Introducing a "Street Fair" Open source Practice Within Project Based Software Engineering Courses

A position paper submitted to 1st Workshop on Open Source Software Engineering

Dan Port
University of Southern California
dport@sunset.usc.edu

Gail Kaiser
Columbia University
kaiser@cs.columbia.edu

Columbia University and USC, as well as many other universities, have software engineering courses where students work in small teams to develop a product of moderate complexity. Typically such efforts are "closed source" where teams do not (or are forbidden) from sharing information and providing assistance to other teams. There are certainly many justifiable reasons for this such as grading uniformity. Yet strong drivers such as grade competition, generally thought to promote quality results (i.e. successful projects), often do not. The authors ran an experiment introducing a style of open source practice called "street fair" with surprisingly positive results.

In describing the Open Source development methodology, Raymond [1] draws an analogy between traditional closed source development and a "cathedral", in which there is a rigid hierarchy among developers, managers, testers, etc. He likens open source projects to Middle Eastern bazaars, in which numerous merchants hawk their wares loudly to passersby. While there is often a small set of principal developers in an open source project, there is little hierarchy among them. Contributors compete to have their modifications inserted into the next release of the system, bringing recognition and reputation. Although the famous open source projects involve hoards of developers, and that is often viewed as one of the key contributions of open source [2], such global scale projects seem likely to remain very rare. We seek to draw concepts from open source for small systems engineering, and vice versa. But the bazaar methodology, in essence, exudes methodology. Although to some extent that's its point, this seems too chaotic and immature, even for small projects. A magazine article [3] expounding the virtues of the "grownup" methodology employed for the extraordinarily bug-free NASA space shuttle software described the dominant image of the software development world as:

> *Gen-Xers sporting T-shirts and distracted looks, squeezing too much heroic code writing into too little time; rollerblades and mountain bikes tucked in corners; pizza boxes and Starbucks cups discarded in conference rooms; dueling tunes from Smashing Pumpkins, Alans Morrisette and the Fugees. Its the world made famous, romantic, even inevitable by stories out of Sun Microsystems, Microsoft, and Netscape.*

This kind of atmosphere is obviously not going to produce well-engineered software systems, outside a few heroic efforts like Linux and Apache. Note that typically bazaars are widely associated with interminable haggling.

We propose to pursue a more controlled style of open development, which we call the "street fair" model. The openness here concerns processes and methods, as well as code. Street fair builds on the two-pronged team-interaction structure we introduced to help our Spring 1999 3156/4156 students deal with the complexities of software engineering models (e.g. requirements, UML diagrams, lifecycle, etc.). First, we provided a common Web-based repository where project materials from *all* teams was intentionally made available to *all* students enrolled in the class. Recent offerings of 3156/4156 and other CS courses (including the software engineering courses at USC) have featured project team websites, generally intended for perusal only by project members, teaching assistants, and the course instructor, but **not** for other students in the class. Second, in addition to the usual lectures and recitations, students attended and often led "workshops" (on software design, team management, etc.) where they openly discussed development issues. Students were encouraged to interact freely with other teams to discover and utilize class "best practices", review other teams' deliverables, test and inspect other teams' code, volunteer technical assistance, and so forth.

In a conventional computer science course, the inter-team interactions promoted by street fair might be considered "cheating", so measures were taken to avoid plagiarism and we encountered none. Since the goal of software engineering education is to teach good practices to the students, the benefits of rapid knowledge and "best practice" propagation far outweigh the cost of extra vigilance on the part of the course managers. Outright plagiarism should be rare and obvious due to the unique makeup of a project and team combination (i.e., different constraints, skill sets, project concepts, etc.). The street fair model might be more problematic in industry, where projects are competing for customer dollars and market leadership rather than curved grades. There the "street" might be limited to within a company or organization, or a cooperating consortium– such as those firms supporting MCC, SPC, Telcordia, etc. Conventional information management access controls could be employed to restrict outsiders to only see process, not product, requiring clear distinction of the two.

One nice but rarely advertised feature of the open source movement is that contributions are color-blind: recipients/beneficiaries of one's ideas and/or scholarly habits cannot distinguish race, color, religion, national origin, gender, sexual orientation, etc. Disability status at worst leads to relatively slow response time, also characteristic of over-committed fully-abled people

This leads us to considering requirements on supporting information management structures, as well as investigating how, and which, information management structures can assist street fair style small system engineering methodology. Consider a real example from the Spring 1999 Columbia courses, which involved the task schedule students were required to develop within the Life Cycle Plan. Team 12-2 looked at the example MS Project Gantt chart from the MBASE task schedule guidelines, but felt it was "too much" for their little plague-ware map terrain generator. Two of their team members were off-campus "video network" students and another on-campus spoke little English. They decided to use instead a Visio template called "box schedule". Prof. Port mentioned in class how much he liked team 12-2's novel schedule model. Several other teams looked up team 12-2's schedule after class and adopted the box schedule template. Eventually five of 33 teams were using box schedule templates for the task schedule in the LCP. But many other teams were either not in class the day this came up, not paying attention, or enrolled in the course's other section so could not hear the instructor's comment about team 12-2.

Humphrey has developed a highly regarded industrial-strength team software process (TSP) as well as a textbook-variant introductory team software process (TSPi) for small teams [4]. However, both processes assume that all team members have previously completed substantial training in his personal software process (PSP [5]). Said background is clearly preferable, but pragmatically unrealistic for small software firms, short-term virtual teams for small-scale open source development, and undergraduate programs with space for only one single-semester software engineering course. Although as true for larger scale efforts, having a well-structured set of software engineering models is vital to the endeavors listed previously. Such software engineering models provide means of managing and communicating complex project development information in a uniform way and hence serves as a framework for an open source style development effort between heterogeneous projects. One such framework we have much experience with is MBASE.

The Model-Based [System] Architecting and Software Engineering (MBASE) methodology was developed by Barry Boehm, Dan Port and others at USC. MBASE is a comprehensive set of detailed guidelines that describe techniques for the creation and integration of software engineering models. The models to be integrated extend beyond Product models such as object-oriented analysis and design models and traditional requirements, to include Process models such as lifecycle and risk models, Property models such as cost and schedule, and most notably Success models such as business-case analysis and win-win. Paramount in the MBASE paradigm is the identification and resolution of the hidden conflicts among the models the software system stakeholders bring to a software project. Intended to support a wide range of large-scale systems engineering endeavors, MBASE is broad and "feature-packed". MBASE is continuously refined in an annual cycle based on analysis of feedback from a combination of industry adopters and graduate students in a two-semester course sequence centered on a team software engineering project that attempts to simulate the full lifecycle of large scale industry projects.

MBASE and the associated street fair model are intended to "jump start" small projects that cannot meet the time, budget, management and other resource requirements of TSP and its cohorts.

Empirical studies have considered variations in team size for certain aspects of the software lifecycle, generally in the context of large-scale software development. For instance, Porter *et al.* [6] describe a study where the code inspection team was kept small (1, 2 or 4 members), and the full team developing that 65kloc system included only six people plus five developers from other projects augmenting the inspector pool. But data was collected over 18 months, implying the project was active at least that long, and the software under development was a supporting tool for an archetype large system, Lucent Technology's 5ESS. Another study involving 5ESS could conceivably have hypothesized that smaller teams might be better, given that Perry and Evangelist [7] found that the majority of problems detected only during system integration (i.e., late) were due to interface errors, presumably among modules developed by different team members who weren't communicating as well as they should.

There are a great number of software development methodologies concerned with "cathedral" large scale software engineering, such as Rational's Objectory process [8], and Jacobson's OOSE [9]. However, MBASE has several distinct characteristics that make it preferable as a starting point: in contrast to many development methodologies, MBASE directly addresses the critical issues of model integration and model clashes. There is already substantial experience adapting the MBASE goal oriented, risk driven milestones (LCO, LCA, IOC, etc.) to course curricula for evaluation. However, we propose to identify at least one other large-scale methodology amenable to our invariant/variant scale-down and street fair oral history approaches, and then conduct a "paper study"; it is not feasible to apply the classroom-based methodology evaluation model to significantly competing approaches at the same time.

Our position is that an open source-style paradigm for public perusal of software development models, processes and methods will significantly improve the quality of project deliverables and ultimately project products themselves. A moderately controlled "street fair" model (in analogy to Eric Raymond's uncontrolled "bazaar") will structure intra and inter team interactions to enable particularly promising MBASE methods created or identified for one project to be openly shared with other projects of a similar scope or nature. More provocatively, the methods created or identified by teams could be opened to public scrutiny, to enable rapid dissemination of "best practices" (and weeding out of bad ideas), similar to the open source movement for code.

References

[1.] Raymond, The Cathedral and the Bazaar, 1997. http://www.tuxedo.org/~esr/writings/cathedral-bazaar/.

[2.] Mann, Programs to the People, Technology Review, January/February 1999. http://www.techreview.com/articles/jan99/mann.htm.

[3.] Fishman, They Write the Right Stuff, Fast Company, 6, December 1996/January 1997, http://www.fastcompany.com/online/06/writestuff.html.

[4.] Humphrey, Introduction to the Team Software Process, Addison-Wesley, 2000.

[5.] Humphrey, Introduction to the Personal Software Process, Addison-Wesley, 1997.

[6.] Porter, Siy, Toman and Votta, An experiment to assess the cost-benefits of code inspections in large scale software development, 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, October 1995.

[7.] Perry and Evangelist, An Empirical Study of Software Interface Errors, International Symposium on New Directions in Computing, August 1985.

[8.] Kruchten., The Rational Unified Process, Addison-Wesley, 1998.

[9.] Jacobson, Christerson, Jonsson and Overgaard, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1993