Int. J. Engng Ed. Vol. 24, No. 4, pp. 648–658, 2008
Printed in Great Britain.

0949-149X/91 $3.00+0.00

An Environment to Help Develop Professional Software Engineering Skills for Undergraduate Students*

RUBBY CASALLAS AND NICOLÁS LÓPEZ
Departamento de Sistemas y Computación Universidad de los Andes, Bogotá, Colombia
E-mail: rcasalla@uniandes.edu.co

In this paper, we present a strategy to help students develop the necessary skills to become effective software engineering professionals. We created a software engineering group, called QualDev, composed mostly of undergraduate students. QualDev places students on real software projects, but with some features to ease their control and evaluation. Our educational strategy is to use active teaching/learning methodologies that enable us to create scenarios with regular self-assessment. There are many challenges related to setting up and maintaining such a software development team; we relate our experience in creating and evolving the QualDev group, its organization, projects, methodologies, and processes.

Keywords: software engineering education; team work; active learning; collaborative learning

INTRODUCTION

EDUCATORS meet many challenges in defining software engineering courses, practicum or software capstones projects. A number of authors [1, 2] have presented some of the issues involved in helping students to develop the skills expected from software engineers, such as the effective use of processes and methodologies to ensure quality.

Beyond obtaining knowledge, the ultimate goal of software engineering education is to help students build their own toolbox full of well-understood methodologies, procedures, teamwork principles, metrics, computational tools, languages, etc, as well as giving them the ability to determine which tools to use in a given context.

The abilities and criteria for making these decisions are much more complicated skills than the ability to apply a specific method to build an artifact. These skills are directly correlated with many of the outcomes depicted by ABET as 3a–3k [3]. Equipping students with these skills is a difficult task, and even more so is assessing whether or not this goal has been accomplished [4]. Furthermore, these skills evolve continuously during the professional life of a software engineer. Therefore, as educators, we feel responsible for helping students to initiate the development of these skills.

Conversely, the SE2004 [5] presents a series of knowledge areas in their curriculum guidelines for undergraduate degree programs in software engineering. Each area includes an extensive number of topics that should be covered, each specified using Bloom's attribute [6] as a topic that requires knowledge, comprehension or application. Attributes related to *application* for a topic require students to develop the ability to use learned material in new and real situations. Creating environments that facilitate the development of these abilities is a challenge for educators.

We have created a software development group, made up of undergraduate students, called QualDev†, which produces high quality open source applications. The team develops tools in the broad domain of software process support, some of which have been successfully used in real-world contexts. The users are members of small sized software development companies as well as other development groups within the university, particularly students on various software engineering courses. The QualDev group receives feedback from these users and has to deal with issues of change management.

Students in the QualDev team have acquired the discipline of following processes based on Team Software Process (TSP) [7] and agile methodologies. Students continuously evaluate the effectiveness of the processes and propose adaptations, aiming for continuous improvement; in addition, tools developed in the team and other open source tools support this improvement.

We established the QualDev group four years ago and over 120 people, mainly students, have participated as part of their undergraduate curriculum either as an elective course, during their senior project, or both; therefore the students can work in the group for one or two semesters.

* Accepted 24 April 2008.

† http://qualdev.uniandes.edu.co

By participating in the QualDev group students have an opportunity to develop and apply the skills needed by software engineering professionals. These skills include the experience of dealing with real clients and, also, the students lead and organize development teams, define their process objectives, and are in charge of assessing their fulfillment. Students can effectively get an understanding of how software processes, methodologies, and tools help to produce high quality software. In this environment, they learn how to work as members of a team and they have to deal with negotiation issues.

The educational strategy for the group is the use of active teaching/learning methodologies such as collaborative learning and mentoring, which makes it possible to create scenarios of regular self-assessment of team functioning. Furthermore, we have defined a structure that eases group administration and control. The role of the instructor has changed, he or she is no longer in charge of defining content and lectures, but is more of a guide who gives advice on the identification of objectives, supports students to achieve objectives and gives counsel when issues arise.

The main difference from other software engineering experiences for undergraduate students is that they deal with not only the correct application and use of processes, methods and tools, but also, and more importantly, with the responsibility of making decisions, and understanding and assessing their consequences. Their decisions range from negotiating plans and establishing commitments to choosing design and technology alternatives. Through all these decisions, students are aware of quality issues, client requirements, project restrictions, and ethical concerns.

There are many challenges related to setting up and maintaining such a group. We need to balance pedagogical and experiential aspects, and students need to understand the usefulness of methodologies in a real context, albeit with restrictions that enable control and assessment of these experiences. Moreover, the team has a high turnover rate, since many students enter and leave the group each semester, which makes knowledge management a critical issue. We have defined an induction process supported by tutorials and guidelines; the cornerstone of this process is a mentoring scheme where seniors help novice students to become productive quickly.

In this paper, we present our experience in creating and evolving the QualDev group, our organization, projects, methodologies, and processes. We show how our strategy indeed helps undergraduate students to develop skills necessary for professional Software Engineering practice correlated with the outcomes 3a–3k recommended by ABET [3]. Furthermore, our approach provides an environment where students develop abilities in some of the most challenging areas of the SE2004 guidelines such as software configuration management and software quality.

Additionally we discuss how we have confronted the challenges to create this environment and the lessons learned in this process. Specifically, we show how the team successfully implemented some strategies such as the mentoring scheme and have made it possible to establish a team that is self-organizing, self-evaluating, self-regulated and continuously learning.

THE CHALLENGES

This section presents the challenges we experienced in building an environment that sustains the development of the skills needed for students to become software engineers. By building this environment, we try to fulfill some of the needs pointed out in [8].

Real but controlled software projects

Most real world projects are ongoing and last for more than four months—the length of a term; furthermore, a developer entering a software company usually has to begin working on a project that has already started and may be a development or maintenance one. Thus, a professional software engineer should be able to interact with clients, other project members, and project documentation to understand, maintain, and evolve the software under construction.

A common strategy to reproduce this environment is to have students interact directly with clients and during a period work on a complete development cycle that ends with the delivery of a finished product. Most co-op schemes are carried out in this way, and it is a perfectly valid approach for gaining experience on certain aspects.

However, this situation might not be ideal for other reasons [9, 10]. First, commitment with client requirements and deadlines might overshadow the importance of other aspects of the process. Secondly, delivering a fully functional program within a semester might not be realistic: industrial projects are usually too long and demanding for undergraduate students. Lastly, during this learning phase, students should give equal or superior weight to other aspects of the process than to client needs.

Our challenge is to provide more control on how this relationship with clients is carried out. External clients should help define requirements, provide feedback on prototypes, and provide change requests for existing applications. Nevertheless, project objectives, milestones, and evaluation should not be exclusively related to client needs; they should be set considering the achievement of learning objectives that aim at strengthening the skills of real software professionals.

With such a scheme, students can make decisions and make mistakes when doing so that would often be inadmissible to clients. For example, a bad design or technology choice can result in an unacceptable tool from a client's perspective;

however, it is a chance for students to understand the impact and costs involved in these choices.

Effective teamwork

Students often have reservations towards teamwork because in previous experiences they have not grasped the need for exploring collaboration among their teammates beyond a typical 'you do that and I do this and the day before we just merge our work'. Usually, students in CS/SE programs dislike and avoid creating dependencies among teammates because it is easier to place trust in oneself than in others. Students need an environment that promotes positive interdependence and encourages students to plan collaborative work, dependencies and commit to their realization.

Furthermore, IT careers are usually perceived as disciplines that do not promote group activities [11]. However, this is not true for the software engineering practice, which requires interaction with clients, interdisciplinary teams and other developers. Therefore, a challenge for our whole community is to destroy the perception that software development is a solitary activity by providing an environment where students can develop teamwork and communication skills.

Inevitably, problems will arise with teamwork; however, students need to learn how to handle these situations and resolve them, rather than try to avoid them. Thus, this environment should have explicit mechanisms to handle situations that involve students renegotiating their commitments and values as much as development results [12]. That is, rather than punishing groups that have teamwork issues, they should be rewarded for properly handling these issues.

Responsibility and commitment to quality

A risk that we have when teaching software process is that students need to place so much attention on learning the process that not enough time is left for them to understand the impact that following a software process has on the software quality. Learning the details of the process becomes the main concern, thus hiding its effectiveness to produce quality software. Because of this, students have problems understanding the role that processes have in ensuring high quality software.

This leads to a difficulty for students in integrating all the elements of the process in a coherent way. Students need to witness the usefulness and increase in productivity and quality when using a process, instead of seeing it as an unnecessary burden to develop software.

In a typical course, students begin with client requirements and execute a development cycle during the semester and, at the end, they turn out a fully functional program. However, it is rarely the case that this program is actually used in a real world context after the semester is finished.

This is the basic problem with this approach: students can only rightly appreciate the true cost of quality (or of low quality) once a tool is in production and under use. Once change requests and defect reports start pouring in, developers can appreciate the real consequences in time and effort needed to respond to maintenance requests.

The challenge lies in providing students with an environment that has long-term development projects. These projects must include defect correction and change request execution, as well as new requirement development. With such projects, students can understand the impact that process and methodologies have in the long run.

Knowledge management

As stated above, a software developer entering a company usually has to begin working on a project that has already started. One of the greatest challenges of this process is introducing the developer to the project, and this includes not only the product itself, but the process, methodologies and tools. This issue is broadly related to knowledge management.

Our challenge here is to provide an environment where students can rapidly learn a new process. This environment should include explicit support mechanisms to guide students in this process. Furthermore, a developer with experience should be able to support a new developer in the process of entering a new project. We would like students to understand the complexity of this process by confronting the challenge of introducing a new student who has no experience of using the process, technology, methodology, or tool. Thus, our environment should provide situations that encourage knowledge transfer, and specific processes, guidelines and mechanisms to evaluate if this understanding was achieved.

QualDev PROJECTS

Before presenting our structure and the pedagogical approach, this section briefly describes the kind of projects on which students work.

The group has embarked on development projects of several applications since its creation four years ago. The projects are long term and are continuously maintained and modified. The applications are in the broad domain of process support for software development, and some of these are used in real-world contexts. The oldest project is a software tool called ChangeSet, which supports software configuration management. A more recent project is a planning and scheduling tool called PlanningTool.

The users are members of small and medium sized software development companies: specifically six software companies are currently using our tools to support configuration management and planning processes. Additionally other development groups and courses within the university use our applications. Both undergraduate students and masters students in the basic and advanced

software engineering courses use our tools for planning and tracking. Currently, more than 150 students use our tools each semester. From these users, the team receives feedback and has to deal with change management issues.

Students receive change requests and rapidly release new versions for each specific client. Students must negotiate the scope and reach of each release, and plan the implementation of change requests and testing and integration activities.

ChangeSet and PlanningTool are the largest projects in development; the group currently has a few more projects in execution. We give more detail below of the main characteristics of these two projects to exemplify the types of projects.

ChangeSet

ChangeSet is a tool that gives support for software configuration management, as defined in [13]. Its main purpose is to provide users with a friendly interface for product management using grouped versioned baselines, and for development tracking using managed change requests. We also designed ChangeSet to fit the requirements in terms of simplicity, user orientation, and abstraction of technical tasks.

QualDev members have refined the requirements, design, and implementation over more than 20 development cycles. The tool is currently in use by external clients who give feedback regarding usability, functionality, and the overall performance of the tool. Clients also use the tool to submit change requests and report errors.

Some of the services offered by the tool include project creation and management, configuration item management, version management including files associated with a version, baseline management, and lifecycle management of change requests.

The tool executes on a J2EE [14] platform, specifically on the JBoss container [15]. Currently, there is a stable web-based client for the tool and an Eclipse [16] client under development.

PlanningTool

When QualDev started, the need for a tool to plan and track the work of each person and of the whole group arose. We chose an open source tool called DotProject [17] to support planning and tracking activities. The reason for this choice, besides its availability, was its user-friendly web-based user interface. We could plan all tasks for every team member and then each member could register time logs anywhere.

However, as the planning and tracking process of the team matured, some new requirements emerged that were difficult to accomplish with DotProject. In particular, loading dozens of tasks each week using the web-based interface was time consuming, likewise for centralizing tracking and accountability activities. This motivated the definition of an own Planning Tool, integrated with DotProject.

PlanningTool is a project that has as objective reducing time consumed by planning activities by means of extensions and customizations for DotProject. PlanningTool is currently a set of modules that extend the functionality of DotProject to improve the management and tracking of software projects. The tool provides a much simpler interface for the project planner to introduce tasks and to acquire tracking reports.

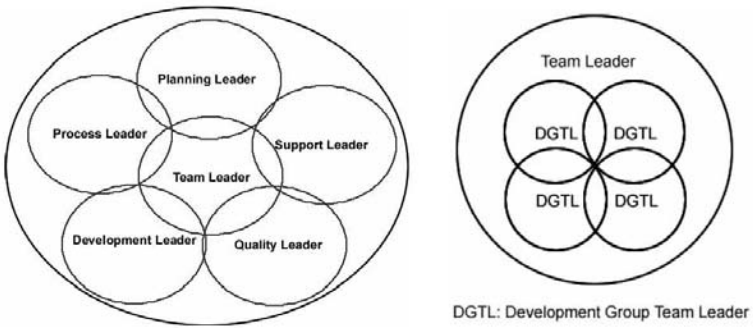
QualDev STRUCTURE

This section presents the team's organization, main processes, basic workflows, and mechanisms to manage and control all the projects.

Participants and roles

Undergraduate students, graduate students, and instructors participate in the team. Students entering the team have previously taken a basic software engineering course and, in most of the cases, a software architecture course. Undergraduate students can participate for one or two semesters. This means that every term a set of students enters their second term while another set of students participate for the first time.

Each semester students are divided into groups of four to six individuals. Students in each group assume a specific role similar to those proposed by TSP [7]; each role has specific tasks and activities



(a) (b)

Fig. 1. (a) Development group; (b) control group.

assigned to it. Each semester there are on average 25 undergraduate students participating in QualDev. The team leader is usually a student in his or her second semester in the team. Figure 1(a) shows a possible structure for a development group.

The leaders of each team and the instructor make up the project control group for QualDev. This group is in charge of overseeing and evaluating all the projects in order to guarantee that the processes are being effective and that quality is assured on each product. Figure 1(b) shows the structure for the control group.

A graduate student (research assistant) is assigned to support each development group. He or she provides support on development issues and problems and helps with tracking of individual performance. Each semester there are on average four graduate students participating in QualDev.

Regularly, we have graduate students in the group; they use our projects to perform experimentation regarding some particular process or the implementation of a technique or method. For example, graduate students have helped with process assessment, and metrics, indicators and report definition. One instructor participates as coordinator; his role is as facilitator and advisor for the team. Since mainly undergraduate students organize the team, the instructor is not the only one in charge of grading; students themselves evaluate and take part in the grading process.

External software companies and other internal development projects participate as clients of the group. They help to define requirements and, more importantly, provide feedback on the use of the tools by means of change requests and defect reports.

Processes

Each group is in charge of one of the projects or at least of a part of it. Groups use a software process based on the key ideas of TSP [7] and adapted by QualDev students through the semesters. When students begin their experience in the team, they have already been trained on the basics of TSP. The QualDev process specifies particular improvements and adaptations proposed by students throughout the execution of development projects and process experimentation since its start. The process defines specific roles with activities and commitments that the student executing the role must fulfill.

Each project executes short development cycles; the duration of a cycle is usually 4–6 weeks, with a launch phase to establish clear and measurable goals and a post-mortem activity to evaluate performance. Students propose and define the goals. This scheme enables a quick reflection on the execution of a cycle, which aims at continuous process improvement. Development cycles are defined independently for each individual development group depending on the objectives. The techniques used to build the software artifacts (e.g. requirement documents, general and detail designs) are all based on Object-Oriented Methodologies [18]. Figure 2 shows a general workflow for a typical development cycle.

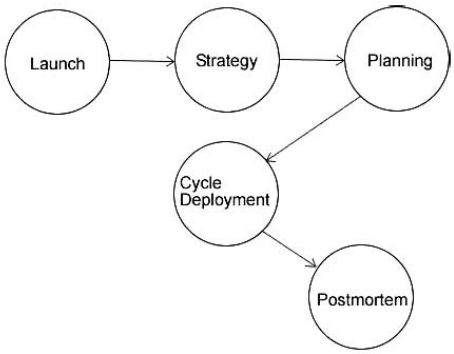


Fig. 2. Development cycle workflow.

The post-mortem process is the main enabler of project adaptability in agile processes. Recognizing which activities are effective in the process and which are not is the main guide to plan process improvement activities. The identification of improvement points and their implementation guarantee continuous improvement and adaptation to constant changes in a competitive environment.

Figure 3 shows the organization for a semester; the launch phase includes the definition of projects, the selection of participants and the induction process that will be presented in the section below on 'Knowledge management'. The induction process takes up the first two weeks of the term and is critical to the success of all projects. During these two weeks, the team has to train and prepare new participants to be productive developers in the group. Afterwards each development group defines its objectives for the cycle with the aid of the instructor and executes a development cycle.

In the middle of the term, the team plans activities to get a perspective of how all the projects are progressing and to give the members a chance to confer with their peers. Subsequently, each project executes another one or two development cycles.

At the end of the semester, the group performs a final post-mortem; this activity is of high relevance

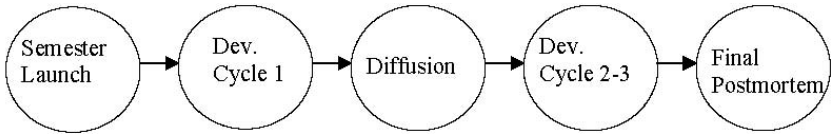


Fig. 3. Semester workflow.

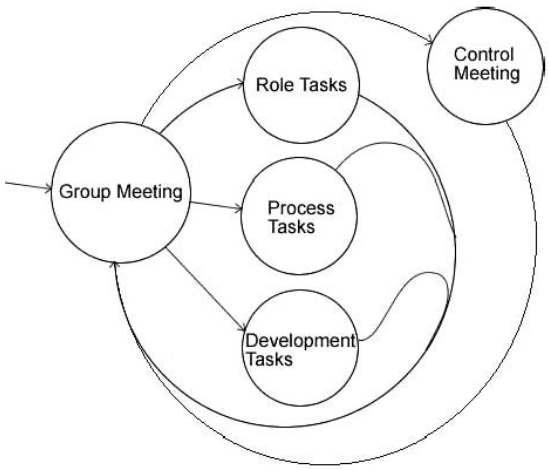


Fig. 4. Weekly workflow.

to fulfilling the goal of continuous process improvement. All participants meet to present the results of their projects. This meeting focuses on the development artifacts produced and the experiences with clients. The team discusses lessons learnt during the term and issues and improvements for the methodologies, the team, and the development process. Students sketch objectives for the next term as well as conclusions about the experience. Figure 4 illustrates the main activities followed during a semester.

During each development cycle, students participate for nine hours per week. Figure 4 shows a workflow for a typical development week. Students spend one or two hours in team meetings, the rest of the time, they plan, develop, and perform role specific activities. The control group meets weekly to assess the evolution of the groups and discusses specific issues that may have occurred.

Grading scheme

Grading for project-based courses in Software Engineering is usually one of the most challenging activities. This is because it is difficult for the instructor to differentiate the results of a project from the results of each participant in the project. Various authors discuss problems associated with identifying how individual students contribute in software engineering projects [19, 9]. In order to cope with this problem, we have defined a grading scheme that receives input from student work on various points during the term in order to determine the individual grade of each student.

Input grades are received from peer evaluations: at the end of each cycle, each participant completes a peer evaluation form. The purpose is to assess the group performance, the contribution and participation of each member in the project, and the quality of the deliverables among other aspects. Additionally, mentors evaluate the induction process of new participants; likewise, new participants evaluate the roles of their mentors.

Finally, the instructor and project control group establish a grade for each team project and an individual grade for each participant each time a cycle ends. All students work the same amount of time weekly and report the time worked; graduate assistants evaluate students weekly based on individual performance considering various aspects such as conformance to process activities, execution of planned tasks, and effective communication with team members. The instructor, during the post-mortem, with input from graduate assistants, evaluates team results based on the definition and fulfillment of group, product and process objectives. We have clearly defined percentages for each of these aspects allowing us to give clear and timely feedback on student performance. However, we still need to improve our assessment strategy by means of a similar strategy to that presented in [20] to include information that helps us to evaluate the *knowledge gained* as perceived by the students.

APPLYING SOFTWARE ENGINEERING

The SE2004 [5] presents a series of knowledge areas in their curriculum guidelines for undergraduate degree programs in software engineering. Each area includes an extensive number of topics that should be covered; each area is specified using Bloom's attribute [6] as a topic that requires knowledge, comprehension, or application. The taxonomy provides a useful structure to categorize the type of attitudes and skills a student should have related to each topic. The attribute application for a topic requires students to develop the ability to use learned material in new and concrete situations. Students should be able to use information, methods, and concepts in different situations and solve problems using their newly acquired skills and knowledge.

We have defined a series of learning objectives that directly address the challenges presented earlier and they are related to the areas of SE2004. These objectives were established bearing in mind some of the outcomes pointed out by ABET as 3a–3k [3].

Moreover, our approach creates an environment for developing skills related to software engineering at the level of application. In particular, our approach introduces undergraduate students to topics in the areas of professional practice (PRF), software management (MGT), Software Process (PRO) and Software Evolution (EVL). These areas are usually a challenge to introduce to students; likewise, it is complex to create an environment where students can apply skills in these areas in a controlled environment.

Professional practice

Professional practice covers the knowledge and skills to practise software engineering in a professional, responsible and ethical manner. This includes skills related to group dynamics, com-

munication and professionalism. The QualDev team specifically addresses developing skills at the level of application for the topics writing and presentation skills. This area also includes teamwork and group dynamics. We present our contribution to the achievement of this topic in the section on 'Effective teamwork', as our approach to confront the challenges of working in teams.

During the last two months of the semester, each student is expected to undertake an individual focus project. Each student sets the objectives of this project; he or she must creatively identify and specify a problem that they can solve in the restricted period. These projects must directly address the needs of the group or of a specific project of the group. Examples and ideas for projects are drawn from past projects.

Students must propose two or three ideas for this project before the middle of the semester. These ideas are discussed with the instructor who helps dimension the project so that it can be achieved within the time limit. After this discussion, the student elaborates a formal report of the objectives and he or she must plan the activities required to satisfy this project.

The student must achieve the objectives he or she established and report the results by means of a document and oral presentation to the group. This approach is similar to common methods of problem-based learning and gives students the opportunity to develop skills at the level of application for the topics of writing and presentation skills.

Software management

Software management relates to skills for planning, organizing and monitoring the software development life cycle. This includes management concepts, project planning, organization and control, and software configuration management. These skills are not commonly included in undergraduate programs at the level of application.

The achievement of QualDev planning and tracking process resides mainly in two facts. First, it rests on the commitment of each student to fulfilling a plan and, more importantly, on the registration of time logs. Owing to the discipline of students, earned values are a useful mechanism to track the progress of projects. Second, the success of the process resides on the level of detail in which the plans are produced; this high level of detail has enabled us to reduce the percentage error in estimated development time to less than 15% on average. Now, students can estimate, for each artifact, in a precise way, the time needed to produce, modify, inspect, and test it.

Thanks to the ChangeSet tool, QualDev members are very aware of software management. This process consists of identification and standards schemes for artifacts, baseline creation, change control, and accountability. The ChangeSet tool gives support for all these activities.

To support source code versioning, we use the CVS tool [21] but its details are hidden behind some more abstract concepts managed by ChangeSet. Currently, students are developing a series of Eclipse plug-ins to support developers in creating a change set (a set of artifacts needed to fulfill a change request or to correct a defect) without having to create branches or tags directly on CVS.

Software process

Software process covers the knowledge related to commonly used software life-cycle processes. Even though the SE2004 does not require students to apply skills in this area, we believe that professional practice requires students to have skills to use methods and concepts of software process in new situations.

From the beginning, students have understood the importance of having detailed definitions of processes for all activities. They rapidly learnt the importance of having the process documented because this helps coach new students entering the group. They realized that if the process is not documented the probability of losing or forgetting good practices is very high.

From the beginning QualDev team has documented process definitions. First in an informal way but now, with the help of graduate students, processes are documented using the Business Process Modeling Notation (BPMN) [22] graphical notation and a set of more specific templates to describe the details. These include templates to define guideline documentations and checklists for process execution. Additionally, we have defined a scheme to organize support documents related to a process and present them in a web page that allows for easy navigation. Figure 5 shows an example of a process definition using BPMN.

Developers enacting the process have guidelines at their disposal on how to produce every artifact. The interests and initiatives of participants drive the definition and improvement of these processes. Furthermore, students with the same role in different groups have paired up on their own initiatives to discuss the advantages or problems that a process may have. Tools and process definition have been improved or modified because of these reflections.

Software process definition has become a project in itself. Students decided to put the process on control configuration management, using, obviously, the ChangeSet tool. Now, they receive change requests for processes by means of this tool. The project control group also plays the role of a change control board, which makes decisions regarding requests such as establishing priorities or planning the execution of a change request.

Software evolution

Software evolution covers the skills related to the process of ongoing support to stakeholders and clients as a result of changing assumptions, problems, requirements, architectures and technologies. The concept expands upon the traditional

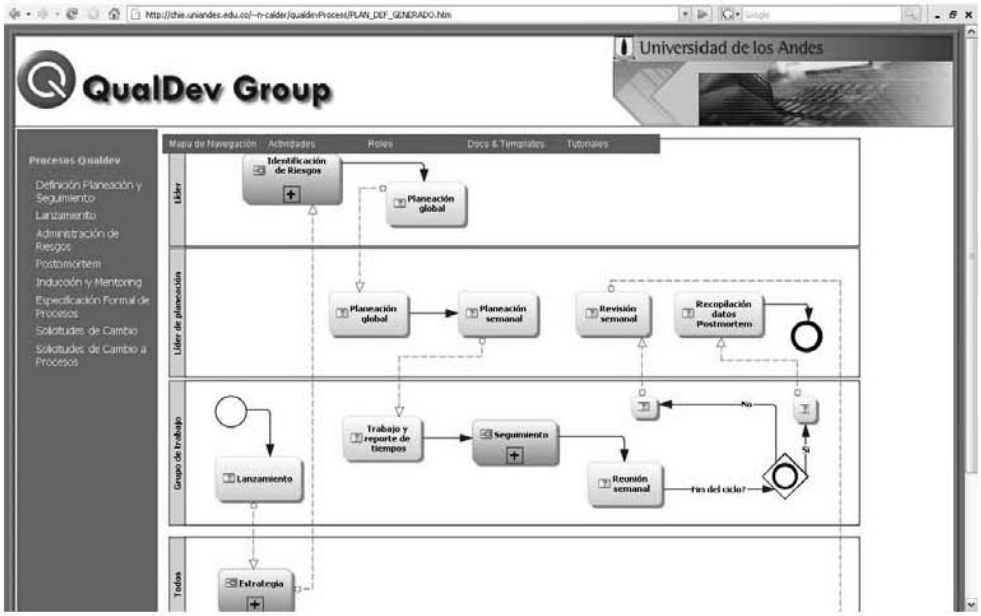


Fig. 5. Process web page.

notion of software maintenance. The SE2004 does not establish topics for the application of skills; however, we believe that professional practice requires an initial exposure to these issues in real but controlled situations.

Software evolution and maintenance makes sense in the group because real clients request changes or adaptations to the tools in use. We have a process that defines how to receive change requests, plan a release with a set of them, develop, and test them, and integrate and deliver the release.

Adding new functionalities to the software has not been a problem for students. Complicated problems arise when maintenance involves changes in technology. For example, with the ChangeSet tool the team has had many problems in trying to keep the tool up to date vis-à-vis the new versions of the JBoss container and the J2EE specification (now JEE5).

The group can evidence the cost of making decisions on the fly regarding upgrading versions or using new promising tools. Students found that if they underestimate the impact of changes, it is impossible to plan the costs with even the slightest accuracy. Since then, the students make prototypes of a solution to test a new technology and to have elements to estimate the costs of changes. As a result of these experiences, students have encountered new situations that require the application of skills related to maintenance and evolution.

CONFRONTING THE CHALLENGES

Real but controlled software projects

In QualDev, external clients give input to projects by means of change requests; otherwise the instructor and graduate assistants mediate other type of interactions with clients. One graduate assistant is in charge of communicating with software companies, and supports them in the implantation of our processes and tools. From his or her work, the instructor and assistants define the projects for each semester and some guidelines or directions for students to define objectives that are reachable yet ambitious.

The projects are long term, so students face the challenges of maintenance activities; additionally, the instructor guides students to define objectives for implementing new requirements. Each group should release one or two versions of the tool with some new requirements and some maintenance requirements.

The instructor encourages students to fulfill the objectives. However, if problems occur, students commit themselves as much to completing the objectives as to other aspects of the process. With this approach, the responsibility of fulfilling client requests and deadlines is as relevant as following the process and, most importantly, committing to process improvement.

When a new project starts, it might be unreasonable to think that students can develop a fully functional tool in just one semester. To tackle this challenge, in QualDev, the instructor guides the students to define their objectives for new projects so that they include designing and defining the architecture for a complete version of the tool, but only implementing a small number of requirements. At the end of the semester, students release a simple tool with two or three requirements. The implementation of the rest of the requirements can continue in the next semester, so eventually, a complete release of the version is produced.

Effective teamwork

QualDev relies on collaborative learning as the main tool to develop skills related to group dynamics and teamwork and to tackle the problems that students have when working in teams. Cooperative learning [4, 23] is a method that requires students to work in teams with specific roles under conditions that include positive interdependence, individual accountability, face-to-face interaction, appropriate use of collaborative skills, and regular self-assessment of group functioning.

Each team establishes collaboratively two or three development cycles in the semester with specific objectives negotiated with the project control group. Members must achieve the objectives as a team relying on each other. Each team meets once a week to track their advance and plan activities for next week. Students negotiate their activities and their dependencies with other members' activities.

All students work the same amount of time weekly and our grading scheme guarantees individual accountability.

The group has various mechanisms to achieve regular self-assessment of group functioning. The grading schemes presented above show some aspects of the assessment of the teams. Additionally, each group is required to evaluate at the end of a cycle their performance as individuals and as a group. During the post-mortem, students must identify the main problems, challenges, and issues that their team had and propose strategies and solutions to these problems.

The means to strengthen teamwork include the face-to-face interactions that occur during weekly meetings. Additionally, pair programming and other group activities are encouraged; other means of interaction such as e-mail, forums, and instant messaging are commonly used.

Responsibility and commitment to quality

Students must be aware of the usefulness, and increase in productivity and quality when using a process, rather than seeing it as an unnecessary burden to develop software. In QualDev the students understand how the process contributes to overall quality of the product. Each semester students face the challenges of maintaining and evolving applications with equal emphasis placed on the product and the process. We believe that with this approach students understand the impact that processes and methodologies have in the long run. As evidence of this, we present some examples of experiences that are commonly faced by students.

As mentioned above, each group produces one or two releases each semester. We have defined a process for defining, integrating, and testing new releases. These releases are put in production as soon as possible. In the case of tools being used in other courses, this process is almost immediate. With this scheme, students immediately receive feedback on the usage of the tools.

On some occasions, tools with bugs were released because a faulty release creation process was in place. Because of this, students redefined their release process to include a more structured approach to testing, before releasing the product and a process to track and classify the type of errors found by users.

Another example of the quality culture in the team is the way in which students face design or architectural mismatch. Students must understand and continue the work of previous students and, in this process, they almost always find some form of mismatch between design and implementation. Under the guidance of the lecturer, the students reorganize their plans to include correcting the defects. In this process, students usually sacrifice the commitment of some objectives to correct bugs or mismatches. Thus, students are faced with the cost of non-quality, experiencing first-hand the impact that processes can have on quality.

Additionally, QualDev team uses a variety of methods and tools to perform its activities. They all have something in common. They are open to the community as well as the tools developed, which are open source. Because of some erroneous decisions made in the past concerning changes in a tool, we now have an established process for evaluating a new technology, documenting the investigation, and developing a prototype. This is a mitigation plan associated with the risk of affecting the success of a project due to a wrong choice of technology.

Knowledge management

Introducing a new developer to the team is an issue that includes not only introducing the specific application that the student will work with, but also our process, methodologies and tools. This condition is aggravated since the group has a high turnover rate: many students enter and leave the team each semester. However, we have defined a mentoring scheme that leverages the high turnover rate to create a process to introduce easily and effortlessly new students to the team.

One of the main advantages of our project is having participants with different levels of expertise. In particular, we have undergraduate students who are in their second term as well as new ones. This means that every term, a set of students enters their second term while another set of students participate for the first time.

We have used this fact to our benefit by defining a mentoring scheme to ease the induction process of new students to the group. Instead of having an instructor or the team leader directly solve every issue that a new participant has, we define a scheme whereby each new participant has a mentor: a second semester student in charge of assisting and helping him or her.

The mentoring scheme helps in various ways: first, it helps new students to be able to put to rest

simple doubts that otherwise might be left; it gives them a quick and clear source of information, no matter how trivial it might be. For example, a student might be embarrassed to bother the instructor with a simple compilation error; he would be more likely to contact his mentor who is not seen as such a distant figure. Secondly, it reduces the load on an instructor or the team leaders who have to control and organize all the students. The instructor can focus only on particular issues, such as a student who might be having problems contacting his or her mentor or is having difficulty working in a team. Finally, it creates self-confidence in the mentor students who can clearly see what they have learned with the group and this helps develop their teamwork and leadership skills.

Each student and his or her mentor must meet at least once a week for the first weeks of the semester. In the beginning, the focus is on installing the development environment, and understanding and using support tools in the way that the process defines. After that, the focus moves to understanding role specific activities and support tools.

CONCLUSIONS AND FUTURE WORK

QualDev team is a professional real world experience where students face diverse challenges regarding teamwork issues, processes, rapid technology changes, and tools that real software companies deal with on a day-to-day basis. The group provides a competitive advantage for students starting their professional careers.

Using long-term projects, which are being used by real clients and require constant maintenance, enables us to create an environment where students can achieve the established learning objectives.

The group is a good example of a course that uses active learning methodologies and is designed to provide students with the skills they will need in their profession. The department and university can learn from experiences and update course material, methodologies, and curriculum based on a better understanding of these aspects.

We are currently starting discussions with industry groups of software companies. We hope that in the future this will enable us to refine our methodologies to fit better the needs of the industry. We envisage, in the short term, establishing relationships with industry to assist in the creation of high-tech companies lead by the students. Students are interested in these skills and our mission will be to facilitate the creation of a bridge between the students and the external sector.

There is still a lot to learn and current issues to solve in our process, organization and methodologies. However, we strongly believe that we have established the base for continuous improvement and this will enable us to offer students a rich environment in software engineering to carry out research and experimentation.

REFERENCES

1. A. T. Chamillard and K. Braun, The software engineering capstone: structure and tradeoffs, *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE'02)*, (2002) pp. 227–231.
2. M. Gehrke, H. Giese, U. Nickel, T. Niere, J. Wadsack and A. Zfindorf, Reporting about industrial strength software engineering courses for undergraduates, *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*, (2002). pp. 395–405.
3. ABET (Accreditation Board for Engineering and Technology), *Criteria for accrediting engineering programs. Effective for Evaluations during the 2005–2006 Accreditation Cycle,* http://www.abet.org, last visited on October 3, (2006).
4. R. Felder and R. Brent, Designing and teaching courses to satisfy the ABET engineering criteria, *Journal of Engineering Education*, **92**(1), 2003, pp. 7–25.
5. IEEE/ACM Joint Task Force on Computing Curricula. Software Engineering 2004, Curriculum *Guidelines for Undergraduate Degree Programs in Software Engineering*, IEEE Computer Society Press and ACM Press, (2004).
6. B. Bloom, M. Englehart, E. Furst, W. Hill, and D. Krathwohl. *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain*, Longmans, USA, (1956).
7. W. Humphrey, *Introduction to the Team Software Process,* Addison Wesley Longman, USA, (2000).
8. K. Surendran, H. Hays and A. Macfarlane. Simulating a software engineering apprenticeship, *IEEE Software*, **19**(5), 2002, pp. 49–56.
9. L. van der Duim, J. Andersson and M. Sinnema, Good practices for educational software engineering projects, *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007)*, (2007) pp. 698–707.
10. H. Vliet, Some myths of software engineering education, *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, (2005) pp. 621–622.
11. L. Williams, L. Layman, K. Slaten, S. Berenson and C. Seaman, On the impact of a collaborative pedagogy on African American millennial students in software engineering, *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007)*, (2007) pp. 677–686.
12. T. Hilburn and W. Humphrey, Teaching teamwork, *IEEE Software*, **19**(5), 2002, pp. 72–77.
13. CMMI® Web Site http://www.sei.cmu.edu/cmmi/ last visited on 2006-10-03.
14. E. Roman, S. Ambler and T. Jewell, *Mastering Enterprise Java Beans*, 2nd edn, Wiley, (2002).
15. JBOSS team. http://labs.jboss.com/portal/ last visited on 2006-10-03.

16. Eclipse project http://www.eclipse.org/ last visited on 2006-10-03.
17. Project Management Software http://www.dotproject.net/ last visited on 2006-10-03.
18. B. Bruegge and H. Dutoit, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems,* Prentice Hall, (2000).
19. O. Hazzan and Y. Dubinsky, *Teaching* a software development methodology: the case of extreme programming, *Proceedings of the 16th Conference on Software Engineering Education and Training, 2003 (CSEE&T 2003)*, (2003) pp. 176–184.
20. G. Mitchell and J. D. Delaney, An assessment strategy to determine learning outcomes in a software engineering problem-based course, *Int. J. Eng. Educ.*, **20**(3), 2004, pp. 494–502.
21. CVS Tool. http://www.nongnu.org/cvs/ last visited on 2006-10-03.
22. Business Process Modeling Notation (BPMN) Information. http://www.bpmn.org/index.htm last visited on 2007-07-05.
23. D. Johnson, R. Johnson and K. Smith, *Active Learning: Cooperation in the College Classroom.* Interaction Book Co., (1998).

Rubby Casallas is an Associate professor in the Department of Systems and Computing Engineering, University of Los Andes, Bogotá, Colombia. She received her Ph.D. in Informatics from the University of Grenoble, France in 1996. Currently she is the coordinator of the Software Construction group at the University of Los Andes. Her interests are Software Engineering Education and Software Product Lines.

Nicolás F. López is an instructor in the Department of Systems and Computing Engineering, University of Los Andes, Bogotá, Colombia. He received his MSc in Systems and Computing Engineering from University of Los Andes in 2005. He is currently a software engineering instructor and leader of the QualDev group.