Use of Microsoft Testing Tools to Teach Software Testing: An Experience Report

Ing. Gustavo Lopez, Universidad de Costa Rica

Gustavo Lopez is a researcher at the University of Costa Rica's Research Center on Information and Communication Technologies (CITIC), where he has worked since 2012. He has contributed to several research projects on software testing and human-computer interaction, and he has also designed and taught training courses on topics related to software testing. Previously, he worked as a Software Engineer at a software development company in Costa Rica. He received his B.S. in Computer and Information Science from the University of Costa Rica in 2011. His research interests include in software testing, human-computer interaction, gender in computer science and computer science education.

Dr. Alexandra Martinez, Universidad de Costa Rica

Alexandra Martinez is an Associate Professor in the Department of Computer and Information Science at the University of Costa Rica (UCR), where she has worked since 2009. She has taught graduate and undergraduate courses in Databases, Software Testing, and Bioinformatics. She has done applied research in software testing and bioinformatics at UCR's Research Center on Information and Communication Technologies (CITIC). Previously, she worked as a Software Design Engineer in Test at Microsoft Corporation in Redmond, WA, and as a Software Engineer at ArtinSoft in San Jose, Costa Rica. She received her Ph.D. in Computer Engineering from the University of Florida in 2007, her M.S. in Computer Engineering from the University of Florida in 2006, and her B.S. in Computer and Information Science from the Universidad de Costa Rica in 2000. She also received a scholarship to study in the Pre-Doctoral Program in Computer Science at the Ecole Polytechnique Fédérale de Lausanne, Switzerland, from 2001 to 2002.

Use of Microsoft Testing Tools to Teach Software Testing: An Experience Report

Abstract

This paper reports our experience using Microsoft testing tools in both graduate and undergraduate Software Testing courses for four semesters. In particular, we used Microsoft Visual Studio *Ultimate* 2010 (including Microsoft Test Manager 2010) and Microsoft Team Foundation Server 2010, which together offer an integrated and comprehensive environment for the application lifecycle management, including test planning, authoring, automation, execution, tracking, monitoring and managing. We assessed our experience in using the tools from the student`s and the teacher's points of view. Based on students' feedback, we found that students not only consider they learned a lot from the labs (where Microsoft tools were used) but also deem the tools easy to use, relevant to the course (supported the learning of course concepts), and valuable for their professional career. On the other hand, based on the teacher assessment, Microsoft tools provide support for the application of many different concepts studied along the course within an integrated environment, reducing the learning curve for students, while offering the added value of an industrial-level tool.

1. Introduction

Software testing is a critical activity in software engineering, accounting for 30% to 90% of the total labor expended in developing software[15]. Yet software testing remains an underestimated activity in the Computer Science training curricula[20]. At the University of Costa Rica, the Bachelor of Science's program in Computer and Information Science offers an elective undergraduate course in software resting, and Master of Science's program in Computer and Information Science offers an elective graduate course in software testing as well. Both are 4-credit-hour courses, with 64 hours of class time in a 16-week semester. The undergraduate and graduate versions of the courses are very similar in their core contents (since the undergraduate course is not pre-requisite for the graduate one), differing mainly on the applied research project (only performed at graduate level), advanced topics presented by students (topics and depth vary according to the level), and the types of activities developed in and outside the class (for example, rich in-class discussions are far more common at the graduate level than at the undergraduate level, since graduate students are more mature and have more work experience). The main goal of both courses is to guide students in the learning of the theoretical foundations and necessary skills for understanding and applying software testing processes, techniques, and activities within the context of software quality assurance. By the end of the semester students are expected to (*i*) understand and explain the fundamentals of software testing, (*ii*) identify best practices for software testing and quality, (*iii*) compare and apply different techniques, levels and types of software testing, as well as (*iv*) plan, manage and implement a software testing process.

In order to effectively teach students how to test real-world software, the choice of software tools, exercises, and lab projects is crucial, and we particularly believe that industry-level tools together with practical labs or projects need to be used in the learning setting. We have observed a tendency to adopt a set of separate open source tools in software testing courses, where each tool serves a different purpose at a particular facet of the testing process, but these tools rarely integrate and while they help students to learn a particular topic of the course, from our perspective, this approach fails to give students an overall, integrated and comprehensive view of the testing process with its activities, facets, and relationships among them. Moreover, we think this approach unnecessarily increases the cognitive load on the students since they need to master several tools in a short period of time and they might get lost in the details of each new tool they are faced with rather than focusing on the "big picture" of the task they are trying to solve (i.e., a concept they are trying to apply and learn). We therefore decided to adopt Microsoft's testing tools, which provide support for all the concepts we want to teach (test planning, authoring, automation, execution, tracking, monitoring and managing) in a single integrated environment that is also widely used in the industry, thus overcoming the aforementioned shortcomings. In particular, we used Microsoft Visual Studio *Ultimate* 2010 (which includes Microsoft Test Manager 2010) and Microsoft Team Foundation Server 2010.

Our contribution is to report on our experience using Microsoft testing tools in the context of both graduate and undergraduate Software Testing courses, offering a guideline for using the tools to apply the different concepts learned throughout the course. This could help colleges and university professors who teach similar courses decide whether it is worth adopting these tools in their courses.

Previous teaching and training experiences on software testing techniques and tool support at academic and industrial settings have been reported by Xie et al.[22] but they consider only developer testing. The main lesson learned is that the use of industrial tools and technologies reduces the load on students during the course and helps them get some experience that could be applied later.

Other valuable experiences presented by Harrison[9] indicate that in order to help students learn both the developer's and tester's perspective of testing, a two-part project is a good approach. The first part of such project consists in developing an application and testing it appropriately (testing from the developer viewpoint), while the second part consists in testing other student´s application (testing from the tester viewpoint). A problem with this approach is that due to the timeline of the course, the course ends up focusing more on the development than on the testing part (the author reports that 55% of the time is spent developing while only 33% of the time is spent testing, and the remaining 12% is spent writing a short reflection paper).

There have also been experiences using "real-world" (industrial) software under test in testing courses[8], as a mean to effectively teach students how to test real software. The major risks of this approach are confidentiality and technical support on software that is under development by others. Garousi[8] states that this approach requires and leads to strong academia-industry partnerships, but points out that it is necessary to achieve certain level of confidence between the parts to start such projects.

Wong[21] describes a way to improve the state of undergraduate software testing education by incorporating core concepts into the Computer Science curricula. Wong states that it is not enough to teach students the fundamentals of software testing, but it is also necessary to expose students to useful software testing tools that are used both in the academia and in the industry. The main issue we perceive is that they use one tool for each testing technique, and this can be cumbersome if more than one technique is used in the term project.

Acharya et al.[1] present two active learning approaches to teach software verification, each used at a different institution. In both cases, there is a clear separation between the theoretical and practical parts of the course. In one of the institutions, the following tools are used: a bug tracking system, a unit testing tool, a mock developing tool, a white box testing tool (including code coverage and cyclomatic complexity), and a mutation testing tool. One of the main problems they face is compatibility among the software they use and the lab machines. Such problems are expected when several tools with different requirements are used instead of a single integrated package.

Chen et al.[4] suggest a teaching approach based on diversity principles with the use of lectures, tutorials, projects, and panels; however, no tools are suggested to achieve the goals proposed by the authors. The proposed teaching approach is based on a lecture-tutorial-project-panel model. The lecture presents the basic concepts, the tutorial offers small examples to illustrate the concepts, the project reinforces the contents of lectures and tutorials through practice, and a discussion with students (panel) allows the difficulties to emerge and be solved in group.

Dukes et al.[6] present a case study where five different tools are used for web application security testing but then again each tool is used independently to addresses a different vulnerability problem (not in an integrated environment). Not all the tools presented in this research are free; two are commercial standalone tools. This prevents students from viewing the vulnerabilities as a whole problem. We think this could be useful for a course that is focused on security testing but not for an introductory course on software testing.

Garousi[7] presents open modern software testing laboratory courseware that is similar to the one we report in this paper, but he uses several tools and SUTs. One of his findings is that testing educators should align the choices of SUTs and tools with the ultimate goal of the course at hand, the type of students, and the time and resources available to the students in the course.

Other forays into improving the teaching of software testing have been reported. For instance, Cowling[5] describes how a staged approach from software engineering is applicable to software testing, and shows that incremental development is not well supported on several curriculums. Martinez et al.[11] present their experience with two reflection mechanisms: a learning journal used in a Software Testing course, and a two-part reflection questionnaire used in a Software Quality Assurance course. Smith et al.[19] explain how they used peer reviews to teach software testing within a Data Structures course, by encouraging collaboration and competition among students.

The rest of the paper is organized as follows. Section 2 describes the context of the courses. Section 3 presents the labware used in the course. Section 4 mentions the implementation and assessment methodology. Section 5 discusses our findings. Section 6 concludes the paper and outlines our plans for future work.

2. The Courses Context

2.1. The Undergraduate Software Testing Course

The undergraduate-level Software Testing course, CI-2400, is a 4-credit-hour elective course in the 4th year of the Bachelor of Science's program, in the Department of Computer and Information Science at the University of Costa Rica. The course is offered regularly, with an enrollment of 15 to 22 students. The class meets twice a week for one hour and fifty minutes during 16 weeks, for a total of 64 hours of class in a semester. All courses at our university have the same grading scale: 0 to 10, and students require a grade of 7.0 or higher to pass the course.

2.2. The Graduate Software Testing Course

The graduate-level Software Testing course, PF-3866, is part of a group of several software engineering courses regularly offered by the Master of Science Program in Computer and Information Science at the University of Costa Rica. It is a 4-credit-hour course with 64 hours of class time in a 16-week semester, and a 2-credit-hour co-requisite lab course where students put theory into practice by developing a large practical project or an applied research project. The class meets once a week for 4 hours with 10-minute breaks every hour.

2.3. Course Objectives and Contents

As it was previously mentioned, the undergraduate and graduate versions of the Software Testing course are very similar in their core contents and objectives, since one is not prerequisite for the other.

The main goal of the courses is to guide the students in the learning of the theoretical foundations and necessary skills for understanding and applying software testing processes, techniques, and activities within the context of software quality assurance.

After completing the course, students are expected to be able to:

- Understand and explain the fundamental concepts of software testing.
- Identify best practices for software testing and quality.
- Compare and apply different techniques, levels, and types of software testing.
- Plan, manage and implement a software testing process.

The course reading materials are based on several reference books and recent papers in the area. The course contents are divided in six units, which are listed on Table 1.

The main difference in content between the undergraduate and graduate versions of the course was the choice and depth of advanced topics presented by students. Because of the co-requisite lab course, graduate students usually chose a topic that was related to the applied research project they were developing in the lab course, which usually made the presentations richer and better since students not only had to grasp the theoretical background of the topic but also put it into practice by means of some experiment.

Table 1. Course Units.

Unit Name	Unit Content
1. Principles of Software Testing and Quality	Basics: quality assurance and control, software testing, verification and validation (V&V), test cases, software defects. The testing process and V&V activities during the software lifecycle. Quality factors.
2. Planning and Managing the Testing Process	Components of a Test Plan according to IEEE 829 Standard, and alternative approaches. Management of the testing process. Defect reporting and tracking. Test metrics.
3. Types of Testing	Static vs. dynamic tests, manual (technical reviews) vs. automated tests, black box vs. white box tests, functional vs. non-functional tests (load & performance, security, localization, usability, accessibility), and regression tests.
4. Levels of Testing	Unit testing, integration testing, system testing, user acceptance testing. Alfa, beta, pre-release (RC) y final tests.
5. Test Design Techniques	Black-box techniques: equivalence partitioning, boundary value analysis, cause-effect graphing, intuition and experience. White-box techniques: control flow testing (statement coverage, decision coverage, condition coverage, decision-condition coverage, multiple-condition coverage, path coverage, basis path coverage), data flow testing (all defs coverage, all uses coverage, all DU paths coverage). Combination Testing (all-pairs technique). Exploratory Testing.
6. Advanced Topics	Student presentations on advanced topics in software testing. Candidate topics include: mutation testing, cloud testing, model-based testing, database testing, performance/load testing, security testing, and automated test generation tools.

2.4. Teaching Approach and Course Evaluation

Our teaching approach for the Software Testing courses is based on Just-in-Time Teaching (JiTT)[17,16,18] and other active learning strategies[2,12,13] such as in-class discussions and cooperative learning activities. Active Learning has been described as the use of techniques that involve students in the learning process, rather than listening to a lecture in a passive way[13]. Students can actively learn by reading, writing, discussing, solving a problem, or responding to challenging questions[12,13]. We adopted active learning strategies since they have been shown to improve student comprehension and retention of material[3] as well as to increase student motivation and higher order thinking[2,13]. JiTT, in particular, enables the student to go to class ready to actively participate in the different activities, and to have a feeling of ownership since classroom activities are tailored to their needs[17]. One of the authors has had previous successful experiences with the use of JiTT in other courses[10].

Following the JiTT spirit, we required students to prepare for class by reading in advance the assigned material, and we made sure students actually read by having online reading tests that had to be completed the night before class. In this way, the teacher could read students' responses and adjust the next lesson based on common difficulties or doubts expressed by students. In-class activities rely on active participation by the students, especially at the graduate level. For example, rich discussions contrasting different viewpoints and work experiences among students are common at the graduate level since graduate students are more mature and have more work experience than undergraduate ones. The majority of the classes combine short lectures, exercises, and discussion on the assigned readings. Some classes are taught directly in the laboratory so that students learn to use Microsoft's testing tools with the assistance of the teacher (although the lab guides were designed to be self-sufficient).

The course evaluation was not the same across the four implementations of the course, varying mainly due to the level of the course (grad/undergrad) but also based on students' feedback and teacher's self-assessment. Some of the evaluation instruments used in the course included quizzes, exams, oral presentations on advanced topics, term projects, lab reports, and reading tests. Not all of these were used in all implementations; for instance, quizzes and reading tests replaced exams in the last two implementations of the graduate course, and lab reports replaced the term project in the last implementation of the graduate course (previously, there were no lab reports although there were lab sessions).

The term project is multiphase and performed teams of 3 to 4 students. The main phases of the project are: (*i*) design and specification of test cases, (*ii*) manual execution of test cases and incident reports, and (*iii*) design, implementation and execution of automated tests. Students implement the project using Microsoft's testing tools, which they learn to use in the labs. By using an integrated environment (Microsoft Test Manager) to specify user stories, test cases, test plans, test runs, and bug reports; they have the added benefit of traceability (from bugs to test cases to user stories) as well as richer queries and reports. Additionally, the same suite of tools (in this case, Microsoft Visual Studio) supports the development of automated tests such as unit tests (with code coverage metrics), user interface tests, load tests and performance tests. It is worth noting that the *Ultimate* version of Microsoft Visual Studio actually includes Microsoft Test Manager, which although is a separate application, it integrates nicely with Visual Studio. The use of Microsoft testing tools in the term project and labs complements the theoretical knowledge with a realistic hands-on experience that will be helpful in their professional career.

2.5. The Students

The majority of our undergraduate students are full-time students, although some work as course assistants at the university. A small percent of students have full-time or part-time jobs outside the university. Students who take the undergraduate Software Testing course are seniors.

On the other hand, all of our master's students work fulltime in different industries, from small and medium-size software organizations to large IT Departments of non-IT companies. Most of the students work on software development, some are software architects or project managers although there is a trend for students to start working as software testers. A small percentage of our students have other job profiles such as service management and support desk professionals.

2.6. The Teacher and Labware Creator

The first author designed and developed the labs, which included written guides, examples, tools, infrastructure, and software under test. He initially created these labs as part of training on software testing requested by an internal software development unit from our university. This training was offered in the context of a research project on software testing and quality where both authors participated and which was supported by the Research Center on Information and Communication Technologies (CITIC) and the Department of Computer and Information Science. At the time of the study, the first author was a researcher at CITIC with over one year of experience as a software tester and training instructor. He also voluntarily helped students during the lab sessions of the Software Testing courses taught by the second author.

The second author was the instructor of the Software Testing courses. At the time of the study, she was an Associate Professor at the Department of Computer and Information Science, with over three years of teaching experience and over two years as principal investigator in research projects. She has a Ph.D. in Computer Engineering, academic background in software testing, and three years of industry experience as a software tester.

3. LABWARE

3.1. The Origins

At the beginning of 2011, a software development unit from our university contracted CITIC's training services to provide both a course on the fundamentals of software testing and a workshop on using Microsoft testing tools to support the software testing process. This unit develops and maintains most of the software used internally in our university. The unit's staff develops software in Visual Basic using Microsoft development tools, thus the practical part of the training focused on learning to use Microsoft testing tools and incorporating them into the workflow of the development team, leveraging their knowledge of the Microsoft environment.

The training was conducted in two stages of 18 hours each: first, a theoretical course on software testing (taught by the second author) and then, a hands-on workshop on the use of the tools (taught mainly by the first author). Two researchers from CITIC (one with expertise in IT and another with expertise in software testing) were in charge of developing the lab guides, examples, and software under test to be used in the hands-on workshop. A set of ten labs was designed, using Microsoft Visual Studio Hands on Labs as a basis but adapting them to fit the needs of the unit for which the training was targeted. Lab guides were written to be easy to follow in order to avoid the need for constant trainer intervention.

The labs covered the main features of Microsoft testing tools such as version control, test case creation and management, load tests, performance test, UI automated tests, unit tests, automatic and manual bug report, coverage metrics and virtual environments using lab management. The labs were refined and corrected after their use in the training workshop. Then, the labs were adapted for use in the Software Testing courses.

3.2. The Tools

The tools we used were Microsoft Visual Studio (VS), Microsoft Test Manager (MTM) and Microsoft Team Foundation Server (TFS), in their 2010 versions. The *Ultimate* skew of Visual Studio includes the Test Manager application. These tools were chosen because they provide an integrated and comprehensive suite for the application lifecycle management, including test planning, authoring, automation, execution, tracking, monitoring and managing.

3.2.1. Microsoft Team Foundation Server

The Visual Studio Team Foundation Server is the collaboration platform at the core of Microsoft's application lifecycle management solution; TFS is an industrial tool that can be used to support either agile or cascade development processes. TFS delivers source control, work item tracking, automated builds, project web access and informative web sites and reporting. TFS is part of Microsoft Team System Servers, which also include build servers and testing servers. TFS is used to store data and manage software projects[14].

3.2.2. Microsoft Test Manager

The Microsoft Test Manager application is a tool introduced along with VS and TFS 2010. It is used to create and organize test plans and test cases, and execute manual tests. MTM is built specifically for testers to be able to interact with other members of the project team. MTM allows the planning, execution, analysis and tracking of test cases associated with one or more software development projects, MTM also allows the automation of some kind of tests. MTM enable testers to access Lab Management that help tests across multiple virtual environments and it is the main tool to support testing and communication with the rest of the team[14].

3.2.3. Microsoft Visual Studio

Microsoft Visual Studio is an integrated development environment used in console, graphic and web applications as well as web services. Visual Studio includes support for testing tasks including unit testing, performance and load testing, and UI automation. It also allows visualization and reporting of relevant work items in the development and testing process [14].

3.3. Labs

Table 2 shows the labs used in the course, along with the content units they support. All units except the first one have at least one supporting lab, allowing students to apply the concepts studied throughout the course.

The first sets of practices are focused on the management part of the tools as it addressed the source control capabilities and the work items management. Microsoft Team Foundation has features designed to assist enterprise software development teams manage the work and help with software defect tracking.

Table 2. Labs per course unit.

Course unit	Supporting Lab
1. Principles of Software Testing and Quality	None
2. Planning and Managing the Testing Process	1. User stories and task management lab 2. Source control lab 3. Test case and test plan management lab 4. Bug reporting and management lab
3. Types of Testing	5. User interface automation lab 6. Performance testing lab 7. Load testing lab
4. Levels of Testing	8. Unit testing lab
5. Test Design Techniques	9. Code coverage lab
6. Advanced Topics	10. Virtual environments lab

The first lab on **User stories and task management** aims to get students familiarized with the tools (particularly, Visual Studio) by teaching them how to manage work items. "Work item" is the generic name for user stories, tasks, test cases, and bugs. Work items are the central elements within Microsoft tools for managing a software development project. Work items' workflow or states can be customized depending on the team's requirements. In this lab, students learn to create and manage user stories (requirements) and tasks, and link them together for traceability purposes (knowing which tasks are associated to which user stories). This is useful because later in the term project students are asked to create and store user stories for the software they are testing, so that test cases and bugs can be traced back to the original requirements.

The second lab is on **Source control.** The main purpose of this lab is to teach students the benefits of using a source control manager, and how to resolve code conflicts that may arise when working with other people on the same code base. For this lab, Visual Studio is used. Clearly, TFS is used in the background, but students interact with Visual Studio.

The third lab is about **Test case and test plan management.** For this lab, we use the Microsoft Test Manager application. Nevertheless, since MTM and VS are part of an integrated tool suite, we can either manage existing test cases (for instance, created in VS) or create new ones from MTM. This lab guides students in the creation of a test plan and corresponding test cases. It also contains informative sections that explain the possible states of a work item, as well as how to manually execute test cases and keep track of test execution statistics.

The fourth lab is on **Bug Reporting**, and it aims to teach students how to adequately report and manage bugs (or defects) using the tools. For this lab, MTM is used. Although creating a bug report may seem simple, students are reminded of best practices recommended in industry standards such as IEEE 829. Special attention is paid to the quality of bug reports when grading the term project.

The next three labs focus on black box testing and non-functional testing, and cover specifically performance tests, load tests, and automated user interface tests (the main testing types supported by the tools). All three labs deal with test automation. These labs are on **UI automation**, **Performance Testing**, and **Load Testing**. Visual Studio is used in all of them. The main purpose of these labs is to enable students to develop advanced automated tests that can be automatically executed as many times as needed.

The next two labs focus on white box testing, particularly unit testing and obtaining coverage metrics. In both cases, Visual Studio is used. The **Unit testing** lab teaches students how to automatically generate partial test code (generated code is just a skeleton), complete the test code with the help of assertions and other test logic, and create data-driven tests from existing tests. This lab and the following one are designed to be executed one after the other, since coverage metrics are obtained from unit tests. The **Code coverage** lab teaches students how to obtain code coverage metrics when running unit tests. Sometimes we have asked students to add more unit tests so as to reach 90 to 95% of code coverage, thus engaging students into a little competition.

The last lab is on **Virtual Environments.** The main goal of this lab is to teach students about the virtualization capabilities of the tools, which allow to create and manage virtual environments for testing. We have to mention that this was the only lab that was not executed in any of the four implementations of the course, due to the infrastructure requirements needed to run it. The only time this lab was executed was during the practical training for which the labs were initially designed.

3.4. The Software Under Test

We used three different software under test (SUT). The first SUT developed was the widely known "triangle" program, which takes three integer values as input, and outputs whether the triangle is scalene, isosceles, or equilateral. We found a java class that coded the triangle program, so we translated it to C# in order to be able to use it from Microsoft´s tools. Then, we purposely injected some bugs into the code so that students would find some bugs during testing. This SUT was used in the unit testing and code coverage labs.

The second SUT was designed to show the source control capabilities of Microsoft tools. It consisted in a set of small classes (one class per student) that would simply output a message with the student's name. This SUT was used in the source control lab, where the students were asked to first modify the code of a classmate, then their own, and later attempt to check in (commit) their code to the source control tool. The purpose was to expose conflicts that could arise with different code versions and to use the tool to resolve them.

The third and most-used SUT was the MVC Music Store application provided by Microsoft as a tutorial application built on ASP.NET MVC. The MVC Music Store is a web application that sells music albums online, and implements site administration, user sign-in, and shopping cart functionality. This SUT is used in the following labs: Load testing, Performance testing, UI automation, User stories and task management, Test case and test plan management. A modified version of this application with injected bugs is used in the Bug reporting and management lab.

4. Implementation and Assessment

4.1. Implementations

There have been four implementations of the Software Testing course that have used the Microsoft testing tools. The first implementation was on the first semester of 2011 and it was the graduate version of the course. The second implementation was on the second semester of 2011 and it was the undergraduate version of the course (in fact, this was the only undergraduate implementation we have). The third implementation was on the first semester of 2012 at the graduate level and the fourth and last implementation was on the second semester of 2013 at the graduate level as well.

4.2. Assessments

Our approach of incorporating Microsoft testing tools in the course was assessed from the students' and teacher's perspectives. The students' perspective was gathered from a survey. Participation in the survey was voluntary and anonymous. The teacher's perspective was obtained from an analysis of strengths and limitations, and a list of lessons learned.

The survey contained questions related to the materials used during the course, including the lab guides for using Microsoft testing tools. Other set of questions focused on the teaching and learning strategies used in the course (for instance, lectures, exercises, discussions, readings, applied research project, and classmate presentations of advanced topics). There was also a set of questions related to the use of Microsoft tools for testing, which are relevant for this study. In each implementation of the course, approximately 15 students participated in the survey. In the next section, we present the aggregated results of student responses across the four implementations of the course.

5. Findings and Discussion

5.1. Students' assessment

Seven of the 35 questions in the survey were related to the use of the Microsoft testing tools and the hands-on labs. The first question was "How much do you think you learned from the labs (using Visual Studio and Microsoft Test Manager)?" The multiple-choice answers were: a lot, a little, nothing, and no-response (NR). Figure 1 summarizes student responses to this question.

We observe from Figure 1 that 73% of the 63 students who completed the survey consider that they learned a lot from the labs, i.e., from using Microsoft testing tools as the practical component of the course. An interesting finding is that the lab practices and term project won the second place (a tie) among the course activities which students learned most from, only surpassed by teacher's lectures and readings (tie for first place).

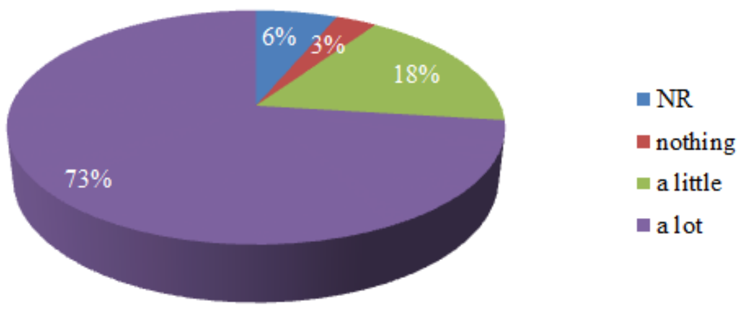

Figure 1. Students' assessment of how much they learned from the labs.

The second question was if the students thought there was a good balance between theory and practice in the course. Here, 79% of the students who completed the survey stated that there was a good balance, although some pointed out that the course was toilsome. The other 21% of the students though that there was not a good balance, with some indicating that there was a big focus on theory and a lack of practice.

Students were also asked to what extent they thought that the Microsoft testing tools were (*i*) easy to use (i.e., user friendly), (*ii*) relevant to the course (i.e., supported the learning of concepts studied in the course), and (*iii*) valuable (i.e., knowing these tools gives added value to the course and to you as a professional). The multiple-choice answers were: a lot, a little, nothing, and no-response (NR). Figures 2, 3 and 4 show the results for these three questions.

From Figure 2 we observe that 86% of the students who completed the survey think that the tools are highly user friendly. This leads us to think that it is good idea to use these tools in an academic setting such as the aforementioned course, since the learning curve is mild and students feel at ease using the tools. In Figure 3, it is not a coincidence that 92% of the students think that the tools are highly relevant to the course since the labs were designed to support the main concepts studied in the course (the same concepts were covered in the training that originated the labware). From Figure 4, we note that 88% of the students state that the tools are highly valuable and provide an added value in their professional lives. We suppose that since most respondents are graduate students who work, most of them possibly knew Microsoft Visual Studio only as a development IDE but were unaware of the potential of its testing features until they took the Software Testing course.

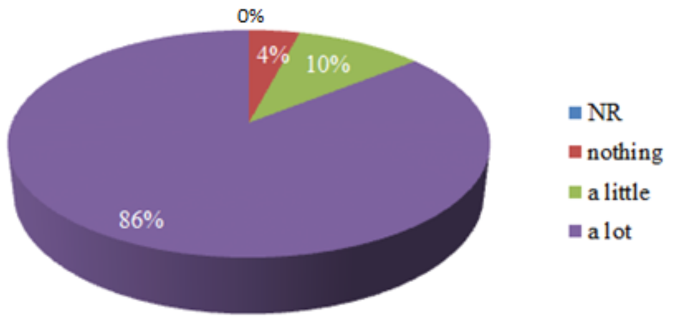

Figure 2. Usability of the tools (user friendliness).

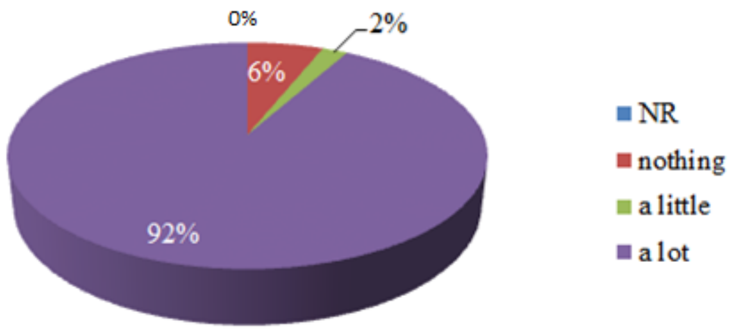

Figure 3. Relevance of the tools to the course (support for concepts studied).

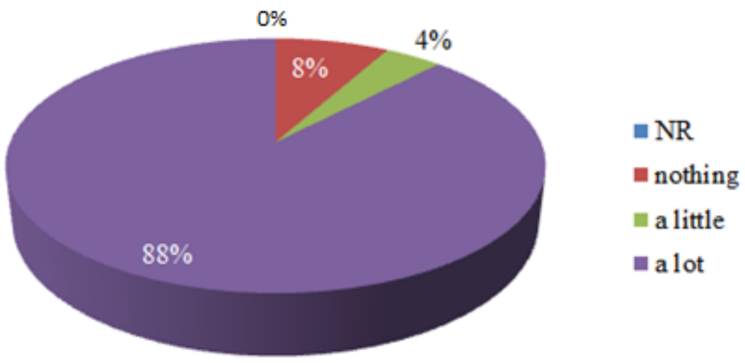

Figure 4. Value of the tools (added value to the course and students).

Another question asked was “Do you think that the use of Microsoft testing tools was enriching for your learning?” Table 3 shows some of the comments students wrote regarding the use of Microsoft tools.

Table 3. Sample student comments regarding the use of Microsoft testing tools.

	Comment
Student 1	I work with. NET, so the tools were very useful for my job. I liked that the teacher was brave enough to use these tools in the course, despite the fact that the university discourages the use of proprietary software. I have nothing against open source software but I do not like the philosophy of rejecting proprietary software even if it is more suitable for teaching.
Student 2	It broadened my knowledge of tools that I can use for testing and how to use them.
Student 3	It’s a good tool and is used by many software development companies.
Student 4	A lot, since this tool is currently used in the industry, working with it drew us closer to the way companies operate nowadays.

We also asked students what other aspects they would have liked to learn in the course about the Microsoft testing tools. Table 4 shows some of the comments made by the students.

Table 4. Sample student comments regarding things they would have liked to learn about the tools.

	Comment
Student 1	The coverage of testing tools was pretty good.
Student 2	Personally I am satisfied with what I learned. I guess there must be much more functionalities than the ones we used but I am happy to know I could use the tool and now it is part of what I can write in my resume.
Student 3	I think that the introduction to the tool allows us to investigate by our-selves anything extra we want to know.
Student 4	I think they are useful tools that should be employed earlier in the course. You can get more out of the tools when people are familiar with them. Labs were performed towards the end of the course and there was no time to truly understand the functionalities; we were simply following the guides.
Student 5	I would like a more robust and complete example on unit testing that includes several classes and functionalities, preferably on a web appli-cation.
Student 6	I would have liked to learn about the Lab Manager.
Student 7	Tests with virtual servers. I found them quite interesting and useful in the "real world".

The last two questions were "What did you like most about the course?" and "What did you like least about the course?" Tables 5 and 6 show some of the comments made by students in response to these questions.

Table 5. Sample student comments about what they liked most in the course.

	Comment
Student 1	The practical part with Visual Studio and the feedback from people who use it.
Student 2	The labs, which I think should be given a little more time and depth.
Student 3	The project, where one could perform the testing process for an ap-plication, from the test planning to the tests execution and reporting defects in the application.

Student 4	What I liked most about the course was the development of the project because it taught me what should be done in a real life project.
Student 5	The use of Visual Studio to implement the tests.
Student 6	The topics and tools used because they favor my work performance, therefore, I believe that the knowledge acquired in this course can be applied in the workplace.
Student 7	Learning testing techniques and use of Visual Studio.
Student 8	The labs and to be able to know a new tool.

Table 6. Sample student comments about what they liked least in the course.

	Comment
Student 1	Using little open source software.
Student 2	The pile of work associated with the team project.
Student 3	The short time devoted to the labs.
Student 4	The fact that we only used one tool. I would have liked to study alternative tools. VS and TFS are very good tools but for those who develop in a different language, they fall short.

5.2. Results from Teacher's assessment

The teacher made a qualitative assessment on the strengths and limitations of the use of Microsoft testing tools, based on her experience across the four implementations.

5.2.1. Strengths

The main strength of having such integrated tool suite to support a course is that students are able to put into practice the different concepts and facets of the testing process within a single environment (IDE), with the consequent benefit of minimizing the learning curve and allowing full traceability from user stories to tasks to test cases to automation to bugs (which would be cumbersome if different tools were used for different parts of the process). Another advantage of using Microsoft testing tools is that they support the concepts studied across all but one of the course units. Additionally, we believe that learning a commercial tool that is actually used in real industrial settings is an added value for students. In fact, a considerable number of graduate students who took the course indicated that they use Microsoft tools in their work, but they did not know about the testing capabilities of these tools, so learning these in the context of the course was a direct benefit for them and their organizations.

A characteristic that we use frequently and that we deem important for teachers is that the tools allow us to monitor and evaluate the individual work and progress of the students, while allowing them to collaborate as team members. This is enabled by the use of individual accounts in the TFS server combined with team projects to which users belong. This is an enormous help for the teacher because if there is a student who is not doing his part of the project, the teacher can easily detect it, and can penalize that student rather than the whole team. The logging mechanisms embedded in the tool also make it more difficult for students to cheat (for example, if a student copies test cases or bugs from a classmate, this will show in the work item's log since the tool store who created the item, who modified it, who copied it, etc.).

It is also worth mentioning that there is a vast amount of documentation, tutorials and videos available from Microsoft to help teachers and students alike to learn how to use the tools (documentation is usually scarce in open source tools).

Lastly, with the recent development of Pex and Moles[*] from Microsoft Research, there is an opportunity for students to use Microsoft tools in combination with Pex and Moles for their applied research project.

5.2.2. Limitations

A major limitation that we found was the overhead of correctly installing and configuring the Team Foundation Server and the user/group permissions adequately. We therefore recommend the teachers who want to adopt this tool in their course to seek support from the IT Department during the installation, configuration and setup.

Another limitation is the non-negligible learning curve for the teacher to master the different aspects of the tool, due to the fact that it is a rich commercial tool with many features and complexities. We recommend allowing at least a month time to learn the tool before starting the course.

One technical limitation with the tools is that the user interface automation (over web applications) only works with some browsers; being Microsoft's Internet Explorer the one that works best. This is a major drawback if other browsers need to be considered during testing.

5.3. Lessons learned

One of the lessons we learned is that it is a good idea to use a tool or tool suite that is easy to use by the students in order to reduce the learning curve for learning the tools and have the students readily focused on using the tools to adequately do the testing (rather than on integrating, solving compatibility issues among different tools, or trying to get the tool to work). Another lesson is that using a tool that students can use at their workplace gives an added value to the course, and since commercial tools are in widespread use in the industry, it makes sense to teach students a good commercial tool rather than an academic-only prototype that has a low chance of actually being used in the industry.

Since the tools are industrial it can require a large set of servers depending on the scale needed. We suggest using a medium size server for the TFS and small size server for the web applications that need to be published for testing. With these two servers all the labs can be conducted, except or the Virtual environments lab, since it requires a virtualization server

[*] Pex and Moles are Visual Studio add-ins that provide code analysis for .NET code and automated white box testing, available at http://research.microsoft.com/en-us/projects/Pex/

running Hyper V and depending on the amount of people running the test at the same time, memory requirements grow. We normally have 16 people working in pairs, for a total of 8 simultaneous accesses, and the memory requirements grow up to 32 GB. Additionally, at least two network adapters are needed.

The hands-on labs approach for the practical part of the course help students to learn the tools without much assistance from the teacher, allowing the teacher to spend more time answering questions related to the application of the theory rather than the specifics of the tools. The written lab guides are also an excellent aid to students because they serve as a reference that can be used at any point during the course or later, if they forget how to do something.

We also found out that the quantity and extension of the labs is about the right workload for students, but it needs to be spread out through the semester and just in time when the theoretical concepts are studied. In the majority of the implementations, labs have been relegated towards the end of the course (due to several reasons), which was indicated by many students as an aspect to improve in the course. Another lesson learned from the students' feedback is that more time needs to be allocated to complete the labs, since it was common that students did not finish on time and had to complete it on their own outside of class.

6. Conclusion and Future Work

We presented an experience report on the use of Microsoft testing tools in a Software Testing course. The tools used were Microsoft Visual Studio *Ultimate* 2010 (including Microsoft Test Manager 2010) and Microsoft Team Foundation Server 2010. Our experience comprises four implementations of the course (i.e., four semesters), one at the undergraduate level and three at the graduate level. The course labware is described, including a detailed list of the labs (hands-on practices) that were used to support each course unit. Our approach of incorporating Microsoft test tools in the course was assessed by the students through an anonymous survey, and by the teacher through a qualitative analysis of strengths and limitations.

Results from the students' assessment indicate that students consider the Microsoft testing tools very easy to use, relevant to the course since they help learning course concepts, and highly valuable for their professional career. Additionally, a majority of students said they learned a lot from the labs (where the tools were used). Some students commented that the labs, term project, and use of VS was what they liked most about the course (both the labs and term project made heavy use of the tools), which we think is a positive sign. On the down side, some students commented that they would have liked to learn other tools as well (particularly, open source ones), others complained about the amount of work required by the term project, and yet others complained about not having enough time for the labs (practical part of the course).

The teacher's assessment indicates that the main advantage of the tools is that students are able to apply the different concepts and parts of the testing process from one integrated environment, thus minimizing the learning curve for students, and allowing traceability and integration of the different test elements across the different parts of the testing process. The main disadvantage is the overhead of properly installing and setting up the Team Foundation Server as well as configuring the user/group permissions adequately. Other strengths and limitations were described, as well as a list of lessons learned.

In the future, we plan to explore the use of a more complex and realistic SUT for the unit testing lab, as suggested by one student in the survey. We would also like to address common complains from students by allocating more class time to the labs and distributing the labs better across the semester. We also plan to incorporate two more labs: one on "Exploratory testing" and one on "Static code analysis", since we discovered that the 2012 version of the tools provides good support for these topics.

7. Acknowledgments

This work was partially supported by CITIC, under the project No. 834-B2-A14, and by the Department of Computer and Information Science from the University of Costa Rica.

8. References

1. Acharya, S. and Schilling, W. Effective Active Learning Approaches To Teaching Software Verification. *American Society for Engineering Education* (2012).

2. Bonwell, C. C. and Eison, J. A. Active Learning: Creating Excitement in the Classroom. *ASHE ERIC Higher Education Report No. 1.* The George Washington University, School of Education and Human Development, Washington D.C., USA, 1st edition (1991).

3. Briggs, T. Techniques for Active Learning in CS Courses. *Journal of Computing in Small Colleges* (2005).

4. Chen, Z., Zhang, J., and Luo, B. Teaching Software Testing Methods Based on Diversity Principles. *Conference on Software Engineering Education and Training* (2011).

5. Cowling, T. Stages in Teaching Software Testing. *International Conference on Software Engineering* (2013).

6. Dukes, L., Yuan, X., and Akowuah, F. A Case Study on Web Application Security Testing with Tools and Manual Testing. *Proceedings of IEEE* (2013).

7. Garousi, V. An Open Modern Software Testing Laboratory Courseware – An Experience Report. *IEEE Conference on Software Engineering Education and Training* (2010).

8. Garousi, V. Incorporating Real-World Industrial Testing Projects in Software Testing Courses: Opportunities, Challenges, and Lessons Learned. *Conference on Software Engineering Education and Training* (2011).

9. Harrison, N. B. Teaching Software Testing from Two Viewpoints. *Consortium for Computing Sciences in Colleges* (2010).

10. Martinez, A. Using JITT in a Database Course. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (2012).

11. Martinez, A. and Jenkins, M. An Experience Using Reflection in Software Engineering, *American Society for Engineering Education* (2012).

12. McConnell, J. J. Active learning and its use in Computer Science. *Proceedings of the 1st conference on Integrating Technology into Computer Science Education* (1996).

13. McConnell, J. J. Active and Cooperative Learning: Tips and Tricks (Part I). *ACM Special Interest Group on Computer Science Education Bull* (2005).

14. Microsoft Developer Network. http://msdn.microsoft.com/dn308572

15. NIST. The economic impacts of inadequate infrastructure for software testing. http://www.nist.gov/director/planning/upload/report02-3.pdf

16. Novak, G. M., Gavrin, A., and Wolfgang, C. Just-in-Time Teaching: Blending Active Learning with Web Technology. *Prentice Hall PTR*, Upper Saddle River, NJ, USA, 1st edition (1999).

17. Novak, G. and Patterson, E. An Introduction to Just-in-Time Teaching (JiTT). In S. Simkins and M. Maier, editors, *Just in Time Teaching Across the Disciplines*. Stylus Publishing (2009).

18. Simkins, S., Maier, M., and Rhem, J. Just-in-Time Teaching: Across the Disciplines, and Across the Academy. Stylus Publishing, Sterling, VA, USA, 1st edition (2009).

19. Smith, J., Tessler, J., Kramer, E., and Lin, C. Using Peer Review to Teach Software Testing. *International Conference on Computing Education Research* (2012).

20. Talon, B., Leclet, D., Lewandowski, A., and Bourguin, G. Learning Software Testing Using a Collaborative Activities Oriented Platform. *Ninth IEEE International Conference on Advanced Learning Technologies* (2009).

21. Wong, E. Improving the State Of Undergraduate Software Testing Education. *American Society for Engineering Education* (2012).

22. Xie, T., De Halleux, J., Tillmann, N., and Schulte, W. Teaching and Training Developer-Testing Techniques and Tool Support. *ACM SIGPLAN conference on Systems, Programming, Languages and Applications: Software for Humanity* (2010).