On the downscaling of the Jazz platform

Experimenting the Jazz RTC platform in a teaching course

Angelo Gargantini[1] and Guido Salvaneschi[2] Patrizia Scandurra[1]

[1] Università degli studi di Bergamo,
[2] Politecnico di Milano

Abstract. We believe that students should use collaborative and project management tools similar to those they will encounter in their professional life. Moreover, we believe that Universities could promote and encourage the use of best practices supported by efficient tools by teaching students their use. IBM Rational Team Concert (RTC) is a good candidate to be used as industrial tool in University courses, since it supports many best practices in a complete collaborative development environment providing planning, source code management, work item management, and build management and it has been already successfully used in industries and software houses. However, we experienced that RTC is too rich in the features it offers and this makes its adoption very expensive in terms of students and teachers time and effort. For this reason we have studied and present in this paper a simplified use of RTC which consists in reducing the concepts (at ontological level) and the features were offered to the students (at practical level).

1 Introduction

In the first courses of computer science faculties students are widely exposed to the use of programming languages such as C/C++, Java or Python. In many cases the following courses give them a higher level approach through the study of design patterns, software architectures and processes and the other common topics of the software engineering field.

The abstractions taught in these courses are fundamental for the ability of students of analyzing problems and be able to project suitable solutions. While these are the conceptual tools that probably will make the difference in their professional career, many of the tools which are widely used in the industrial production of software are usually neglected in graduate courses. This is the case of tools that support project planning and monitoring, task assignment and team collaboration, and process adoption enforcement. Sometimes even configuration management systems are presented only in a theoretical lesson and their use is not subsequently imposed as a standard practice in software development in university projects.

On the other hand, we found that also many software companies, especially small enterprises like those operating near our University, are reluctant to adopt

best practices (like configuration management and structured collaborative environment) because they estimate that the cost of their adoption, even if supported by tools, will not compensate the gained efficiency. In this scenario, the Universities can promote the adoption of these best practices by teaching to the students the concepts at conceptual level and the use of tools at practical level, and in this way, lower the cost of their use.

We believe that the adoption of a collaborative development environment can be an effective way of teaching source code management in parallel with planning and process monitoring topics. One of the issues of introducing these type of tools in a course is that they are usually dimensioned for industrial-size applications. Being conceived for the software development related to an entire product line, they allow for the management of several teams, multiple time lines and different releases. This complexity constitutes a barrier for students who have to deal with a slow learning curve in order to reach a point in which they can effectively take advantage of the tool.

We have experimented the use of Jazz Rational Team Concert (RTC) [2, 6, 5] in a graduate project course which integrates some frontal lessons of advanced programming and software engineering topics. We spent a decisive effort in reducing the complexity of the tool by simplifying the ontological model of the entities to which the students are initially exposed. In our opinion this is essential to reduce the *barrier to entry* and make a tool like this suitable for an introductory course. While this process by no means made Jazz usage trivial, we succeeded in the goal of achieving a good familiarity of the students with the tool. This was mostly due to Jazz flexibility which allows for its adoption in huge projects but makes it usable with some care also for very small works.

This paper is organized as follows. Section 2 provides a brief overview of RTC and its features. Section 3 presents the conceptual model of the RTC features and their simplifications we adopted. Finally, Section 4 reports some related work, while Section 5 presents some conclusions and describes our future directions.

2 Elements of collaboration in the Jazz RTC environment

RTC is a collaborative development environment which provides support for project planning, source code and build management, task assignment, project health monitoring and reporting. The tool also integrates a process support.

RTC is part of a suite of IBM tools for team collaboration such as Rational Quality Manager which is a test management environment for test planning, workflow control, tracking and reporting, and Rational Requirements Composer, an environment for requirements elicitation and definition. RTC is released either as a server-side repository with a web interface, a build system engine and as a development environment integrated with the Eclipse IDE [1] or as a VisualStudio [3] plugin.

An RTC repository allows us to define different `Project Areas` to which users who collaborate to the same project subscribe. It is also possible to organize developers in separate teams assigning a `Team Area` to each of them. Tasks

(development activities or bug fixes, etc) are assigned to the users identifying `Work Items`. The overall project evolution is monitored through time lines which establish milestones and releases.

In addition to the support for project management, RTC offers all the traditional functionalities of a configuration management system, such as versioning, patch application, conflict resolution and code locks. It is possible to configure a separated server for long-running builds, fully integrated with the RTC environment.

As a general consideration, we observe that many aspects of RTC are thought for very large scale projects. For example, the support of many collaborating teams with different time lines for the same project is commonly used for `parallel development`, such as, for example, in activities where some developers are working on implementing the features of the next release of a product while other teams are involved in the fixing of bugs of the current product release. Another example of this large scale approach is given by the assignment of tasks: a developer retrieves its tasks by querying the system through a set of selection conditions. This schema applies well in an environment with a huge number of tasks and an analyst in charge of the project planning that assigns the different tasks to the developers and whose role is in principle different from that of the developers.

3 Downscaling the Jazz RTC environment

In this section, we expose the choices that we made for adopting Jazz as a collaborative framework for a graduate project course. Since the tool is very powerful – it exposes a huge set of entities with complex semantics and relations among them – we decided to make a set of key simplifications in order to make it usable by the students with a reduced initial effort. Our starting point was the development of a *metamodel*[3] of the RTC concepts to adopt and simplify.

We tried to simplify as much as possible relations among concepts, for example by reducing in some cases n-ary relations (with multiplicity $1..n$) to unary relations. These relations simplifications are clearly documented in the RTC metamodel (see the following subsections) by grey boxes indicating the new cardinality. We removed non necessary entities or made their presence transparent to the students (e.g. by adopting a single default category when a hierarchy of nested categories could be defined). In the case we really removed an entity or a relation, we added in the metamodel a grey cross on the class or association representing the concept; instead, when we wanted only to make an entity a bit transparent to students, the entities are shown in the diagrams with an empty

[3] In software engineering, *metamodeling* is the construction of a collection of concepts (things, terms, etc.) within a certain application domain. A model is an abstraction of phenomena in the real world; a metamodel is yet another abstraction, highlighting properties of the model itself. In practice, metamodeling implies the development of UML-like class diagrams to describe and analyze the relations between concepts.

cross on them. In some cases, a simplification is made by restricting the possible instances of an entity; this is only reported in the description text as well.

We categorize RTC concepts in three perspectives: *Work Organization*, for concepts related to the developers' work organization; *SW Artifacts Management*, for concepts related to the management activities of the SW artifacts being produced; and *Development Process*, for concepts pertaining to the development process. In particular, we adopted a simplified *Agile Model Driven Development* process [4] whose schedule organization was modeled inside Jazz. The concept of *p*rocess in Jazz is somehow complex, involving a huge set of general choices such as defined roles, access rights associated to each role, a default timeline or available work item types. To this purpose, we exploit the *Jazz Simple Team Process* which allows team members to perform any kind of modifications denying anything to outsiders and defines a very simple set of default values which best suites our needs.

3.1 Work Organization

Fig. 1 shows a fragment of the RTC concepts related to the Work Organization perspective. First of all, we decided to eliminate the concept of *team area* (see class `TeamArea`) which is used in Jazz for organizing a small group of developers who are in charge of a subset of the whole project. Since projects are quite small, we decided to remove this intermediate way of aggregating developers. So, in our case, each project is assigned to a *project area* and each team of students (users of the RTC platform) has a dedicated project area which is not accessible by other teams.

In the general case, *users* can be associated to an *area* with different roles. The common application of this feature is to separate developers who have access to code modifications, and users who are in charge of planning and monitoring the evolution of the project allowed to modify plans and task assignments. We decided to make a strong simplification adopting a unique possible *t*eam member role (as instance of the `Role` class) which allows for every action inside an area. For this reason, in the metamodel portion shown in Fig. 1 the multiplicity of the association ends terminating on the `Role` entity are reduced to one. This essentially removes the concept of "role" in our schema, because each user who is assigned to an area has the team member role inside that area.

Inside Jazz RTC, a *work item* is the way of keeping track of the tasks and issues that a team needs to address during the development cycle. Work items are associated to an area and keep a reference to the user who created them, while ownership indicates that a certain task is assigned to a specified user. Developer activities make the state of work items advance. To reflect this, work items have attributes such as *time estimation* in order to complete the task, a *severity* and *priority* evaluation, a *state* value such as *new* or *resolved*, and a *type*. We decided to keep the set of available work item types small, allowing only for the *Defect*, *Task* and *Enhancement* types.

Finally, in Jazz it is possible to define work item *categories* inside a project area. Categories can be nested and organized in a hierarchy. The aim of categories

is to assign a work item to one of them; developers who are interested to a certain category subscribe themselves to it in order to be notified when the state of an item belonging to the category changes. We tried to make the concept of work item category transparent to the students keeping a default root category to which all the work items belong to. Usually, students are free to organize items in categories and trigger the notification mechanism on their own.

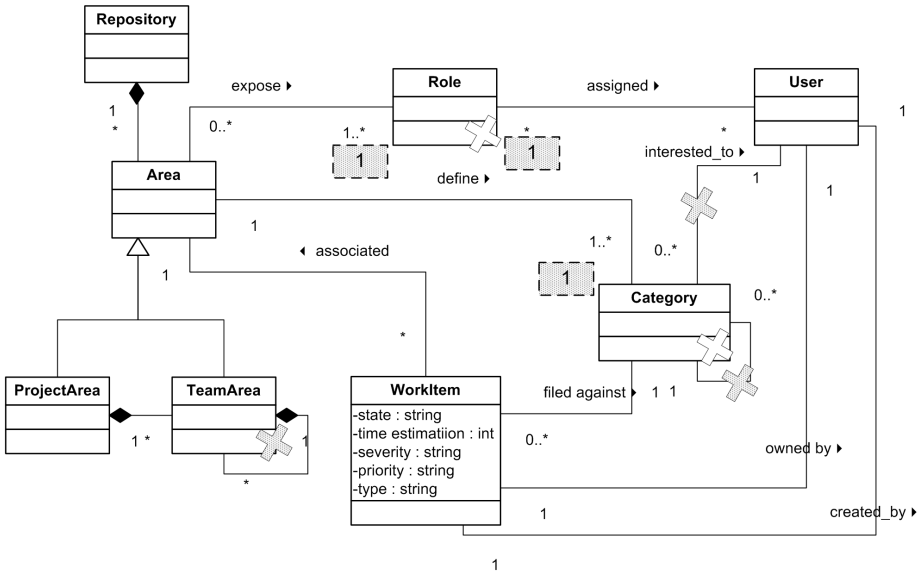

Fig. 1. RTC metamodel: Work Organization

3.2 SW Artifacts Management

Fig. 2 shows a fragment of the RTC concepts related to the SW Artifacts Management. *Repository workspaces* are used in Jazz as a remote copy of the developer's work. They can be used for backup and as a source for delivering a change set to a branch. A developer can own multiple repository workspaces inside the same repository. We limited each student to a single repository workspace, instead of the many ones that the platform could make available to him or her; this workspace is used both to backup the work and to deliver later a *change set* to a *stream*.

A single stream acts as a central repository for all the students participating to the same project. This is related to the fact that we have adopted a single *timeline* (see next subsection 3.3) for the project, so the unique repository of the area is like an instantaneous photograph of the current state of development of

the project. This is quite different with respect to the complete functionalities of Jazz RTC which allows us the adoption of many streams to contain the same *component* (read: SW artifact) at different states of development (i.e. in different versions).

It is common to divide a big project into several components, each of which exports a subset of the system functionalities and has an explicit interface and dependencies on the execution context. We decided to extremely simplify this by making each project constituted of only one component which contains all the functionalities that the project must implement. This choice basically removes the original concept of component from the platform, because each versioned file is assigned to the same component. We believe that this is a bit extreme choice, being this solution suitable only for very small projects; even if the use of components in RTC is discouraged because we believe that the presence of different components makes harder to understand the basics of code versioning at the beginning, students are encouraged to apply principles of component-based development to their software artifacts, but this will be not traceable in RTC. However, experts students can in the end also exploit the RTC component concept taking advantage of this feature on their own.

Changes to a component are represented in terms of *change sets*. While a developer modifies the code inside his or her local workspace, Jazz RTC keeps track of the differences between the local workspace and the (remote) repository workspace and adds these modifications to one ore more change sets. A change set is always associated to only one component. The developer can commit the change sets to his or her repository workspace, making it synchronized with the local workspace. With a successive delivery operation, it is possible to apply the change sets collected in a workspace repository to a stream, sharing the modifications with other users of the same stream. The history of the change sets applied to a component is stored inside a repository, so that the sequence of changes can be inspected and under certain conditions undo operations are also possible.

In the RTC platform it is possible to take a picture of the state of development of a component as "base point" for further use/development by creating a *baseline* in the component's change history. When a component is created inside a repository it is initialized with an "initial" baseline representing the empty component before the addition of any change set. We decided to remove the concept of baseline from our model, as we considered it non essential for our purposes.

Another concept we decided to remove is the concept of *snapshot* that is a repository object including exactly one baseline for each component in a repository workspace. This is useful for recreating a workspace configuration which is considered important. Snapshots and baselines can be delivered in the same way of change sets and in this sense can be seen as collection of change sets; this is the reason why in our diagram we have a relation between snapshots, baselines and change sets.

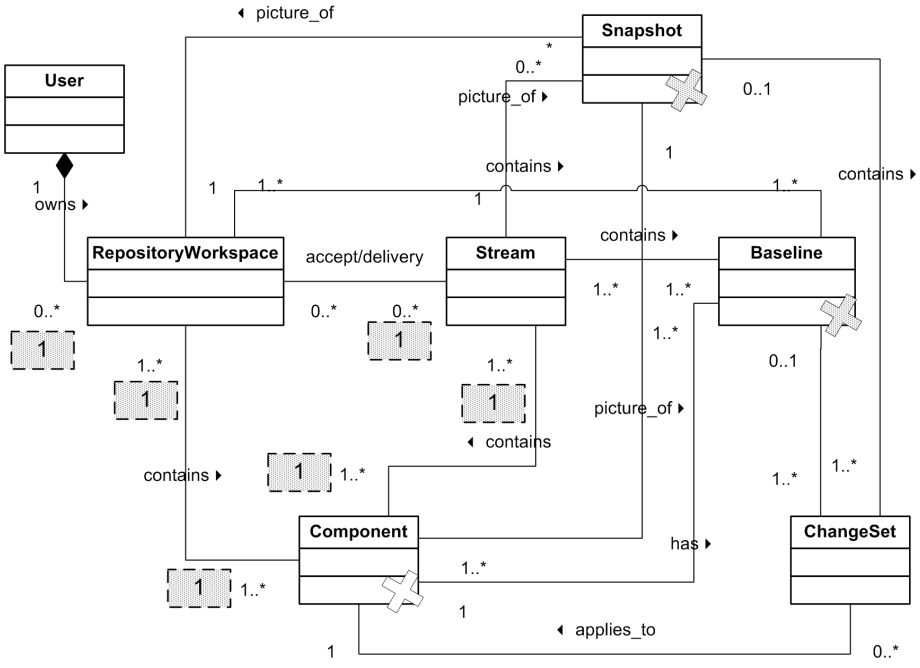

Fig. 2. RTC metamodel: SW Artifacts Management

3.3 Development process

As it can be natural in an advanced course, we expect from the students a decisive analysis effort before the implementation phase. Students are required to create modeling artifacts that describe their architectural design choices and even envision several alternatives evaluating the advantages of each solution. For this reason we believe that a process centered on the use of high-level models in the UML style, for example, is the best suited for our teaching purposes. However, the modeling effort should be functional to the comprehension of the solutions and its result should be immediately applied. These considerations together with the small size of the teams and the ease of interactions between students lead us to the choice of an "agile process" that easily allows for simple management and rapid prototyping.

The AMDD[4] process is the agile version of a model-driven development process. In a model-driven development process, like the OMG's Model Driven Architecture (MDA), extensive models are created and refined throughout the development to guide the implementation of the final source code (that is possibly generated, at least in part, in an automatic way from models). With AMDD, models are involved in an agile way by creating and using in particular phases only those modeling artifacts that are good enough to drive the development activity. AMDD is best suited for small and co-located teams, which is a common case of the students groups in an university project. We adopted a simplified version of the AMDD process also because we believe this can help in achieving our twofold goal: on one hand, we want to force the students at using a systematic approach to their work, on the other hand we want to provide them an agile methodology without excessive constraints.

Fig. 3 shows the lifecycle of the AMDD-like development process we adopted. Each box represents a development activity. Essentially, it comprises the typical AMDD activities of Envisioning, Iteration Modeling, Model Storming and Development, organized as follows. The envisioning (iteration 0) includes two main sub-activities: *Requirements Envisioning*, for collecting and representing initial functional requirements (usually by text and *use case* models), and *Architectural Envisioning*, to produce an initial SW architecture model (in terms of UML component/deployment diagrams) following the *component-based development* concepts and principles (components, provided/required interfaces, hierarchical assembly, reuse, etc.). Then, an arbitrary number of iterations can follow. Each of such next iterations is organized in three (self-explanatory) phases: *Modeling*, *Model Storming*, and *Development & Testing* (in parallel). The time indicated in each box represents the length of an average session. The basic idea is that perhaps you model for a few minutes then code for several hours.

In order to encode our AMDD process within the RTC tool, we considered the RTC concepts shown in Fig. 4. In the RTC environment, it is possible to create different *timelines* inside each area: in a big project, it is common to have different time schedules inside a project, such as the timeline for a stable release, and the timeline for a beta version of a next release. However, the goal of the students is usually to develop a stable version of their software for the deadline of

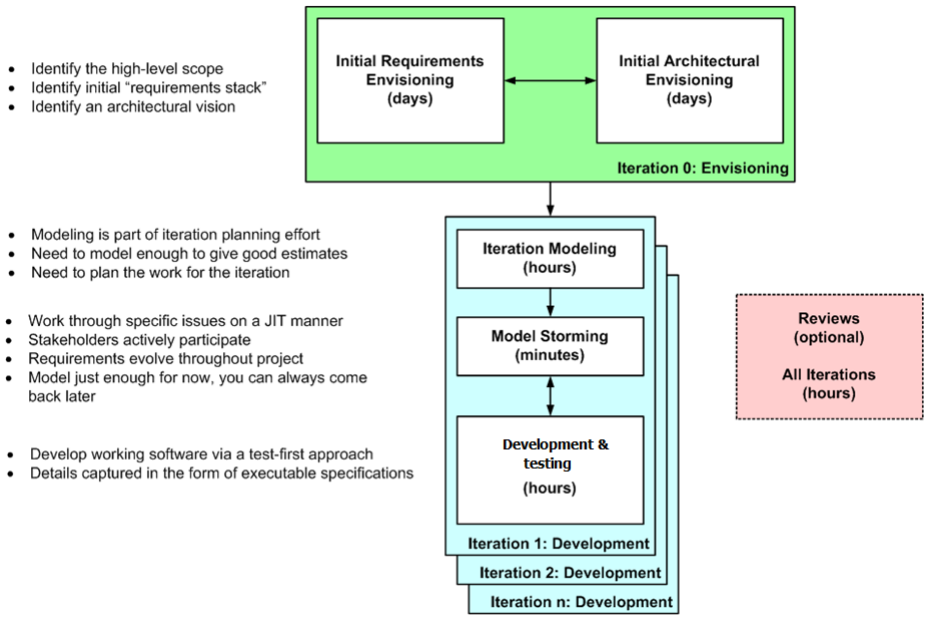

Fig. 3. AMDD development process adapted from [4]

the exam, which can be easily achieved with a single timeline containing several intermediate milestones.

Each timeline can contain several *iterations*, eventually nested, representing the progressive evolution phases of the project. Inside RTC, *plans* are used to manage work items in the context of the given time constraints. Plans are used to modify work assignments for team members, and, being synchronized with the status of the work items, they can be used to track the progress of the work. While it is possible to create many plans for each iteration, we limited this feature, choosing to associate a single plan for each iteration in the timeline. We also decided to associate plans only to the lowest level iterations in the timeline and to limit the type of plans that can be used. In fact, in the RTC environment it is possible to define not only iteration plans, but also plans associated to a *team release*, to a *project release*, and others. This type of plans are useful for the high level monitoring of a project progress, after deciding to keep track only of certain work item types defined as top-level as describing high level tasks. We decided to simplify these aspects by keeping only the iteration plans which are used for the fine-grain monitoring of the progress of each iteration.

The screenshot in Fig. 5 shows the RTC timeline of a typical project following our AMDD process. It can be intended as an instantiation of the metamodeling concepts represented in Fig. 4. Our AMDD process time schedule structures the timeline of the project. Students are encouraged to create the timeline in Jazz

RTC (see Fig. 5), filling in the tool the iterations and adding open work items to each iteration as a "todo" list for each phase of the project.

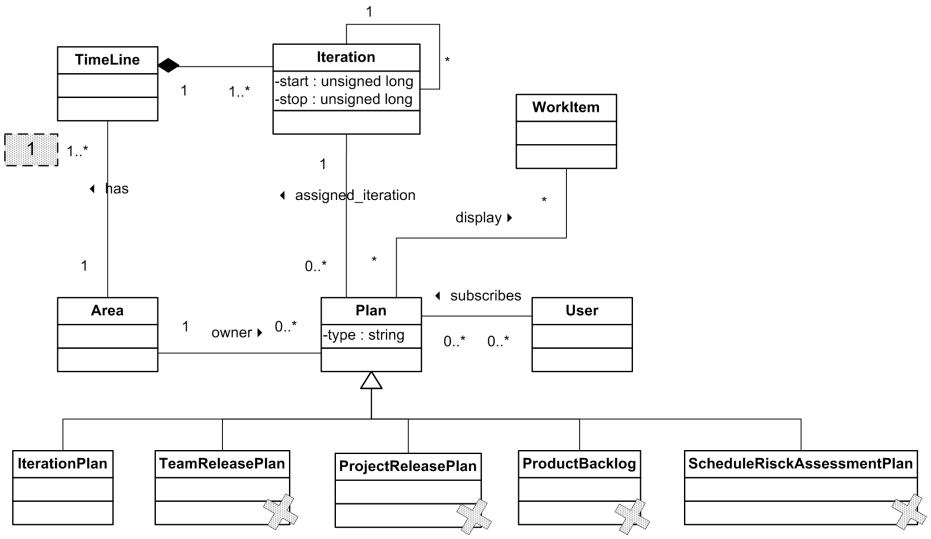

Fig. 4. RTC metamodel: development process perspective.

4 Related work

In this paper, we presented a down-scaling approach of the Jazz RTC platform experimented during a software design/programming course of the engineering faculty at the University of Bergamo. We adopted this technique for allowing students (divided in small teams) to develop and deliver the design/software artifacts of their project works (assignments for the final exam) in a collaborative and traceable manner. Though the proposed approach applied numerous simplifications to the original Jazz vision [5], it is advantageous for the obtained usability of the product and because it helps us to incorporate within the RTC tool software development best practices on agile planning, traceability, iterative development, management of different releases etc., exactly as we explained them during theoretical frontal lessons.

Very few papers exist in the literature – see, for example, [9, 8] – on the use of the Jazz RTC tool at university courses to support teaching software engineering practices. These papers summarize the objectives and the most visible advantages (similar to ours) obtained using RTC from the teaching point of view; but, to the best of our knowledge, none of them explain clearly the concepts, simplifications and configurations adopted to downscale the RTC platform. Instead,

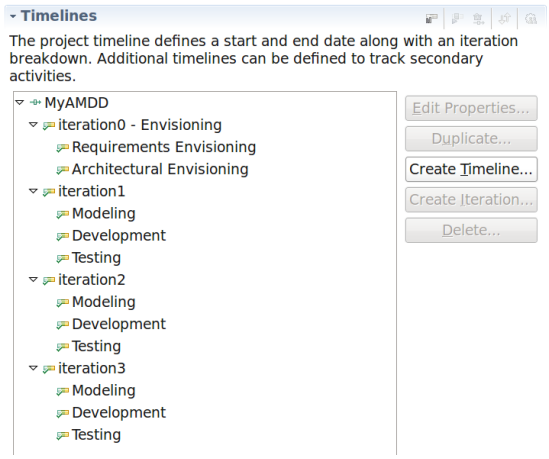

Fig. 5. The simplified AMDD process within RTC

inspired from the work in [7], we illustrated, trough the metamodeling technique, in a transparent and systematic way the pruning effects of our downscaling approach for teaching purposes.

5 Conclusions and future directions

In this paper, we presented the motivations and the work done for downscaling RTC in order to be used in an academic course. The effort required to understand where RTC could be simplified was remarkable but it has allowed a fair explanation of the use of the platform in a reasonable time. We hope that the students will explore other concepts and features during their autonomous use to prepare the final project of the exam.

At the light of our experience, we found that starting from a very rich tool with a considerable number of features and try to simplify it, requires a greater initial effort than the scenario in which the tools is initially very simple and offers very few functionalities and new concepts and features are later added by means of plugins. We believe that if RTC offered initially a simple clean environment which could be enriched by plugins and auxiliary tools, it would have a greater spread especially among small companies which are not ready to embrace the entire jazz/RTC philosophy and which still require a long maturity path in order to adopt all the best practices supported by the tool itself.

The proposed approach allows for its adoption in very small teams, but, as future work, we aim at refining/revising our technique to further dimension the RTC platform for students academic-size applications in order to allow for the management of larger teams by including specific “development roles” and multiple time lines. We also plan to conduct a series of interviews with students

to gather their impression and reactions to Jazz, and to use their experience to design our next iteration of our downscaling approach.

References

1. Eclipse Project, http://www.eclipse.org
2. IBM Rational Team Concert, http://jazz.net/projects/rational-team-concert/
3. Microsoft Visual Studio, http://msdn.microsoft.com/it-it/vstudio/default.aspx
4. Agile Model Driven Development, http://www.agilemodeling.com/essays/amdd.htm
5. Cheng, L., Hupfer, S., Ross, S., and Patterson, J. *Jazzing up Eclipse with collaborative tools*. In Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology Exchange (Anaheim, California, October 27 - 27, 2003). Eclipse '03. ACM, New York, NY, 45-49.
6. Frost, R. *Jazz and the Eclipse Way of Collaboration* IEEE Software, IEEE Computer Society, 2007, 24, 114-117
7. P. Maresca, A. Cotugno, S. Mignogna, R. Longobardi, A. Donatelli, R. Gangemi. *Business process Eclipse Editor (BEE)* Proceedings of the 3rd Italian Workshop on Eclipse Technologies, Bari, Italy, November 17-18, 2008.
8. A. Meneely and L. Williams. *On preparing students for distributed software development with a synchronous, collaborative development platform.* SIGCSE Bull. 41, 1 (Mar. 2009), 529-533.
9. Jaroslav Prochazka. *How Jazz Rocks Teaching Iterative Software Development - Utilizing IBM Rational Team Concert at the University of Ostrava.* CSEDU 2009 - Proceedings of the First International Conference on Computer Supported Education, Lisboa, Portugal, March, 23-26, 2009 - Volume 2, INSTICC Press, 2009