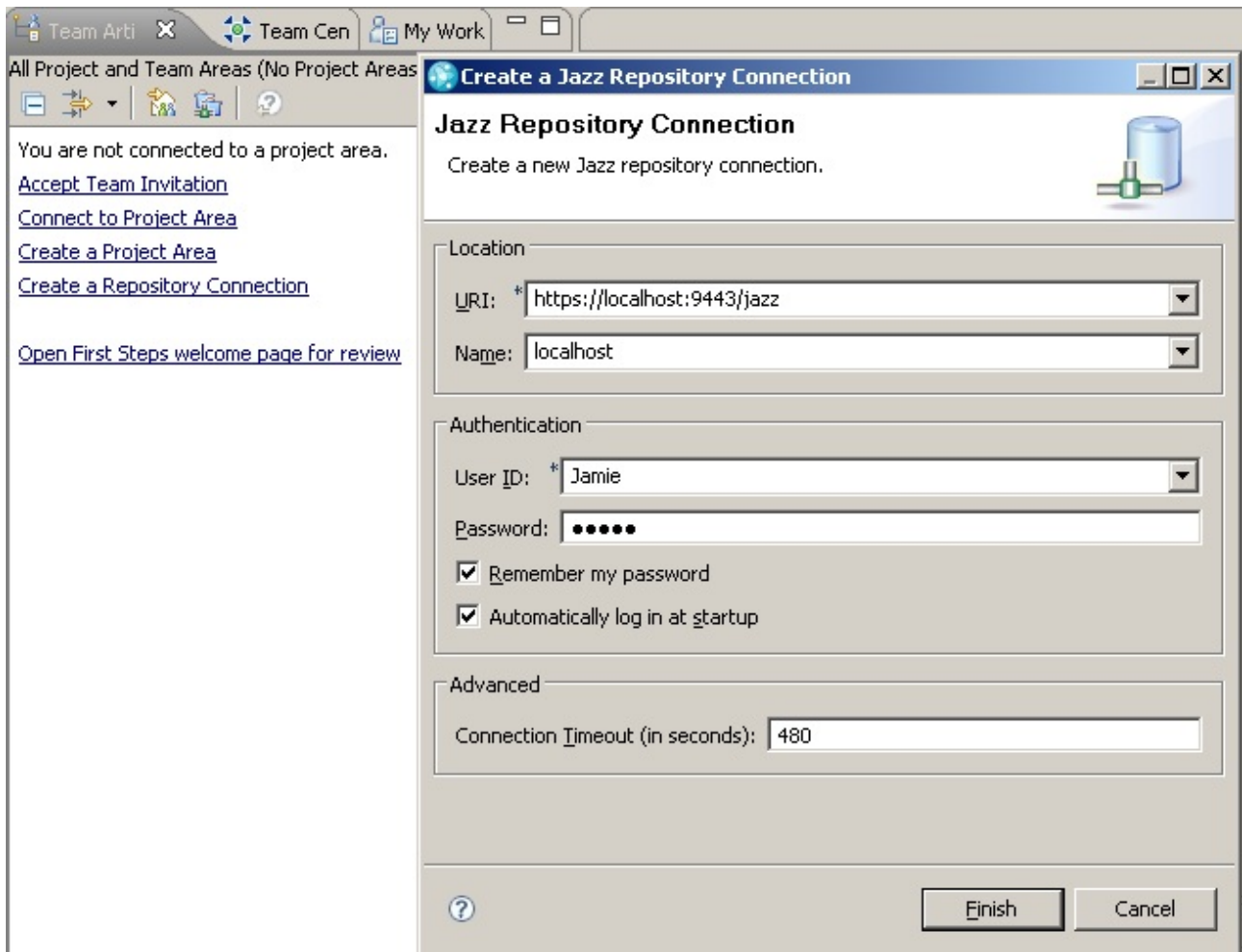
Server Status		Memory Usage	
Database Status	Connected	Maximum Memory Allocation	1500 MB
Server Uptime	4 hours, 12 minutes	Current Memory Allocation	197 MB
		Free Memory (of Current)	31 %

Installed Products	
Jazz Foundation - Jazz Team Server	1.0.0.2 iFix 5 (I20101021-2131)
Rational Team Concert	2.0.0.2 iFix 5 (I20101021-2131)

License Status	
Server License	Rational Jazz Team Server - Express-C
Server License Status	Permanent
Floating License Server	Not Configured
Floating License Client	Not Configured

Server VM	
Version	1.5.0
Vendor	IBM Corporation
Name	IBM J9 VM
Details	J2RE 1.5.0 IBM J9 2.3 Windows 7 x86-32 j9vmwi3223-20091104 (JIT enabled) J9VM - 20091103_45935_IHdSMr JIT - 20091016_1845_r8 GC - 20091026_AA

Picture 2: Screenshot of the status in the web interface of the team server for Rational Team Concert.



Picture 3: Creating a Repository Connection between the user and the server in Rational Team Concert.

**Create or Import Users**

### User Information

Enter the user information.


Name: \*

User ID (case sensitive): \*

A default password equal to the User ID will be set.

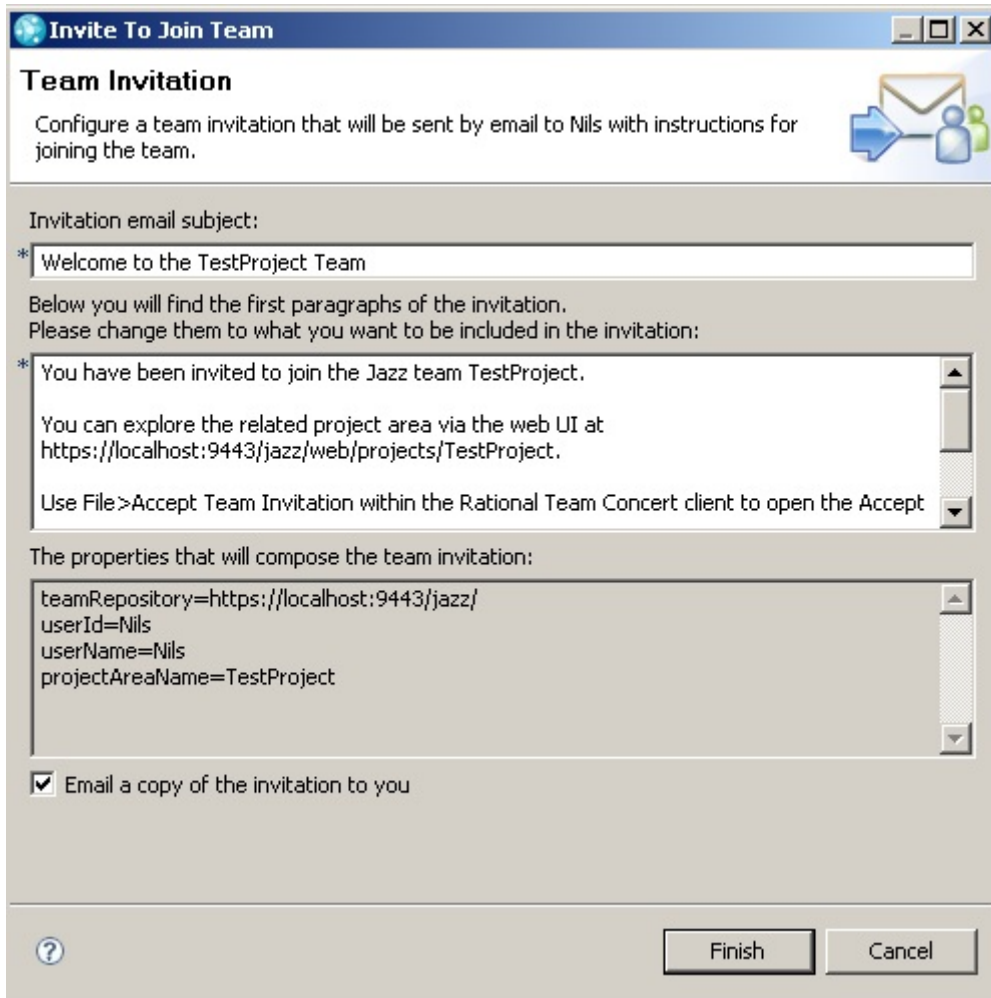
Email address: \*

Photo:

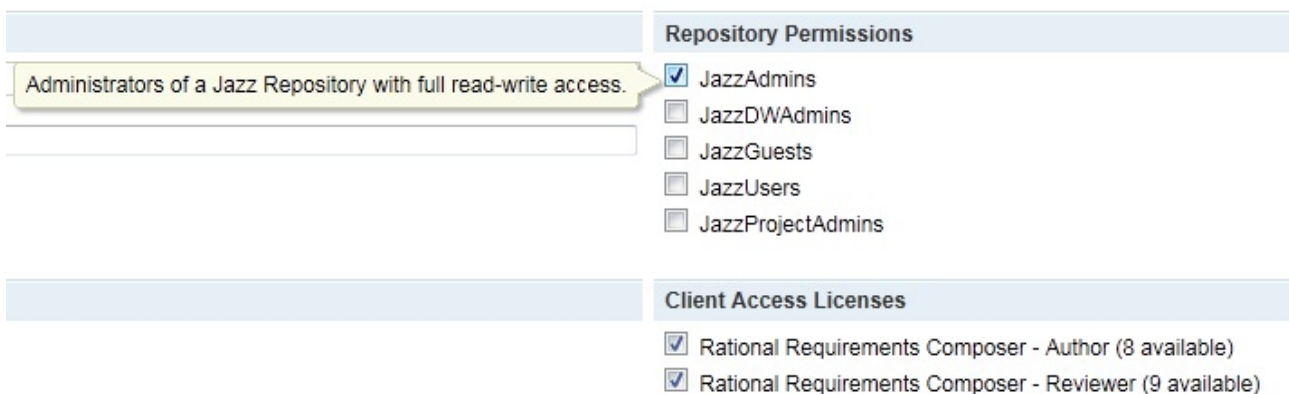


Will be scaled to 100 x 100 pixels

*Picture 4: Adding or Creating User to a project.*



Picture 5: Inviting User to Project in Rational Team Concert as an administrator.



Picture 6: Example of description for licenses and repository permissions on the servers.