Open Source and Agile Software Development in Corporates: A Contradiction or An Opportunity?

WH Morkel Theunissen, Andrew Boake, Derrick G Kourie
mtheunis@cs.up.ac.za; andrew.boake@up.ac.za; dkourie@cs.up.ac.za
Espresso Research Group
Department of Computer Science
School of Information Technology
University of Pretoria
Pretoria, 0001, South Africa

Abstract

Currently the software industry is facing two phenomena that are challenging the dominance of traditional approaches to developing software, namely agile software development (ASD) and open source software (OSS) development. Each have the potential to provide certain benefits to various software development efforts. However, although they are compatible in some respects, they are also contradictory in other ways. This poses an awkward dilemma for institutions that wish to exploit the benefits of both. This paper investigates the tensions between the two phenomena and deeper challenges in bridging the gap between them.

I. Introduction

Currently the software industry is facing two paradigms that promise enormous benefits and a revolution of software development. On the one hand, agile software development has emerged as a fast-paced, nimble means of developing software for customers. On the other hand, there has been an increased adoption of Open Source Software (OSS) and its style of development by corporate players, in the hope of maximising reuse and, in turn, reducing costs.

The following two subsections sketch the emergence of these two paradigms.

A. Open Source Software

The past several years have seen subtle changes in the OSS culture. These changes are primarily due to the increased adoption of OSS by companies such as IBM, Hewlett-Packard (HP), Sun Microsystems, Novell and many more, who are promoting OSS products and participating in OSS projects.

Stallman institutionalised the culture of sharing source code and, through the GNU General Public License (GPL), he defined the rules for- and the means of doing it . However, the culture of sharing code had been around from very early on. It was common practice in the 1970's to share the code of some function or application with other programmers. The initial version of Unix, written at Bell Labs by Thompson and Ritchie, is a concrete example of this sharing culture [13]. However, during the 1980's, companies that employed programmers increasingly discouraged and eventually disallowed the practice. The enforcement of license agreements was the prime means of exercising this control.

Stallman experienced the effect of this software commercialisation, including the negative effect that it had on his own development environment. The GNU Not Unix (GNU[1]) project was started because of his displeasure in these practices [8]. The project was built on his belief in freedom of software, which is expressed through the Free Software Definition [5]:

> " 'Free software' is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech,' not as in 'free beer.'
>
> Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software:
>
> - The freedom to run the program, for any purpose (freedom 0).
> - The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
> - The freedom to redistribute copies so you can help your neighbour (freedom 2).
> - The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.
>
> A program is free software if users have all of these freedoms."

The GNU project grew into the Free Software Foundation (FSF), which was formed to take over the administrative responsibilities of the GNU project.

[1]A recursive acronym that is a common pun in the OSS culture.

During the 1990's the commercialisation of the Internet and its increased exposure to- and utilisation by more and more users sent shockwaves through the whole software development industry. One of the tremors was the rise of a new generation of Free Software proponents and the birth of a Free operating system called Linux that was actually working—as opposed to the dream of the GNU Hurd[2]. Another tremor was that commercial companies—especially ISPs—were faced with the fact that the Internet was being increasingly built on open standards and free software such as BIND, HTTPD (which became Apache) and Sendmail.

The Internet became an enabler for the community of developers who believed in Free/Libre and Open Source Software (FLOSS). They were able to share code, ideas and experiences in a cheap and easy fashion on a large distributed scale. Through this, FLOSS was revitalised and thrived to such an extent that it has moved from its so called hobbyist use to all areas of business.

Some businesses saw this as an opportunity, and even as a weapon against competitors. Netscape adopting an OSS strategy through the Mozilla project is a case in hand. IBM's discontinuation of their internal web server and their subsequent adoption of Apache instead is another example. This increased interest by prominant software vendors (starting in about 1998) started to influence the development culture of FLOSS. Businesses experimenting with Free software found the GPL to be too restrictive and the term 'Free' to be misleading (since it seems to imply *gratis* and that money cannot be made from it). Due to this and Netscape's plan to release their Communicator suite's source code to the community, certain of the dominant free software proponents felt the need to came together to address this problem. Out of this meeting emerged the idea of relabelling their efforts as **Open Source Software** and the creation of the *Open Source Definition* (OSD) [10], [11]. The Open Source Initiative (OSI) was formed to regulate and promote the OSD. They are responsible for certifying licenses as conforming to the OSD. To date over 50 licenses have been certified as OSD compliant, including the GPL, LGPL and BSD [9].

B. Agile Software Development

On another front, the Internet boom stimulated a revolution in the development methodology sphere. Many developers felt that developing at Internet time required a reconsideration of the traditional development processes. The latter did not seem to offer the results needed within the time frames associated with these new projects. Moreover, multi-year projects, so typical of the former era, were being replaced by projects that were to be completed in the order of months. These considerations spawned a number of methodologies that were called the 'light-weight' methodologies. In early 2000 the proponents of these various methodologies came together to discuss their commonalities. This resulted in the adoption of the name *agile* instead of 'light-weight'; the formulation of the *Agile Manifesto* and the formation of the *Agile Alliance* [6]. The *Agile Manifesto* states the agile beliefs as [1]:

> "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more. "

Since the formation of the Agile Alliance there has been an increased interest and adoption thereof. Large organisations, including NASA, Chrysler and financial institutes are practicing agile software development and reporting success [15], [2], [3].

The above discussion has highlighted two possibly unanticipated phenomena that are increasingly influencing the *modus operandi* of software development. In this paper we will be investigating the relevance of both agile software development (Section II) and OSS (Section III) to the corporate environment. A comparison between OSS development and agile development is drawn in Section V. In addition, we also investigate the influence that companies potentially have on the OSS culture (Section VI). Section VII highlights the discrepancies between agile software development and OSS development that needs to be bridged to potentially gain maximum benefit for corporate software development. To conclude, a summary of the findings of this paper are presented in Section VIII.

II. Agile Software Development

Section I pointed to the importance of agile software development. This section expands on the matter. Increasingly, businesses have become aware of the strategic importance of Information Technology (IT) in their daily activities as well as its potential for providing a competitive advantage. Moreover, in the quest to stay competitive, companies are often prepared to invest significant resources in rapidly duplicating, and in some cases enhancing, technologies introduced by their competitors. (For example, technologies such as short messaging systems (SMS) were rapidly introduced by competitors once they had made

[2]A operating system kernel as part of the FSF/GNU effort that was still on the drawing boards at that stage (*circa* 1992).

their first appearance in the market place.) This means that competitive advantage of innovative technologies has a limited lifetime. To maximise profit, businesses seek to take a project from an idea into deployment in as short a timespan as possible, often within a few months if not weeks. This exerts extensive pressure on the software developers who are responsible for the implementation.

From an Agile perspective, traditional software development methodologies involve comprehensive ‘ceremony’[3] which places a burden on software developers and slows the implementation. This was one important perception that gave rise to the different agile software development methodologies. These methodologies aim to be people-oriented and are seen as being in opposition to the alleged process-oriented nature of traditional methodologies.

The benefits claimed for following an agile methodology are: faster development, delivering features that the customer actually needs and being adaptable to changes.

III. OSS

Traditionally OSS projects tended to revolve around infrastructural related solutions such as the operating system, web and mail servers, etc. Recently, however, OSS projects expanded to solve corporate problems such as office productivity suites, groupware, etc.

As more and more OSS projects aim to solve corporate problems instead of the mere itches of hobbyist programmers, the question of whether and how to apply best practices associated with commercial software development has become an issue of concern in OSS projects.

An increasing number of large corporations are adopting and contributing to OSS. The extent to which they engage in the production and use of OSS ranges from simply using an OSS product to driving an OSS project. The different levels of OSS engagement are briefly described below, starting off with the least involvement and progressing to the management of an entire OSS project. Although certain benefits are associated with each of these engagement levels, each level also requires specific resource commitments, as will be indicated in the discussion.

- *Simply using a product*. At this level one simply acquires the OSS product in either source or binary form and uses it to fulfil a need. In most cases this requires a negligible expenditure of resources. As a matter of courtesy, one may register as a user of the product, but this is merely to indicate one’s support for the product.

  The primary benefit of engaging OSS at this level, is the low investment required to acquire software to address one’s needs.
- *Modifying a product without sharing the modifications*. At this level of OSS engagement, one may decide to customise an OSS product to suite specific needs. For a variety of reasons it might be preferable to keep these changes internally instead of sharing them with the community. For example, it might be that the modifications include royalty and/or patent regulated elements, or represent significant competitive advantage.

  At this level, the degree of resource investment increases in proportion to the extensiveness of the modifications to the original OSS product.
- *Modifying a product and contributing the changes back into the community*. Here, again one acquires the product and makes changes to suite one’s particular needs. However, arrangements are then made to integrate these changes back into the original project or to make the changes available to the community in some other way. The resource expenditure at this level of OSS engagement will also vary in proportion to the extent of the contribution.
- *Initiating and/or managing an OSS project*. Usually, at this level, significant resources will have to be invested into an OSS project. Participation at this level usually becomes necessary when no one else is willing or able to address a need and/or when one is the leader in the project’s solution domain. The most noteworthy benefit is the ability to steer (at least to some extent) the direction of the project.

  Examples of this level of engagement include Netscape’s initiative in undertaking the Mozilla project, Sun with OpenOffice, and IBM with Eclipse.

Companies such as IBM, HP, Sun and Novell have hundreds of their own developers contributing to OSS projects, including projects such as the Linux kernel, Apache and Gnome. Some of the prominent projects are even sponsored and/or managed by these companies. Examples are Eclipse by IBM and OpenOffice by Sun.

An underlying reason for OSS gaining relevance in the corporate environment is the notion of reuse. Software development best practices advocate the reuse of software components as far as possible. OSS libraries offer more than black-box views on components—they offer the ability to see into a component as a white-box. Using libraries developed by the OSS community may not only reduce the cost of acquisition, but may also reduce the risk involved when errors are found with the libraries. Having access to the source code enables the team to verify the implementation and to correct mistakes. This is in contrast to relying on proprietary components, where developers are left emasculated and disempowered in the face of errors.

[3]This is Agile jargon for activity that does not immediately and directly contribute to the attainment of the required product.

Furthermore, the notion of utilising hundreds, even thousands, of volunteers to help develop one's software appeals to large number of businesses and managers[4], provided that they can exercise some level of control over the final product.

These potential benefits have made OSS seem attractive to various corporate software developers and their companies.

It is important however to realise that OSS development may be more adequately described as being a *style* of development, rather than a process (see [4], [14]).

IV. A Farming Analogy

A *farming analogy* or parable, in which software development and farming are compared, offers a deeper understanding of how OSS functions and how corporate adoption may influence the OSS culture.

Anyone with a basic horticultural knowledge should be able to grow and cultivate vegetables. She may even do so in her own backyard. However, to farm on any significant scale, one needs certain basic elements: a patch of ground, seeds, water, fertiliser, equipment, time, knowledge, experience, labourer(s) and a desire to harvest useful crops.

In the software development context, seeds map to a conceptual idea, or—in the OSS case—to some initial code or design that starts off the project. Developers can be compared to the labourers that do the farming.

The patch of ground where the crops are cultivated and harvested corresponds to various 'areas' where OSS is developed. As in agriculture, these may provide for community-based activity. SourceForge is a prime example of space donated by a non-profit organisation, in this case, Open Source Technology Group Inc. (OSTG)[5]. In other instances, a patron may donate a patch of land for exclusive use by a certain community. In the software development context this maps to a specific project, for example OpenOffice, sponsored by a certain company, in this case Sun Microsystems. Other more mature projects might even be self-hosted, such as the Linux kernel and Apache.

The farming equipment would map to software tools used by developers, such as version-control software, issue trackers, forums, website, integrated development environments (IDEs), mail servers and clients, etc.

The OSS development style is comparable to a community-driven farming undertaking in which a number of community members with a common interests in cultivating a particular crop come together and labour as a team to reach their goal. It may be that a person who is particularly knowledgable about the crop, will initiate and 'manage' the effort. These members will work hard at sowing, watering, fertilising and nurturing the crop. They will gratefully accept assistance from passers-by who offer to help out with specific needs. They will not mind others harvesting from their land as long as they adhere to certain basic household rules. In the case of OSS development, this is expressed in an OSI compliant license.

The conventional and alternative approach to both software development, and to farming, is to undertake it as a commercial or proprietary endeavour. In the farming case, a business enterprise tills a patch of land that is enclosed by high walls to keep others out and to protect its cultivation secrets. Such a farming method aims at mass production. The enterprise may set up a retail outlet to sell produce to consumers. The produce will be nicely packaged and ready for consumption. There may well be certain constraints and certain procedures to follow when using the product, and liability disclaimers if these procedures are not followed. The parallels with commercial software development are obvious.

The emerging phenomenon of corporations participating in OSS efforts, yields an interesting mix of these approaches. Here the farmer employs labourers to work on assigned crops. Labourers may even be given portions of the land, provided certain crops are planted and a certain proportion of the harvest is assigned to the farmer. The farmer might then ship this produce to one of her closed—as in proprietary—processing plants. These plants add value to the original produce, which is then sold to consumers.

In the case where commercial entities are the primary sponsors of projects, one might compare this to the feudal system. This system presents the community with both dangers and opportunities. In such a system, crops are cultivated under the patronage of a landlord who expects not only a portion of the crops, but also loyalty and obeisance in return. In OSS one might also be able to describe this patron and *vassal*[6] relationship as symbiotic, in that both derive benefits from the relationship. Whether the Eclipse project, and other similar projects will emerge as having this character, remains to be seen.

This analogy reflects some aspects in the current state of play in software development. It also hints at some of the pending contentious points with regard to OSS development within the corporate sector. The analogy serves as a backdrop for the remainder of this paper.

V. OSS Development is not Agile Software Development

It has been alleged that OSS as a style is just another instance of agile software development [14]. To assess the validity of this allegation, a comparison between the stereotypical OSS style and Extreme Programming (XP) is made in Subsection V-A. Thereafter, the OSS development model's compatibility with Agile principles, as outlined in the Agile Manifesto, is assessed (Subsection V-B).

[4]However, it should be noted that only a small number of projects ever reach this magnitude of contribution.

[5]OSTG is a community-driven media network on the Web and a wholly owned subsidiary of VA Software Corporation.

[6]'A person under the protection of a feudal lord to whom he has vowed homage and fealty : a feudal tenant"[7].

A. *XP vs OSS*

Table I summarises the differences between the characteristics of XP and the characteristics commonly associated with developing traditional OSS projects. XP was selected as a representative agile methodology because of its predominance. As stated before, OSS projects by their very nature are not standardised, and so many deviations from the 'typical' characteristics listed in Table I are to be expected.

	XP	OSS
Team Size	2-10	1-20 Core developers + unspecified number of volunteers
Developer Location	Centralised and Co-located	Distributed
Releases	Fixed time schedule	*Ad hoc*, when core developers feel that it is appropriate
Requirement selection	User-driven	Community-driven
Communication	Primarily verbal	Primarily E-mail and Issue trackers
Methodology	Disciplined	Customised to core developers personality
Type of developers	Usually highly skilled and experienced	Skilled volunteers

TABLE I
COMPARISON BETWEEN XP AND OSS DEVELOPMENT CHARACTERISTICS

It is clear from Table I that there are similarities as well as differences between the approaches. Both development methods rely on a small group of developers. OSS projects tend to be coordinated by a small core development team that may be classified as 'full time'. This core team is assisted by casual developers who submit patches and by users who report on problems. Agile teams are, of course, conventional full time employees of a company.

One of the most distinctive differences is in the way communication takes place. Agile development methodologies regard verbal communication between members as important and XP even have as an absolute requirement that team members should be co-located for face-to-face communication. OSS development, on the other hand, is by nature distributed, and therefore relies on remote communication.

The agile model is highly centred, emphasising collaborative team work and requiring the team to accept collective responsibility for their activities. OSS development, on the other hand, is diffuse, relying on voluntary individual efforts and resulting in limited individual responsibility.

Another difference lies in the formality level associated with each method. This difference derives from the origins of the respective development styles. Although agile methods were conceived of as an alternative to rigid and regimented software engineering processes, they were nevertheless intended for corporate environments in which management would still commit to strong, but different, software engineering principles in which team members enforce discipline. The OSS heritage is entirely different: it is driven by passionate people who undertake voluntary development and who neither want nor require supervision. This difference in developmental culture may well be the primary obstacle in trying to drive OSS development from within the context of a commercial environment. Linked to this difference is the way in which views on schedules differ. In the case of agile, the schedule is an important element of the developmental process, the entire developmental methodology being geared to better meet and deal with schedules. In the case of OSS, the notion of a schedule may not be very important at all, and may be subject to the whims and fancies of the core developer(s), and the inherent stability of the preceding changes/features.

B. *OSS and the Agile Manifesto*

The foregoing showed that there are indeed quite radical differences, but also correspondences, between agile and OSS developmental styles and traditions. To further compare these two styles, it is of interest to enquire whether and how the OSS developmental style and culture adheres to the principles outlined in the Agile Manifesto[1].

Allowing for wide divergences in OSS styles, one may argue that OSS developers, by and large, adhere to the following principles:

- "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"

However, the OSS culture does not sit comfortably with the remaining agile principles. In the following paragraphs, these principles will be discussed in relation to the OSS culture, style and ethos:

"Business people and developers must work together daily throughout the project". In a number of agile methodologies this means that the business representatives have to form part of the development team, as in XP's 'on-site customer' practice. It is immediately evident that the distributed nature of OSS development prohibits the close collaboration with customers that is prescribed by the agile principles. The fact that OSS development may rely on forums, mailing lists, Instant Relay Chat (IRC), issue trackers etc. to interact with the customers, is not an entirely convincing adoption of this key agile principle, which so strongly emphasises person-to-person interaction.

To some extent, the fact that OSS developers are also users of the product which they develop, may suggest positive adherence to this aforementioned agile principle. After all, this means that there is always an instantaneously available 'on-site customer' who understands the technical jargon. However, to be locked into this situation, means to have no scope for creative interactive dialogue with another, and to cast the developer in the role of the domain expert (which may or may not be valid). Most seriously, however, it means that the developer becomes the judge and jury of her own trial, as it were, and there is no independent assessment of the software until it is released into the public domain, at which point the core decisions about its design and functionality would already have been taken. Instead, the agile perspective would require that the community, as the users of the product, should be steering the direction of the project from the start.

Moreover, it should be noted that in some 'modern' OSS projects the developers may no longer even be users of the product. A project such a Compiere, involving the development of an Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) system, is not intended primarily for the use of the developers. In fact, the core developers are paid employees (who may or may not be domain experts, as stated before). From an agile perspective, this type of project environment most certainly needs to interact with customer representatives who should ideally form part of the development team.

"Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done". On the surface it may seem as if OSS development does indeed adhere to this principle. However, in the agile context, the reference to 'environment and support' points to matters such as financial support, hardware provision, training, the physical development area, access to on-site customers and 'protection' from business and managerial concerns. The idea is for management to provide for the developer a space where she can be totally focused, without other mundane concerns intruding into her working environment. In the context of OSS development, the environment and support are self-supplied - i.e. it is up to the developer herself to create the physical and emotional space for focused tranquillity.

"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation". Once again the distributed nature of OSS does not usually permit face-to-face communication. In [3] Cockburn analyses the communication burden under different scenarios and concludes that no other medium can outperform face-to-face conversations. All substitute media such as IRC, Instant Messaging (IM), e-mail or forums, do not come close to providing the same communication opportunities as co-located developers. However, OSS advocates may counter that the reliance on remote communication has not detracted from the overall success of OSS.

It is therefore incumbent on agile software development to accommodate the remote communication needed for OSS development if it is to share in the successes of OSS. This is further addresses in VII.

"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly". The stereotypical OSS projects do not comply with this principle. The practice of evaluating the development process in any formal sense is not part of the OSS style or culture. However, it is part of the OSS culture to adjust to circumstances, and to this extent, adjustments to a development process may well be made if it became apparent that it is no longer efficient. An example of precisely such an adjustment, is to be found in the history of the Linux kernel. In late 1998, the Linux kernel was on the brink of being forked because Torvalds was unable to keep up with the patches being sent to him. He was thus unable to meet the demands and needs of the community. On the one hand, he had many other commitments and on the other hand, the growth of the Linux kernel in scale and popularity had outstripped the ability of a single individual to manage the integration of patches and releasing of the official kernel. Torvalds thus became a bottle-neck for the development effort and some senior members of the core development group became frustrated, threatening to fork the kernel development. A meeting between some of the senior stakeholders was held to resolve the problem. The outcome of the meeting was the adoption of the BitKeeper version-control tool and the formalisation of the lieutenant system [8]. In this sense, the core group of developers of an OSS project were able to adjust the development process in order to align it with evolving requirements.

"Simplicity—the art of maximising the amount of work not done—is essential". Although much OSS development , steeped as it is in the Unix tradition [13], seems to encourage simple design as a means to early release, there is no explicit OSS

manifesto or list of principles that stipulates such a disposition amongst those who contribute to OSS development. Neither is there any explicit process which stipulates this kind of striving for simplicity. Instead, in OSS development, what is encouraged is working software that is open to the scrutiny of others. On the one hand, this may *indirectly* encourage simplicity in individual coding efforts, but coding style ultimately remains uncontrolled and subject to the particular tastes of individual developers.

At another level, the OSS style of development appears to be quite the opposite of *maximising the amount of work not done*. It involves an open invitation to all and sundry to try their hand at developing some or other feature. One might almost say that it tries to *minimise the amount of work not done*. It is more analogous to, and indeed as wasteful as, evolution—allowing various participants to grope for solutions in all directions, and allowing those that are fit for survival to endure on their own merits.

"The best architectures, requirements, and designs emerge from self-organising teams". To enable distributed participation and scalability the prominent OSS projects rely on highly modular architectural designs. This type of architecture seems to be the hallmark of the successful OSS projects (eg. Linux, Apache, Eclipse). These modular architectural implementations require significant upfront design decisions and planning. A modular architecture is essential if contributors are to add their own extensions seamlessly without an in-depth knowledge of the existing code base. In traditional OSS projects, these architecture and design decision tend to be made by the primary maintainer, although lieutenants may be consulted and provide suggestions.

Such upfront architectural design contradicts the ASD notion of an emerging architecture. In this milieu, agile teams derive an architecture through collaborative white-board discussion sessions, when the software's future direction warrant refactoring of its simple design. The design decisions tend to be only based on the simplest solution for the current need, without too much regard for potential future requirements.

This approach is of course eased by small co-located teams. A large OSS community faced with a major revision update, will be relatively drastically affected.

The above subsections have shown that, notwithstanding certain similarities, there are substantial differences between the Agile methodologies and the OSS style of development. However, 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. Indeed, there are important contemporary forces bringing pressure to bear on the traditional OSS development culture. The next section considers the impact of recent, seemingly paradoxical developments, that are apparently poised to have a strong influence on the traditional OSS culture: the influence of big business.

VI. The Potential Influence of Commercial Interests on OSS

This section identifies some of the influences that the commercialisation of OSS projects may have on the OSS style of development. As mentioned before, the adoption and increased support of OSS projects by commercial companies is bringing about some changes in the traditional style of OSS development. For example, the Eclipse project has introduced the notion of fixed time schedules for the delivery of milestone releases. One could say that the project contains a high-level project plan that outlines the features that will be available at each milestone. It also lists when feature freezes commence, it indicates how stabilisation testing will proceed, is based upon test driven development and carries out nightly builds. In all of these ways, Eclipse's project management is enforcing a development process that resembles more traditional software development, and that is somewhat foreign to the traditional OSS development style.

This example illustrates that commercial interests are undoubtedly having an impact, albeit in some limited contexts, on the development culture of OSS. Whether the overall influence will be positive or negative is a matter of opinion, and remains to be seen. However, in the next two subsections (VI-A and VI-B) some of the potentially negative and positive influences are discussed.

A. Possible negative influences

Traditionally, OSS development has been carried out by volunteers who contribute to the projects for personal reasons. These volunteers have typically been idealistic, passionate and highly motivated. Corporate participation may dampen this passion, in as much as developers are reduced to being treated as if they are conventional employees or consultants.

There is a possibility that a company could overwhelm or hijack a project by throwing full time developers and other resources into it and eventually taking over the entire management of the project. By acquiring managerial control of the core development team, the company would then be able to steer the project in directions that redound to its own benefit, with little regard to the rest of the community. Fortunately the very nature of OSS allows for forking of the code, which would enable the community to regain control of the project if they do not agree with the direction of the steering group.

A scenario in which individuals and especially organisations continuously take products from the OSS community without ever contributing anything back to it, poses a potential threat to the OSS system. For the OSS paradigm to work, there needs to be a balance between taking and contributing[7]. Corporate entities using OSS will need to keep to this spirit, even though their use of OSS is motivated by financial gain. For some in industry, this will require a significant mind-shift from a hardcore capitalist philosophy, to a more benign socially responsible disposition.

B. Possible positive contributions

Companies can contribute resources to an OSS project in various ways. These could include:

- *Hardware* – For example, in the case of the Linux kernel, specialised hardware could be donated with the aim of building support for that specific equipment. Another example would be to provide servers, workstations or test equipment to the developers.
- *Financial* – Financial contributions are always in demand by OSS projects. Expenses such as salaries for full time developers, hosting costs, etc. could be covered by this type of contribution.
- *Facilities* – This could include office space for developers. It could also include the provisioning of hosting capabilities in expensive and sophisticated server farms. Another contribution could be the provision of conference facilities and the hosting of developer meetings.

Not only are companies able to contribute physical resources to the development effort; they are also able to contribute less tangible resources such as knowledge, experience, expertise and paid developers. The knowledge and experience that companies are able to bring to the development effort may enhance the applicability of the product under developed. They could also bring much needed man-power to niche projects, where the number of voluntary developers is low. The development of the Linux kernel is a case in point: IBM has assigned a number of its internal developers to help with the project. As a result they not only participate as normal Linux kernel developers, but are also able to develop and test extensions to the project that are based on IBM hardware and technology.

One of the most important potential advantages that could flow from companies participating in the OSS effort, is if they would make available their capacity for providing commercial support to users of OSS products. This would not only encourage the adoption of OSS products, but would also give newfound credibility to OSS projects. Such an endorsement of OSS would amount to an acknowledgement that OSS is capable of competing with proprietary products and would diminish perceptions of OSS inferiority.

Novell and IBM are examples of traditional proprietary companies that have embraced OSS products and who have incorporated them into their product and service offerings, providing this type of commercial support.

The foregoing suggests that companies could have a radical influence on OSS, both positively and negatively. An assessment of the social and moral consequences is however beyond the scope of this paper.

The next section extends on the contention points between the OSS and agile approaches.

VII. Tension between OSS and Agile

In the previous section the possible influences of corporate involvement in OSS development was surveyed. Here, Subsections VII-A and VII-B highlight adaptations that will have to be made to both agile and OSS development if they are to be reconciled. This discussion will assume that agile software development is increasingly representative of corporate software development efforts. This stance derives from the discussion on the importance of agile software development (Section II), and as such concludes that companies investigating the adoption of OSS have also concluded that agile offers the corporate software development methodology required for fast-paced, nimble contemporary development.

A. Tensions between OSS and Corporate Culture

The corporate environment places certain requirements on the software development process to enforce accountability from employees. This poses a number of strains on a development team that functions in such an environment. Some of these requirements are highlighted below.

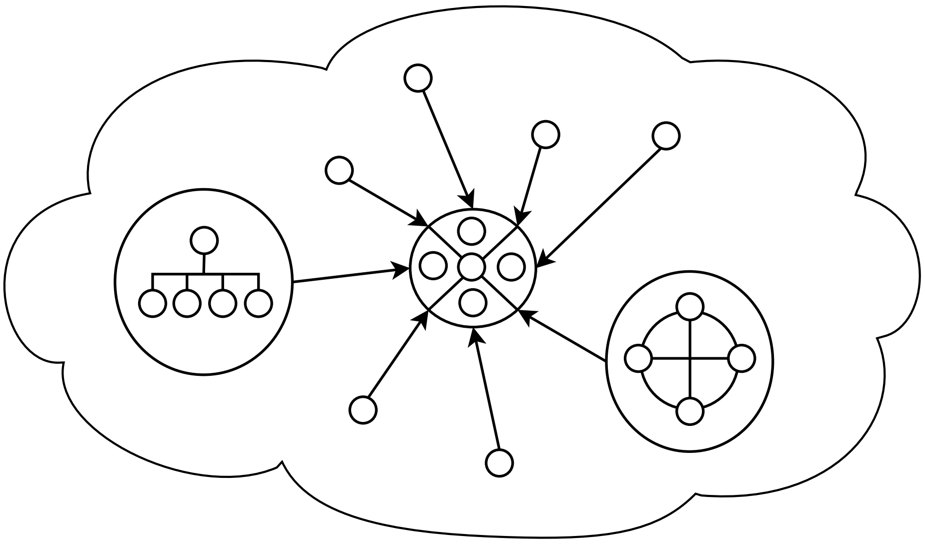
- *Monitoring of developers.*
  In an environment where remuneration for work is the norm, there is a need to manage and monitor employees. Traditional OSS projects have not been subjected to this, due to the voluntary nature of the development. However, in the corporate paradigm, when a manager assigns tasks to her subordinates she would like to be able to track their progress and their activities for resource balancing purposes. This scenario can become even more complicated when an OSS development

[7]This has echoes of the Marxian aphorism: “From each according to his abilities; to each according to his needs”

style is used internally. Monitoring what developers are contributing to different and disjoint OSS efforts is difficult. 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. Notwithstanding, a manager has to ensure that developers complete essential tasks, instead of working on random OSS-associated tasks that the developers regard as interesting or fun.

- *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 felt 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 account for cash flow, Return-On-Investment (ROI), etc. This also reduces the volatility associated with deploying solutions.
- *Quality Assurance Processes*.
  OSS by its very nature allows for extensive peer review. One of the underlying concepts of OSS is the so-called 'Linus' Law' that states: "Given enough eyeballs, all bugs are shallow." [12]. However, although some OSS projects may apply certain rules prior to accepting contributions (patches), there are 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. Pair-programming[8] is an example of the extent to which agile methodologies apply code and peer review procedures. Here, code is constantly and in real-time under review by another developer, providing a certain level of assurance regarding the quality of the code. Code reviews may also be seen as an extension of the 'monitoring of developers' requirement discussed above.
  Agile also encourages the use of test-driven development supported by automated testing to further pursue quality assurance. The use of a test-driven approach is, in itself, a significant shift in the approach to traditional development approaches that emphasise phased testing only after certain stages. To properly employ this practice requires a supportive culture with knowledgable and skilled developers.
- *Support of Heterogeneous Software Development Methodologies within an OSS project*.

Fig. 1. Heterogeneous development methodologies interacting within OSS



When organisations participate in OSS projects they might rely on a software development methodology for their internal development that may or may not be compatible with the OSS style. A fictitious scenario to illustrate the possible cultural mixes in an OSS project is shown in Figure 1. The cloud refers to the OSS community. The six small circles not contained in any other circle indicate individual OSS developers. Arrows indicate that they interact with five core developers of the OSS project who are shown in the larger centre circle. The centre core developer is the lead or primary maintainer and the surrounding four, her so-called lieutenants. Organisations participating in the project are indicated by the large circles on either side of the project circle. The organisation on the left uses a more formal hierarchical approach to development, whereas the organisation on the right pursues an agile approach. In the figure, the organisations contribute transparently as single entities. In certain projects, organisations may have developers forming part of the core team or may as a single

[8]A practice where two programmers work together on one machine. One uses the machine to implement the current feature, while her partner thinks strategically (thus globally) about the proposed solution.

entity represent the core development team.
Given the above scenario, the OSS development process may need to explicitly account for the fact that some contributions could come from organisations—as a single entity—and not only from individuals. As stated before, these organisations may be adhering to different internal development methodologies and these may be reflected in their contributions. The core development team as well as other developers need to realise this and respect certain peculiarities in their contributions. The developers within the various organisations should, in turn, concede the validity of other development methods.

The factors discussed in the forgoing focus on tensions between OSS and the corporate culture in which it is arguably the OSS culture that will require the greatest adaption. These factors may seem rudimentary, however they embody some of the core differences between corporate and part-time voluntary development. In our quest to find synergy between corporate/agile software development and OSS development these points of tension will need to be addressed. The means of addressing them will require further research.

B. Tension between Agile and OSS development

Just as OSS development will need to accommodate a corporate development style, so also may agile developers within corporate institutions need to adapt to certain OSS practices that are not completely aligned with purist agile principles. This subsection highlights some of the potential adjustments that might be needed.

- *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 agile approach depends extensively 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 style. 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 OSS development style requires *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' 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 totally opposite mode of communicating information between developers may prove to be a severe obstacle in the quest for synergy.
- *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 over many aspects of the project will be lost. This could be a disturbing prospect for these developers and should be taken into account when the team interacts with the OSS community.
  The view on timeschedules is associated with the 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.
- *Good 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 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 when participating in the OSS community.

These points would seem to suggest that adapting OSS to the corporate environment may not be the only stumbling block in our quest for synergy. Adapting the agile culture to enable participation in OSS projects seems to require changes to some fundamental principles. These adaptions may prove the greatest test of the touted adaptability of agile approaches.

The above two subsections are not 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. Instead, they offer a starting point for a deeper analysis of the contention points when reconciling these approaches. This adaptation to both approaches may also be facilitated through the use of appropriate tools. A assessment of the requirements for such tools is beyond the scope this paper, but is part of the research interest of the authors.

The tension between OSS development and agile software development may be attributed to two fundamental differences between the cultures. These differences may be summarised as:

- Distributed *vs* Centralised location
- Voluntary *vs* Remunerated work

These differences run deep, and any effort at reconciling the two approaches will need each to recognise, and adapt to, the others' approach.

VIII. Conclusion

This paper has set out to investigate two contemporary phenomena that have acquired considerable momentum within the software engineering industry: agile software development and open source software development. We have briefly looked at their origins (Section I) and relevance (Sections II and III).

A 'farming analogy' was used to give some insight into the OSS culture, into the role that companies may play in this community, and how they might influence—and be influenced by—OSS (Section IV).

Section V provided a comparison between agile software development and OSS development. Here it was seen that the assertion that OSS is simply another instance of agile development, is supported neither by a comparison of their development characteristics, nor by an assessment of OSS against the principles listed in the Agile Manifesto.

The conclusion that OSS development is not simply another agile software development instance compelled an investigation into the possibility of bridging the two approaches. The reason being the hope of gaining the benefits that each brings to software engineering in the corporate environment. This investigation revealed that there are a number of elements in the respective approaches that will require compromises from both sides to enable reconciliation and synergy.

The paper also highlighted the arduousness of describing OSS development due to the corporate involvement with some OSS projects. Some of the projects that are being adopted or driven by corporate participants, seem to have adopted certain of the software engineering practices associated with traditional corporate software development, thus confirming the influence that corporate involvement is having on the OSS development culture. Eclipse is a prime example of this trend. These developments stress the importance of further research into synergy between OSS development and corporate/agile software development. Research into the means of adapting each approach, either by enabling tools and/or by adjustments to the cultures, is needed.

References

[1] "Manifesto for Agile Software Development." [Online]. Available: http://www.agilemanifesto.org
[2] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.
[3] A. Cockburn, *Agile Software Development*. Pearson Education, Inc, 2002.
[4] M. Fowler, "The new methodology," Online, April 2003. [Online]. Available: http://www.martinfowler.com/articles/newMethodology.html
[5] Free Software Foundation, "The free software definition," Online, 2004. [Online]. Available: http://www.fsf.org/philosophy/free-sw.html
[6] J. Highsmith, "History: The Agile Manifesto." [Online]. Available: http://www.agilemanifesto.org/history.html
[7] "Merriam-Webster Online." [Online]. Available: http://www.m-w.com
[8] G. Moody, *Rebel Code: Linux and the Open Source Revolution*. Penguin Books, 2002.
[9] Open Source Initiative, "The approved licenses," Website, 2004. [Online]. Available: http://www.opensource.org/licenses/
[10] ——, "History of the OSI," 2004. [Online]. Available: http://www.opensource.org/docs/history.php
[11] ——, "The open source definition," Webpage, 2004. [Online]. Available: http://www.opensource.org/docs/definition.php
[12] E. S. Raymond, "The cathedral and the bazaar," *First Monday*, March 1998. [Online]. Available: http://www.firstmonday.org/issues/issue3_3/raymond/index.html
[13] ——, *The Art of Unix Programming*. Addison-Wesley, September 2003.
[14] ——, "Discovering the obvious: Hacking and refactoring," June 2003, weblog entry. [Online]. Available: http://www.artima.com/weblogs/viewpost.jsp?thread=5342
[15] W. Wood and W. Kleb, "Exploring XP for scientific research," *IEEE Software*, vol. 20, no. 3, pp. 30–36, May-June 2003.