On the Prospects and Concerns of Integrating Open Source Software Environment in Software Engineering Education

Pankaj Kamthan
Department of Computer Science and Software Engineering, Concordia University, Montreal, Quebec, Canada

kamthan@cse.concordia.ca

Executive Summary

Open Source Software (OSS) has introduced a new dimension in software community. As the development and use of OSS becomes prominent, the question of its integration in education arises. In this paper, the following practices fundamental to projects and processes in software engineering are examined from an OSS perspective: project management; process, workflows, and collaborative activities; modeling and specification; deployment of standards; documentation; and quality assurance and evaluation. Based on a pragmatic framework, the prospects of integrating OSS in a traditional software engineering curriculum are outlined and concerns in realizing them are given. In doing so, the cases of the adoption of OSS process model, use of OSS as a Computer Aided Software Engineering (CASE) tool, OSS as a standalone sub-system, and open source code reuse are considered. We present some of the trade-offs that could help educators in decision making towards the use of the OSS environment in software engineering pedagogical contexts. The significance of openly accessible content in general and its relation to OSS in particular is briefly highlighted.

Keywords: Constructivism, Open Content, Software Engineering, Software Modeling, Software Process, Software Quality, Software Reuse

Introduction

The steady rise of Open Source Software (OSS) (Raymond, 1999) over the last few decades has made a noticeable impact on many sectors of society where software has a role to play. Indeed, the current information technology movement could be safely attributed to the relatively inexpensive, easily installable, and readily available OSS for a variety of computing devices.

The GNU is Not UNIX (GNU) software, the TeX mathematical typesetting system, the X Window System, the Linux, OpenBSD and their utilities, the K Desktop Environment (KDE), and more recently the Apache Software Project and the SourceForge, are some of the prime examples through which OSS is changing the way software is developed and used.

Editor: Dion Goh

The discipline of software engineering (Ghezzi, Jazayeri, & Mandrioli, 2003) was born out of the need of introducing order and predictability in large-scale software development. Software engineering advocates a systematic approach to the development of high-quality software within the given time, budget, and other organizational constraints. Over the last few decades, software engineering has been also playing an increasingly prominent role in computer science and engineering curricula of institutions around the world.

This paper examines the interplay between traditional software engineering and Open Source Software Development (OSSD) from an educational standpoint.

The rest of the paper is organized as follows. We first outline the background and motivation necessary for the discussion that follows and state our position. This is followed by a detailed treatment of core software engineering practices in the light of OSS. Based on that, we then discuss the use of OSS in software engineering education (SEE). Next, some challenges and directions for future research are outlined. Finally, concluding remarks are given.

Background and Related Work

In this section, we review the definition and basic characteristics of open source, and provide motivation for the interplay of OSS and SEE.

Definition of Open Source

The concept of open source can mean different things in different contexts (Gacek & Arief, 2004; Perens, 1999) and it is therefore crucial to articulate a precise definition.

For the purposes of this paper, we will use “open source” as a single encompassing term that satisfies the following conditions: (1) non-time delimited, complete software whose source is publicly available for (re)distribution without cost to the user, (2) imposes minimal, non-restrictive licensing conditions, and (3) is itself either based on non-proprietary technologies or on proprietary technologies that conform to (1) and (2).

For example, the Java programming language is itself proprietary but its specification is publicly available and there is software based on Java that would satisfy (1) and (2). Any software that does not fall into this category is termed as non-OSS. As an example, freeware may not completely satisfy (1) and therefore is non-OSS.

OSS encourages intellectual “freedom”. The notion that software should be isolated from control by a small group of vendors was pioneered under the auspices of the Free Software Foundation (FSF). Since then this idea has been adopted and extended in many ways, including that by the Open Source Initiative (OSI) and by Creative Commons.

The freedom of examination, modification, and redistribution; reduction of vendor reliance; reduction of production costs; and augmentation/flexibility in the number of software options available to users from which to choose from, are some of the advantages commonly touted of the OSS.

In the following, by an “OSS environment” we will mean the situation or mindset that includes project, process, product, and people involved in the development of an OSS.

The Engineering of Open Source Software

As the use of OSS increases, the question of *how* they are actually engineered garners interest. An engineering perspective towards OSS is necessary for a variety of reasons: OSS may be adopted

and used in critical areas of an organization and so need to be carefully examined with respect to non-OSS alternatives, OSS installed in an organization may need to be maintained over time and therefore need to be well-understood by maintenance engineers, and current OSS practices could be of interest from an academic (say, teaching, learning, and research) standpoint.

Although OSS itself has a long and rich history, it is only in recent years that a software engineering viewpoint towards it has been taken (Spinellis & Szyperski, 2004; Vixie, 1999). The annual workshops in recent years under the label of *Open Source Software Engineering* have also created an awareness of this important area.

Open Source Software in Education

As the use of OSS gains prevalence, the issue of its outreach in an educational context arises. In this paper, we take the position that students studying software development should be exposed *early* to this rapidly growing area.

In fact, the use of OSS in computer science education has been emphasized in recent years (Attwell, 2005; González-Barahona, Heras-Quirós, Centeno-González, Matellán-Olivera, & Ballesteros-Cámara, 2000; Liu, 2003). It has also been suggested (Cusumano, 2004) that developing OSS could also help students in their future career paths.

However, the current studies of OSS-based education are limited in one or more of the following ways: the discussion is often confined to the case study of a specific OSS, do not highlight the problems associated with introducing OSS, do not address software engineering exclusively, or ignore aspects of software engineering that OSS do not address. One of the purposes of this paper is to address these concerns.

Elements of Software Engineering and its Education, and their Manifestations in Open Source Contexts: A Perspective

In this section, we look at six broadly classified areas that are common in most SEE contexts, namely that of project management; process, workflows, and collaborative activities; modeling and specification; deployment of standards; documentation; and quality assurance and evaluation. These areas overlap with the knowledge areas of the Guide to the Software Engineering Body of Knowledge (SWEBOK) (Abran, Moore, Bourque, & Dupuis, 2001) and of the Software Engineering Education Knowledge (SEEK) (IEEE-CS/ACM, 2004).

We examine the extent to which these areas are realized (or not) in an OSS environment. In doing so, we inherently set *constraints* on the use of OSS environment in SEE, which is discussed in the next section.

Project Management

Managing a software project is important for its eventual success. We shall limit our discussion largely to measuring success and team, time, and configuration management.

The goals of developing software in educational and OSS contexts are different (Shaw, 2000). In software engineering, the software product is a means to the end, not an end in itself. It has been reported (Cusumano, 2004) that OSS often lacks precise specification of goals and as a result fails to define "success."

In SEE, there is a price for not performing up to the expectations or not working to full potential, and is often exhibited by an impact on the grading differential. On the other hand, the reason for abandoning an OSS project are often not given or made public.

Although software engineers are often bound by organizational or professional code of ethics, this is not the case in OSS. OSS is carried out on an "honor system." Specifically, there are little or no repercussions for not following up on work or on schedule, or stalling the project altogether. This flexibility may be attractive in a professional context but does not scale well in an educational setting.

In lieu of mimicking real-world software projects as well as due to natural limitations of schedules at educational institutions, there are inevitable time constraints associated with course projects. However, OSS projects are usually carried out without a monitoring entity and on a volunteer basis, and are not time bound.

In a software engineering setting, there is a notion of a team, whereas in OSSD there is a *community of practice* (Koohang & Harman, 2005) that is a group of individuals that are informally bound to collaborate on a shared task. There are fundamental differences between the social structure of a team of students in a software engineering environment and the participants in a community of practice in the OSSD. In general, software engineers working on a software project in a professional or learning context are collocated while those in OSS developers form a distributed community (Crowston & Howison, 2005; Thomas & Hunt, 2004) within a social network (O'Reilly, 2005). Assuming responsibility and accountability ― both individually and as a team ― are at the heart of software engineering. On the other hand, there is no inherently hierarchical team structure in OSS. There is usually a core group that contributes the most with a sporadic participation by others (Michlmayr, Hunt, & Probert, 2005).

There are also notable differences with respect to "fluidity" in social bonding between a team and a community of practice. In a software engineering context, students most likely belong to the same institution, may take multiple courses together, and may often work on the same projects together. These students also may be related on a personal level (as roommates, siblings, and friends) and/or the relationships may evolve to become personal. On the other hand, the open source community of practice forms an open, loosely networked, and "virtual" ecosystem. This ecosystem is dynamic and "organic" as OSS developers are loosely related and move across projects and coalesce (Weiss, Moroiu, & Zhao, 2006).

In the author's experience with the practice of SEE, configuration management is not as pervasive in educational software projects as it is in OSS and is usually limited to version control and backups. Student team leaders have reported in their final project presentations that their teams do benefit from installing a Concurrent Version system (CVS) system accessible via the Web for their document and source code deliverables. However, the distributed nature of contribution as well as the desire of the developers to be able to disseminate "up-to-the-minute" code has led to a usually strong support for configuration management (version control, bug tracking, or build management) (Asklund & Bendix, 2001) in OSSD. In fact, it is quite common in an OSS environment for the developers to post nightly builds for others to experiment and to solicit feedback.

Based on "post-mortem" activities, such as project retrospectives (Derby, Larsen, & Schwaber, 2006; Kerth, 2001), organizations continually strive to improve the effectiveness of their software projects while remaining cost-friendly. However, there is little evidence to suggest that being carried out in OSS projects.

Process, Workflows, and Collaborative Activities

In software engineering, students are normally introduced to both prescriptive and agile process models (Boehm & Turner, 2004). The former are often rigid/bureaucratic and involve heavy use of documentation. The latter allow flexibility by virtue of sensitivity to the social and organizational environment in which software is being created and involve lightweight documentation. The selection and adoption of a process model depends on the context of a team environment and on the characteristics of the application domain underlying the software to be developed.

The OSSD process, known as the "Bazaar model" (Vixie, 1999), is rather flexible. However, it is not subsumed by any of the conventional software process models. Indeed, OSSD can benefit from some of the agile practices such as openness to change and cotemporal/parallel development (Warsta & Abrahamsson, 2003). As an example, many of the practices of Extreme Programming (XP) (Beck & Andres, 2005), which is an agile process model, are applicable to OSS (Nishinaka, 2001). However, *Onsite Customer*, *Pair Programming*, and *Collective Ownership*, are three of the key practices of XP that do not scale well in the distributed, non-proximal environment of the OSS.

The Bazaar model also differs from the iterative process models that embrace certain aspects of agility such as the Unified Process (UP) (Jacobson, Booch, & Rumbaugh, 1999) process framework and based on it the Rational Unified Process (RUP) (Kruchten, 2004). For example, RUP has a strong emphasis on customer involvement and is model-driven (Beydeda, Book, & Gruhn, 2005), both of which are not a commonplace in OSSD.

Software process workflows typically include (Ghezzi et al., 2003) software requirements (problem definition), software design (high-level view of the solution), implementation (low-level working solution), and testing (verifying whether the solution in fact matches the problem). In an OSSD process (Raymond, 1999), software requirements are usually not publicly available, the focus on design is informal, and there is much attention on implementation and in some cases on testing. Indeed, several OSS utilities (notably for properly structuring the source code and for unit testing) have been created just to support the last two phases.

It is a commonly held belief in the software engineering community that the quality of a software process directly impacts the quality of the software product, and therefore much research in the last two decades has focused on means for software process improvement. Indeed, process maturity is an integral topic in many courses related to software process engineering. In traditional software process environments, organizations can make use of the Capability Maturity Model (CMM) (Paulk, Weber, Curtis, & Chrissis, 1995). However, apart from some isolated cases (Golden, 2004), there seems to be little systematic effort towards addressing the maturity of the OSSD process.

Modeling and Specification

Modeling, particularly during early phases of software development, is playing an increasingly important role in activities and deliverables in software engineering (Beydeda et al., 2005) and its education (Cowling, 2005). Early modeling is crucial from the point of view of understanding the problem and solution domains in an implementation-neutral manner, opportunity for experimentation at low cost, and control and prevention of problems that can propagate into later stages of operation.

Modeling in its different degrees of formality plays a central role in both XP and UP and is a determinant of the process maturity of an organization. Some form of modeling is introduced in most practical software engineering courses.

The Unified Modeling Language (UML) (Booch, Jacobson, & Rumbaugh, 2005) has emerged as a standard language for modeling the structure and behavior of object-oriented systems, and its use in the last few years in SEE has increased dramatically. The author has recommended a proper use of UML (Kamthan, 2004) for domain and use case modeling in several courses. However, there is little evidence of use of UML, and in general of any form of systematic modeling, in an OSSD process.

Formal specifications are also integral to many courses in software engineering (Alagar & Periyasamy, 1998) where the safety requirements or design of a critical system need to be precisely (mathematically) expressed. However, once again, there is little evidence to support the use of mathematics in OSS problem or solution domains of a software system for analysis or synthesis, respectively. This evidently limits the use (and even the use in part) of OSS in safety-critical software. A similar argument holds for real-time or embedded systems, whose definition and design has also gradually begun to depend on formal specifications.

Deployment of Standards

There are a variety of reasons for introducing and adhering to standards in software engineering. Standards provide a common ground for a team, streamline efforts, and when applied well, are known to contribute towards quality improvement (Schneidewind & Fenton, 1996). Lack of standardization can often lead to communicability problems (among humans) and interoperability problems (among machines).

The author has been a strong proponent of the use of standards throughout SEE, has made mandatory use of standards from the Institute of Electrical and Electronics Engineers (IEEE) and International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) for process documents, and strongly encouraged standardized definitions of programming languages and corresponding compilers/interpreters from American National Standards Institute (ANSI) and Ecma International.

The use of standards in the OSSD process (Raymond, 1999) is usually confined to implementation-level concerns. The OSS approach serves as a platform for trying out new technologies and developing "proof-of-concept" implementations. For example, in case of OSS for Web Applications, the deployment of "standards" is centered on text description languages such as the HyperText Markup Language (HTML) or those based on the Extensible Markup Language (XML) and presentation languages such as the Cascading Style Sheets (CSS), to name a few.

Documentation

The role of documentation is usually accentuated in software engineering for visibility and for communication. The documentation forms a means for introducing visibility within a software process. The role of communication is central to any software development and the documentation acts as the *message carrier* within the communication infrastructure of a software project.

Almost all software process models support some form of documentation although the degree of documentation may vary between those process models that are bureaucratic and those that are flexible. The courses related to technical communication and programming methodology early in the curriculum form the basis of internal documentation of software developed in later courses. In some cases, creating external documentation (say, user manual or a help system) may also be required.

In contrast, availability of documentation remains one of the issues in OSS use and reuse (Raymond, 2004). In OSS, the notion of process documentation is not always adopted and followed. The internal documentation in OSS tends to focus more on the implementation rather than early

phases (of requirements or design). The documentation at times may not be complete or may only be sketchy. At times, help or tutorial documents are not updated to synchronize with the latest code releases. The OSS style of writing currently in place at times tends to be informal rather than technically inclined to the issue at hand.

Still, the presence of external documentation in OSS is relatively common, albeit not as rigorous as, say, commercial software. The degree of "good" external documentation for an OSS also seems to vary with respect to its usage and feedback. For example, much of the software under the Apache Software Project is well-documented and tends to evolve frequently.

It is critical that the aforementioned issues are put into context and pointed out to the students *early*, especially if it is their first contact with a systematic use of documentation in software. It becomes increasingly harder with the passage of time to change perceptions and habits that have built and coagulated.

Quality Assurance and Evaluation

There are different views of quality (Wong, 2006). In software engineering and its education, there is much emphasis on quality in all aspects of software (project, process, product, and occasionally even people).

The issue of OSS quality in general, and concerns of performance, security and usability in particular, has been addressed (Halloran & Scherlis, 2002; Michlmayr et al., 2005; Schmidt & Porter, 2001; Seidel & Niedermeier, 2003).

Although comprehensive studies are still lacking (Aberdour, 2007), there are many OSS that appear to exhibit "high" quality. However, the approach to quality assurance and assessment in OSSD does not appear to be *systematic* (Porter et al., 2006) and therefore the results do not seem to be repeatable. For example, there is little evidence in OSSD of any of the available quality models (Fenton & Pfleeger, 1997) being adopted and followed. In OSS, peer reviews are used as a technique for an informal evaluation whereas formal inspections are apparently non-existent. In general, comprehensive collections of test cases, test suites or test harnesses are rare, and broad testing is even rarer. More importantly, participation is voluntary and monitoring of the degree of testing is almost non-existent. The linear relation of the number of bugs found to improvement of quality proposed by the OSSD process (Raymond, 1999) and endorsed by others (Koohang & Harman, 2005; Verma, 2006) is a bit simplistic, and has indeed been termed as a "fallacy" from a software engineering perspective (Glass, 2003).

One of the approaches taken in SEE for addressing quality in a concrete and repeatable manner is the following: to improve a property of a given entity (say, a process or a product), we must be able to quantify that property. Therefore, the issue of quality is closely related to that of measurement (Fenton & Pfleeger, 1997). For example, if we wish to improve space-efficiency, we could measure the source program file size and in turn the lines of code (or number of characters); to improve structural complexity of a program, we could measure the number of decision structures, the number of parent-child classes, the number of method calls, and so on. The resulting data could then be analyzed (statistically or otherwise). Once again, there is little evidence to support rigorous measurement efforts in OSS contexts.

Table 1 summarizes the departures of the OSS environment from software engineering and its education discussed previously. These put the scope of OSS within context for the next section. We contend that highlighting the differences between the two could be useful to the students participating in a software project involving the OSS environment.

Table 1. The deviations of the OSS environment from software engineering and its education

Project	• The precise estimates of schedules and details of other aspects that provide the overall picture of the software project plan are usually absent in OSSD.
Process	• The traditional OSS process model does not explicitly support any customer involvement in its phases, an aspect that is important for today's interactive software systems. • There is little or no evidence to suggest early modeling in OSSD that could be used as inspiration for similar domain contexts. • There is minimal trace between phases of a process and that from phases to process artifacts in OSSD. • The rationale for design decisions, including the use of algorithms and data structures which led to implementation, is sporadic in OSSD.

Having compared OSSD to traditional software engineering and its education, we now turn our attention to *realizing* the OSS environment in SEE.

Implications of the Open Source Software Environment in Software Engineering Education

Our approach for integrating the OSS environment in SEE, namely OS2SE2, is presented in Table 2.

The entries within a row or a column are placed in no particular order of significance. However, the order of entries across the rows and columns *is* relevant: the entries in higher rows depend on the entries in lower rows and the entries in the columns on the left depend on the entries in the columns on the right.

Table 2. A high-level view of the OS2SE2 framework for deploying the OSS environment in SEE

<table>
<tr><th>Nature</th><th>Type of Deployment</th><th colspan="4">Meta-Concerns</th></tr>
<tr><td rowspan="2">Practice</td><td rowspan="2">• OSSD Process Adoption
• OSS as a Computer Aided Software Engineering (CASE) Tool
• OSS as a Sub-System
• OSS for Reuse</td><td rowspan="3">Planning</td><td rowspan="3">Teaching and Learning Goals</td><td rowspan="3">Feasibility</td><td rowspan="3">Legality</td></tr>
<tr><td>Theory</td><td>• OSS for Pedagogy
• OSS for Learning</td></tr>
</table>

The following is a summary of the entries in the OS2SE2 framework:

- **Theory and Practice.** There are different (but not necessarily mutually exclusive) ways in which OSS can be used in SEE (Kamthan, 2006): (1) for pedagogy and (2) for learning, and (3) adopting the OSS process, (4) as CASE tools that support software produc-

tion, (5) as one of the sub-systems, or (6) for the purpose of source code reuse. We note that (1) and (2) are theoretical in nature, while (3)-(6) are practical in nature. The practical aspects can all occur within the same software project and must be in agreement with the theoretical base that has been created.

- **Planning.** The inclusion of any external entity, regardless of the cost, must be appropriately planned to reach its full potential. The absorption of the OSS environment in SEE is no exception.
- **Teaching and Learning Goals.** The approaches for theory and practice and the OSS adoption plan needs to be aligned with teaching and learning goals to which the contributing factors include the pedagogical aims of the institution and the learning context. They are likely to be entrenched in teaching strategies and learning theories adopted, vary across institutions (say, a polytechnic school and a University), differ with respect to overall program curriculum, and vary with respect to the cultural and prerequisite technical background knowledge and skills of students.
- **Feasibility.** Since software engineering is a practical discipline, all the aims and activities from their initiation to their completion involving OSS should be feasible. The feasibility analysis is evidently related to decision making (Clemen, 1996).
- **Legality.** The laws regarding OSS vary across jurisdictions (say, Canada, Germany, and Russia), and therefore any use, manipulation, and/or development of OSS must be legally acceptable in the place where it is carried out.

The precise articulation of the teaching and learning goals, of the criteria and techniques to be adopted for carrying out a feasibility analysis, and of legal issues, is beyond the scope of this paper.

We now discuss the entries of Table 2 on planning and on theory and practice in more detail.

Planning the Deployment of OSS Environment in SEE

There is usually a long bureaucratic process for selection, adoption, and inclusion of any new didactic instrument at educational institutions. It is likely that educators planning to adopt the OSS environment may have to present a formal proposal that includes a rationale. We hope that the following guidelines assist in that regard and can alleviate some of the tedium involved:

- **Nature of OSS.** Carefully select an OSS keeping in perspective that some students may not be equally excited about them as they may be for industry projects offering commercial incentives (Andrews & Lutfiyya, 2000). To that regard, independent surveys of OSS applicable to different project contexts (Woods & Guliani, 2005) could be useful.
- **OSS for Careers.** Look into the usefulness for the OSS for building future careers of students as, for example, some OSS for same domains are more broadly used in industry than others.
- **OSS Technical Considerations.** Given the same problem domain, there may be more than one OSS and (like other software) they may not necessarily be all the same. For example, check the history of the OSS and see if the evolution has been stable; verify claims particularly related to quality (for instance, look into the degree of testing carried out); check the availability of any non-trivial (representative) examples and how well they work; check whether the OSS is sufficiently documented before recommending its use; and check the terms and conditions of the associated license (if any).

Indeed, close collaboration with systems administrators of the corresponding Departments can be quite useful in such a decision making. An incremental approach starting from a minimal and well-defined list of OSS is highly recommended.

Use of OSS for Pedagogy

OSS could be deployed in various ways for the purposes of teaching in a classroom. The availability of source code of OSS provides a unique opportunity for the educators to experiment. As compared to the "toy" theoretical examples in textbooks, the OSS "real-world" contexts can often provide better opportunities for teaching intricate concepts.

The openness of OSS in contrast to non-OSS becomes all the more valuable when a deep knowledge of internals of software is necessary for understanding. This is particularly the case in systems software courses where, for example, the design of an operating system kernel of an OSS such as that of Linux can be discussed. The source code internals of software (that is usually larger in scale than those accompanying the commonly used textbooks) can be shown and aspects of its design and quality can be debated in the class.

Apart from technical aspects, it has been the author's experience that OSS can also serve as a starting point for discussing social aspects of software engineering like software ethics (Qureshi, 2001) and licensing issues. For example, how well a given OSS follows the principles put forward by the Software Engineering Code of Ethics and Professional Practice (SECEPP) of the Association for Computing Machinery (ACM)/Institute of Electrical and Electronics Engineers Computer Society (IEEE-CS) Joint Task Force on Software Engineering Ethics and Professional Practices are worthy of examination and class discussion.

Use of OSS for Independent Learning

OSS provides a useful workbench for learning. For example, OSS can be used for self-learning purposes outside classroom (say, at the library or at home). Indeed, the ascent of affordable personal computers, high-speed Internet connectivity, and the use of the Web as an information base are having a major impact on the way students study and learn at home.

Among the different theories of learning, individual constructivism (Piaget, 1971) has emphasized "learning by doing". Indeed, there is parity between OSS and elements of constructivist approach to learning (Koohang & Harman, 2005). The availability of OSS source code provides a unique opportunity for students to experiment at some depth and thereby enhance their knowledge and improve upon their skills.

We note, however, that the lack of sufficient documentation and timely technical support, if at all, can pose obstacles for putting this into practice.

Adopting the OSSD Process

OSS can be used as a basis for assigning course projects on topics similar to the application domain of the OSS itself. The openness of the source code can help educators judge the feasibility of such as software project for a given team size and the time permitted. Assuming it is legal within the jurisdiction, OSS could also be used as a basis for reverse engineering: given a certain OSS, students can be asked to create a high-level design model or visualization of its source code, or to refactor (Fowler, Beck, Brant, Opdyke, & Roberts, 1999) its source code to improve some of its quality attributes while still preserving its functionality.

As part of a course project, students could be made to "simulate" the OSS environment for developing software by adopting the OSS process and the practices in it. The resulting software will then itself be an OSS and whose development could be made open to public. As an example, SourceForge could provide a medium for development and distribution, while Sakai could be used for collaboration among teams.

However, this may be the most challenging of all the applications of OSS in SEE. First of all, this will likely require extra effort on part of the educator and this may not be in line with the administrative requirements of mainstream courses. As apparent from the previous section, the Bazaar model requires a different mindset from traditional approaches and may need to be "tailored" for an educational use. For example, instilling the sense of teamwork in physical proximity and collectively experiencing the issues that go with it are an important part of learning. Also, some institutions discourage coursework outside their confines and expect ownership of the final product.

Another issue is about exercising fairness in evaluation. For example, once a team has set up a place on SourceForge, should it be allowed to solicit help and feedback from those in the OSS community not registered in the course? What is the impact of openness of source across teams for an identical project topic? These questions need to be addressed and satisfactorily answered prior to any OSS initiative in education.

OSS as a CASE Tool

We need software to develop software, and OSS could prove a useful aid as a Computer Aided Software Engineering (CASE) tool. There have been reports (Martin & Hoffman, 2007) of successful use of OSS as a CASE tool in small organizational setting.

Some examples OSS that could be adopted are Apache Maven for project management, MediaWiki for fostering team-wide communication, WinCVS (client)/CVSNT (server) or Subversion for version control, ArgoUML as a UML modeler, IBM Eclipse for multi-purpose authoring environment, Doxygen for a general programming language documentation or CCDoc for C++-specific documentation, Bugzilla for issue tracking, Apache Ant for building, and JUnit for unit testing, to name a few. Indeed, the author has successfully used all of these in University courses on object-oriented software design.

In spite of their usefulness, certain hindrances of adopting OSS as a CASE tool remain. OSS are not always feature-rich in comparison to their non-OSS counterparts, technical support can be scarce, may not be timely, and is not guaranteed (if sought from people external to the institution), the OSS utilities used may not be interoperable with each other, or students may find "all-in-one" multi-utility packaged commercial integrated development environments (IDE) more convenient to use for programming purposes than individual isolated pieces of software.

OSS as a Sub-System

Reinventing the wheel is considered inertia in software development and, at times, is not practical. For example, it is not always realistic to develop everything that is required from scratch for a non-trivial software project.

OSS can be used as auxiliary software and can thereby support the software system under development. In that regard, OSS support has in general been exemplary. A systematic approach for creating Web Applications has been termed as Web Engineering (Kappel, Pröll, Reich, & Retschitzegger, 2006), and OSS has played a crucial role in advancing this discipline.

Indeed, the author's experience with the support of OSS in Web Engineering for applications such as Course Registration System, Fine Art Auction System, Software Patterns Management System, and Student Personal Information Portal, has in general been quite encouraging.

Still, as in the previous section, the issue of technical support remains. Also, not all educational institutions may be willing to allow the installation and use of software for a one semester use for a single course.

Elements of OSS for Reuse

This approach to OSS in software engineering advocates reuse of portions of OSS code in assignments or as part of the system under development in, say, a course project. Examples of reusable entities include object-oriented software frameworks and libraries. Such opportunities of reuse ameliorate the tedium of writing the entire source code from scratch, particularly for routine primitive functions such as creating a user interface menu bar, finding the singular value decomposition of a matrix, or drawing an elliptic hyperboloid.

We note here that reuse is neither truly free, nor automatic. It is well-known from the COCOMO II cost estimation model (Boehm et al., 2001), reuse comes at a price of learning and adapting to new situations. Efforts of reuse that are not an integral part of planning at the outset of a software project can be detrimental to productivity (Long, 2001) and maintainability.

Some issues in OSS reuse are: students treating reused code as a "black box" without really understanding the internals, the degree to which reuse should be allowed, and that of appropriate acknowledgement.

There are some partial resolutions to these issues. For example, to minimize reuse of OSS without reflection, students could also be questioned to reflect understanding of any reused code during project demonstration or otherwise. Given more than one option for the use of an OSS as a subsystem, students could be asked how and why they chose one over the other. Possible criteria for choice of a sub-system could be: availability, ease of installation, interoperability with the system being built, portability, and past experience. The educators could set criteria for the degree of reused open source code and make it known to students a priori. The students could be asked to formally declare any open source code reuse and to provide precise articulation to that regard, say, in their process documents.

Finally, the issue of evaluating work based on reuse, particularly when it has to be balanced against originality, remains a challenge and raises several questions. For example, if the usability of software `A` (45% original, 55% reuse) is deemed much better than software `B` (55% original, 45% reuse), should `A` be graded higher than `B` if it is known that it was the reuse in `A` that made the difference? Similarly, should a team be penalized for *merely selecting* a software library that they did not know at the time of use had subtle security flaws that only became explicit after repetitive use? Should the degree of reuse be related to the project schedule? For example, should the differential in the amount of reuse across project teams be reflected in their respective time lines for the final project submission? The precise answers to these questions are likely to depend on the context of reuse.

Example of a Course Project with OSS

In one of semester of an undergraduate course on software engineering, the author supervised a project that required the students to build a Course Registration System as a Web-based application. The choice of the topic was based on the familiarity of students with the domain.

The system would enable users to use the Internet/Web to check the list of available courses, their descriptions, and pre-requisites; the courses they are currently registered in and related logistics (instructor, schedule, and classroom location); and be able to request the administrators to register in a particular course. It would enable administrators to add or delete courses, or modify the information on an existing course, and register a student (based on a variety of criteria including necessary pre-requisites and limits on the class size).

The class of about 50 students was divided into teams of size around 10 each. One student from each team voluntarily took upon the role of the team leader so as to manage the team and to act as the liaison between the author and the members of the corresponding team.

The students were given complete freedom of choice of the underlying technology except that the process artifacts and the final system would have to be entirely based on OSS.

The teams on their own set up and used MediaWiki to collaborate. The author and teaching assistants were given access to these groups. The posting and responding to messages became an implicit determinant of the level of participation and contribution by members of a team and helped the author question those that were passive. It also prevented the process to become a "black box" and helped in timely submission of the intermediate deliverables and of the completion of the project itself.

The teams initially used CVS for configuration management of documents and models but found it a bit difficult to learn at first. Later on, some teams switched to Subversion.

The process documents (specifically, project management, requirements, design, and test plan) were created using Open Office and the models (specifically, domain and use case models) were based on UML and created using ArgoUML. All modeling was carried out in a collaborative manner (Kamthan, 2005).

There was heavy emphasis of *conceptual* reuse in form of software design patterns (Appleton, 1997; Van Duyne, Landay, & Hong, 2003) and the students made broad use of it. The opportunities for source code reuse were deliberately limited to 10% and students were asked to explicitly declare any reuse. At the same time, they were encouraged to reuse functions that provide simple functionality such as basic (username/password-based) authentication, navigation, and searching.

For the final product, some teams used Amaya as the user agent on the client-side and the Apache Web Server along with JBoss and Apache Tomcat for dynamic delivery of resources on the server-side, while the others preferred the combination of Mozilla FireFox and MySQL/PHP Hypertext Preprocessor (PHP). This differential was attributed to the diverse background of courses that the students had previously taken.

Upon completion of the project, each member of the team was asked to submit (in confidence) a short individual report summarizing their project experience to the author. The individual report served as an informal retrospective.

The project was grades based on two set of marks, one that applied to individual students in a team and another to the respective team as a whole.

There were five main challenges faced during the duration of the project:

1. At times, some students articulated that the responsibility of installing and administering certain OSS on their own was, particularly during heavy course load, a bit burdening.
2. The need for a requirements management (such as for traceability) was felt but no suitable OSS options were available. This issue was partially resolved by making the

process documents available on the Web and using hyperlinks to represent relationships.

3. Although students found the learning curve of ArgoUML to be low and the rich number of export formats to be useful, they also noted that the support for UML and the diagram quality were inadequate.
4. Since most of the "industrial strength" software available for quality assurance and evaluation has been traditionally commercial, addressing quality control was another challenge. The students used the OSS tools from the World Wide Web Consortium (W3C) for automatically evaluating the quality of documents in Extensible HTML (XHTML) and the style sheets in CSS. However, the evaluation was limited to conformance to syntax and rudimentary checking for accessibility, usability, and security.
5. Due to security considerations, the students were not allowed to run any network software (and therefore the sub-systems necessary for the project) on the University computer network. The resolution found to this issue was that the running and testing of the executable software would be done on a laptop computer belonging to one of the students (preferably the team leader, if possible). In retrospective, this made the students appreciate the possibility of having configuration management on the Web (that in turn was based on an OSS). The latest version of all source code could always be created in an independent manner, uploaded, and made available to the rest of the team rather than being centralized on a single computer.

In general, the students found carrying out the project to be a rewarding experience. The students came away with the impression that for certain early aspects of the process, OSS is yet to evolve at the level of competing commercial software. However, they found the OSS support at the implementation phase to be satisfactory in general. They were excited by the freedom and flexibility to try new OSS out on their own. At the same time, the students did find the author's intervention to be useful. For example, they did appreciate input in suggesting possibilities and assisting in selection. Although some students did not have the requisite background in the aforementioned technologies but were (with the help of their peers and teaching assistants) willing to learn based on the likelihood that these technologies will useful in other courses and their future careers. Indeed, some of the students after graduation have found employment in industrial sectors that use OSS.

Guidelines for Incorporating OSS in SEE

In Table 3, based on the prior discussion and our experience with OS2SE2, we present some of the trade-offs that could help in decision making towards the use of the OSS environment in SEE contexts.

Table 3. A perspective on the use of OSS in SEE.

General/Administrative	*Possibilities* • The prospect for educational institutions to be able to make available a broad collection of software without incurring heavy costs as well as be able to provide OSS utilities for which there are no commercial parallels. • The flexibility of trying out different OSS and examining them at any level of desirable detail prior to making a commitment. *Obstacles* • There may be competing interests such as an institution's long-term commitment to a commercial software vendor. • There may be reluctance from system administrators to install, administer, and maintain the OSS if it is only meant to be used in one semester and/or in a single course. • There may be reluctance from faculty's administrators to adopt an OSS on a "large-scale" (such as for several courses or an entire program)
Teaching/Learning	*Possibilities* • The opportunity for both teachers and students to experiment (for example, with source code internals) more freely, which can be in agreement with the spirit of a constructivist approach to teaching and learning. • The prospect for students to incrementally develop their own personal collection of tools specialized for various tasks (modelers, compilers, debuggers, and so on) in a software project within minimal cost, if at all. • The opportunity for students to contribute to an existing OSS in various directions (such as reengineering, reverse engineering, discovery of software design patterns, or extensions via implementation of further modules). • The opportunity for students to participate in the development of an OSS for their own software projects. *Obstacles* • In many cases, there are no explicit guarantees for technical support when needed. • The use of OSS, particularly those whose scope of testing is not known, may not be suitable in context of safety-critical software. • It can be difficult to make objective assessments of software projects that make broad reuse of open source code.

Some Directions for Future Research

The work presented in this paper can be extended in a few different directions, which we now briefly discuss.

It may be useful to conduct a formal survey to empirically "validate" some of the ideas proposed in this paper and, indeed, the author has attempted to do so. However, for that to be effective, the data size of the participants (teachers and students) should be reasonable large and the participants should have roughly similar level of skills and/or experience with the set of OSS being used in the statistical analysis.

In recent years, there has been an increasing crosspollination between OSS and e-learning (Koohang & Harman, 2005) as exemplified by the emergence of OSS projects such as Sakai, EduZope, and Moodle. The Open Sources Education (Tadeusz & Ostrowska, 2006) is a platform for e-learning that has been applied to management courses in Universities in Poland. An investigation into the effectiveness of the adaptation and use of these OSS e-learning projects within the SEE context would be of interest.

Among the possible domains that OSS addresses (Nakakoji & Yamamoto, 2001), it would be of interest to examine the ones more congruent to software engineering. OSS has already had a major impact on Web Applications and Web Services but their broad use in real-time and embedded systems is yet to be seen. In these cases, it will remain important with regards to the goals of learning that the attention of the students remain on the topic at hand rather than on the manipulations of the technologies (Hadjerrouit, 2005).

The consideration of human factor is important in both teaching and learning. In feedback to the author over the years, students find it important that the subject being communicated is fun to learn, and OSS can provide that avenue (Luthiger, 2005). Computer games offer a variety of technical challenges related to user interface design/interaction design and incorporation of three-dimensional (3D) graphics. They can also introduce many of the software metaphors (Boyd, 1999) without resorting to unnecessary terminology. Introducing such games as part of software projects (Rucker, 2002), and the use of OSS libraries to realize that, would be of interest.

Among the open source possibilities, this paper focuses mostly on *software*; a natural extension of this work would be to look into the use of *Open Content* (excluding source code) in software engineering. The aim of open content is to "facilitate the prolific creation of freely available, high-quality, well-maintained content" (not including software). The significance of Open Content for education in general has been highlighted (Attwell, 2005). The continually increasing price of textbooks, none of which may be suitable as-is to a given course, is one motivation for open content in SEE.

The Directory of Open Access Journals (DOAJ), CEUR Workshop Proceedings, and the Informing Science Institute are examples of services that make their quality controlled scientific and scholarly publications in various areas related to software engineering available to the general public electronically and free-of-charge. MIT OpenCourseWare, Rice Connexions, and Open Course are initiatives for making educational resources used in courses at one institution available to the public at-large. Such services could help level the playing field and open new vistas in research-oriented higher educational contexts in the software engineering discipline, particularly where affordability is an issue.

OSS and Open Content are not mutually exclusive. In fact, OSS such as the Open Conference System and the Open Journal System can create the environment for making scholarly Open Content available and manageable.

Students need to acquire knowledge of the theoretical underpinnings of an OSS in order to build such systems in the future. Indeed, learning objects in form of Open Content books and journals and openly availably course material describing the use and applications of an OSS are *complementary* to the OSS itself. It is *together* that they can move towards a paradigm shift: a complete open environment for modern SEE.

Conclusion

From its modest beginnings, OSS has today reached the level of maturity that it could be embraced as well as criticized, but not dismissed as an ephemeral "social experiment." The progress in human civilizations has often been measured by a movement away from intransigence; OSS is one part of this evolution.

If the predictions of software business models (Cusumano, 2004; Feller, Fitzgerald, Hissam, & Lakhani, 2005) are correct, OSS and non-OSS will continue to co-exist. Both OSS and non-OSS have their own share of strengths and weaknesses, are most likely to co-exist, and any approach to software development should take them into consideration collectively.

There is much that software engineering and commercial OSSD can learn from each other (Asundi, 2001). Indeed, recent industrial support of OSS efforts has led to mutual benefits. The support of Linux by IBM, of Mozilla FireFox by (the now defunct) Netscape, and of Java Community Process (JCP) by Sun Microsystems, are exemplars in this regard.

If one of the goals of SEE is to prepare students for their future careers, we must look at the OSS objectively. OSS has much to offer to SEE, however, as this paper has shown, the transition from one to the other is hardly straightforward. However, the adoption of OSS in education need not be seen with skepticism but rather with cautious optimism. The proposal of OS2SE2 framework in this paper is one step in that direction.

In conclusion, OSS is bringing about significant change in the way software is being developed and used. To embrace this change requires a reflection and reexamination of the current state of the curriculum. For that to come to a realization, the current software engineering culture (Wiegers, 1996) in educational institutions will need to evolve.

Acknowledgments

The author would like to thank his undergraduate and graduate students from courses over the years for their participation, and Hsueh-Ieng Pai (Concordia University, Montreal, Canada) and the reviewers for feedback and suggestions for improvement.

References

Aberdour, M. (2007). Achieving quality in open source software. *IEEE Software, 24*(1), 58-64.

Abran, A., Moore, J. W., Bourque, P., & Dupuis, R. (2001). *Guide to the software engineering body of knowledge - SWEBOK*. IEEE Computer Society.

Alagar, V. S., & Periyasamy, K. (1998). *Specification of software systems*. Springer-Verlag.

Andrews, J. H., & Lutfiyya, H. L. (2000). Experiences with a software maintenance project course. *IEEE Transactions on Education, 43*(4), 383-388.

Appleton, B. A. (1997). Patterns and software: Essential concepts and terminology. *Object Magazine Online, 3*(5), 20-25.

Asklund, U., & Bendix, L. (2001). Configuration management for open source software. *First Workshop on Open Source Software Engineering*, Toronto, Canada, May 15, 2001.

Asundi, J. (2001). Software engineering lessons from open source projects. *First Workshop on Open Source Software Engineering,* Toronto, Canada, May 15, 2001.

Attwell, G. (2005). What is the significance of open source software for the education and training community? *The First International Conference on Open Source Systems* (OSS 2005), Genova, Italy, July 11-15, 2005.

Beck, K., & Andres, C. (2005). *Extreme programming explained: Embrace change* (2nd ed.). Addison-Wesley.

Beydeda, S., Book, M., & Gruhn, V. (2005). *Model-driven software development*. Springer-Verlag.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S, Clark, B. K, Horowitz, E., Madachy, R., Reifer, D., & Steece, B. (2001). *Software cost estimation with COCOMO II*. Prentice Hall.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley.

Booch, G., Jacobson, I., & Rumbaugh, J. (2005). *The unified modeling language reference manual* (2nd ed.). Addison-Wesley.

Boyd, N. S. (1999). Using natural language in software development. *Journal of Object-Oriented Programming, 11*(9), 45-55.

Clemen, R. T. (1996). **Making hard decisions: An introduction to decision analysis** (2nd ed.). Duxbury Press.

Cowling, A. J. (2005). The role of modelling in the software engineering curriculum. *Journal of Systems and Software, 75*(1-2), 41-53.

Crowston, K., & Howison, J. (2005). The social structure of free and open source software development. *First Monday, 10*(2).

Cusumano, M. A. (2004). Reflections on free and open software. *Communications of the ACM, 47*(10), 25-27.

Derby, E., Larsen, D., & Schwaber, K. (2006). *Agile retrospectives: Making good teams great*. O'Reilly Media.

Feller, J., Fitzgerald, B., Hissam, S. A., & Lakhani, K. R. (2005). *Perspectives on free and open source software*. MIT Press.

Fenton, N. E., & Pfleeger, S. L. (1997). *Software metrics: A rigorous & practical approach*. International Thomson Computer Press.

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). *Refactoring: Improving the design of existing code*. Addison-Wesley.

Gacek, C., & Arief, B. (2004). The many meanings of open source. *IEEE Software, 21*(1), 34-40.

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (2003). *Fundamentals of software engineering* (2nd ed.). Prentice-Hall.

Glass, R. L. (2003). *Facts and fallacies of software engineering*. Addison-Wesley.

Golden, B. (2004). *Succeeding with open source*. Addison-Wesley.

González-Barahona, J. M., Heras-Quirós, P. D. L., Centeno-González, J., Matellán-Olivera, & Ballesteros-Cámara, F. (2000). Libre software for computer science classes. *IEEE Software, 17*(3), 76-79.

Hadjerrouit, S. (2005). Designing a pedagogical model for web engineering education: An evolutionary perspective. *Journal of Information Technology Education, 4*, 115-140. Available at http://jite.org/documents/Vol4/v4p115-140Hadj50.pdf

Halloran, T. J., & Scherlis, W. L. (2002). High quality and open source software practices. *Second Workshop on Open Source Software Engineering*, Orlando, Florida, USA, May 25, 2002.

IEEE-CS/ACM. (2004). *Software engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering (SE 2004)*. Institute of Electrical and Electronics Engineers Computer Society (IEEE-CS)/Association for Computing Machinery (ACM) Steering Committee. August 23, 2004.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process*. Addison-Wesley.

Kamthan, P. (2004). A framework for addressing the quality of UML artifacts. *Studies in Communication Sciences, 4*(2), 85-114.

Kamthan, P. (2005). Pair modeling. *The 2005 Canadian University Software Engineering Conference (CUSEC 2005)*, Ottawa, Canada, January 14-16, 2005.

Kamthan, P. (2006). Open source software in software engineering education: No free lunch. *The 2006 Canadian University Software Engineering Conference (CUSEC 2006),* Montreal, Canada, January 19-21, 2006.

Kappel, G., Pröll, B., Reich, S., & Retschitzegger, W. (2006). *Web engineering*. John Wiley and Sons.

Kerth, N. L. (2001). *Project retrospectives: A handbook for team reviews*. Dorset House.

Koohang, A., & Harman, K. (2005). Open source: A metaphor for e-learning. *Informing Science Journal, 8*, 75-86. Available at http://inform.nu/Articles/Vol8/v8p075-086Kooh.pdf

Kruchten, P. (2004). *The rational unified process: An introduction* (3rd ed.). Addison-Wesley.

Liu, C. (2003). Adopting open-source software engineering in computer science education. *The Third Workshop on Open Source Software Engineering*, Portland, Oregon, USA, May 3, 2003.

Long, J. (2001). Software reuse antipatterns. *ACM SIGSOFT Software Engineering Notes, 26*(4), 68-76.

Luthiger, B. (2005). Fun and software development. *The First International Conference on Open Source Systems (OSS 2005)*, Genova, Italy, July 11-15, 2005.

Martin, K., & Hoffman, B. (2007). An open source approach to developing software in a small organization. *IEEE Software, 24*(1), 46-53.

Michlmayr, M., Hunt, F., & Probert, D. R. (2005). Quality practices and problems in free software projects. *The First International Conference on Open Source Systems (OSS 2005)*, Genova, Italy, July 11-15, 2005.

Nakakoji, K. & Yamamoto, Y. (2001). Taxonomy of open source software development. *First Workshop on Open Source Software Engineering*, Toronto, Canada, May 15, 2001.

Nishinaka, Y. (2001). Open source software developments in XP Style. *First Workshop on Open Source Software Engineering*, Toronto, Canada, May 15, 2001.

O'Reilly, T. (2005). *What is web 2.0: Design patterns and business models for the next generation of software*. O'Reilly Network, September 30, 2005.

Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. (1995). *The capability maturity model: Guidelines for improving the software process.* Addison-Wesley.

Perens, B. (1999). The Open Source Definition. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open sources: Voices from the open source revolution*. O'Reilly & Associates.

Piaget, J. (1971). *Genetic epistemology.* W. W. Norton & Company.

Porter, A., Yilmaz, C., Memon, A. M., Krishna, A. S., Schmidt, D. C., & Gokhale, A. (2006). Techniques and processes for improving the quality and performance of open-source software. *Software Process: Improvement and Practice, 11*(2), 163-176.

Raymond, E. S. (1999). *The cathedral & the bazaar.* O'Reilly & Associates.

Raymond, E. S. (2004). *The art of UNIX programming.* Addison-Wesley.

Rucker, R. (2002). *Software engineering and computer games*. Addison-Wesley.

Qureshi, S. (2001). How practical is a code of ethics for software engineers interested in quality? *Software Quality Journal, 9*(3), 153-159.

Schmidt, D. C., & Porter, A. (2001). Leveraging open-source communities to improve the quality and performance of open-source software. *First Workshop on Open Source Software Engineering*, Toronto, Canada, May 15, 2001.

Schneidewind, N. F., & Fenton, N. E. (1996). Do standards improve product quality? *IEEE Software, 13*(1), 22-24.

Seidel, W., & Niedermeier, C. (2003). Open source software: Leveraging software quality in the industrial context. *First Workshop on Open Source Software in an Industrial Environment (OSSIE 2003)*, Erfurt, Germany, September 23, 2003.

Shaw, M. (2000). Software engineering education: A roadmap. *The Twenty Second International Conference on Software Engineering (ICSE 2000)*, Limerick, Ireland, June 4-11, 2000.

Spinellis, D., & Szyperski, C. (2004). How is open source affecting software development? *IEEE Software, 21*(1), 28-33.

Tadeusz, K., & Ostrowska, T. (2006). The open sources education: A real time education. *The Seventeenth Annual Information Resources Management Association International Conference (IRMA 2006)*, Washington, D.C., USA, May 21-24, 2006.

Thomas, D., & Hunt, A. (2004). Open source ecosystems. *IEEE Software, 21*(4), 89-91.

Van Duyne, D. K., Landay, J., & Hong, J. I. (2003). *The design of sites: Patterns, principles, and processes for crafting a customer-centered web experience.* Addison-Wesley.

Verma, S. (2006). Software quality and the open source process. In E. Duggan, & J. Reichgelt (Eds.), *Measuring information systems delivery quality* (pp. 291-310). Idea Group.

Vixie, P. (1999). Software Engineering. In C. DiBona, S. Ockman, & M. Stone (Eds.), Ope*n sources: Voices from the open source revolution.* O'Reilly & Associates.

Warsta, J., & Abrahamsson, P. (2003). Is open source software development essentially an agile method? *The Third Workshop on Open Source Software Engineering*, Portland, Oregon, USA, May 3, 2003.

Weiss, M., Moroiu, G., & Zhao, P. (2006). Evolution of open source communities. *The Second International Conference on Open Source Systems (OSS 2006)*. Como, Italy. June 8-10, 2006.

Wiegers, K. (1996). *Creating a software engineering culture.* Dorset House.

Wong, B. (2006). Different views of software quality. In E. Duggan & J. Reichgelt (Eds.), *Measuring information systems delivery quality*. Idea Group, 55-88.

Woods, D., & Guliani, G. (2005). *Open source for the enterprise.* O'Reilly Media.

Biography

Pankaj Kamthan has been teaching in academia and industry for several years. He has also been a technical editor and participated in standards development. His professional interests and experience include software quality, markup languages, and knowledge representation.