Integrating tools and frameworks in undergraduate software engineering curriculum

Christopher Fuhrman, Roger Champagne, Alain April
Department of Software and IT Engineering
ÉTS, University of Québec
Montréal, Canada
{*christopher.fuhrman,roger.champagne,alain.april*}*@etsmtl.ca*

***Abstract*—We share our experience over the last 10 years for finding, deploying and evaluating software engineering (SE) technologies in an undergraduate program at the ÉTS in Montreal, Canada. We identify challenges and propose strategies to integrate technologies into an SE curriculum. We demonstrate how technologies are integrated throughout our program, and provide details of the integration in two specific courses.**

***Keywords*-software engineering curricula; tools; frameworks; technology; integration;**

I. Introduction

Software engineers who have recently graduated are expected to be familiar with technology that is being used in industry. The *Software Engineering 2004 Curriculum guidelines for undergraduate degree programs in software engineering* [1] states: "Engineers use tools to apply processes systematically. Therefore, the choice and use of appropriate tools is key to engineering." Appropriate process tools are part of the essential areas that a curriculum must address. Development frameworks and application programming interfaces (API) are also essential. In this paper, we refer to all these elements as simply "technologies".

The areas where such technologies apply include (but are not limited to) project management and planning, requirements engineering, architecture and design modeling, coding, and testing. Universities that educate and train software engineers therefore have a responsibility to prepare students to be familiar with the technologies that are appropriate in these areas.

Several challenges exist which make fulfilling this responsibility difficult for educators. As mentioned above, these technologies span a wide spectrum of areas, but they are also constantly evolving. How does an educator decide which technologies are appropriate to use in the curriculum? Choosing the technologies is only part of the problem. Educators must also acquire, teach, deploy and maintain these technologies, given the constraints present in academic environments.

For example, some technologies have costly licenses which make them difficult to acquire. Even if an academic license exists or the technology has an open-source license, there may be little documentation about using the technology in an academic environment. On the other hand, many technologies are open-source and popular, and provide educators with excellent resources that facilitate integration into the curriculum.

Faculty are under constant pressure to perform research. The costly investment of time to integrate the newest technologies into undergraduate courses is not always seen as the best return for their academic career. At the same time, some of the technologies that are useful in their research can also be useful in industry and therefore in the curriculum. The expertise with the technologies is often present with the faculty and their graduate students. An important question is how to leverage that expertise in an undergraduate setting.

On the technical side of things, deploying the technologies in a learning environment is complicated by several factors. Laboratory spaces are shared among several courses, and so one physical workstation is time-shared and must run different technologies, sometimes under different operating systems. Technologies are often complex and have their own environmental requirements. It is necessary to test the installed technologies to make sure they work properly in the learning environment. Laboratories have a multiplicity of workstations, whose environments should be identical. Finally, to keep these environments free of viruses, information technology (IT) staff must apply patches to the software regularly (ideally more often than once a semester).

In this paper, we present an experience report of more than 10 years of teaching an undergraduate SE program at the ÉTS in Montreal, Canada. After explaining the background of our undergraduate SE program, we present our strategies for selecting and integrating technologies into the curriculum of this program. With respect to specific challenges we share the strategies that have proven successful to us. Finally, we give some concrete examples of how technologies are integrated into several courses, before we end the paper with discussion and conclusions.

II. Context

The education system in Quebec is slightly different from the typical situation in North America. Students attend secondary school for only five years, followed by a post-secondary institution called a CEGEP, offering college-level programs that either prepare students for entry into

university (with two-year pre-university programs) or train students for technology-oriented careers, the latter being roughly the equivalent of community colleges elsewhere in North America.. Therefore a student attending university would two years in pre-university CEGEP, followed by three or four years of university for an undergraduate level program. However, ÉTS is exceptionnal because its programs are for students who have gone through the three-year CEGEP programs (career/technology). A consequence of this specificity is that all ÉTS students are technicians in a discipline related to their engineering program prior to their admission. In the case of SE, this means the students have three years of programming education and experience prior to their arrival at ÉTS. We are therefore able to teach concepts such as OOD, OOA and design patterns during the first year, and software architecture during the the second year, to name a few examples.

ÉTS is exclusively an engineering school, and is part of the University of Quebec network of universities. The school is relatively young, having been created in 1974. As of October 2011, the total number of active students was approximately 6,300, including close to 5,000 undergraduate students, and a little over 1,300 graduate students. It is the fourth largest engineering faculty in Canada, in terms of the total number of undergraduate students. The school offers undergraduate engineering programs in six disciplines, including SE, which is the focus of this paper.

All engineering programs at ÉTS integrate cooperative (co-op) education. Students must take three co-op terms, which last four months each. The school places 2,400 students in 1,100 companies each year for these co-op terms. The students are paid during their co-op terms, which are important revenue sources for them during their studies, but also makes the relationship one of employee/employer. Because of incoming students' technical background and the co-op nature of our programs, ÉTS is considered a very "hands-on" school, which it is by design and mission. It is also worth noting that according to 2004 statistics, 96% of enterprises in Québec are Small and Medium-Sized Enterprises (SMEs), and 83% of those employ four people or less [2]. This reality must be taken into account in the design of our programs and courses, especially given the fact the Québec's economy includes 25% of Canada's IT firms [2].

All Engineering programs in Canada are subject to accreditation by the Canadian Engineering Accreditation Board (CEAB), a federal organization. Moreover, all the students graduating from our engineering programs are directly eligible to become members of the Quebec Order of Engineers. We mention these points because they have direct impacts on how we design our programs, which are audited every three to six years by the CEAB, and are also audited by a provincial government agency at roughly the same frequency (but not necessarily in sync).

The undergraduate program targeted by this paper is managed by the Software and IT Engineering Department,which currently has 20 full-time faculty (averaging each 10 years industrial experience), and roughly half that number in support staff, all categories considered. There are currently 430 active students in the SE program, and 240 active students in the IT Engineering program, for a grand total of 670 undergraduate students. We mention the IT Engineering program here because the same faculty are responsible for both programs' curricula.

Finally, all undergraduate courses at ÉTS have associated exercise or lab periods. For each course, there are 13 weekly three-hour periods dedicated to the course (typically lectures) which include the mid-term exams, and 12 weekly two-to-three hour practical work periods (typically in a computer lab).

III. Challenges

A. Selecting appropriate technologies

When the undergraduate programs were put in place in 2001, it was the faculty and Teaching Assistants (TAs) who chose the technologies to use in the practical part of the courses the were responsible to create. Often, these technologies were the same ones used in the research activities of the professors, and so they and their graduate students (who often are available to teach courses or laboratories) were also familiar with the same technologies.

Once courses are being taught, however, there is opportunity for improvement via feedback from students and industry. Figure 1 illustrates how this flow of expertise and feedback cycles through the various stakeholders in the education process. Through course evaluations (formal and informal), round-table discussions with students and industry, program accreditation, surveys (formal and informal), the department gathers feedback about the appropriateness of the technologies that are integrated into courses.

Perhaps the most important link with industry comes from feedback via the co-op courses. During each of the three co-op courses at ÉTS, students spend one semester in a company, and that experience allows bi-directional sharing

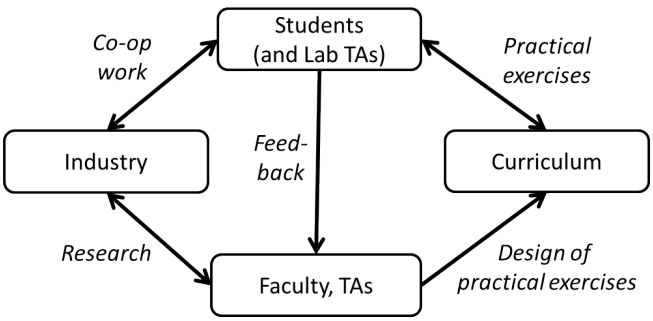


Figure 1. Flow of expertise and feedback for choice of technologies

of information about technologies. In their co-op work, students can suggest or use the technologies they learned about in their courses. In the other direction, students will recommend to faculty (formally or informally) integrating technologies that they used in co-op work. This feedback allows for evolution of courses in an iterative and agile fashion.

As an example, the first-year object-oriented analysis and design (OOAD) course was set-up in 2001 as a C++ programming course using CORBA. By 2003 a group of students had signed a collective petition stating that the technologies and practical exercises in that course were not useful for their work in industry. They used examples from their co-op work to make their point. A departmental decision was made to re-design the practical part of the course, to use Java and Eclipse, as well as the Unified Process. These changes even influenced the methodology taught in the course and even resulted in a change of textbook [3]. The project was transformed into a semester-long development of a web-based application, which is submitted in three iterations for evaluation.

Over the years, this OOAD course has continued to adapt to changing trends in technologies. The latest version of the project integrates Google Maps and Hibernate. The suggestions for changes often come from proactive students or Lab Assistants (LAs,) who are eager to prepare tutorials or "Hello World" examples using these technologies. The changes are approved by faculty in charge of the courses. Informal surveys of first semester students showed an equal proportion of students with backgrounds in .NET and Java. As a result, in the first-year OOAD course they are permitted to code the semester-long project in either .NET or Java, with an understanding that LAs are prepared to support programming questions in Java.

When the SE program began, students were constantly asking for a source-code version-control technology to use in the context of their practical course work. It was difficult to find a general solution to this problem, due to the large number of laboratory exercises it could be applied to. Open-source solutions such as CVS or academic-license solutions such as IBM Rational ClearCase both provide adequate features for use in an academic environment. However, they both require a lot of administrative overhead to manage the creation and sharing of repositories used in the context of any course. There was no technology to easily allow students to create their own repositories and manage sharing permissions. This "problem" was turned into a business opportunity for one of our early graduates, who produced a technology called Academic Version Control (AVC) that is still in use today by many students at ÉTS. AVC has a web-based front end allowing any authenticated student or faculty to create and manage sharing of CVS repositories.

There are other aspects we considered when choosing technologies. A technology that's only backed by a graduate TA or LA is risky to integrate because these people will generally move on after a year or two. Many commercial technologies come with academic site licenses. However, our experience is that SE students want to be able to use technologies on their own computers (to work outside the lab space), and academic licenses do not always permit that. Open-source technologies are usually excellent choices, especially when they have a large community. They are likely to have tutorials or sample programs to flatten the learning curve. Commercial technologies with academic licenses do not always come with good tutorials, in some cases because the vendor charges for training of the technology. Sometimes vendors allow instructors to sit-in for free on training sessions with a local company. However, some companies make trainers sign non-disclosure agreements, because they want to ask questions relating to their intellectual property. In this case, unless the instructor has a relationship with the company, there's no free training.

B. Technical challenges

As mentioned above with respect to choosing a technology, it takes technical expertise to successfully integrate it into the curriculum. It's not necessary to know all the dimensions of a technology (as we discuss in the section below on Learnability). Similarly, undergraduate courses can benefit from some of the more rudimentary features of a technology.

Because of the large variety of technologies and the important number of courses that use them, our department opted for configuring many of the labs with either multi-boot hard drives or virtual machine capabilities. Several of the workstations can boot Windows or Linux, or the Mac workstations can run virtual machines with Windows. Obviously these workstations are more complex to configure and replicate, but the benefit is that a single seat can be used for multiple courses. One disadvantage to this approach is slightly less availability of lab spaces outside of reserved lab periods. To facilitate configuring the machines, *ghosting* of hard drive images is used as much as possible. IT staff validate the image of a configuration, and then it can be replicated to all the destination machines overnight. This approach also helps when security updates need to be applied usually once a month.

Having IT support staff who are knowledgeable and motivated despite the complexity and wide range of the technologies is important. IT support in an academic environment is already a challenge, let alone trying to integrate these technologies into many undergraduate classes. IT staff who are talented often enjoy challenges, and it's been our experience that these kinds of challenges are positive. We discuss turnover of IT staff in the section on Political challenges.

IT support staff do not provide much integration of the technologies into a course. For that, it's back to the

expertise of the instructor or the LAs, which can mean a lot of work. A strategy that works well at our university is that the administration funds project proposals that seek to integrate new technologies in to courses (or provide updates to existing ones). A typical project is to pay a LA who's familiar with the course to integrate/update some technology into a set of laboratory exercises. If the scope of the work is large enough (e.g., development of a software technology to be used by students), it can even be considered a co-op work term for the student. In this case, the professor in charge of the course becomes an industrial supervisor.

C. Learnability challenges

Many technologies are complex and have voluminous (or in some cases, poor) documentation. It is neither interesting nor feasible to have students learn all the functions of a given technology in the context of a lab exercise. Therefore, technology learnability [4] is a challenge because it has an impact on the amount of time needed to accomplish the learning goals of the lab sessions.

To overcome this problem, instructors first decide which functions are applicable, and then they can provide task-focused tutorials that allow students to become familiar with only the important functions of the technology in the context of the lab exercises. A tutorial can be in the form of a simple web page (including screen shots) documenting the steps to follow to perform a task with a specific technology, or it can be a video created using a free video capture tool that can be later annotated on Youtube. Sometimes such tutorials already exist for popular open-source technologies. Students can be asked to familiarize themselves with the tutorials before the lab session, allowing instructors to focus more on the exercise. This strategy is analogous to a student preparing for discussions in a course by having already done the readings.

Apart from streamlining the laboratory exercises, tutorials serve as a simple functional test of the installation of the technologies. IT staff can follow the steps in the tutorials (as if they were students) to verify that a technology has been properly installed in the lab environment. This is useful to avoid unpleasant surprises on the first day of the lab sessions.

D. Political challenges

At many universities with tenure-track positions, a professor is implicitly (or sometimes explicitly) encouraged to invest most energy in research activities. Strategies that help reduce the risk of research faculty spending too much time on integrating technology into undergraduate programs include organizing capstone projects that have pedagogical and technical goals, having a budget to hire undergraduate or graduate students who can perform the integration or updates (clever LAs are great candidates), creating a positive team environment for LAs among several courses where they "own" the lab assignments, etc.

Another political challenge relates to evolutions in IT support. Traditionally, computer science departments kept their own IT support staff who were specialists in the areas of technology needed in the department. However, with the distillation of IT into everyday life, academic institutions may push to have IT support resources spread across more than one academic department. Our experience is that some human resources can be shared, but it takes some who are dedicated to the department if a level of quality is to be expected when technology integration is high.

This leads to the challenge of turnover in IT staff. Qualified IT staff are always at risk of leaving, perhaps because salaries or conditions are better elsewhere. The strategy we propose is to accept that IT staff turnover is a reality. Much like students, talented IT staff in academic environments will not be there forever. So it's an ongoing necessity to identify, hire and train young talent to maintain the quality. Although this is a difficult challenge, we found that by integrating modern technologies into curriculum, the talented people are motivated and interested by those same challenges, despite the pressures of the first week of courses.

Finally, integrating cloud computing into academia raises a political issue, at least in some universities where there are rules or laws about where student data is stored. Yale University decided to postpone switching its mail services to Gmail [5], stating concerns about the vagaries of laws and governments, given that every piece of data is stored in three random data centers that could be in many places throughout the world. We mention this point because it extends to other features of Google, such as Google Apps, which look promising to integrate into curriculum. Many students already collaborate on team projects in the cloud with these technologies, but using their own private Google accounts. It is not clear what legal implications there are, if an institution would force students to use these technologies by integrating them into a curriculum.

E. Maintainability, reuse and deployment challenges

SE educators have limited/finite resources. In an effort to maximize efficiency when preparing "courseware" (examples, tutorials, assignments,...), we propose three important aspects:

- the ability to reuse such artifacts across multiple courses (for instance, examples or tutorials introducing a technology used in multiple courses);
- the effort required to maintain these artifacts over time;
- the effort required to deploy these artifacts so students gain access to them.

From the reuse point of view, course material is very similar to software components, e.g. finding the right "granularity" for a reusable tutorial or example requires careful consideration, which requires more effort than developing a "one-shot" version of the same artifact. If an example is to be reused in a mandatory first year course and also in a

senior elective or even a graduate course, it must indeed be carefully designed and implemented. However, a reusable course material artifact should in turn contribute to reduced effort when it is reused. The notions of coupling applies here also. Each artifact should be as loosely coupled to others as possible, in order to improve potential reuse and maintainability.

Maintainability of courseware is especially important when the courses discuss or use concepts or technology that evolves rapidly, which is certainly the case in a lot of SE courses. From this perspective, course materials also exhibit similar characteristics to software. Once an educator commits to a certain technology, changing it becomes harder as time goes by, because of the time invested in using it and developing other artifacts (examples, tutorials) around it. Selecting a specific technology then requires careful consideration and outlook. It can be very frustrating to invest substantial effort over long periods of time to support a specific technology in courses and have that technology suddenly disappear or become "unsupported".

One would be tempted to think that with modern means (web sites, wikis, ...), deploying course materials is not an issue. One of the challenges we run into is deploying materials that are reused across courses. Our web server is structured in a "one site (root) per course" fashion, for all kinds of valid reasons (access control, simplicity). However, in such a structure, examples, tutorials and simple web pages that are reused in multiple courses need to be copied in many places. with the associated potential problems: the need to copy any update to such material in multiple places, the problems caused by forgetting one of these places, etc.

IV. Overview of technology integration in undergraduate SE curriculum

In this section we present some of the technologies and where they are integrated into the curriculum. Figure 2 shows how various types of technologies are integrated into courses. The courses are presented in the table in chronological order relative to the SE undergraduate program, showing that students are exposed to technologies used in industry as early as the first year. We included the special project and capstone projects that show almost any technology is optional, since those projects are individually decided and managed. They share a common element of a mandatory project management technology, however. The list of technologies we include is not complete (because of space constraints), and it should not be construed as a promotion of those technologies or their vendors.

We distinguish between technologies that are optional or mandatory in the context of a course. For example, in the OO Analysis and Design course, students develop a web-based application. The simplest solutions can be done with Java in Eclipse using Derby for a database. Some students prefer to be more ambitious or use other programming languages (e.g., C# in .NET). However, all projects must show unit tests using JUnit.

The following section describes in more detail how these technologies are integrated into specific courses.

V. Examples of technology integration in courses

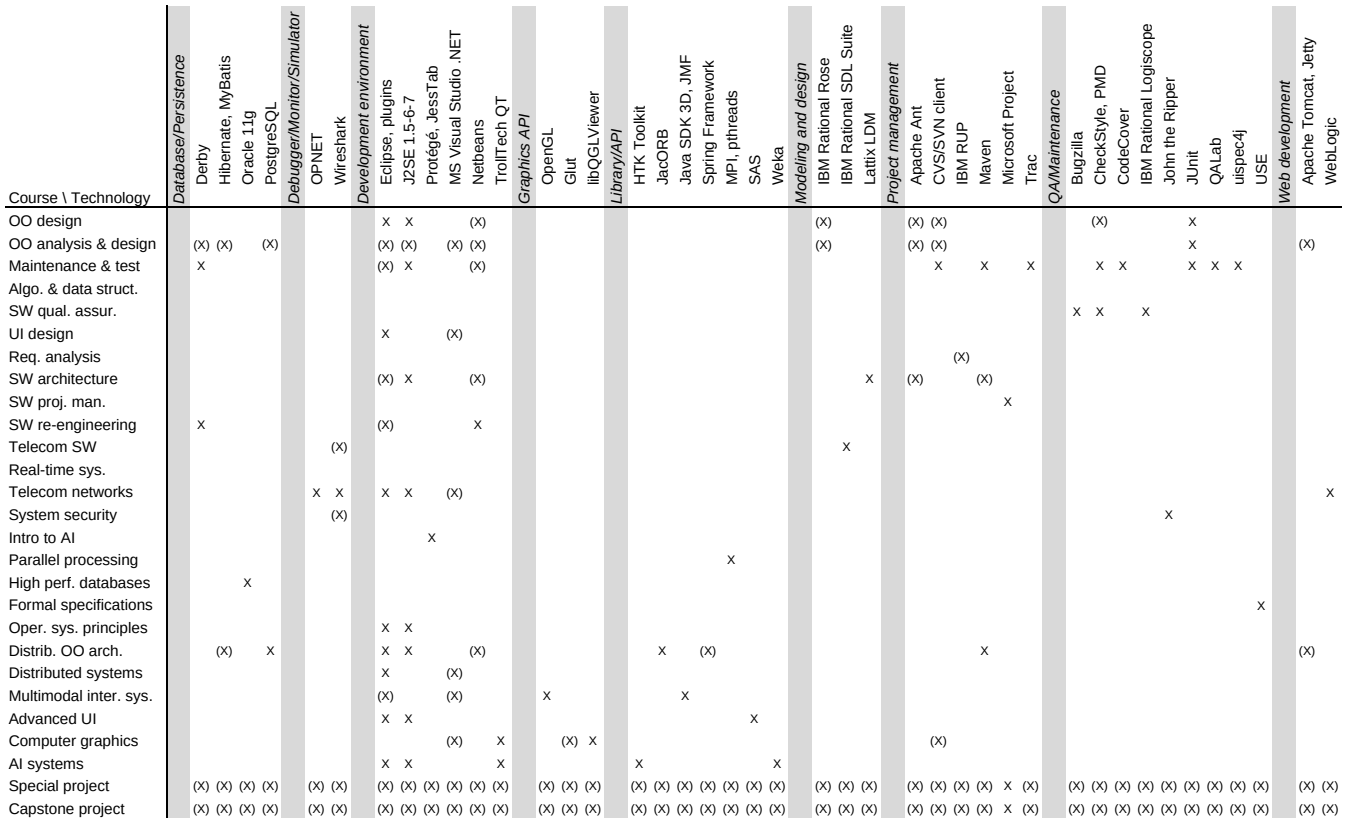
This section describes the technologies integrated in two courses in our SE program. In previous sections we have mentioned other courses as brief examples, but here we provide details. The first example is a second-year course titled "Test and Maintenance", and the second is a senior elective titled "Distributed Object-Oriented Architecture". In both cases, exposing students to realistic industrial environments is a high-priority objective. Moreover, given the fact that a majority of our graduates are likely to work in a small business environment, students are forced to perform some relatively "low-level" tasks, such as creating accounts on servers and configuring their environments themselves, because they likely won't have anyone to do it for them. This requires extra guidance and knowledge from the LAs and professors, but we concentrate on simple things we are comfortable with.

The two courses were selected for description here because of their relatively high technology content, and also because each illustrates one of the two main deployment strategies we adopted for our lab environments. In the first strategy, a virtual machine (VM) holding the main technologies required is deployed for each team of 3-4 students, and some technologies are installed on the individual workstations in the computer labs. In the other strategy, there is a single central server used by the entire group, and almost all the technologies they need are deployed on the individual workstations.

A. Test and maintenance course

This course was first offered during the Summer 2008 term. Ideally, a full course would have been created on each of the main topics, but we could only add one course. The two halves were developed by two different professors, hence the two topics are discussed sequentially. The course is offered twice a year to groups ranging from 50 to 70 students. The pedagogical value of the strategy adopted in the lab assignments for this course was the focus of another paper presented elsewhere [6].

During the first half of the term, the practical objective is to expose students to a mature software maintenance environment. Students are given a relatively small application that has a user interface and uses a database. Its size is manageable (six Java classes, approx. 2,100 lines of code) and it has the potential to highlight the salient issues in performing maintenance on an application developed by someone else. It has several problems (poor separation of concerns, bugs, no documentation). The assumption is that an undisciplined



Course \ Technology	*Database/Persistence*	Derby	Hibernate, MyBatis	Oracle 11g	PostgreSQL	*Debugger/Monitor/Simulator*	OPNET	Wireshark	*Development environment*	Eclipse, plugins	J2SE 1.5-6-7	Protégé, JessTab	MS Visual Studio .NET	Netbeans	TrollTech QT	*Graphics API*	OpenGL	Glut	libQGLViewer	*Library/API*	HTK Toolkit	JacORB	Java SDK 3D, JMF	Spring Framework	MPI, pthreads	SAS	Weka	*Modeling and design*	IBM Rational Rose	IBM Rational SDL Suite	Lattix LDM	*Project management*	Apache Ant	CVS/SVN client	IBM RUP	Maven	Microsoft Project	Trac	*QA/Maintenance*	Bugzilla	CheckStyle, PMD	CodeCover	IBM Rational Logiscope	John the Ripper	JUnit	QALab	uispec4j	USE	*Web development*	Apache Tomcat, Jetty	WebLogic
OO design										x	x			(X)															(X)				(X)	(X)						(X)				x							
OO analysis & design		(X)	(X)		(X)					(X)	(X)		(X)	(X)															(X)				(X)	(X)										x					(X)		
Maintenance & test		x								(X)	x			(X)																			x							x	x			x	x	x				
Algo. & data struct.																																																		
SW qual. assur.																																								x	x		x							
UI design										x			(X)																																					
Req. analysis																																			(X)															
SW architecture										(X)	x			(X)																x			(X)			(X)														
SW proj. man.																																						x												
SW re-engineering		x								(X)				x																x																				
Telecom SW								(X)																																										
Real-time sys.																																																	
Telecom networks							x	x		x	x		(X)																																				x
System security								(X)																																					x						
Intro to AI												x																																					
Parallel processing																									x																								
High perf. databases				x																																													
Formal specifications																																															x		
Oper. sys. principles										x	x																																						
Distrib. OO arch.			(X)		x					x	x			(X)								x		(X)													x											(X)	
Distributed systems										x			(X)																																				
Multimodal inter. sys.										(X)			(X)				x						x																										
Advanced UI										x	x															x																							
Computer graphics													(X)		x			(X)	x																(X)														
AI systems										x	x				x						x						x																						
Special project		(X)	(X)	(X)	(X)		(X)	(X)		(X)	(X)	(X)	(X)	(X)	(X)		(X)	(X)	(X)		(X)	(X)	(X)	(X)	(X)	(X)	(X)		(X)	(X)	(X)		(X)	(X)	(X)	(X)	x	(X)		(X)	(X)	(X)	(X)	(X)	(X)	(X)	(X)		(X)	(X)
Capstone project		(X)	(X)	(X)	(X)		(X)	(X)		(X)	(X)	(X)	(X)	(X)	(X)		(X)	(X)	(X)		(X)	(X)	(X)	(X)	(X)	(X)	(X)		(X)	(X)	(X)		(X)	(X)	(X)	(X)	x	(X)		(X)	(X)	(X)	(X)	(X)	(X)	(X)	(X)		(X)	(X)

Figure 2. Courses (in chronological order) with integrated technology: X=mandatory technology, (X)=optional technology

developer left the company with no documentation, and all that is available is the source code for the application. According to our experience, this is a realistic situation in low-maturity SMEs. The actual maintenance work they must realize is described in Table II.

Following the maintenance part of the course, the focus switches to software testing. The overall objective of this portion of the lab is for students to acquire minimal experience with basic black- and white-box testing techniques, and also with basic testing frameworks, by designing, implementing, executing, and reporting test results, at various levels (unit, integration, system).

This course uses a fairly rich set of technologies. When we selected them, our goal was basically to use at least version-control and ticket management systems. We also wanted to use analysis technologies to give various indicators on code quality, such as violations of programming standards. As previously stated, the students must perform system administration tasks, by design. At the time we were selecting the lab environment for this course, there was a trend towards virtualization in our department. We consequently decided to supply each team (3-4 students) with a Linux-based virtual machine (VM) hosted in a server room and containing some of the software they need pre-installed, but not completely configured. Table I summarizes the technologies used for this course, identifying those installed on the team VMs, on individual workstations in the lab, and in some cases in both places.

Table I
TECHNOLOGIES: TEST AND MAINTENANCE COURSE

Workstation	VM
Eclipse, CodeCover & Metrics plugins	SVN (version-control)
Apache ant (build technology)	Trac (ticket management)
Checkstyle, PMD, QAlab (code quality)	Apache httpd (web server)
Visual Paradigm (UML from code)	
Junit, uispec4j (testing frameworks)	

There are five lab assignments, summarized in Table II ("M" = maintenance, "T" = testing).

In this specific case, there were initially few *technical challenges*. The professors developing the initial version of the course and lab assignments had experience with some of the technologies, were willing to invest effort to learn others, and delegated part of the work (VM development and configuration) to a senior student in the context of his capstone project. This student had prepared a similar VM for a student club. It took the student 25 hours of effort to prepare the VM. IT support staff took care of deploying the configured VMs on the institutional servers. The choice of

Table II
LAB ASSIGNMENTS: TEST AND MAINTENANCE COURSE

Description and tasks	Weeks
M1 - Environment setup: test VM, finish configuring Trac and SVN, create accounts for team members and customer, create roles and milestones, install application source code in SVN, generate quality analysis reports, create set of tickets according to findings.	2
M2 - Reverse engineering and refactoring: produce likely design (UML diagrams), refactor code to improve maintainability, start fixing defects according to tickets raised, document all changes in tickets	2
M3 - Add new functionality: estimate effort for changes, obtain "customer" approval, perform changes on dedicated SVN branch	2
T1 - Black-box testing: system level automated functional testing via the GUI with uispec4j	3
T2 - Test-Driven re-engineering and white-box testing: incremental layering of architecture, introducing unit and integration tests	3

LAs was important here, and people who had used most of the technologies were selected in order to offer students adequate support in the lab. One of the technical challenges we did not foresee was this environment's scalability. Initially, this course was the only one to use VMs. However, as other courses adopted the VM approach, there was a problem with hardware resources to accommodate this popular approach. To solve this problem we did two things: 1) added memory to the servers, and 2) changed from the freeware VMs to the Enterprise version (VMware Academic Program) whose performance is better. It took a competent analyst roughly two days of effort to configure the VMs for this course with the new Enterprise virtualization package, but the analyst pointed out that it took him four to six weeks of full time effort to learn how to configure the Enterprise version of VMware for another course (with good prior knowledge of how to configure the freeware version).

The *learnability challenges* were important in this case. A minority of students had been exposed to the technologies prior to taking this course. Trac and SVN are two examples of technologies that have a lot of documentation available, to a point it's intimidating for a novice. We mitigated this by carefully looking at "mainstream" documentation for these technologies, and provided students with well targeted pointers to specific artifacts in the lab assignment descriptions. We did not supply any tutorials for this course, but designed the lab assignments in such a way that students would learn by actually performing targeted tasks. A simple example was designed for use in class and as an example in the lab, to illustrate usage of the various eclipse plugins in the testing portion of the course.

There were no *political challenges* to speak of in this case. Apart from the scalability problems mentioned above, and the normal updates of technologies that occurred in the last years, there were no *maintainability, reuse or deployment* problems either.

Upgrading the environment last summer required important effort from the professors and LAs also, mainly because we decided to start using Maven in this course. Maven has a relatively important learning curve, and everyone was learning Maven during the upgrade. The core Maven functionality is quite limited, and a multitude of plugins must be used, each having its usage and configuration specificities. Therefore, a course-specific Maven tutorial was written, in such a way that students start with the source code and completely configured environment, and progressively write their Project Object Model (POM, main Maven build script) by copying and pasting sections from the tutorial. The main Maven concepts are explained incrementally throughout this tutorial. It was successfully used by a group of 70 students and was globally considered a positive addition to the course. We have no official feedback on student appreciation of Maven yet, as course evaluations will be performed at the end of the current term. Table III summarizes the effort put into upgrading the lab environment and assignments this term. More effort will be required later this term, as the testing portion of the course is about to start as of this writing.

Table III
EFFORT: LABS UPGRADE, TEST AND MAINTENANCE COURSE

Who	Effort (hours)	Description
IT support	160-240	Learning to configure Enterprise VMware (not specific to this course, but required)
IT support	12	Configuring and deploying the new VMs for 70 students (26 teams)
Professors	65	Recruiting LAs, coordinating the whole effort, learning maven, writing a course-specific tutorial on Maven, validating the new lab assignments produced by the LA, evaluating new candidate applications for the assignments
LA	80	Learning maven, configuring and testing all the plugins, testing new versions of applications on the VM, writing new versions of the assignments, validating tutorials produced by the profs, training two other LAs

B. Distributed Objected-Oriented Architecture course

This course is a senior-level elective. It was first offered in 2003, and is since then offered once or twice a year to groups of 30 to 55 students. The core topic of the course is middleware, and the main objective is to expose students to two important types of architectures in distributed systems, namely distributed objects (e.g. CORBA) and server-side component architectures (JEE).

In this course, the lab environment is deployed differently than the course described in the previous section. There is a single central VM used by the whole group, hosting a database server. All the other technologies required for the lab assignments are deployed on individual computers

(windows-based PCs) in the department's computer labs. The environment is 100% Open Source, allowing students to reproduce it on their own computers. Moreover, the shared VM is made available outside the university through a VPN connection, enabling students to access the servers from anywhere. This is an important aspect for students, allowing them to work on their lab assignments outside their assigned lab period, even if the labs are used by other courses.

This course also uses multiple technologies. The selection criteria was somewhat different in this case. The CORBA implementation that came with Sun's Java SDK was used initially. We later switched to JacORB, mainly because it was a more complete CORBA implementation, allowing us to cover a broader subset of CORBA concepts in the course. For the JEE part, when the course was initially created, the professor responsible for it had no experience in JEE, and relied on experience from colleagues from elsewhere who suggested a fully open-source set of technologies. Table IV summarizes the technologies currently used for this course that are installed on individual workstations.

Table IV
TECHNOLOGIES: DISTRIBUTED OO ARCHITECTURE COURSE

Technology
Eclipse IDE JEE edition with m2e and m2e-wtp integration plugins
JacORB (Java CORBA implementation)
Apache Maven (project management)
Apache Tomcat (JEE web container)
PostgreSQL clients (psql, pgadmin)
Hibernate, MyBatis (persistence)
Spring framework (JEE development)

There are three lab assignments in this course, each lasting four weeks. Table V summarizes these assignments.

Table V
LAB ASSIGNMENTS FOR DISTRIBUTED OO ARCHITECTURE COURSE

Description and tasks
1 - Introduction to distributed objects with CORBA: develop a simple distributed application on at least two hosts, given the interface specifications (IDL) for all remotely-accessible components.
2 - Introduction to JEE: develop a subset of functionalities from lab1 in JEE using servlets and JSP only (no EJB) and a relational database.
3 - JEE and frameworks: re-implement the same application as in lab2, introduce at least one web framework and at least one persistence framework. Framework choices left to students.

This course had numerous *technical challenges* over the years. Initially, we used Sun's reference implementation (J2EE 1.3) as our J2EE environment. The professor, who also served as LA, had no experience with this technology. Luckily, the IT support technician at the time did; he had setup everything, and it just worked. Nowadays, the professor needs to give IT support detailed instructions on which technologies to install and how. This is done via a dedicated page published by the professor on the course web site, which is used by IT support, the students who wish to install the technologies on their own computers, and the professor himself, who uses this as a reminder for himself from term to term. This issue highlights the fact that it is hard to find IT support personnel that know ALL the technologies required in a SE program.

One of the main technical challenges with this course's technology is that some of these technologies evolve rapidly (IDE, plugins). In the past two years, the professor spent between 20 and 40 hours each time the course is offered just to upgrade all the technologies and test them together. The course examples discussed below are especially useful to test the environment from term to term.

IT support recently developed a technique to remotely add individual Eclipse plugins (to an already deployed Eclipse distribution) on all the lab computers without requiring a complete re-imaging of the whole disk. This new capability proved very useful when we discovered we were missing a crucial plugin in the labs, four weeks after the term had started. Until recently, it was department policy that no changes to the lab technologies could be performed during the term.

Since this course introduces a number of complex technologies, *learnability challenges* were addressed by investing an important effort (over many years) to supply students with small running examples, each consisting in a small application highlighting a specific aspect of a technology. We currently use 11 such examples (2 for CORBA, 1 for database access, 4 for "basic" JEE, 4 for JEE-related frameworks). Most of the examples were initially taken "as is" from the various technology distributions used or on the Web. Some were modified, and all were extended with an Ant build script and instructions explaining the core concepts the example focuses on, and how to deploy it. These examples are actually being reused by colleagues in other courses.

To improve learnability, each lab assignment starts by "forcing" students (a small portion of the grade is assigned to this) to follow a tutorial (based on one of our examples mentioned above). Each of these tutorials takes a full lab period (two hours) to complete. We get sustained positive feedback on these tutorials in the course evaluations. They also enable us to confirm that the technologies are correctly deployed, the students have correctly configured their environment, and they are ready to start the "real assignment" in a timely fashion (they must demonstrate their running tutorial results the 2nd week of each lab assignment).

We recently introduced Maven in this course also. This yielded two important advantages, compared to Ant (which we were using before): 1) automatic resolution of external dependencies (e.g. required external libraries) at compile/run time; 2) deploying the examples on the course web site simply meant creating a ZIP archive containing the source code, the Project Object Model (Maven script), and the

README.html file (no external JARs). However, we needed to learn about another set of Maven plugins we hadn't needed in the other course (IDL compiler, JEE container deployment, database access). So far, we replaced the Ant build script by its Maven equivalent for 7 of our 11 examples, learning maven in the process. This took 42 person-hours this term alone (this effort is specific to this course). Since this course uses a simpler environment than the Test and maintenance course, the Maven tutorial developed for the other course was simplified for this course. This took 16 person-hours. Also, utility scripts and sample configuration files were developed to ease the students' learning of Maven and its use in the school's infrastructure. The fact that student accounts reside on network drives required some adjustments compared to using a local drive, since by default some applications (such as Maven) store useful files in the user's local account. Altogether, we have so far spent (this term alone) 18 person-hours in preparing the lab environment (setting up technologies, writing sample configuration files and related instructions, testing technology deployment in the labs). This does not include IT support effort to deploy the tools in the labs.

There were no *political challenges* to speak of in this case either. One *maintainability* issue we regularly ran into in the past was keeping the various tutorials up-to-date. This was complicated by the fact that we used to include screen shots from the IDE, which basically changes every time the course is offered. One practical way to reduce this problem is to refer to menu item sequences instead of screen shots. We find this easier to maintain over time. In terms of *deployment*, adding Maven to the technology set greatly facilitated deployment at two levels: the examples supplied to students via the course web site, described earlier, and also the submission of lab assignments to the LA. In both cases, external libraries need not be included, since they are resolved and downloaded by Maven at compile time. For the lab assignments, students are forced to use Maven, which imposes a standard project (folder) structure. Compiling, running and grading the assignments is much easier, since all their submissions have the same structure.

VI. Related work

Different proposed curricula in SE have been studied by Mishra *et al.* [7], who recognize the importance for technologies. They suggest that it is in the practice courses (e.g., capstone projects) where the students discover and explore new technologies and development methodologies. In our approach, technologies are integrated into the majority of SE courses, although not to the same extent. Rusu *et al.* [8] show that having faculty with strong ties to industry who supervise capstone (senior year) projects is an effective way to improve student implication in real-world SE projects.

Toth points out [9] that SE programs must go beyond teaching just theory and increase on practical aspects. He proposes using open-source projects as the basis for capstone projects. Although this approach is excellent for capstone projects, in curriculum where co-op work is done as early as the first year, practical work must be introduced earlier if students are to be ready for their industrial contact. Boloix *et al.* discuss tool learnability [4], which cannot be ignored especially with some of today's very sophisticated technologies.

Chen *et al.* [10] consider the benefits of cooperative education (co-ops) where students spend time in industry working on real-world software projects. The authors stress that not only do students gain useful work experience that relates back to their theoretical work, but it also improves relations with industry. Reichlmay [11] documents the advantages of cooperative education, but also discusses the benefits in terms of faculty and research collaboration. Since students and faculty are the forces of change who are closest to a curriculum, it makes sense that they be connected to industry.

In terms of specific tools, Skevoulis and Makarov [12] conclude that by using appropriate formal methods tools in courses, students learn from them and have better attitudes towards formal methods. In [13], Fuhrman demonstrated that a tool supporting lightweight modeling and formal methods can be used to help interpret informal (real-world) specifications in an undergraduate telecommunications software course (see Figure 2). This course has since been updated with technologies supporting the Specification and Description Language (SDL) that provide lightweight model-checking.

For integration of version-control technologies into curriculum, Parada *et al.* [14] propose a solution using a popular source-code control technology (Subversion). They provide an interface to instructors to create workspaces for students. As far as we can tell from the paper, this technology is not designed to allow students to create their own workspaces. Milentijevic *et al.* [15] consider the benefits of using version-control in project-based learning. Their solution documents two roles, supervisor and student, and it's the supervisor who must define the project.

VII. Conclusion and future work

Many different tools and frameworks are used in practice in SE. In an ideal world, all students would graduate with a good understanding and knowhow related to the use of many of theses tools. In this paper, we discussed our experience in integrating tools in various SE courses. After describing our context, we described various types of challenges faced by the many stakeholders involved in such technology integration in curricula. These challenges are the main ones we have been facing for 10 years in our SE program. We highlighted the numerous aspects to consider when choosing tools and identified the main technical, learnability, and political challenges. We also discussed issues such as course material

maintainability, reuse and deployment. We gave an overview of the numerous tools deployed in our SE program, and described in some details the specific challenges we faced when integrating tools in two courses using rich technology sets, including recently added technology (Maven, VMware upgrade) and the associated effort required by the various stakeholders involved. The effort data reported here could perhaps be considered an upper bound, as the professors involved invest more time than average in the preparation of their lab environments.

One reality that strikes us term after term is that our students collectively know a lot about technologies, and they like working with technologies. They are a valuable source of information about the latest technologies, and in terms of knowledge, they will always outnumber the few educators teaching any given course and labs. Open and honest discussion with students about tools should be encouraged in class, via course-related discussion groups, and their opinions should be carefully considered by educators. In our context, the fact that students take three co-op courses adds even more feedback of this nature. These co-op courses in turn force us to insist on tools throughout the program, including in the early stages, so that students can bring some value to the organizations that hire them for their co-op terms.

We are convinced that pre-existing industrial experience of faculty, TAs, and IT support staff is essential with technology integration in curricula. Another helpful aspect is the existence of resources (a budget) to maintain and evolve laboratory environments. The LAs assisting us in the upgrades described here were paid for their work by the School. The professors and IT support staff's time is considered part of their normal work load.

Another aspect to keep in mind is that most of our graduates will work locally after they finish school. Our entire program strategy is very much aligned with our local context, and we assume this would be true for many universities.

In terms of future work, we plan to monitor technology usage in the labs. This would enable the various stakeholders (educators, IT support, department management) to get data on the usage of the various tools, which in turn would help identify tools that should no longer be maintained. This is likely to simplify the maintenance of the environment on these workstations.

ACKNOWLEDGMENT

The authors would like to thank Patrice Dion, Pierre Dumouchel, Francois Coallier, Lucie Caron, Sandra Ranger, Christian Desrosiers, Luc Duong, Nadjia Kara and Carlos-Teodoro Monsalve, who supplied information that was used in this paper. We also wish to thank the anonymous reviewers, whose comments allowed us to improve this paper.

REFERENCES

[1] IEEE Computer Society, "Software Engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering," http://sites.computer.org/ccse/, Aug. 2004, date accessed Oct. 20, 2011.

[2] Government of Canada - SME Financing Data Initiative, "Small business financing profiles," http://www.sme-fdi.gc.ca/eic/site/sme_fdi-prf_pme.nsf/eng/h_02088.html, date accessed Oct. 22, 2011.

[3] C. Larman, *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*, 3rd ed. Upper Saddle River, N.J.: Prentice Hall PTR, 2005.

[4] G. Boloix and P. N. Robillard, "CASE tool learnability in a software engineering course," *IEEE Transactions on Education*, vol. 41, no. 3, pp. 185–193, Aug. 1998.

[5] D. Tidmarsh, "ITS delays switch to Gmail," http://www.yaledailynews.com/news/2010/mar/30/its-delays-switch-to-gmail/, Mar. 2010, date accessed Oct. 20, 2011.

[6] R. Dupuis, R. Champagne, A. April, and N. Seguin, "Experiments of adding to the experience that can be acquired from a software project course," in *QUATIC '10*, Sep. 2010.

[7] A. Mishra, N. Cagiltay, and O. Kilic, "Software engineering education: some important dimensions," *European Journal of Engineering Education*, vol. 32, no. 3, pp. 349 – 61, 2007.

[8] A. Rusu and M. Swenson, "An industry-academia team-teaching case study for software engineering capstone courses," in *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual*, Oct. 2008, pp. F4C–18 –F4C–23.

[9] K. Toth, "Experiences with open source software engineering tools," *IEEE Software*, vol. 23, no. 6, pp. 44–52, Dec. 2006.

[10] J. Chen, H. Lu, L. An, and Y. Zhou, "Exploring teaching methods in software engineering education," in *4th International Conference on Computer Science Education*, Jul 2009, pp. 1733 –1738.

[11] T. J. Reichlmay, "Collaborating with industry: strategies for an undergraduate software engineering program," in *SSEE'06*. New York: ACM, 2006, pp. 13–16.

[12] S. Skevoulis and V. Makarov, "Integrating formal methods tools into undergraduate computer science curriculum," in *Frontiers in Education Conference, 36th Annual*, Oct. 2006, pp. 1 –6.

[13] C. P. Fuhrman, "Lightweight models for interpreting informal specifications," *Requirements Engineering*, vol. 8, no. 4, pp. 206–221, 2003.

[14] G. H. A. Parada, A. Pardo, and C. D. Kloos, "Towards combining individual and collaborative work spaces under a unified e-portfolio," in *ICCSA'11*, 2011, pp. 488–501.

[15] I. Milentijevic, V. Ciric, and O. Vojinovic, "Version control in project-based learning," *Comput. Educ.*, vol. 50, pp. 1331–1338, May 2008.