Learning Software Testing using a Collaborative Activities Oriented Platform

Bénédicte TALON  
Université Lille Nord de France  
BP 719 – 62228 Calais  
talon@lil.univ-littoral.fr

Dominique LECLET  
UPJV-MIS  
33, rue St Leu 80039  
dominique.leclet@u-picardie.fr

Arnaud LEWANDOWSKI  
Université Lille Nord de France  
BP 719 – 62228 Calais  
lewandowski@lil.univ-littoral.fr

Grégory BOURGUIN  
Université Lille Nord de France  
BP 719 – 62228 Calais  
bourguin@lil.univ-littoral.fr

Abstract

This publication describes a teaching initiative of software testing that makes use of a collaborative environment. Learning software testing is in fact a domain which needs practice in order to develop the necessary know-how. We led a project-based experiment with a collaborative environment of a new kind. We wanted to verify its ability to serve our particular learning context. In this article, after a quick definition of software testing, we discuss the problem of teaching software testing. We then present a project-based organization to teach skills dedicated to software testing. We describe the generic collaborative working platform we have used in our pedagogical context. We conclude with some results of an experiment.

1. Introduction

The management of defective software is a major and expensive activity. It is a significant challenge for companies and consequently for the universities that have to educate the future CS professionals.

One of the authors had to deal with the challenge of teaching software testing to students.

It is essential to put the students in situations close to reality. These situations allow them to meet the requirements of tests and to show them the interest of a good organization.

We thought about proposing to students a project related to the production of software tests. The teaching objectives did not aim at educating experts but were rather centered on targeted knowledge and competences like knowing: the various families of test, useful specifications for the tests, how to practice some principal tests, how to produce documents related to the test, and how to organize a test project.

The teaching device presented in this paper is the result of a collaboration between researchers working on a pedagogical method [1,2] and others working on software environments supporting collaborative activities [3]. Together, they have produced a collaborative environment that serves as a technological support of a teaching device dedicated to test.

2. Teaching Software Testing: a Challenge

Software testing is a software engineering activity that has “the intent of finding errors” and not of demonstrating “that errors are not present” [4]. Testing implies executing code but also “activities such as design reviews, code inspections, and static analysis of source code (…) even though code is not being executed in the process of finding the error or unexpected behaviour” [5].

Even if crucial, this activity remains a difficult and generally underestimated activity in the Computer Science training curricula. Indeed, detecting the different failure modes of a complex software is difficult [4,6].

According to Whittaker [6] “This difficult, time-consuming process requires technical sophistication and proper planning. Testers must not only have good development skills — testing often requires a great deal of coding — but also be knowledgeable in formal languages, graph theory, and algorithms”.

Our CS University Degree proposes two years to train CS practitioners who will work in a project team. A Software Engineering unit is charged to sensitize students to techniques and activities of software production. In this unit, we must initiate them to software testing and develop rigor in the processes they use, which is a hard task due to the profile of these low experimented students.

We applied the MAETIC pedagogical method which offers a rigorous framework to the development of professional skills. This method has been instrumented thanks to CooLDA, a tailorable collaborative platform.

3. An experimented pedagogical method

MAETIC is a pedagogical method which organizes a project-based pedagogy. MAETIC has been built to develop and acquire professional skills. In the project-based approach, the learner builds its knowledge and know-how thanks to the project [7]. MAETIC describes activities managed by a group of students and directed
towards a concrete production. The teacher activities consist to animate, not to decide.

MAETIC rests on 5 phases which correspond to traditional activities of project management. During each activity, students have to realize sub-activities and to produce deliverables.

Communication and Information Technologies (CIT) bring students many advantages [8]: work at distance, distribution and sharing of documentation, communication. CIT also present many advantages for the teacher such as centralized and at distance control, communication with students, regular follow-up, etc.

4. The COOLDA platform

CooLDA is a theory-based environment i.e. an environment built in respect of what Activity Theory tells us on the human activity. Further information about this approach and the results we obtained can be found in [3].

Our main goal is not to create tools, like a new discussion tool, or a new text processor. Rather, we want to create an environment that integrates existing tools. According to us, each tool supports one kind of activity. When several tools are used in parallel by a group of actors, they generally serve a more global activity than the particular activity they were designed for. For example, a group may use in parallel a chat, a web browser, a text processor, and a mailing software. Each of these tools supports a particular activity (online discussion for the chat, etc.) but they do not know each other. Then, our purpose is to provide an environment that can create the context of use of the different tools involved in a global cooperative activity managing the links between its different (sub-)activities.

To achieve this, we have created an activity model [3] (figure 1). Each activity is linked to a resource that proposes operations. A resource corresponds to a software tool, like a mailing component, for example. A user is an actor in the activity, as he plays a role in it, role that allows him to do actions. An activity can be linked to other ones, when the role of one of its actors implies that this user plays another role in another activity. Finally, an activity is an instance of a task, which constitutes an activity model, or pattern, that can be reused.

These concepts have been implemented in CooLDA, a distributed environment based on the Eclipse platform and dedicated to the support of collaborative activities [3,9]. The system can be adapted to the emergent needs of the users during the activity. We built this environment as generic as possible, in order to be able to support any kind of activity – provided that the appropriate tools to integrate exist.

5. The Software Testing Learning Scenario

The learning scenario was conceived in order to respect the various steps of MAETIC, while taking into account the constraints (8 meetings with one meeting per week) and the targeted competences (practice of various testing techniques, use and realization of documentation related to the test, organization of the team work).

The lectures are made available through the platform in a specific shared folder. A scenario indicates to the students what to read and when. A Teacher’s logbook recalls which activities to carry out each week and explains which resources to use or produce. Models of document are given via the platform in order to assist students.

The program sources to be tested are programs written in C by others former students. Students are thus confronted with realistic code, with common errors and bugs. Along with the source code, software requirements and functional specifications are provided. They allow students to understand precisely the software specifications and to build a correct test plan.

Evaluation sheets are also provided to facilitate the testing process and assist students in reporting the discovered errors. Students have to produce and deliver test reports which are placed on a centralized repository. The teacher can easily access them later thanks to the platform.

6. Preparing COOLDA for Software Testing

The major part of the work consisted in ‘translating’ the MAETIC method organized for software testing in the CooLDA’s activity model (Figure 1). This has been done by representing the MAETIC method by five inter-connected activity models (also called tasks) corresponding to the five steps of the method.

Each activity model describes a particular step of the process. In this kind of platform, designed to propose tools, the user chooses the activity he/she wants to join.

We have identified tools that could suit the needs of the MAETIC activities. These tools had to be pluggable in CooLDA, which relies on the Eclipse platform. For instance, we choose Office Integration Editor plug-in that integrates OpenOffice.org tools in Eclipse. We also use Eclipse's integrated CVS client and Eclipse’s CDT (C/C++ Development Tooling), a set of plug-ins dedicated to the development and test of C/C++ programs. We have
also adapted Eclipse’s integrated web browser, and integrated a chat plug-in. Other tools fostering collaboration are provided by CooLDA itself (awareness view, perspectives sharing facility, etc.).

In order to finely integrate these tools, we have identified (by introspection on the tools themselves) which methods could be use for their integration and for their piloting. These methods have been wrapped into Operations in the activity models. Finally, specific Actions have been defined for each role, in order to determine how these tools should be configured and piloted.

7. The Prototype

A student connects to the environment and chooses, for example, to join the ‘planning’ activity (Figure 2). Then CooLDA retrieves automatically the last versions of the shared documents that are managed through a CVS, it pilots the Office Integration Editor plug-in in order to open the right document(s), and it connects the Chat plug-in to the right server and channel, with the user’s name.

![Figure 2. Screenshot of a planning Activity](image)

On the figure 2, we observe several tools: a Teacher’s logbook (tab labeled “Journal de bord”) automatically opened showing the last information on the learning unit, a Group’s logbook (tab labeled “Journal de bord”), a chat, a project explorer. Joining an activity starts the activation of the tools involved in this activity (In figure 2, the spreadsheet is launched for the planning).

The environment is completely integrated. Students don’t need to shift from an environment to another. They can manage their project, consult the teacher’s logbook, manage their own logbook, discuss, realize reports and execute tests in the same global environment.

8. Conclusion and Perspectives

An experiment has been undertaken during year 2008. We noticed that students appreciated the project, because they had the feeling to have learnt and understood the test preoccupation. The teacher strongly appreciated the centralization, in her environment of the follow-up of the groups: availability of documents and logbooks, integrated into the same environment offered her an easy follow-up of the groups’ activities and consequently an easy control and assessment. The nature of the interface directed towards the activities facilitates the organization of students’ work

The originality of the platform, compared to other Learning Management Systems, lies in its intrinsic characteristics. It is a completely integrated work environment: it can be seen as being at the same time a collaborative environment for project management (offering chat, documents sharing, awareness) and a development environment (dedicated to software testing).

9. References