INCORPORATING OPEN-SOURCE SOFTWARE DEVELOPMENT INTO COMPUTER SCIENCE AND SOFTWARE ENGINEERING EDUCATION AT UNIVERSITY LEVEL

Linh N. Vu [1], Ti Kean Tan [1] & Prapat Maneerat [1]

[1] Department of Computer Science and Software Engineering
Faculty of Engineering
The University of Melbourne

ABSTRACT: Recently, open source software development (OSSD) has established itself to be an efficient and successful methodology to produce high quality software. However, it has not been incorporated into computer science and software engineering (CSSE) education at Australian universities, despite OSS products being widely used in these academies. Our aim is to show how the educational model derived from open source methods for software programming can help to develop more well-rounded CSSE professionals compared to the conventional proprietary educational practices in Australian CSSE courses. To achieve this aim, we firstly investigate the expectations the industry has for CSSE graduates. Secondly, we review the existing proprietary educational model to examine its problems and raise ethical concerns. Thirdly, we conduct a case study of final year CSSE students at the University of Melbourne. Finally, we introduce open source software development and its derived educational model. We compare the two models to show the benefits the open source alternative can offer to CSSE education in Australian universities. We conclude by providing justifications for the implementation of this educational model.

INTRODUCTION

OSS, as defined by the Open Source Initiative, is ”software that must be distributed under a license that guarantees the right to read, redistribute, modify and use the software freely” [Ini04]. The significance of the OSS development model to CSSE professionals is that they can freely learn from the open source code, modify and reuse it, and participate in globally collaborative projects.

In CSSE education at university level, OSS development practices are conducive for learning because of its educational nature. Firstly, it promotes programming knowledge sharing, which is the root of education. Secondly, it provides students with practical experience in large scaled collaborative projects. Thirdly, it supports risk-taking, rewarding and building from failure. Lastly, it encourages releasing early and often, obtaining frequent constructive peer reviews and user feedbacks, which leads to a more fruitful and self-assured education.

However, despite the aforementioned benefits, being a relatively new methodology, the OSS development model has not been widely accepted in Australian universities. Therefore, in this paper, we aim to show that the educational model derived from this development methodology can develop more well-rounded CSSE graduates compared to the current proprietary one.

Firstly, we will investigate the expectations the IT industry has for CSSE graduates. This is essential to determine the skills CSSE graduates need, not just to satisfy the needs of future employers, but also to fulfill their duty to lifelong learning as professionals, outlined in the IEEE-CS/ACM Software Engineering Code of Ethics and Professional Practices (SECEPP)[T.04].

Secondly, we will review the existing proprietary educational model in Australian universities to see whether it satisfies the above expectations. We will raise problems and ethical concerns regarding its implementation.

Thirdly, we will conduct a case study of final year CSSE students at the University of Melbourne, who are the would-be product of this educational model, to show some examples of said problems.

Finally, we will introduce the open source educational model as an alternative, by comparing these

two with regards to the issues raised from our case-study interviews. We will show that the proposed model will provide students with higher quality training for the workplace via its tight integration with collaborative projects, as well as benefit the CSSE community and the general public. We will also discuss the possible obstacles to a successful implementation of our model.

INDUSTRY EXPECTATIONS FOR CSSE GRADUATES

A report by the IT&T Skills Task Force in June 1999 indicated a strong growth in most areas in the ICT industry, and an anticipated difficulty in finding enough suitably qualified graduates in the future [D.04]. As such, there has been many government initiatives aimed at producing enough graduates to satisfy the demands of the industry.

However, despite a dramatic increase in the number of ICT graduates in recent years, employers continue to complain of skill shortages. The Australian Computer Society's 2003 Remuneration Survey shows a continuing decline for ICT skills, with many ICT professionals losing their jobs [D.04].

Why has the increased number of ICT graduates Report of the Discipline Review of Computing Studies and Information Sciences Education not satisfy the demands of the industry? Have the skills of ICT professionals become obsolete? Are there other highly sought after skills that employers look for, but cannot find in recent ICT graduates? In other words, are university courses providing the necessary set of skills that employers seek in graduates?

Generally, employers do not complain about the lack of technical skills. This means the universities have done an excellent job in that respect. However, employers do point out some non-technical attributes that they wished were available in graduates. These attributes are presented in the following.

Oral communication skills

Employers mentioned oral communication skills first and foremost. The myth of the lone programmer is long gone. ICT professionals need to communicate with management, co-workers and clients. Real world jobs require professionals to apply specialized leading-edge technology to solve problems as well as the ability to work as a member of a team [JJ03]. Studies have shown that software development is not an isolated activity. ICT professionals spend more than half their time in the workplace interacting in various ways with co-workers and clients [PDEG94].

Professional technical writing

Most projects in the industry require extensive documentation throughout the entire software development life cycle. These documents need to be clear, professional, and easily understood by different stakeholders. Hoewever, professional technical writing is a rare skill that is highly sought after in the industry [JJ03].

Project Managements skills

Employers are constantly on the look out for future leaders. Most of the time, the first step into a management position is being a project manager. Also, considering that a large portion of real world projects either fail, go over-budget, or extend over-time, sound project management skills could contribute greatly to a company's bottom-line.

Concern for users In an increasingly competitive industry, companies have become customer-focused. This means that user support personnel not only require technical skills, but also a genuine concern for users. Unfortunately, computing professionals are not renowned for their helpful attitudes. IT professionals have been accused of being rude, not caring about user problems, and making users look stupid [J.98].

Work experience

Employers often complain that graduates lack the ability to learn from on-the-job training and understanding of their business processes [D.04]. In an industry where technology advancements become obsolete very quickly and average company sizes are less than 10, employers do not have the time and resource to train graduates for a long period of time. Most graduates are expected to have transferable skills and had completed industry placements [D.04].

CURRENT EDUCATIONAL MODEL

According to Tavani (2004), CSSE educators in universities are also regarded as CSSE professionals. Therefore they are ethically required to advance the integrity and reputation of the profession and ensure that their ”products”, the CSSE graduates, meet the highest professional standards possible, as suggested by the SECEPP core principles. The educators shall ”constantly seek new channels, methods and technologies to reach and intrigue the students ... teach them skills that they can apply in the real world” [JS03]. A successful educational model shall support the educators to achieve their goals.

One of the main factors that influence the CSSE educational model, as Gruba *et al.* [GPJ04] points out, is employer and industry viewpoints. As shown in the previous section, employers are not finding the right skills in current graduates. This suggests that the current educational model is not effective, and probably needs an overhaul. Before we do that, we must first attempt to uncover the flaws of the current model, covered in the remainder of this section.

Corporatized education

Faber (2002) argues against the corporate influence on CSSE education, which has become the driving factor of the current educational model. Similarly, other critics have described and criticised this move to corporatise education, where knowledge of systems, techniques, tools, and processes can become a competitive advantage and more valuable when it is restricted from public use. Such restrictions have little effect on market practices already based on economies of supply and demand; however, they conflict with the academic culture of knowledge creation, education and learning.

The corporatisation of education is evident as employers exert their pressures directly on universities to revise CSSE courses to reflect industry demands [GPJ04]. It proposes the proprietary university, where knowledge created by university researchers and students are withheld from broad public dissemination and even from fellow academics.

Weakening of the academic community

The emphasis on proprietary knowledge weakens the academic community. It indirectly encourages academics to work in isolation, thus making the dissemination of ideas and practices problematic. The lack of knowledge and resource sharing leads to the unawareness of latest innovations and technologies, which can result in ”reinventing the wheel” issues [SJ04].

Another negative consequence of working in isolation is the weakening of the relationship within the academic community, especially between students and experienced academics. Gruba et al observes that as a result of this, the students will not participate in further education after undergraduate level. This means a loss of potential researchers for the academic community, thereby reducing the pool of knowledge even further.

Scarcity of resources

The scarcity of financial and human resources is another issue with the current educational model.

For example, despite team projects being arguably the most important part of the education for would-be software engineers, they are quota-ed as they are disproportionately expensive to mount [GPJ04]. Even for large firms, they cannot afford to run today’s project scales in-house [S.04]. Furthermore, the high cost of proprietary software does not help the situation.

The issue of insufficient funding also leads to the scarcity of suitably qualified staff, which slows down

the improvement of the syllabi. It also does not allow the educators to focus on students of high calibre, as they need to cater for mediocre students in order to ”fill the quotas” [GPJ04].

Ineffective collaboration

Collaborative learning projects offered in current CSSE courses is heavily criticised for being ineffective.

Faber argues that outside of universities, few programmers ever start from scratch, yet university projects and assignments continue to perpetuate students working on solitary, new projects. He decries these academic projects for being stagnant and only addressed to one audience (the professor) for a one-time purpose, lacking real-world awareness and concern for real users

Since projects are determined by the length of academic terms, students may initiate a project, work on it for several weeks, and then complete it for credit before the term ends. Faber criticises this practice for not teaching students the importance of effective documentation, notetaking and actual collaborative work because no one ever continues the projects they initiate. As a result, students do not learn the importance of good project management, such as how to gracefully and appropiately hand-off projects, how to strategically break projects into stages or how to mentor new initiates into their projects.

Being in isolated, purely academic projects, students lack the experience of examining their roles in a larger community of peers from various backgrounds and specialist areas, domain experts, authorities, novice users etc. Hence, they cannot master desirable transferable skills such as communication and social skills as required by the industry.

These problems associated with the current educational model raise four main issues that violate the IEEE-CS/ACM SECEPP core principles [T.04]. Firstly, it fails to produce the products, that the CSSE graduates, that meet the highest professional standards possible. Secondly, since students can be regarded as customers of universities, the model fails to act in their best interests by not equipping them with fully refined skills before they enter the workforce. Thirdly, it does not satisfy the expectations of the employers. Finally, by taking a corporatised approach to education that values secrecy and proprietary knowledge, it does not act in the best interests of the public.

We will show examples of the issues highlighted in this section with a case illustration involving an interview with final year CSSE students at a leading university in Melbourne, Australia.

CASE STUDY

We conducted several interviews with CSSE students in their final year of an undergraduate degree at the University of Melbourne. Our aim was to find out, from their point of view, whether they have satisfied industry expectations, and whether the course has prepared them well to achieve such qualities of a CSSE professional.

We acknowledge that the number of interviewees has not been high enough to claim any statistical significance. However, our aim is to provide an example of the current educational model in practice. We believe that, since the University of Melbourne is one of the leadings tertiary institutions in Australia, it can reasonably exemplify the issues with CSSE education at university level in this country.

Here is the list of questions we asked them:

1. Throughout university, graduates have learnt various skills that have helped them in their work-place. However, these new professionals often continue their education to acquire new skills long after graduation.

   What other skills do you expect to gain after graduating?

2. What are your long term career ambitions? In the next five years? In 10 years?

3. Do you think that you’re on course to achieve your goals?

4. The following are the industry expectations of CSSE graduates:

- Oral communication skills
- Technical writing skills
- Project management skills
- Concern for users
- Work experience

Do you think you have these skills now?

5. Have collaborative projects at university prepared you well for the workplace?
6. Do you think that the current curricula needs to be improved in order to help you develop as a well-rounded CSSE professional? Please state your suggestions.
7. Will you continue with higher education after graduating? Why?

Please see the Appendix for the answers we received from the interviewees (for privacy reason, they shall remain anonymous). The interviews showed various problems with the curent curricula, which we will cover and address in our proposed educational model.

THE OPEN-SOURCE EDUCATIONAL MODEL

The aforementioned problems associated with the current CSSE educational model in Australian universities beg the question: *what is the better alternative?*

In recent years, OSS has seen spectacular successes. It has evolved from a seemingly idealistic philosophy driving ”home-grown” projects to an established and successful development paradigm that produces high quality software such as the Linux operating system kernel, the Apache webserver, BIND, MySQL, Perl, PHP etc. Therefore, OSSD, as claimed by OHara and Kay [JS03], can serve as ”a channel, method and technology to learn and teach computer science”. OSSD has the potential to expand group work beyond the classroom to include much larger projects and more distributed teams, as well as introducing the students to the larger science/engineering communities. We see this as the main benefit that this software development paradigm has over the current educational model in CSSE courses in Australian universities. Hence, our main goal is to derive an educational model from OSSD philosophy and practices to overcome the limitations of the existing model, in order to produce high quality professionals that meet the employers expectations.

To build an open-source educational model, we need to examine the two questions: *what is OSS* and *what are its practices?*

OSS definition

The following key conditions, as outlined by Feller and Fitgerald [JB00], define OSS:

1. The source code must be available to user.
2. The software must be redistributable.
3. The software must be modifiable, and the creation of derivative works must be permitted.
4. The license must not discriminate against any user, group of users, or field of endeavor.
5. The license must apply to all parties to whom the software is distributed.
6. The license cannot restrict aggregations software.

We recommend further readings at http://www.gnu.org and http://www.opensource.org for a complete understanding of various aspects of OSS.

OSS development practices

Eric S. Raymond, president of the Open Source Initiatives (http://www.opensource.org), wrote a classic article about OSS, called ”The Cathedral and the Bazaar” [S.00]. In this article, Raymond summarised his OSSD experiences into 19 lessons for software programmers, which have become the recommended practices of this paradigm. These lessons, when applied in a university environment, will be very beneficial to the education of future CSSE professionals. Here are the relevant ones that will form the basis of our proposed model:

1. *Every good work of software starts by scratching a developer's personal itch.*
2. *To solve an interesting problem, start by finding a problem that is interesting to you.*
3. *Good programmers know what to write. Great ones know what to rewrite (and reuse).*
4. *When you lose interest in a program, your last duty to it is to hand it off to a competent successor.*
5. *Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.*
6. *Release early. Release often. And listen to your customers.*
7. *Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.*
8. *If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.*
9. *Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.*
10. *”Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.”*
11. *Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.*

The whole list of 19 lessons can be found in Eric Raymond's full article ”The Cathedral and The Bazaar”, linked in the Referrences section.

Faber attempts to use some of Raymonds recommendations as a basis to form an open source educational model for CSSE courses at university level that consist of the following practices:

1. Problem-based learning
2. Working through texts, working through drafts
3. Encouraging risk-taking, inquisitiveness and invention
4. Handing off projects and mentoring new students
5. User testing
6. Converting drafts to final products
7. Collaborative development
8. Rewarding and building from failure

Although Fabers proposed model has most of the main advantages of OSS education, its structure might be ineffective and confusing for those who are used to the conventional model, e.g the current CSSE educators and students, the target audience of this paper. Since this demographic is familiar with the SDLC, and both OSSD and proprietary software development paradigms comply with that, we will present our proposed model in phases corresponding to the SDLC to make comparisons between the two models more intuitive. It also highlights the benefits of open-source model more effectively, as the main advantage that this model has over the current proprietary model is related to software projects run by groups of students, a claim which we will thoroughly analyse.

Our proposed model

Making projects open-source

The core requirement to implement and operate our proposed educational model at universities is to make student projects open-source. Documentation, codes and even binaries of such projects can be put under a public license that preserves the open-source definitions, which can be based on an existing license such as the GNU GPL, LGPL, BSD, MPL, etc. They can be organised in an online repository similar to sourceforge.net, the largest repository for open-source developers. The benefits of this are tremendeous, including:

- Providing a great knowledge resource for CSSE students
- Maintaining student projects, preserving good ideas and implementations
- Inviting experienced academics or developers from other institutions or the industry to contribute to this resource.
- Creating work experience reference for graduates resumes.
- Establishing a strong CSSE community
- Providing the general public with free software products.

We will elaborate on those in the upcoming sections.

Starting a project

CSSE students at Universtiy of Melbourne always start their projects from scratch. Firstly, this does not teach them about reusability of ideas, designs and codes. Secondly, as argued in our previous section about the current educational model, this situation rarely happens in the real-world. Thirdly, a single project handed out to hundreds of students will fail to ”scratch the itch” of many. Raymond states this problem in his ”Bazaar” model by pointing out that ”necessity is the mother of invention”, however too often, software developers spend their days ”grinding away for pay at programs they neither need nor love”. That is in the context of the proprietary software development paradigm, but we can see how it clearly reflects on the current educational model.

In our open-source model, the first key idea is that students shall be able to choose projects that they find interesting, meaningful, motivating and compelling. It is a more effective way for them to learn correlative knowledge, skills and aptitudes. Rather than having a supervisor dictating that this material must be learned, the students would come to realise that to produce a high quality work for the project that they genuinely care for, they would need to learn these things.

The second key idea is to get students to pick up an existing project and start from mid-stream. This is the most common situation in a work environment, however, is not introduced to the students we interviewed. The project can be an existing one from the universitys open-source repository, or from other online open-source ones such as sourceforge.net. While learning how to get into an existing project, the students would learn the real importance of effective documentation, since poor documentation would make it very hard to grasp the ideas and procedures of said project.

Requirements analysis

This phase in the SDLC is usually where market research and feasibility study are conducted. The purposes are to have a better understanding of the target audiences needs, to discover requirements for the software application or to explore the system that the developers intend to build. Alternatively, it could be about requirements gathering, in which case there is a specific client with specific needs that the developers need to study about. The majority of software in the market belongs to the first case: there is no single, specific client but there are many potential end-users with a similar need of an application that can solve their problems.

However, our interviewees have not been introduced to this first, more popular case. They are usually given a specific requirement specification made by the lecturer (acting as the client) or gather the requirements themselves from a single imaginary and purely ”academic” client (meaning a client made up for the purpose of teaching only). As a result, the students are not presented with concerns for real end-users and desirable real-world features of a software system (feasibility, modifiability, expandability, scalability, etc.).

In the open-source model, students shall be encouraged to conduct a market research to realise the needs of their target audience for such software in a real-world context. By having considerations for real end-users and doing market research, students would realise the importance of social and communication skills. They shall also need to research the current offerings in the market, or past/existing projects of other students or researchers at their university to see which they can reuse and how they can make their product stand out from the rest. This will help them to develop critical thinking skills and a competitive mind needed at the workplace.

Design

Learn from successful projects

At the University of Melbourne, design knowledge is passed on to students in the form of theory from textbooks and lectures. Using this method, it is hard to justify if a design decision is good or bad without a working example. There are many projects in the OSS community that have been major successes, such as Linux, Apache, MySQL, PHP, BIND, OpenOffice.org, Mozilla.org, etc. The source code, documentation, log file, and other material associated with these projects can be a rich source of reference as to how a good design decision was made.

Most projects in the OSS community are simple in design. According to Eric Raymond [S.04], design should be robust and simple, instead of cute and complicated. To quote Antoine de Saint-Exupery, who was an aircraft designer and a poet, ”Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.”

One of the reasons why OSS development has been a success is because the design ideas are simple enough for the average developer to comprehend. This simplicity easily convinces developers of a projects potential, and thus encouraging their participation.

Considering that learning software design from OSS project is hardly rocket science and is free, OSS projects could provide a good source of reference to gain work experience employers are looking for.

Learn from past failures

The University of Melbourne bases its assessment on examinations, standardized testing and rote learning. In this model, there is not much room for experimentation and failure.

Students are expected to complete projects from scratch within a 6-month or 12-month period. This short time frame does not allow any experiments with design decisions, as this stage affects the rest of the development life cycle. To avoid failures which can be costly in terms of opportunity cost and

course fees, students are more inclined take the tried and tested methods to solve their problems. As such, students are rewarded for following what instructors recommended, and most likely punished for inquisitive experiments gone wrong.

While this model may solve assessment and discipline issues, it does not teach students how to solve open-ended problems, nor does it allow much opportunity for developing an inquiring mind and risk-taking.

In the OSS model, failure is a common and anticipated part of development process. It allows students time and space to run into conceptual obstacles, to undertake directions that may not work, and to hit walls in their development.

In an open source classroom, students would be presented with half-solved problems, failed solutions, and dead ends. Students are then expected to engineer new solutions from these materials. This encourages learning from past failures.

As Eric Raymond notes, the most striking and innovative solutions come from realizing that ones concept of the problem was wrong.

Learning from past failures and recovering from such failures is an integral part of project management, given the high failure rate of software development in the industry. Students of the OSS model would be well-prepared in dealing positively with failures, making them a valuable asset in any organization.

However, incorporating this change into the curricula raises an interesting problem for highly motivated students. The notion of failure could be hard to swallow for these students who have not been allowed to experience failure or who correlate failure with self-esteem, future success, and self-image. This issue will have to be looked into at some stage.

Implementation

Learn from review

It is generally accepted that formal technical reviews are useful in finding errors, ironing out inconsistencies and learning from others. A review can benefit the writer, the reviewer and the project altogether. The writer can learn from recommendations and mistakes pointed out by the reviewer. The reviewer, especially junior programmers, can learn from the coding style of others. As for the project itself, review ensures that errors are corrected before it is integrated into the system, ensuring a bug-free project that is ready for release at any moment.

Despite the significant benefits of reviews, they are often overlooked when deadlines loom and when human resource is scarce. Some also argue that developers shy away from having their codes scrutinized because they might feel attacked personally.

This is a common problem at the University of Melbourne. Most students have many projects on hand from different subjects, and are often hard pressed for time. As a result, code review is overlooked. Even in occasional cases where reviews are conducted, reviews are done by project team members or supervisors. This leads to a groupthink mentality, where members cannot critically review the codes generated within a team.

In contrast, code review is an automatic process in the OSS model. Codes submitted by students will be instantly reviewed by hundreds of developers who approach it from different angles. In such environment, defects are found and fixed quickly because there are many ”eyeballs” looking for the problem [S.00].

No doubt, this approach of learning through mistakes and positive feedback will provide a better learning experience.

Learn to reuse

CSSE students at the University of Melbourne take on same projects in a subject. Thus, code reuse, especially from peers, is a form of plagiarism. In a standard testing environment like Melbourne University, students are rewarded for coming up with novel solutions, starting with nothing more than conceptual frameworks from lectures and textbooks.

However, outside of a university environment, few developers ever start with a blank screen. Most projects are a continuation from previous work by other developers. Students are forced to work with existing ideas and solutions and build upon it, as mentioned earlier.

In the OSS model, students are encouraged to look at similar projects at the start. By analyzing codes from various sources, students can reuse or rewrite these codes for their own system. This saves time and increases reliability. As Eric Raymond has noted, *"Good programmers know what to write. Great ones know what to rewrite (and reuse)"*.

Many projects in business environment are built using the Rapid Application Development (RAD) model. Developers in these models rely heavily on reusable software components for short time-to-market to meet market and time pressures. By learning about code reuse and making it a habit, graduates would be valuable addition to most software companies.

Testing

Currently, most projects given to students at Melbourne University are often imaginary. The purpose of these projects is only to demonstrate the concepts taught in a subject. When subjects are taught this way, students do not see the implication of their projects to real users. Testing of their software is quite limited since students only think of all possible test cases that markers would be testing. Students fail to realize that real life projects need to take into account different stakeholders.

The OSS model, however, will force students to generate their test cases according to the real user requirement. For this to hold, students have to collect information about all possible uses of their software. On top of that, students will also be assisted by hundreds of other testers in the OSS community. These testers could be their technical peers, novice users, and other experienced developers from outside of the university. A combination of all test procedures increases the average reliability of such software compared to proprietary software.

Raymond points out that with a large enough beta-tester and co-developer base, almost every problem will be debugged rapidly. This is because every problem is an obvious fix to someone. With a huge team of testers solving all kinds of problem in which they themselves are familiar with, almost every problem will be fixed in no time.

By doing so, not only they will acquire rapid code improvement and effective debugging [D.02], they will produce software that directly addresses the concerns of users in the real world. The OSS model also allows students to refine their skills and realize the significance of testing before releasing to public.

Furthermore, students will gain real life experience when they are testing their programs, they will need to consider the user and the possible consequences if their program fail, make a decision then take appropriate actions, and this will develop student to have audience awareness.

Maintenance

Students are always taught that software maintenance is the most costly phase in the SDLC. However, as mentioned previously, projects are often wasted after submission since their purpose is only to help students understand the concept of the topic that they are learning. Hence, this practice does not correlate with the emphasis on software maintenance and potentially lead to higher cost of the project when the students start developing real software systems as they do not know the effective procedures to patch and improve their program after it has been released into the market.

With the OSS education model, students can attract feedbacks and suggestions from peers and end-users for their projects, as they are submitted and maintained in the online repository. They can use these assessments to continue refining their applications. Students would be truly involved in software maintenance and would actively learn maintenance methods instead of just going over them in lectures, tutorials or exams.

A common issue with software projects is that certain members of the development team may lose interest at one stage, causing the work to be incomplete, or forcing it into early retirement. The open-source model allows and encourages students to mentor new initiates into the project in order to maintain and possibly expand the development. Mentoring and handing off project to newcomers is arguably the duty of a software developer, according to Raymond.

A direct benefit of this is that students would see themselves part of a larger trajectory of work [D.02]. They would gain mentoring skill and project management skill, which would be useful in the workplace where frequently new employees need to be guided into the companys projects. Addtionally, this would encourage knowledge sharing and collaboration and enlarge the CSSE community.

Once again, the practical importance of good documentation and technical writing skill is highlighted here. To keep the project alive after losing interest in it, students would need to produce clear and detailed documentation to ensure that others can pick up and continue from where they left it. To effectively mentor newcomers, theys would need to explain the features and procedures of the project, which is best done by showing them how to ”Read The Friendly Manual”.

With proper maintenance of software programs, it will help to prolong the life of the software and accelerate the technology improvement by reducing the possibility of people reinventing the wheel, and emphasize on the reusability of the program.

Community and public

The OSS educational model also brings tremendous benefits to the CSSE community and the general public. As CSSE professionals collaborate, exchange knowledge and share experience, it strengthens the community. This would lead to closer relationship between students themselves and between students and experienced academics. Students would be more inclined to maintain such relationship and continue to contribute back to the community even after graduation, such as continuing with postgraduate study and becoming researchers.

By producing better trained CSSE graduates, the proposed educational model would help to increase the quality of ICT services for the general public. This would improve the reputation of CSSE professionals and raise public confidence of the profession. Additionally, with more skilled developers participating in open-source projects, the public would benefit from having more high quality software products for free.

Limitations

Open source: not the be-all and end-all

Unfortunately, going open-source does not automatically solve every CSSE educational issue, nor does it save an existing project that suffers from ill-defined goals or spaghetti code or any of the software engineering's other chronic ills. As Jamie Zawinski, ex-principal of the Mozilla project, correctly observed: *”Open-source is not magic pixie dust”* [S.00]. In 1999, Zawinski resigned from the Mozilla project, complaining of poor management and misopportunities. At that time, Mozilla was still far from passing the critical threshold to production usability. However, recently, it has become a great success, one of the leading open-source projects, that is well-respected in the CSSE community and delivers top-notch products (web-browsers, email clients and enterprise-grade bug tracking software).

Mozilla has managed to provide an example simultaneously of how open source can succeed and how it could fail. Whether the model can succeed, either in the industry or in an educational environment, largely depends on the human factor: those responsible for implementing it and its users.

Reluctance to curriculum changes

Implementing this new educational model, or introducing anything new to the curricula for that matter, is not a simple task. According to Gruba *et al.* , universities and CSSE staff are reluctant to change their syllabi due to staff issues such as the availability of suitably qualified staff and workload, and the pressure from student subject evaluations, both of which often are the result of financial matters.

Sessional staff, such as tutors, may not be qualified to teach the new materials. Even experienced senior staff may need to be retrained to run new or updated subjects. This is costly for the departments and therefore changes to the syllabi are often not being considered.

Staff workload is another issue. Gruba *et al.* note the lack of willingness among staff to add to their teaching load to design and offer new subjects. Moreover, staffs are under pressure from the universitys student subject evaluations. Those who cannot consistently maintain average scores higher than ”neutral” are hindered in their search for promotion, and the department itself are penalized financially. Due to this, understandably, they are reluctant to initiate change, to experiment with innovative teaching techniques, or to teach challenging material.

Student abilities

Ideally, all students should have the right material needed for CSSE educators to deploy the curricula that create graduates of the highest possible calibre. However, Gruba *et al.* show that the exigencies of filling quotas for both local and international students mean that the weaker students in the courses sometimes do not meet the expectations in terms of maths or English skills, or in breadth of knowledge in other ways. Facing the open-source curricula that stresses the development of communication, technical writing skills and various other soft skills could be too overwhelming for such students. Hence, educators usually take the path of least resistance and have to drop revolutionary ideas for the curricula to cater not for the excellent students that they remember years later but for the mediocre ones that muddle their way through the course, never excelling and sometimes failing.

Designing and implementing an open-source CSSE educational model is a big task, and the details of which are beyond the scope of our paper. We acknowledge these limitations, which can be the obstacles to the success of incorporating open-source practices in CSSE curricula. To further explore this incorporation, we will need to conduct additional feasibility study of this model by interviewing more stakeholders of CSSE education at universities, as well as examples and analysis of similar incorporations at other universities around the globe. These tasks will be left for a future paper to address.

CONCLUSION

In the present study, we have identified the expectations that employers have for graduates, but the current CSSE educational model in Australian universities cannot deliver. These expectations include oral communication skills, technical writing skills, project management skills, genuine concern for users and work experience. The current model fails to deliver these qualities in their graduates due to certain flaws: corporatization of education, weakened academic community, scarcity of financial and human resources, and ineffective collaboration. These problems are exemplified by our case study of final year CSSE students at a leading university in Melbourne.

We have introduced the open source software development paradigm, and studied the research and recommendations by CSSE educators and open-source experts. From our study, we have proposed a more effective educational model, which could address the shortcomings of the current model. Our model is heavily based on collaboration between students and academics in software projects, which would help students to develop collaborative skills needed for the industry. It also strengthens the academic community and benefits the general public.

We have identified certain limitations of our proposed model: the limitations of open-source itself, the reluctance to curriculum changes, and student abilities. A thorough feasibility study of the implementation of our model would be needed to address these issues. Unfortunately, it is beyond the scope of this paper, and we hope to resolve this in future studies. It would be interesting to witness this model in practice, and the impact it has on the education of CSSE students.

APPENDIX: CASE STUDY INTERVIEW SAMPLES

Here are the sample responses from the case study interviews.

1. **Throughout university, graduates have learnt various skills that have helped them in their workplace. However, these new professionals often continue their education to acquire new skills long after graduation.**

   What other skills do you expect to gain after graduating?

   "I expect to gain additional technical skills through self-learning or taking extra classes.

   I also expect to gain other non-technical skills, such as negotiation with clients, project management, and knowledge about business processes."

   "I would like to be able to programming skill in different computer languages, with good problem solving skills, the ability to examine, analyze, and solve problem, ability to communicate well, such as documentation, presentation, and oral communication. "

   "I would like to gain workplace-related skills through work experience, seeing that I have had no industry-based learning in this course."

2. **What are your long term career ambitions? In the next five years? In 10 years?**

   "Within five years, I expect to be a system analyst, who act as a middle person between my company and clients. I see myself in this role, as I perceive myself as to be a good communicator with adequate technical expertise.

   Within 10 years, I hope to start my own software company. Taking the role of director, I will set the direction and vision for the company, hiring experts and junior programmers from various skills and background. "

   "I'm too sure with the career position and stuff, but I know that I'd like to get reasonable salary. A programmer sitting in a box and programming some software program would do me just fine. "

   "Within five years, I expect to gain as much industry experience as I can, as well as improving my technical skills. I hope to be able to start my own IT consulting company further down the track."

3. **Do you think that you're on course to achieve your goals?**

   "At the moment, University provides a good foundation in technical knowledge. I believe that the non-technical skills I wish I have can only be acquired through real work experience."

   "Since I want to be a good programer, I reckon that University does provide sufficient programming skills, and problem solving skills. Although, with respect to the program documentation for SLDC, I think the university provide too little on that. Other skills that I want to acquire, it may required a bit of time and my full interest as well, which I don't think I have done that so far."

   I am confidence that I am. However, I wish this course could help me more with work-related experience and more practical technical skills. I have to learn most of them on my own.

4. **The following are the industry expectations of CSSE graduates:**

   - **Oral communication skills**

- **Technical writing skills**
- **Project management skills**
- **Concern for users**
- **Work experience**

Do you think you have these skills now?

"I believe I have the communications skills needed at university level. However, I have not proven myself in a working environment yet.

I have taken part writing a few software process documentation in a University subject. However, it was only a simulated case study (i.e. no actual software was built). I think the real test would come when I have to write documentation for a real software.

I was a project manager for the above mentioned simulated case study. It was a good experience, and I have learnt some important lessons. There is still more to learn in terms of project management.

Throughout university, all projects have not taken into account real users. The only stakeholders were markers and lecturers. Thus, I have never built a system that put the concern of users foremost.

I have not taken part in any industry based placements. "

"I dont think I have sufficient technical writing skills to provide my client documentation, which allow the client to correctly and fully utilize the program. Nor I have proper project management skills since little of the subjects offer team projects. And since, the projects are mostly imaginary projects, we dont have the real concern for users.

Regarding to work experience, I can say that I have no real work experience at all, except for the programming for the given projects in each subject. "

"I think I have oral communication and technical writing skills covered. I have been taught project management skills in this course by being a project manager in a software engineering subject. I have also had work experience in the industry, however that did not come from the university. However, I have not had experience with real-world software projects and real end-users, which I hope to gain once I enter the workforce. I wish projects at uni would be more real and involve real users. I would have learnt more from that."

5. **Have collaborative projects at university prepared you well for the workplace?**

   "Im not sure, as have no idea what collaborative work in the workplace is like. "

   "I had limitted experience working with other people, and it's not really related to IT. So I'm not too sure if the university prepared me for the IT collaborative work or not. However, I felt that the current collaborative projects is not having much effect on me, since there are too few collaborative projects and also the people I am working with are close friends, unlike in the workplace that people come from different background with different level of knowledge. "

   "I have collaborated with my peers, who have as much knowledge about the workplace as I do, which is none. Although it does provide me the experience in working with other people, I would prefer to have some experienced members in my project teams to guide us through confusions. I would also like to do the software projects for real users with real needs, instead of imaginery users with fake needs created just for the academic purpose of the project."

6. **Do you think that the current curricula need to be improved in order to help you develop as a well-rounded CSSE professional? Please state your suggestions.**

   "There has been some good subjects (ie. 433-343) that have provide communication skills, technical writing skills, and knowledge about professional issues.

   One improvement I would suggest is incorporating industry-based placement into the curricula to provide job experience."

"Not really. I think the IT industry is developing quite fast, so the curricula should develop as well, such that the students/learners expose more to the recent technologies and methodologies available.

I suggest that the curriculum should expose learners more to the current practices that will help them get the job after graduate, rather than giving them only the basic and letting them learn for themselves on the job or on their spare time. "

The current curricula are clearly outdated. They should continously incorporate new methods and technologies. For example, web development is popular these days, however my department has not done much about it. They can't even get their own websites right!

I would like to have more collaborative projects, even in earlier years, which have industry-based components, so that we will gain more work experience. I would like to have experienced people working with us in our projects to put us on the right track.

7. **Will you continue with higher education after graduating? Why?**

   "At the moment, no. Since I dont have any work experience, that is the main priority for me after graduation. I might continue with higher education in the future. "

   "Yes, I dont think I know nor confident enough to be able to work effectively under the competitive environment as in the workplace. "

   "Not at the moment. As I lack work experience, I'd rather go out into the industry and work for a while. Moreover, I don't feel like being a part of the research community here, as I don't know anyone personally."

REFERENCES

[D.02] Faber B. D. Educational models and open source: Resisting the proprietary university. In *Proceedings of the 20th annual international conference on Computer documentation*, pages 31–38, Toronto, Ontario, Canada, 2002. ACM Special Interest Group for Design of Communications.

[D.04] Hagan D. Employer satisfaction with ict graduates. In *Proceedings of the sixth conference on Australian computing education*, volume 3, pages 119–123, Dunedin, New Zealand, 2004. ACM International Conference Proceeding Series.

[GPJ04] Sondergaard H. Gruba P., Moffat A. and Zobel J. What drives curriculum change? In *Proceedings of the sixth conference on Australian computing education*, pages 109–117, Dunedin, New Zealand, 2004. Australian Computer Society, Inc.

[Ini04] Open Source Initiatives. Open source faq. *http://www.opensource.org*, 2004.

[J.98] Teague J. Personality type, career preference and implications for computer science recruitment and teaching. In *Proceedings of the 3rd Australasian conference on Computer science education*, pages 155–163, The University of Queensland, Australia, 1998. ACM International Conference Proceeding Series.

[JB00] Feller J. and Fitzgerald B. A framework analysis of the open source software development paradigm. In *Proceedings of the twenty first international conference on Information systems', International Conference on Information Systems*, pages 58–69, Brisbane, Queensland, Australia, 2000.

[JJ03] Tan J. and Phillips J. Challenges of real-world projects in team-based courses. *The Journal of Computing in Small Colleges*, 19(2):265–277, 2003.

[JS03] O'Hara K. J. and Kay J. S. Open source software and computer science education. *The Journal of Computing in Small Colleges*, 18:1–7, 2003.

[PDEG94] Staudenmayer N. Perry D. E. and Votta L. G. Open source software and computer science education. *People, organizations, and process improvement*, 11:36–45, 1994.

[S.00] Raymond E. S. The cathedral and the bazaar. *http://www.catb.org/ esr/writings/cathedral-bazaar/cathedral-bazaar/*, 2000.

[S.04] Raymond E. S. Up from alchemy. *IEEE Software*, 21(1):88–90, 2004.

[SJ04] Carbone A. Sheard J. From informal to formal: creating the australasian computing education community. In *6th Australasian Computing Education Conference (ACE2004)*, volume 30, pages 291–297, Dunedin, New Zealand, 2004. Australian Computer Society, Inc.

[T.04] Tavani H. T. *Ethics and Technology: Ethical Issues in an Age of Information and Communication Technology*. John Wiley and Sons, Inc, 2004.

Recommended readings:

- http://www.opensource.org : Opensource.org is a web site of Open Source Initiative, here you should find information about open source including definitions, licenses, news and updates.
- http://www.gnu.org : GNU.org provides the background of GNU projects, and it is also the web site of the Free Software Foundation (FSF).
- http://www.catb.org/∼esr/writings/cathedral-bazaar/cathedral-bazaar : The full article of Eric Raymond's famous article "The Cathedral and the Bazaar".
- http://www.sourceforge.net : the largest repository of Open Source code and applications available on the Internet. SourceForge.net provides free services to Open Source developers.