Corporate-, Agile- and Open Source Software Development: A Witch's Brew or An Elixir of Life?

Morkel Theunissen, Derrick Kourie, and Andrew Boake

Espresso Research Group, Department of Computer Science,
University of Pretoria
{mtheunis,dkourie}@cs.up.ac.za, Andrew.Boake@absa.co.za

Abstract. The observation that the Open Source Software development style is becoming part of corporate software development, raises questions about its compatibility with traditional development processes. Particular compatibility questions arise where the existing corporate development style is in the agile tradition. These questions are identified and discussed. Measures that can be taken to avoid clashes are suggested. An example illustrates how Corporate-, Agile- and Open Source Software could intersect, and SPEM modelling is employed to show how corporate processes would need to adapt to accommodate the new scenario.

Keywords: Open source software development, Agile software development, Corporate software development, Compatibility.

1 Introduction

> *"Double, double, toile and trouble; Fire burne, and Cauldron bubble."*
> —Macbeth (Act IV, Scene 1)

A dispassionate consideration of the cauldron of forces at play in corporate[1] software development, may well raise the question of whether two contemporary (and apparently orthogonal) approaches belong in the brew: Agile Software Development (ASD) and Open Source Software Development (OSSD). Over the past decade, the impact that each of these paradigms has had on the software development industry has grown, and there are signs that this trend will continue. It therefore seems relevant to consider the extent to which these paradigms are mutually exclusive, and, conversely, whether synergies between them can be found. Could blending them into corporate software development processes produce an elixir of life[2] or will they combine into a poisonous witches' brew?

[1] The term *corporate* is used to reference a medium to large enterprise that has its own in-house software developers.

[2] Using this metaphor does not imply that we believe that a silver bullet might be at hand—to reach for another well-used metaphor in software development contexts. However, sobriety does not negate the validity of aspiring to an optimal approach in software development endeavours.

B. Meyer, J.R. Nawrocki, and B. Walter (Eds.): CEE-SET 2007, LNCS 5082, pp. 84–95, 2008.

Although not universally practiced, ASD is already widely represented in many industries and, to this extent, is already in the cauldron. Open Source Software (OSS), too, plays a significant role in many corporate contexts. However, this role varies widely: from simple usage of OSS (as in the provision of internet infrastructure components and development tools); to attempts by corporates to leverage the energy inherent in OSSD by managing such projects (the Eclipse project of IBM being a prominent example). There are many variants between these extremes, such as making casual contributions to OSS as a byproduct of using it. An example of just such a scenario will be presented below. Some corporates have even adopted an in-house OSS development style (HP's POS approach being a case in point [1]).

The present discussion excludes scenarios such as the latter, namely where the corporate fully controls the development style. It also excludes simple, uninvolved OSS usage. The corporate OSSD activity that remains and that is the subject of the present discussion (i.e. development which is only partially controlled by corporates), though fairly limited, nevertheless has been growing quite significantly, and could become a disruptive force in the development processes into which it is supposed to blend.

For the purpose of this discussion the extent of corporate participation and the OSS project's scope and size will not be considered. Instead a more generic view of the forces at work is taken. Scope and size influences are therefore a matter for further study.

It should be noted at the outset that OSSD is not a formally defined development methodology. Although every OSS community uses its own process, there is nevertheless a common overall philosophy. For the purposes of this paper OSSD is taken to refer to development that is more or less in line with common principles that have emerged from prominent OSS projects.

Section 2 briefly surveys the nature of corporates, difficulties they are likely to encounter when attempting to engage in OSSD, and managerial adjustments that will be needed. This discussion does not specifically refer to ASD. Instead, the connection between ASD and OSSD is left to Section 3. Here it is pointed out that the OSSD and ASD paradigms apparently embody contradictory attributes, and that consequently, any attempt by a corporate to simultaneously engage in both would appear to be particularly fraught with difficulties. The section also points to ways in which the identified tensions could be managed. An example to illustrate further the kinds of difficulties that might be anticipated is given in Section 4 and SPEM is used as a medium for illustrating ways in which processes might be adapted to alleviate difficulties. Section 5 proposes general ways of ameliorating difficulties that have been identidifed, before concluding in Section 6.

2 OSSD and Corporate Culture

OSS refers to software developed by a movement that values a distributed, open, collaborative development model, as well as the free distribution and

modification of its software. As mentioned before OSS already plays a significant role in many corporate software development contexts and as such have been scrutinised by many. Our aim is only to highlight the elements relevant to the discussion at hand.

The corporate environment conventionally places certain requirements on the software development process to enforce employee accountability. This imposes a number of stresses on a development team in such an environment—stresses that will inevitably be accentuated when attempting to engage in OSSD. The following items illustrate some of the clashes that could occur between common corporate culture and that of people typically engaged in OSSD:

- *Monitoring of developers*
  In an environment where remuneration for work is the norm, there is a need to manage and monitor employees, and this is generally taken for granted by regular corporate employees. However, participants in traditional OSS projects are not subjected to such regulation, due to the voluntary nature of the development. In the corporate paradigm, once a manager has assigned a task to a subordinate, it is normally assumed that the manager will track the subordinate's progress and activities, and respond appropriately. This scenario can become complicated when an OSSD style is used internally. It is difficult to monitor what developers are contributing to different and disjoint OSS efforts. Furthermore, it may be difficult for management to assess the importance or relevance of an OSS contribution that is not directly used by the organisation.
- *Fixed time schedules*
  Traditional OSS projects live by the principle of "release often", but these releases are largely *ad hoc*, occurring whenever the core maintainers feel that it is time to do a release. Within the corporate environment there is a need to link different software development projects to fixed time frames so as to support business-driven goals such as taking advantage of market windows and managing financial aspects such as Return-On-Investment (ROI), and IT-driven goals such as coordinated roll-out of related projects.
- *Quality Assurance Processes*
  OSSD, by its very nature, encourages extensive peer review. One of the underlying notions in OSS is expressed in the aphorism known as Linus' Law: "Given enough eyeballs, all bugs are shallow." [2]. However, although some OSS projects may apply certain rules prior to accepting contributions (patches), there are generally no formal OSS code review processes (in particular between the core members). In contrast, in both the agile paradigm, and in many other traditional software engineering approaches, code review procedures are adhered to more diligently.

It is incumbent on corporate management to take cognisance of the contradictions between these styles of producing software, and to manage them as and when needed. These difficulties notwithstanding, creative management solutions should be sought where these conflicts arise most prominently. Sections 4 and 5 illustrate these matters in a specific example. In general, it remains the

responsibility of a manager has to ensure that a developer completes essential tasks in due time and in compliance with the requisite quality. Work on random OSS-associated tasks that might be regarded as interesting or fun should be relegated to secondary status, if tolerated at all within the work hours.

There are other aspects to the OSS paradigm that corporate software developers need to understand. One of these aspects is the legal standing of software development, specifically in relation to open-source licenses. Corporate developers generally know about *proprietary* licensing, but they might not be prepared for the variety of OSS licenses and their inter-relationships. OSS developers need to be able to distinguish between the different licenses and their compatibilities. For example, consider the implications if one wanted to link in modules from a library that was issued under the GNU General Public License version 2 (GPLv2), while one's own code was distributed under the Berkeley Software Distribution (BSD) license? An OSS developer would need to know that the BSD and GPLv2 licenses are compatible, *but only if* the BSD code is distributed under the revised[3] BSD version [3,4].

Furthermore, managers of software developers would need to realise that if they have both closed-source and open-source projects in their portfolio, then they should isolate the developers from one another to ensure that no 'contamination' of code takes place.

3 OSSD and ASD

ASD values individuals and their interactions, working software, customer collaboration and responding to change. The primary drivers for ASD are speed and flexibility. Born out of a desire to reduce the overhead caused by over ceremony of traditional software development, the principles of ASD have gained increasing acceptance by corporate developers, under pressure to produce quality software at a rapid pace. As in the case of OSSD, ASD too is an overall philosophy with many variants, each of which finds its application in different development teams.

3.1 OSSD Is Not ASD

Superficially, ASD and OSSD have many aspects in common, including the early delivery of useful software, the valuing of feedback, basing scope and design primarily on utility, and an informed and productive developer community. Indeed, It has been alleged that OSS as a style is just another instance of ASD [5]. To assess the validity of this allegation, we have investigated the extent to which the generic development model of OSS as set out in literature (for example, [2,6]) complies with the principles set out in the *Agile Manifesto* [7]. Furthermore we have compared the stereotypical OSS style and Extreme Programming (XP). Both of these investigations are reported in [8]. Another study by Warsta and

[3] Without the advertisement clause.

Abrahamsson [9] further highlights the differences and similarities between OSS and ASD. Our findings are summarised below.

ASD was born within the corporate world and consequently has a strong focus on certain elements that are not associated with traditional OSSD. These include: *co-located teams*; *assigned* team membership; and *remunerated* employment which brings along a concomitant obligations and hierarchical relationships. Furthermore the *client* role tends to be played by a *non-programmer* who may be the business-user. In contrast, OSSD traditionally starts off with a single developer who is simultaneously the 'client', in that the software to be developed is intended to address a personal need (which could be work related). As the project grows other developers around the world may contribute or even eventually take over the project.

In further assessing the two paradigms, we noted that "the discussion was mainly (but not exclusively) in reference to the stereotypical traditional OSS development style. In reality, the culture surrounding OSS development is neither monolithic nor static". The same viewpoint was taken in regard to ASD. The compliance investigation showed that—at least to some extent—OSSD is indeed compatible with the following list of principles taken verbatim from the Agile Manifesto:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design enhances agility.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Build projects around motivated individuals ... and trust them to get the job done.

However, the remaining Agile Manifesto principles are not at the core of the stereotypical OSSD approach. These are:

- Business people and developers must work together daily throughout the project.
- ...Give them the environment and support they need...
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.
- Simplicity—the art of maximising the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organising teams.

The investigation in [8] therefore concluded that there are distinct differences between the two approaches. As a result, the scope for synergy between them is constrained, due to tensions that are likely to arise between teams who follow these opposing principles.

3.2 Adapting ASD

Consider a typical scenario where individual co-located agile development teams collaborate either with other such teams scattered around the world in an OSSD style, or with external OSS projects. Figure 1 provides an example of two Agile teams contributing to an OSS project. An alternative scenario would be where one of the agile teams –as a entity– form part of the core team of the OSS project. This subsection highlights some of the potential adjustments that might be needed.

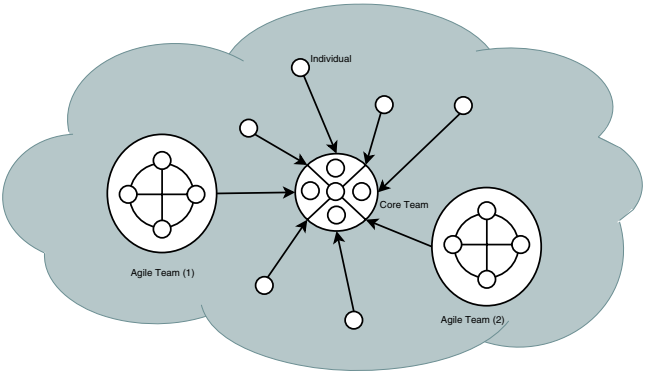


Fig. 1. Agile Teams participating in an OSS project

– *Adapting to remote communication*
From the agile perspective, accommodating a different way of communicating between developers might perhaps be the most challenging. As stated before, the stereotypical agile approach depends extensively on co-located teams who rely on face-to-face verbal communication between members and the availability of on-site customers. This is typically not the case within an OSS development team. Furthermore, the daily routine of an agile team is usually rigorously controlled. Typically, an agile team will start the day off with a short stand-up meeting, followed by a three to four hour focussed session of uninterrupted development. They may then break for lunch, followed by yet another focussed session. During these focussed sessions the developers are typically prohibited from using telephones, e-mail, IRC or any form of external communication, both inbound and outbound. In contrast the OSSD style requires almost *constant* access to communication media such as e-mail,

IRC and the Web. These media, which facilitate 24/7 flow of information will seem extremely 'noisy'[4] to developers accustomed to the agile style.

An added problem is the need to translate and transmit the verbal communication between co-located developers to other distributed external developers. This need to continually document and electronically broadcast the typically informal verbal communication between agile developers may prove to be a severe obstacle in the quest for synergy between ASD and OSSD.

- *Relinquishing of control*
  Agile developers are accustomed to having a large say in the decision making processes that control the direction of a project and the development style and culture within the project. However, when the team is simply yet another contributor in a larger community of developers, some of this control (possibly over *many* aspects of the project) may be lost. This could be a disturbing prospect for ASD developers and should be taken into account when the team interacts with the larger OSS community.

  The view on time-schedules is related to this control issue. Agile proponents advocate fixed, (though short) time cycles to illustrate their progress to the client and to verify the appropriateness of the evolving system. Although the OSS culture is also to deliver frequently, the inclination is to only deliver when the deliverables are useful and stable. This difference perhaps has its roots in the different drivers present in the two approaches. The primary driver in ASD can be seen to be frequent business deadline-driven releases. On the other hand, the primary driver in OSS is delivery of quality software to the community of developer/users.
- *Good OSS community citizenship*
  Agile developers need to realise that they are no longer the centre point of the development effort, but part of a larger community of developers with a deeply rooted culture—largely based on the Unix culture [10]—that has been around for a number of decades. Agile developers will therefore have to gain an understanding of the OSS culture to ensure that they adhere to the underlying, sometimes unwritten, rules of participating in the OSS community.

These points indicate that agile subcultures within parts of the corporate structure would need to adapt if OSSD is introduced into that structure. Indeed, it would seem necessary to compromise on some fundamental agile principles. Accepting these compromises may be a significant test of the very claim made by agile community, namely of being pre-eminently open to change and adaptation.

Of course, the extent to which the above comes into play in a specific project is dependent on the level of engagement between the agile team and the OSS project[5]. An example would be where the development team is responsible for

[4] Although ASD sessions may often be 'noisy' in sense of frequent discussion, the discussions tend to be focussed on immediate tasks at hand. OSSD interruptions, on the other hand, tend to be disparate and less focussed.

[5] These levels of engagement could be: simply *using* an OSS product; *modifying without sharing* the OSS product; *actively contributing* to an OSS project; or *managing* an OSS project.

developing an intranet application that *uses* the *Tomcat* server and *contributes* to the *MyFaces* library. In the aforementioned example one would expect that the tension-points arising in regard to the *Tomcat* project will be somewhat different to those experienced in regard to the *MyFaces* project.

The foregoing, largely a further elaboration of ideas first mooted in [8], does not purported to be an exhaustive list of possible adaptations that need to be addressed in order to gain synergy between agile on the one hand and OSS on the other. It is merely the starting point for a deeper analysis of the contention points and ways of reconciling them.

4 An Illustrative Example

The preceding sections have highlighted the need to take the tension-points into consideration when defining a process for a project that intends to combine OSSD and ASD. Clearly, many problems can be solved by amending the development process(es). However, the ability to adjust the development process depends on the control that one has over the project. Furthermore, the adjustments to the process will be based on the perspective of the team under consideration.

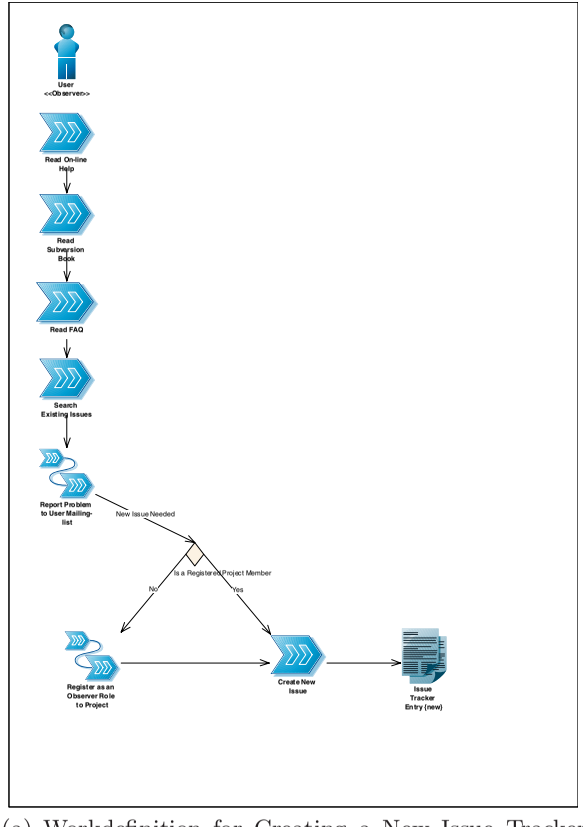
To illustrate the point, consider the following fictitious example: Team Bravo is assigned to develop a conference-room booking system, to be deployed on the intranet of the team's organisation. For the development, team members have decided to use *Tomcat*, *MyFaces*, *Hibernate*, *MySQL* and *GNU Linux*. In addition, they use *Eclipse* and *Subclipse* as development tools. During the course of the project the team extends *MyFaces* with additional components and submits these to the *MyFaces* project. Additionally, a bug is discovered within *Subclipse* and a bug report is filed with the *Subclipse* project, this report containing a *JUnit* test-case to illustrate the problem. Later on, the *Subclipse*-bug is classified as impeding the project and a patch to correct the problem is developed and submitted.

The *Subclipse* project has a predefined process for submitting bug reports. In addition, good project management requires that Team Bravo, too, should follow some internally defined sub-process in submitting a bug report. Clearly, the latter sub-process needs to interface to the *Subclipse* one. A similar situation would hold in the contribution of additional components to the *MyFaces* project.

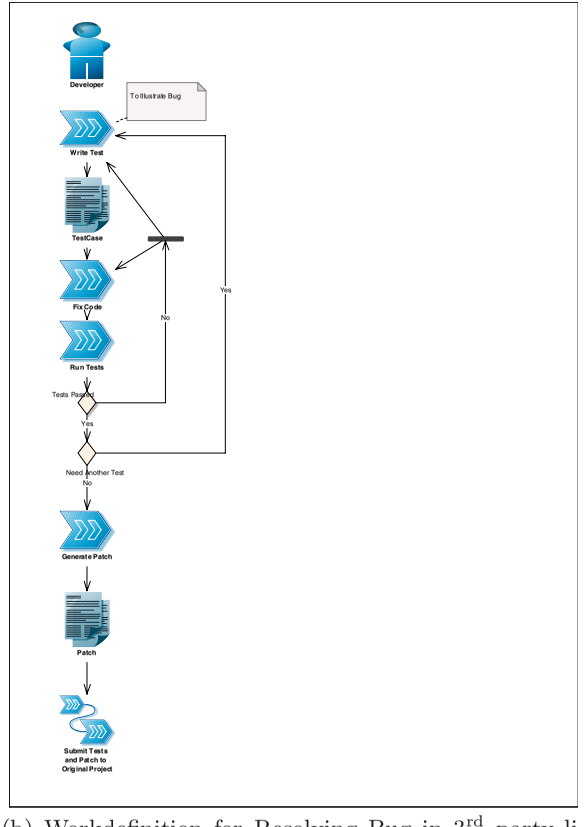
Figure 2(a) depicts the process specified by the *Subclipse* project-page [11] for submitting an *issue* into their issue tracker. The notation used is the *Software Process Engineering Metamodel* (SPEM) version 1.1 [12]. The aforementioned figure represents an Activity diagram to describe the specific *Work Definition*[6]. The *Activities*[7]: *Read On-line Help*, *Read Subversion Book* and *Read FAQ*

[6] "Work Definition: A Model Element of a process describing the execution, the operations performed, and the transformations enacted on the Work Products by the roles. Activity, Iteration, Phase, and Lifecycle are kinds of work definition" [12].

[7] "A Work Definition describing what a Process Role performs. Activities are the main element of work" [12].



(a) Workdefinition for Creating a New Issue Tracker Entry in Subclipse



(b) Workdefinition for Resolving Bug in 3rd party library

Fig. 2. Illustrative Workdefinitions

require that a user should first refer to the existing documentation for possible descriptions on how to resolve the problem that they are experiencing. If these activities are deemed unsuccessful, then the user should search the existing issues in the Issue Tracker database. If this, in turn, is also unsuccessful, then the user should report the problem to the *Users mailing-list*.

The *Subclipse* project requires the aforementioned activities as a filtering mechanism to reduce unnecessary entries in the issue-tracker. If need be, the user mailing-list will direct the user to file an entry in the issue-tracker. Embedded in the figure is another work definition: *Register as an Observer Role to Project*. This refers to the additional activities that are required when the user is not yet registered with the project. In the same way *Report Problem to User Mailing-list* encompasses the process of interacting on the user mailing-list.

The internal process that Team Bravo has to follow to resolve the bug is illustrated in Figure 2(b). The figure indicates that Team Bravo uses a test-driven approach to write the bug-fix, as required by their ASD-compliant development process. *Submit Patch to Original Project* represents an abstract sub-process to follow when reporting the problem and providing a solution to a 3rd party's project. In the above example the abstract sub-process should be superseded by the *Subclipse: Creating a New Issue Tracker Entry* process specified in Figure 2(a) and described above. In this way, the general project process can be customised to incorporate interfaces to other projects.

The foregoing endorses the notion of defining a *process per project*, as advocated by a number of methodologists, including Cockburn [13]. In the case of the conference-room booking system project, not only did the overall process depend on the project's internal sub-processes, but it also had to take account of the sub-processes of other external projects. In practice, the set of external projects to be incorporated may vary from one project to the next. This is evidently a typical consequence of incorporating an OSSD approach into a corporate development effort.

5 Proposals

The previous section gave a practical example of the kind of inter-project interaction scenario which a corporate development team incorporating OSSD could face. Numerous additional illustrative examples and scenarios could no doubt be cited. However, the present limited example already introduces a number of ideas that lead to concrete proposals for dealing with these tensions.

1. Note that, in general, there will be interaction points between sub-processes of the internal project and sub-processes of the various external (OSS) projects. The tensions previously mentioned, i.e. tensions between OSSD and ASD and/or traditional corporate culture, are most likely to be encountered at these interaction points. It would seem, therefore, that one can at least start to deal with these tensions, by articulating—either formally or informally—a sub-process at each such interaction point.

Taking this approach, one is able to specify the corporate development process with minimal (but non-zero) concern for the external projects process. The external projects processes are then only plug-ins that are realised in one's own process on an "as-needed" basis. This contains the tensions between the different, possibly opposing processes and encapsulates them at the defined interface points.

2. Another possible practice to consider, is the introduction of an additional role to the development process: that of a *liaison officer*. This role should be adopted whenever one either uses or contributes to an OSS project. In essence, someone would be designated as responsible for acting as a communication conduit between the projects. The responsibility of this role would be to gain and maintain an appropriate level of understanding of an external project with regard to the respective processes and the evolution of the artifacts. This knowledge would then be disseminated to the rest of the team, as required by the given project.

   An example of this would be assigning developer Jane to liaise on project Subclipse on the team's behalf. Whenever a new release is available, she will inform the rest of the team and aid with the adoption thereof by the team. This role might be seen as a substitute for the product representative from commercial companies from which software product are acquired.

   Furthermore, when deemed necessary, the role may be assigned to multiple persons, for example, on a per module basis for large external projects. The need to assign this responsibility to multiple persons may be particularly important in a context where an ASD approach such as Extreme Programming is being used, since the latter places emphasis on maintaining a level of human redundancy.

   The role of *liaison officer* would vary for each given project and for each external project, its importance being determined by factors such as those listed in the previous section.

6 Conclusion and Future Work

The implications of introducing OSSD into a corporate environment have been considered, and particular references has been made to the implications of accommodating both OSSD and ASD. An example illustrated the kind of situation facing corporate software developers who attempt to develop in an ASD style while collaborating with distributed partners.

In the example, the focus of the development team was to *use* OSS products. However, had the ASD team formed all or part of the *core* of an OSS project, then a number of other issues would need to be addressed. These have not been considered here, and are left for future study. However, it is clear that the *process-per-project* notion will feature strongly in any such consideration.

Clearly, when mixing in different ingredients from the software development processes and/or practices on offer, it would be wise to be weary about the resulting brew. It could turn out to poison or paralyse the project. On the other hand, it might be a elixir—an enabler of a successful and enduring project.

References

1. Dinkelacker, J., Garg, P.K., Miller, R., Nelson, D.: Progressive open source. Technical Report HPL-2001-233, Hewlette-Packard Laboratories, Palo Alto (2001)
2. Raymond, E.S.: The cathedral and the bazaar. First Monday (1998) (Accessed: 2007/06/27), http://www.firstmonday.org/issues/issue3_3/raymond/index.html
3. Rosen, L.: Open Source Licensing: Software Freedom and Intellectual Property Law. Prentice Hall Professional Technical Reference (2005)
4. Free Software Foundation: Frequently asked questions about the GNU GPL (2006) (Accessed: 2007/06/27), http://www.gnu.org/licenses/gpl-faq.html
5. Raymond, E.S.: Discovering the obvious: Hacking and refactoring. Weblog entry (2003) (Accessed: 2007/06/27), http://www.artima.com/weblogs/viewpost.jsp?thread=5342
6. Feller, J., Fitzgerald, B.: Understanding Open Source Software Development. Addison-Wesley, Reading (2002)
7. Cunningham, W.: Manifesto for Agile Software Development (2001)(Accessed: 2007/06/27), http://www.agilemanifesto.org
8. Theunissen, W., Boake, A., Kourie, D.: Open source and agile software development in corporates: A contradiction or an opportunity? Jacquard Conference, Zeist, Holland (2005)
9. Warsta, J., Abrahamsson, P.: Is open source software development essentially an agile method? In: Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering, Portland, Oregon, USA, pp. 143–147 (2003)
10. Raymond, E.S.: The Art of Unix Programming. Addison-Wesley, Reading (2003)
11. Subclipse Project Team: Subclipse issue tracker (2006) (Accessed: 2007/06/27), http://subclipse.tigris.org/project_issues.html
12. Object Management Group: Software process engineering metamodel specification. formal 05-01-06, Object Management Group (2005)
13. Cockburn, A.: Agile Software Development. Pearson Education, Inc., London (2002)