In Search of the Sweet Spot: Agile Open Collaborative Corporate Software Development

W.H. MORKEL THEUNISSEN, ANDREW BOAKE AND DERRICK G. KOURIE
University of Pretoria

Corporate software developers are faced with many difficulties. Development windows are decreasing; scale and complexity are increasing; business requirements are vague and changing; and the underlying technology moves ever on. Agile methods have emerged as leading contenders to tame these challenges. Small teams, face-to-face communication, an emphasis on simplicity and a selection of development best practices contribute to software development which is relevant, yet fast and flexible. At the same time, Open Source Software is increasingly providing infrastructure, tools and components to companies. Progressive development teams are beginning to work in more open, collaborative, and distributed ways. In some respects these practices are similar to agile practices, but in important ways, very different. Yet, both are important and offer unique benefits. This paper discusses the prospects of combining the two in the context of corporate software development, and the approach we suggest to do this.

Categories and Subject Descriptors: D.2.9 [**Software Engineering**]: Management—*Software process models*

General Terms: Management

Additional Key Words and Phrases: Agile Software Development, Hybrid, Open Source Software Development, Software Process Engineering Metamodel, Corporate, Progressive Open Source

1. INTRODUCTION

The past decade has seen the emergence of two "new" software development paradigms. On the one hand, the rise and acceptance of agile approaches has been witnessed in commercial software development; on the other hand, the use of open source software (OSS) as reliable enterprise solutions has increased substantially. The adoption of OSS by commercial Information and Communication Technology (ICT) players has even lead to the evolution of associated OSS development (OSSD) practices and culture traits within these companies.

As such, software development managers would be well-advised to evaluate these two paradigms and to consider their influence on their individual environments and circumstances. However this evaluation is further complicated by the debate on whether OSSD is just another agile approach or not [Warsta and Abrahamsson 2003; Raymond 2003b]. The authors contributed to this debate in [Theunissen et al. 2005], concluding that the general OSSD approach does not comply with the Agile Software Development (ASD) definition, as set out in the Agile Manifesto [Agile Manifesto URL]. Consequently, a software development manager needs to decide which – if any – of these approaches to follow. An alternative solution is however raised and explored in [Theunissen et al. 2005]: that of a hybrid approach. The present paper sets out to extend this notion of a hybrid process in an attempt to find the sweet spot towards an agile open collaborative corporate[1] software development approach.

Note that, as with any proposed solution to the general problem of software development, it would be precipitous to claim that the solution is a *silver bullet* [Brooks 1995]. Instead, the paper's goal is to provide comments on the processes one needs to consider, and thus to facilitate decisions on the appropriate approach for one's circumstances.

Section 2 will sketch issues currently relevant to corporate software development. After establishing the

[1]In this paper we will use the term *corporate* to refer to medium to large enterprises that have their own in-house software developers

Authors Address:
WHM Theunissen, Espresso Research Group, Department of Computer Science, School of Information Technology, University of Pretoria, Pretoria, 0002, South Africa; `mtheunis@cs.up.ac.za`
A Boake, Espresso Research Group, Department of Computer Science, School of Information Technology, University of Pretoria, Pretoria, 0002, South Africa; `andrew.boake@up.ac.za`
DG Kourie, Espresso Research Group, Department of Computer Science, School of Information Technology, University of Pretoria, Pretoria, 0002, South Africa; `dkourie@cs.up.ac.za`

software development needs of corporates we will briefly describe how ASD (Section 3) and OSS (Section 4) have emerged to address certain aspects of these needs. HP's Progressive Open Source model is briefly described in Section 5 as an example of how OSSD may be applied in corporates. Section 6 will introduce the possibility of forging these two separate approaches together in the hope of attaining a hybrid approach to maximise on the benefits that each provides. The section will also highlight some of the difficulties associated with bridging these approaches. Section 7 builds on the notion of a hybrid process by introducing the approach we are following to define and evaluate it. To conclude, a summary of the findings of this paper are presented in Section 8.

2. CURRENT SOFTWARE DEVELOPMENT DRIVERS IN CORPORATES

Although it would be hard to definitively and comprehensively enunciate all factors currently driving corporate software development, few would differ with the assertion that the following list is representative of some of the most important factors:

—The increasing size and complexity of systems
—The pressure to decrease development windows ('Internet time')
—The rapid technological changes
—The need to accommodate changing business needs in the requirements
—The fact that software is driving business' competitive advantage
—The need and will to rely on standardisation to achieve a measure of flexibility
—The aspiration to maximise reuse whether through component based development, integration with legacy systems and service oriented architecture (SOA)
—The emergence of ubiquitous computing where solutions may span multiple platforms (servers, desktops, PDAs, cellphones)

Both ASD and OSS/OSSD address a number of these factors, as will be discussed in the following two sections.

3. AGILE SOFTWARE DEVELOPMENT

During the mid 1990s, a number of software development project managers and consultants realised that the traditional software development approaches were not able to adequately address the types of projects that were emerging during the Internet era. To address the problem, they proposed various so-called *'light-weight'* methodologies. Included in these methodologies is Extreme Programming (XP) [Beck 2000], Feature Driven Development (FDD) [Coad and De Luca 1999], The Crystal Family [Cockburn 2002], Pragmatic Programming [Hunt and Thomas 1999], Dynamic Systems Development Method (DSDM) [DSDM URL] and SCRUM [Schwaber 1995].

In February 2001 proponents of these different 'light-weight' methodologies' met to investigate the possibility formulating a common set of principles to characterise these otherwise diverse approaches. This meeting found common ground between the represented methodologies and resulted in the formation of the Agile Alliance [Agile Alliance Homepage] and the declaration of the Agile Manifesto [Agile Manifesto URL], the latter outlining common principles of these so-called *agile* methodologies [Highsmith].

The Agile Alliance holds *communication*, *simplicity* and *best practices* (with a people-oriented emphasis instead of a process-oriented one) in high regard, and enunciate this in the Agile Manifesto's in terms of a list of values as follows:

> "*Individuals and interactions* over processes and tools
> *Working software* over comprehensive documentation
> *Customer collaboration* over contract negotiation
> *Responding to change* over following a plan" [Agile Manifesto URL].

Since the formation of the Agile Alliance, the adoption of agile approaches by software developers has increased dramatically, which testifies to the viability of these methodologies. It also suggests that these approaches address many of the current software development needs, in particular: the need to *develop at 'Internet time'*, to exploit opportunities offered by *rapid technological changes*, and to have the capability of dealing with *changing business requirements.*

In contrast, the potential vulnerability inherent in an ASD project is its dependence on group tacit knowledge and group face-to-face communication. Consequently the argument may be raised that if such a project was temporarily suspended it might be difficult to re-initiate and/or maintain it.

4. OPEN SOURCE SOFTWARE

During the 1960s and 1970s, the programmers of the day freely exchanged code. This sharing culture came to be seen as a threat to companies who were trying to exploit the commercial value of these programs. As a result, during the 1980s, companies started to clamp down on the code sharing culture by enforcing the use of proprietary licenses. This commercialisation of software and consequent erosion of the code sharing culture gave rise to the GNU Not Unix (GNU[2]) initiative by Stallman. The GNU initiative grew into the Free Software Foundation (FSF) with its definition of Free Software [Free Software Foundation] and other ideological standpoints, including the GNU General Public License (GPL) [Free Software Foundation]. (Note that the 'free' in free software denoted here, refers to the liberty of software and not to the cost associated with the software.)

As the Advanced Research Projects Agency Network (ARPANET) matured, the increased utilisation of an open architecture ensued [Raymond 2003a; Moody 2002]. Furthermore, the *de facto* standards that formed the core of services on the network were built on free software. These services included the Domain Name System (DNS) provided by Berkeley Internet Name Domain (BIND); e-mail services enabled by Sendmail; web content provisioning services provided by the National Center for Supercomputing Applications' (NCSA) HTTPd[3] server (which became Apache) which in turn was based on the – publicly available – hypertext system developed by Berners-Lee [Moody 2002]. When ARPANET became the Internet and was adopted for commercial use, all these free software solutions were drawn into the corporate limelight as businesses extended their presence onto the Internet.

The increased use of the Internet during the 1990's ushered in a new era for the culture of the OSS movement. Not only did the Internet enable a new generation of developers to collaborate and share code in a distributed fashion at a negligible cost, but it also saw the birth of the *free* and open source operating system, Linux. Linux, and the development culture surrounding it, has caused a revolution in the ICT market and its underlying software development approaches. Not only did it provide a platform on which to host services at a fraction of the acquisition cost of the traditional UNIX solutions, it also brought the OSSD culture into large corporations. This came about through a progression: from using OSS solutions to building internal infrastructure at low cost such as print-,file- and intranet web servers; to customising these solutions by providing features that empower the administrator and support for a wider range hardware; and finally, to contributing these changes back into the community. The using and customising activies fuelled each other to further support the OSS paradigm, which naturally spread further into other areas of the company. Similarly, the developers were increasingly using, adopting and contributing to OSS tools that aided them in development.

As more and more corporates started to become interested in this *free* software, the need to remove the misconception associated with the term "free" was realised. A group of free software proponents formed the Open Source Initiative (OSI) [OSI Homepage] which adopted the Open Source Definition (OSD) [The Open Source Initiative].

Examples of the impact that OSS has had on commercial ICT players include:

—Netscape releasing its browser's source code as the Mozilla project [Cusumano and Yoffie 2000; Moody 2002].

—IBM's adoption of Apache in preference to its own Domino web server [Moody 2002].

—HP creating and adopting the Progressive Open Source (POS) approach to software development [Dinkelacker et al. 2001; Fink 2003].

—Sun's move towards an open source culture with the release of OpenSolaris [4] [OpenSolaris Project] and using a community development approach for their Java Development Kit 6.0 (JDK 6.0 codenamed Mustang) [5] [J2SE 6.0 Project].

The aforementioned paragraphs have briefly highlighted the history and trends with regard to Free/Libre/Open Source Software (FLOSS)[6]. The following paragraphs will illuminate the importance of OSS and OSSD for corporates.

The first matter to consider is that OSS/OSSD supports **reuse**. Software development best practices dictate that one should reuse as extensively as possible. Not only will this allow one to focus on the specialisation and uniqueness of the solution, it should also reduces the development time.

[2]A recursive acronym that is a common pun in the OSS culture.
[3]HTTPd is an acronym for HyperText Transport Protocol daemon
[4]Will be released in 2005 under the OSI-approved Common Development and Distribution License (CDDL) Version 1.0
[5]Released under the Java Research License (JRL) which is not OSI approved.
[6]A hybrid term to encompass both Free (also know as Libre) and Open Source Software

Another potential benefit of OSSD is the availability of a larger developer base, extending beyond the human resource capabilities of a company. When following an OSSD approach, a related human resource benefit is the ability to maximise on the skills available from in-house developers, irrespective of their geographical location. This reduces the need to co-locate all the software developers at a geographically central place. An example would be where a project requires, for example, an assembly language specialist as well as a multimedia developer: the assembler specialist could be located at the company's branch on one continent whereas the multimedia team could be located on another. Traditionally this scenario would have demanded that one of the developers would have to move to the other developer's location, or alternatively, the company would be forced to hire an additional developer – with the needed skillset – at the project's geographical location. Within an OSSD environment the need for co-located development is removed because of its inherent distributed characteristic. Furthermore it opens the door to the use of contractors in a just-in-time (JIT) fashion.

OSS may also be used as a strategic 'weapon' for a company/organisation to gain a dominant market share of a specific technology. An example is JBoss using the OSS characteristic of its J2EE Web Application solution to try to become the preferred option for an installation and development base.

There are, of course, several disadvantages to OSSD. For example, the complex social structure may inhibit participation. Its virtual nature is sometimes experienced as impersonal. Although OSSD's disadvantages will not be addressed in more detail in this paper, some may well be ameliorated by the approach that we advocate.

5. HP'S PROGRESSIVE OPEN SOURCE

Hewlette-Packard (HP), having observed the prospering OSS projects such as Linux and Apache, decided to investigate the OSSD model. Out of its study evolved the *Progressive Open Source* (POS) "software engineering paradigm for large corporations" [Dinkelacker et al. 2001]. Dinkelacker et al. [2001] describes this approach in more detail and Fink [2003] places it in the larger HP strategic context. Here, we present no more than a very brief overview of its main concepts.

As implied by its name, the POS approach seeks to "leverage the power of Open Source Software methods and tools for large corporations in a progressive manner: starting from completely within the corporation, to include partner businesses, and eventually complete Open Source."

It is a software engineering paradigm that is based on a three-tiered model for software development. These tiers are: inner source, controlled source, and open source. The tiers reflect the level of exposure, visibility and access control associated with a project. To adopt the model in a company means to ensure that company developers (or at least the majority of developers) follow an OSSD approach. The inner source tier refers to projects that are restricted to company developers only. Controlled source denotes the extension of the project's access to include selected corporate partners. The final tier indicates that a full open source project is being conducted – i.e. one that is open to the global community of developers.

The benefits of POS as stated by Dinkelacker et al. [2001] include:

—Utilisation of a larger developer base

—Utilisation of previously inaccessible skills

—Rapid team re-deployment

—Potential for partner involvement, thus an even greater developer base

Even though the idea of POS development holds promise, the truth of the matter is that adopting the POS approach in a company presents a number of challenges. These matters will not be dealt with in detail here. For further information, refer to Dinkelacker et al. [2001], who acknowledges the problems and classifies them in terms of organisational and technological infrastructure problems.

A final point worth mentioning is that the authors of the POS concept were aware that, if a company is to adopt OSSD, a paradigm shift in its views of its software products will be required. In the OSS paradigm, the end-product to be pursued is source code, as opposed to a binary artifact that is the end goal in the proprietary/traditional context [Dinkelacker et al. 2001; Fink 2003]. When presented with a binary deliverable, the customer has a very limited insight into the product and can only judge the end result. However, the fact that the customer has access to the source code, exposes the company to customer evaluation at an entirely different level. As a result, matters such as having well-documented code that is readable and that is based on an understandable architecture, increase in importance, since these too can be seen and judged by customers.

6. OSS AND ASD AS SOURCES OF TENSION

The current state of play is therefore the following: Firstly, it is recognised that most companies still follow traditional software development methodologies. Secondly, the one hand, the POS approach as described in

[Dinkelacker et al. 2001] appears to have advantages for such companies, and there are many contemporary pressures nudging companies in that direction. Thirdly, as reported in Section 3, an increasing number of companies are either exploring or applying ASD. The question naturally arises: can these two development approaches, ASD and OSSD, be reconciled — much in the same way in which POS has reconciled OSSD and 'traditional' software development. For this reason the paper sets out to explore the possibility of **applying OSSD in an agile environment**.

A number of studies [Raymond 2003b; Warsta and Abrahamsson 2003] have put forward the notion that OSSD is simply another instance of ASD. On the surface, this might seem to be the case, especially if one were to observe a typical team that is following one of these agile approaches. Members are inclined to pride themselves on their ability to *use* OSS tools and libraries – thereby maximising reuse (one of the creeds of ASD) in the process. Furthermore, these individuals will invariably support and promote the OSSD culture. However, these ASD teams seldom produce OSS solutions. To do so would be to move beyond the ASD focus, which is aimed at rapid delivery to a specific customer.

In Theunissen et al. [2005], it is argued that OSSD is *not* simply another ASD instance, and that the two approaches in fact have distinct underlying principles. These conclusions are arrived at by evaluating the extent to which the agile principles (listed in the agile manifesto [Agile Manifesto URL]) are manifested in the generalised OSSD approach used by the prominent OSS projects and described in the literature [Feller and Fitzgerald 2002; Raymond 1998].

Does this then mean that OSSD and ASD are mutually exclusive, or is it possible to find a creative synergy between the two approaches? This question requires a two-phase answer. In the first place, the *tension points* generated by the two approaches need to be identified. This was done in [Theunissen et al. 2005] and is summarised in Subsections 6.1 and 6.2. In the light of these tensions, the second phase of the question is answered in Section 7. That section proposes a hybrid approach that explicitly takes account of the identified tensions.

6.1 Tensions between OSS and Corporate Culture

The corporate environment places certain requirements on the software development process to enforce accountability from employees. As highlighted below, these requirements often conflict with practices that OSS developers take for granted.

—*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 style is used internally. Monitoring which 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 the traditional OSS value system, 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 feel that it is time to do a release. By contrast, 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." [Raymond 1998]. 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. In fact, company code reviews may be seen as an extension of the 'monitoring of developers' requirement discussed above.

6.2 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 would arguably be the most challenging adjustment imposed by the introduction of an OSS culture. 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 not the case in a typical OSS development context. Furthermore, the daily routine of an agile team is usually rigorously controlled. Typically, an agile team starts the day off with a short stand-up meeting, followed by a three to four hour focused session of uninterrupted development. They may then break for lunch, followed by another focused session in the afternoon. During these focused 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 often requires frequent access to communication media such as e-mail, IRC and the Web. These media, which facilitate 24/7 flow of information will appear as extremely 'noisy' to developers accustomed to the agile style.

—*Managing internal and external communication.*
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 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.

—*Delivery schedules.*
The view on time schedules 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. Instead, in an OSSD context, they become 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.

In the section to follow, a process is advocated that is a hybrid between ASD and OSSD. Such a process will clearly need to address these and other ASD/OSSD-generated tension-points.

7. TOWARDS A HYBRID PROCESS

In describing the hybrid process characteristics, we provide below a general description of our overall strategy, we provide a small illustrative example of one of the work definitions, and we discuss the role of tools in the hybrid context.

7.1 The Overall Approach

The preceding sections have made mention of traditional corporate software development, as well as the drift towards both ASD and OSSD. Figure 1 diagrammatically depicts the way in which these various development styles can come together. At the centre, it shows what we call the 'sweet spot': a hybrid process that embodies the best of ASD and OSSD, that maximally reconciles tension points, and that is suitable for corporate adoption.

It is conceded that such a sweet spot is a somewhat idealised notion that may or may not be fully attainable in practice. Nevertheless, it seems worthwhile to articulate it in theory, and to strive after it in practical contexts. It is the long-term goal of the current research project to provide a comprehensive theoretical articulation of such a sweet spot or hybrid process, to evaluate it, to implement it in practical contexts and to assess its acceptability. This is clearly an iterative incremental activity. At this point, however, we report on early work undertaken on this intended trajectory.

Our theoretical articulation of this hybrid process rests on two supporting pillars: the identification of subprocesses, and the articulation of those subprocesses in an appropriate notation. In regard to the latter, it was

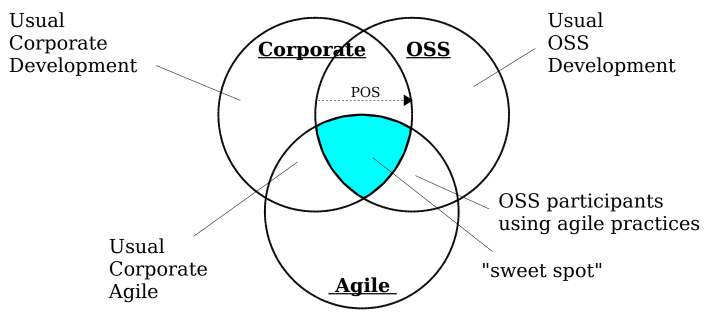

Figure 1. The hybrid process in relation to corporate, agile and OSS

decided to use the *Software Process Engineering Metamodel* (SPEM) Specification (version 1.1) [Object Management Group 2005] as a notational device to define the relevant subprocesses. SPEM is a recognised industry standard for modeling software engineering processes. It is an Object Management Group (OMG) adopted specification and is based on the Unified Modeling Language (UML). That it is suitable in the OSSD context, is illustrated by the work of Lonchamp [2005], where SPEM has been used to model the release management processes of two well-known OSSD projects: the NetBeans IDE and the Apache HTTP Server projects.

In seeking to identify subprocesses, we take as a fixed-point objective that the overall process being defined needs to be as agile as possible. At the same time, it is recognised that various corporate contexts may impose limitations on the degree of agility that is feasible. As a consequence, taking a cue from Cockburn [2002], we suggest a *methodology per project* approach. We are therefore engaged in defining a number of *Work Definitions*[7], which in turn should be selectively combined to produce the appropriate process for a particular project.

7.2 A SPEM Example

As an example, consider the *Submit New Feature Patch* work definition which has been identified as one subprocess in the proposed hybrid methodology. This SPEM work definition is given in Figure 2, relying on the appropriate graphical UML notations as extended by SPEM.

Two *Process Roles*[8] are shown in the figure: the *Casual Developer* and the *Core Developer*. For the purposes of discussion, below we refer to roles as if they represented a single individual. In this case, the casual developer is seen as an external contributor to the project, while the core developer is one of the project's core team members. It is assumed that the core developer has write access to the project repository as well as in-depth knowledge of the code base, project policies etc. Note that in this work definition, it is assumed that the casual and core developer are different individuals. There could be numerous variations on this work definition, reflecting various contexts.

The work definition refers to *Documents* in five different contexts. The references are as follows:

—*Patch* indicates the difference between the published code base and the local customisations.

—*Issue Tracker: Feature Entry* {*new*}/{*closed*}/{*resolved*} refers to different versions of the same document. This document tracks the progress of the feature throughout the development process.

—*Changelog* refers to a list of changes to the product, usually categorised by release number.

The *Activity*[9] for this work definition is: *Submit Feature Request* in terms of which a developer (indicated as a casual developer in the figure), uses the project's issue tracker and writes a request for an extension of the system.

Finally, *Submit New Feature Patch* contains three other work definitions, each of which require their own SPEM specification:

[7] "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" [Object Management Group 2005].
[8] "A Model Element describing the roles, responsibilities and competencies of an individual carrying out Activities within a Process, and responsible for certain Work Products." [Object Management Group 2005]
[9] "A Work Definition describing what a Process Role performs. Activities are the main element of work." [Object Management Group 2005]

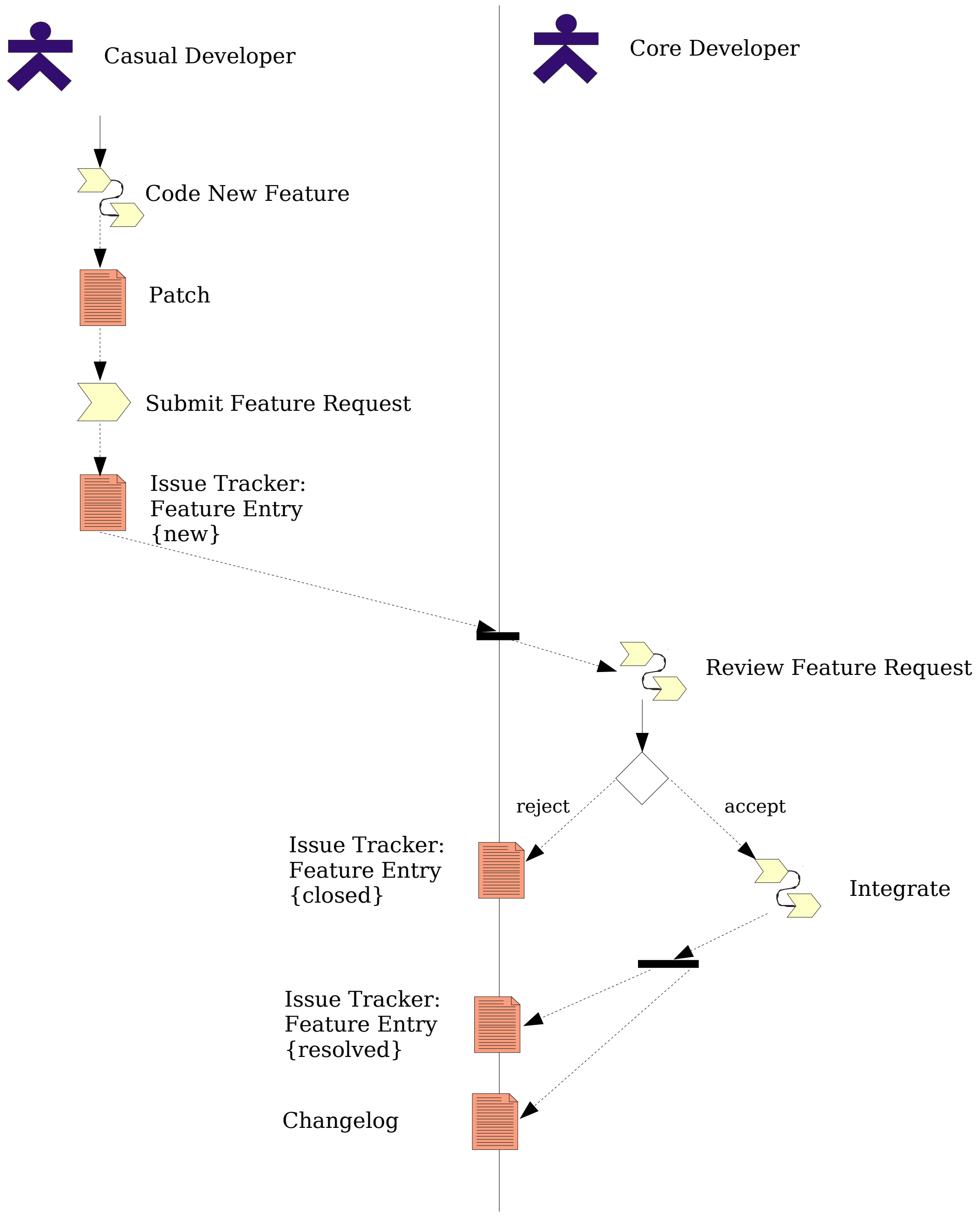

Figure 2. SPEM Work Definition: Submit New Feature Patch

—*Code New Feature* defines the process of developing the code. A developer (again indicated as a casual developer in the figure) may extend the code base to solve the need being experienced.

—*Review Feature Request* refers to the process whereby the core developer performs a review of the submitted request and its enclosed patch, verifying its applicability to the overall project strategy (or roadmap). Furthermore the code's quality and adherence to the project policies are verified.

—*Integrate* defines the process of integrating new code into the existing code base.

The methodological characteristics of these latter two work definitions are not specified in the figure. Their later definition may imbue them with agile properties, or any other process characteristics for that matter. In this example the *Submit New Feature Patch* work definition closely reflects the process followed in traditional OSSD. Its hybrid nature (i.e. where ASD and OSSD are combined) is a result of the fact that either *Code New Feature* and/or *Integrate* could be conducted in an ASD fashion.

We are currently engaged in specifying numerous work definitions, of which the above is typical. In many cases, these work definitions relate exclusively to either ASD or OSSD, but are nevertheless applicable to the proposed hybrid process. There are, however, additional work definitions that are specifically needed in the hybrid process. For example, management may require work definitions to track the time expenditure of corporate developers in relation to their involvement in various projects.

7.3 Tools as an Aid to a Hybrid Approach

The hybrid approach would impose new demands on management, and would ideally need to be supported by a new generation of software tools. Two examples are mentioned below.

One requirement that is prominent in large corporates and often under-emphasised in OSSD, is the need for upfront design. Even agile approaches engage in a measure of upfront design, albeit in an agile way: for example, agile modeling and the XP planning game can be construed as upfront design activities. In the hybrid approach, these agile team decisions, generally made in a face-to-face environment (aided by whiteboard annotations), need to be communicated to the distributed participants. This should be done as transparently as possible to limit the burden placed on the developers – thus supporting agility. Tools may aid in this. An example of adding upfront design capabilities to OSSD tools would be to have an issue-tracker that is able to refer to version-controlled UML models as part of the requirement specifications.

In line with the POS nature of the hybrid approach, corporate developers are usually afforded the opportunity to select one or more projects in which they will participate. In these circumstances, it would be useful to have tools that track a developer's contribution and time expenditure on disjointed projects. A line manager (in a matrix organisational hierarchy) will then be able to track the developers being supervised, thus ensuring that these developers are providing value to the employer. However, the fact that some of these projects might be "external" could cause complications, since internal tools would not track the employer activities on such projects. A tool to be used in the hybrid context would need to address this potential complication.

8. CONCLUSION

The current corporate software development environment is continuously being influenced by a wide range of factors, including technological changes, methodological changes, business requirement changes and trends. These influence the required speed, agility, complexity and scale of software development. It has been pointed out that two contemporary software development approaches have increasingly been meeting these challenges. OSSD in a corporate context was explicitly considered in relation to HP's POS approach.

Based on the evidence that ASD is well underway in some corporates while OSSD is being used in others, we raised the idea of combining the two approaches. Though difficulties of finding synergy between these two approaches were raised, a potential reconciliation through a hybrid approach, containing extensions to both of the existing processes is suggested.

The use of SPEM to define the proposed hybrid methodology has been proposed, and illustrated by an appropriate example. As an immediate research agenda, the SPEM definition of this hybrid process is to be completed, after which the process definition will be evaluated using the Software Process Improvement and Capability dEtermination (SPICE) standard [SPICE Homepage]. In conjunction with these activities, the role of tools to support the process is also being investigated.

ACKNOWLEDGMENT

The research presented in this paper are funded through the National Research Foundation (NRF) and the CeTEIS THRIP grant. This research forms part of the Espresso Research Group's ongoing activities.

REFERENCES

Agile Alliance Homepage. Homepage for the Agile Alliance. Online. http://www.agilealliance.org – 2005/04/01.

Agile Manifesto URL. Manifesto for Agile Software Development. Online. http://www.agilemanifesto.org – 2003/05/20.

Beck, K. 2000. *Extreme Programming Explained: Embrace Change.* Addison-Wesley.

Brooks, J. 1995. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition.* Addison-Wesley.

Coad, P. and De Luca, J. 1999. *Java Modeling in Color with UML.* Prentice Hall.

Cockburn, A. 2002. *Agile Software Development.* Pearson Education, Inc.

Cusumano, M. and Yoffie, D. 2000. *Competing on Internet Time: Lessons from Netscape and its battle with Microsoft.* Touchstone.

Dinkelacker, J., Garg, P. K., Miller, R., and Nelson, D. 2001. Progressive open source. Tech. Rep. HPL-2001-233, Hewlette-Packard Laboratories, Palo Alto. September.

DSDM URL. DSDM Homepage. Online. http://www.dsdm.org – 2003/05/15.

Feller, J. and Fitzgerald, B. 2002. *Understanding Open Source Software Development.* Pearson Education Limited.

FINK, M. 2003. *The Business and Economics of Linux and Open Source.* Prentice Hall PTR.

FREE SOFTWARE FOUNDATION. The free software definition. Online. `http://www.fsf.org/philosophy/free-sw.html` – 2004/10/27.

FREE SOFTWARE FOUNDATION. Gnu general public license. Online. `http://www.gnu.org/licenses/gpl.html` – 2005/03/24.

HIGHSMITH, J. 2001. History: The Agile Manifesto. `http://www.agilemanifesto.org/history.html` – 2002/02/01.

HUNT, A. AND THOMAS, D. 1999. *The Pragmatic Programmer.* Addison-Wesley.

J2SE 6.0 Project. Project homepage for Mustang (J2SE 6.0). Online. `https://mustang.dev.java.net` – 2005/04/01.

LONCHAMP, J. 2005. *Software Process Modeling.* The Kluwer International Series in Software Engineering, vol. 10. Springer, Chapter Chapter 1: Open Source Software Development Process Modeling. Unpublished.

MOODY, G. 2002. *Rebel Code: Linux and the Open Source Revolution.* Penguin Books.

OBJECT MANAGEMENT GROUP. 2005. Software process engineering metamodel specification. formal 05-01-06, Object Management Group. January.

OpenSolaris Project. Project homepage for OpenSolaris. Online. `http://www.opensolaris.org` – 2005/04/01.

OSI Homepage. Homepage for the Open Source Initiative. Online. `http://www.opensource.org` – 2005/04/01.

RAYMOND, E. S. 1998. The cathedral and the bazaar. *First Monday.*

RAYMOND, E. S. 2003a. *The Art of Unix Programming.* Addison-Wesley.

RAYMOND, E. S. 2003b. Discovering the obvious: Hacking and refactoring. `http://www.artima.com/weblogs/viewpost.jsp?thread=5342` – 2004/10/18.

SCHWABER, K. 1995. Scrum development process. In *OOPLSA'95 Workshop on Business Object Design and Implementation.*

SPICE Homepage. Spice: Software process improvement and capability determination. Online. `http://www.sqi.gu.edu.au/spice/` – 2005/07/20.

THE OPEN SOURCE INITIATIVE. The open source definition. Online. `"http://www.opensource.org/docs/definition.php"` – 2004/04/17.

THEUNISSEN, W., BOAKE, A., AND KOURIE, D. 2005. Open source and agile software development in corporates: A contradiction or an opportunity? Jacquard Conference, Zeist, Holland.

WARSTA, J. AND ABRAHAMSSON, P. 2003. 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, 143–147.