Agile Principles and Open Source Software Development: A Theoretical and Empirical Discussion

Stefan Koch

Vienna University of Economics and BA, Department of Information Business, Augasse 2-6, A-1190, Vienna, Austria
stefan.koch@wu-wien.ac.at

Abstract. In the last years, two movements have been widely discussed in the software development community: Agile and open source development. Both have faced some of the same criticism, and both claim some of the same benefits. This paper poses the question whether open source software development is in accordance with agile software development principles and therefore well within the planning spectrum. To this end, the general principles of both movements are detailed and compared, and some empirical data from open source software development projects is given on any similarities and dissimilarities uncovered.

Keywords. Software Development, Agile, Open Source, Software Metrics, Coordination

1 Introduction

Agile software development has been proposed as a solution to problems resulting from the turbulent business and technology environment faced by organizations engaged in software development [8]. Several methods like Extreme Programming (XP) [1], Scrum, Lean Development or Adaptive Software Development exist that embody the principles of this approach as laid down in the Manifesto for Agile Software Development. While there is some evidence, mostly based on singular projects, of positive practical application of these methods, there is a lively discussion on this topic [2,13]. On the other hand, Boehm in his analysis [3] sees both the agile and more plan-driven approaches as having a responsible center, and argues for a risk analysis of a project's characteristics for determining the right balance of disciplines in each case [4]. All of these contributions have not yet ended the ongoing debate, even if some first empirical results on both use of agile principles and the results in cost, productivity and quality have already been published [15], and seem an important step towards this end. There is one point in the discussion that has as yet been largely omitted: Like agile development, there has been another movement which has received much attention in the last years: Open source software development. Also in this case there is considerable debate about benefits and efficiency [11,5,17]. While any

J. Eckstein and H. Baumeister (Eds.): XP 2004, LNCS 3092, pp. 85–93, 2004.

discussion of agile development contains the words cowboy coding, unplanned and undisciplined hacking or similar terms, from which this movement is seen as different, the term open source is not mentioned. Many of the same arguments brought to bear against agile development are also faced by open source development. So, using the planning spectrum introduced by Barry Boehm, the question is where to place open source development? On the far end with the hackers or more towards XP and agile methods? In order to facilitate this classification, this article seeks to compare both methods, and tries to give some first empirical results on any similarities and dissimilarities. Besides clarifying terms and classifications, open source projects and their wealth of data [10,12] might form additional testbeds for agile development methods, and both movements might learn and benefit from each other.

2 Agile Software Development

Both the business and technology environment continue to change at an increasing pace. In software projects, this leads to more frequent changes during the life cycle. Therefore the main question is how to better handle these changes, not stop them by anticipating and including them in the requirements definition, while still achieving high quality and timeliness. To this end, a group of people created the Manifesto for Agile Software Development that values individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, responding to change over following a plan, while explicitly acknowledging the value of the latter items. Furthermore there is a set of principles giving more details, stressing the importance of quality in design, especially in simplicity, the honesty of working code, delivered early and continuously, for communication between developers and sponsors, short feedback loops, the importance of motivated and competent individuals interacting face to face and welcoming change even late in the development. There has been a lot of criticism, one of them being that planning, processes and documentation are essential, and agile development is an excuse for hackers to do as they like, coding away without planning or design [13]. Barry Boehm on the other hand sees agile methods as having a responsible center with a fair amount of planning, and as an important advantage of this movement to draw hackers away from pure hacking [3]. Another point of criticism is the reliance on individual competency, craftsmanship, working together in self-organizing teams in intense collaboration including customers [6]. It is argued that the number of developers possessing excellent technical knowledge paired with the necessary social skills is naturally limited. Also the possible size of agile development teams is discussed. While successful projects with up to 250 people are cited [6], others see a limit of 15 to 20 people.

3 Open Source Software Development

Open source (or free) software has generated much interest in the last years, especially following the rise of Linux and several similar projects like GNU project's

utilities and libraries, the Perl and Tcl programming languages, and the Apache Web server. Regarding these examples, the notion that software of considerable quality can result from this form of development can not be dismissed. Open source software (using the Open Source Definition) is software under a license that fulfills several criteria, giving the user more rights than most other terms of distribution. These include the free redistribution, the inclusion of the source code, the possibility for modifications and derived works, which must be allowed to be distributed under the same terms as the original software, and some others. One example for a license that fits these criteria is the well-known GNU General Public License (GPL) advocated by the Free Software Foundation, which imposes even stricter regulations. While these definitions pertain to the legal terms of distribution, there is also a distinct form of development associated with open source software. The guiding principle is that by sharing source code, developers cooperate under a model of rigorous peer-review and take advantage of "parallel debugging" that leads to innovation and rapid advancement in developing and evolving software products. The best and most widely used description of this development form is an article by Raymond titled 'The Cathredal and the Bazaar', in which he contrasts the cathredal model of commercial software development with the bazaar model of open source using fetchmail as a case study [14]. In this article, he gives several lessons, which form the guiding principles of this form of software development. These therefore constitute the counterpart of the principles behind the agile manifesto and need to be compared with those and reality in open source projects. The criticism faced by the open source development paradigm has several main arguments, the first being that finding and correcting bugs late in the life cycle during coding incurs very high costs [11], a point also discussed in the context of agile development [16]. In addition, effort by people looking for bugs, but not being able to find or fix them, is hidden by spreading it. The inattendance to analysis, requirements engineering and design causes additional limitations due to architectural problems, hiding of useful code, etc. On the other hand it is argued that due to the high modularity of open source code, which is much more stringently enforced to allow more people to work in parallel, and because the context of an error is not lost due to fast release cycles, the costs for fixing bugs in code are not much higher [5].

4 Comparison and Empirical Data

Using several key areas mostly denoted by the principles of agile development and lessons from Raymond's description of open source software development, both movements are compared. Empirical data is used where appropriate to further emphasize and confirm any similarities and dissimilarities. The data employed is derived from several empirical analyses including Apache and Mozilla [12], GNOME [10] and an analysis of Sourceforge, a repository providing free services like version-control or mailing lists for several thousand hosted open source projects. The main idea for this empirical research was to use existing data on the projects available to the public, especially the version control sys-

tems that stores every single check-in of a file by a programmer with additional information like the number of lines-of-code changed, and mailing lists.

4.1 Software Process

Both agile and open source development are no description of a software process as envisioned by this research area. They consist of a set of principles for a software project. While agile in contrast to open source development has several approaches and methods that embody these principles, some of these, e.g. XP, do not have much more detail on the software process, but also restrict themselves to general guidelines. On the other hand, several open source projects have devised elaborate process descriptions, e.g. for release management [9].

4.2 Craftsmanship, Chief Programmers, and Individuals

Agile software development focuses on individual competency and on motivated individuals ('Build projects around motivated individuals.'). This attitude is also described by the term craftsmanship [6], and manifests in pair programming, collective code ownership and mentoring in XP, or chief programmers in FDD. In open source development, Raymond gives a possible explanation for the free effort contributed by using the craftsmanship model, in which the pure artistic satisfaction of designing beautiful software and making it work drives the developers. Empirical data on open source projects show an interesting distribution of the effort invested. While a large number of people participate in the development by giving feedback or testing, a smaller number contributes to the source code, and an even smaller number is responsible for the vast majority of the outcome. In the Apache project, while over 3,000 people submitted problem reports, a core group of 15 programmers out of 400 is responsible for 88% of the lines-of-code [12]. In the GNOME project, which is much more diverse containing several sub-projects, the top 15 of 301 programmers added 48% of the total lines-of-code, while clustering hints at a still smaller 11 person core programmer group. In total, about 1,900 people participated by showing some activity on the mailing lists [10]. Analysis of the Sourceforge repository shows that of more than 12,000 programmers in 8,600 projects, the top 10% are responsible for about 80% of the total source code. Regarding the distribution in single projects, all 65 projects with more than 500k lines-of-code added and at least 5 developers were analyzed. These projects range up to 88 programmers with a mean of 17 persons per project. In the mean, only the top 20.4% of the participating programmers (2.8 people) were necessary to reach 80% of the project's source code. In open source development, each project therefore seems to center around a small number of highly competent and motivated individuals. These individuals, at least some of them, will need to have the social and communication skills necessary to bring larger numbers of people to the project, and hold them there. Open source development can therefore be termed chief programmer teams, as a large number of supporters center around a small inner circle of programmers responsible for most part of actual design and coding.

4.3 Team Size

While successful agile projects of up to 250 people are cited [6], most authors see a size limit at about 15 to 20 persons due to the tightly coordinated teamwork necessary. Both of these ranges are within the bounds of open source projects. As demonstrated above, large projects like Apache or GNOME number a few hundred programmers, but there is empirical evidence for a very small core team in the range of 10 to 20 persons. In smaller open source projects, these numbers decrease accordingly, down to one highly motivated developer with a small team of supporters. The core team seems to have a size of about 5 to 20%, resulting in a group within whom frequent and even personal interactions are easier. While Boehm in his analysis of agile development correctly remarks that 49.9999% of the world's software developers are below average, he concedes that a project does not necessarily require uniformly high-capability people. This seems to exactly mirror the situation in open source projects, where highly capable chief programmers are supported by a larger number of participants.

4.4 Self-Organizing Teams

Agile development stresses the importance of self-organizing teams that are able to rapidly adjust to changing requirements and new challenges ('The best architectures, requirements, and designs emerge from self-organizing teams.', 'At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.'). This requires common focus, mutual trust and respect, and intense collaboration. In open source development, a common focus is ensured, as all participants voluntarily join and therefore follow the goals, which might available implicitly in a vision or example like an existing commercial system. In order to efficiently apply the manpower available to the project, self-organization is strictly necessary. As each participant needs only to do what he wants, self-selection will lead to each one doing what he does best and most efficiently. As Raymond writes '...open source hackers organize themselves for maximum productivity by self-selection...'. For the GNOME project it has been shown that the number of participating programmers in each month, at least until the time of operation, closely follows the efficient manpower distribution proposed for commercial software projects [10]. As there is no central management, the community of developers really seems to be able to organize itself accordingly. Of course, coordination is still necessary also in this form of project, and data from the GNOME project shows that the activity on the mailing lists was strongest during the build-up in active programmers, while declining afterwards [10]. This hints at some sort of briefing or introduction necessary for newcomers. If the data of the GNOME project's source-code versioning system is analyzed, in the mean only 1.8 programmers are found to work together on a single file, even larger files are worked on by only a few programmers. This indicates a division of labour on a higher level. In the Apache project [12], data on the problem reports show that this activity is more widely spread than actual programming, the top 15 people only produced 5% of the total reports, and only three of these were also among the core programmers. This again hints at a high degree of

division of labour, with the tasks that can more easily be performed in parallel being spread out more than others. In the Sourceforge repository, in the mean 1.2 programmers work an a given file, but as this number might be distorted by the large number of small projects, analysis of the 65 large projects (more than 500k lines-of-code, at least 5 developers) was again undertaken. While the number within this group is slightly higher with 1.5 programmers, it is still rather small and near the GNOME project, further enhancing the findings given above. In addition, when the data from the Sourceforge repository is analyzed using text parsing of the commit log messages by the programmers, distinctive names of design patterns show up at about 10% of the projects. One of the main benefits often associated with patterns is improved communication between developers. In fact, there is a significant positive correlation between the number of participants in a project and the number of different patterns used. This indicates that larger teams seem to have an increased need for the improved communication provided by patterns. As this usage is not prescribed, this can be seen as an additional example of self-organization for maximizing efficiency.

4.5 Team Co-location

Agile development aims at close, personal contact and collaboration within the development team ('The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.'), while open source development is performed by large numbers of developers scattered throughout the world [7]. While the empirical data given above suggests that most work is done by a small inner circle of programmers, which could and also do meet in person sometimes, that the self-organization works remarkably well, and is enhanced especially by the Internet medium, this difference remains. In fact, open source software development sees the Internet with all its tools including mailing lists, source code versioning, e-mail, maybe even video-conferencing, as a means for achieving collaboration that is sufficient ('Provided the development coordinator has a medium as least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.').

4.6 Customer Interactions

In agile development, the continuos interaction and collaboration with the customers is paramount. A short feedback loop is necessary to be able to respond quickly to new information like changes in the requirements. These principles are embodied in practices like customer on site in XP. In fact, agile development sees a development team as spanning organizational boundaries and therefore including customers. The same attitude is at the heart of open source development: Users should join the development community and become co-developers in order to more rapidly improve the software ('Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.', 'Release early, release often. And listen to your customer.', 'The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the

latter is better.'). Of course, the necessary precondition for this is the availability of the source code. Even more so than in XP, there is indeed collective code ownership. As has been detailed above, in larger open source projects thousands of people, read users, participate to some degree, not necessarily coding, which only a minority actively does, but in reporting errors, filing change reports or claiming additional functionalities and requirements. In the GNOME project, the nearly 1,900 participants in the mean contributed 10.6 separate postings to the diverse mailing lists. In the FreeBSD project, 82% of the developers indicated that they received some form of feedback on their code, either as a problem report or a bugfix performed on the code [9]. Therefore the open source process is specifically designed to allow customer collaboration, in fact depends on it.

4.7 Early Delivery of Working Code and Feedback

Agile development sees working code as the 'primary measure of progress'. To use this measure in the collaboration with customers, and in order to ensure short feedback cycles, frequent releases of working software are intended ('Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.'). These practices increase the motivation for all participants, allow for easier discussion of the current status and therefore increased chances to uncover necessary changes and efficient possibilities for incorporating them ('Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.'). In open source software development, frequent releases are also propagated ('Release early, release often.'). This has several reasons, including that a larger number of users and co-developers should test und debug the code, thus faster finding and correcting any errors ('Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.'). In order to minimize the lost time spent by participants looking for problems which have already been found or even solved, everyone needs to be kept at the current status. This is achieved by releasing new versions even with only a small number of changes. During the early days of Linux, new releases could occur daily. In the FreeBSD project, approximately 200 developers have been granted commit authority, and any change committed by these individuals results in instant creation of a new release [9]. There is an additional effect of releasing open source software often: Keeping developers constantly stimulated and rewarded, as they have new challenges to rise to, and at the same time see the results of their prior work take shape in the form of improved software. This motivational aspect has been mentioned by 81% of the FreeBSD developers [9]. These points correspond remarkably well: Working software is released often to facilitate change, to keep the effort for rework under control, to ease collaboration with the users and customers, and as a primary measure of progress.

4.8 Changing Requirements, Good Design, and Simplicity

Closely connected to frequent releases is the attitude of agile development towards change. As release and feedback cycles are short, changes in the require-

ments happening anyhow due to the turbulent environment can easily be uncovered, and can more easily be implemented than at the end of the development ('Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.'). Incorporating these changes in addition needs an appropriate design ('Continuous attention to technical excellence and good design enhances agility.'). Therefore agile development stresses the importance of having a simple design that allows for easy changes ('Simplicity - the art of maximizing the amount of work not done - is essential.') and refactoring. The same is also inherent in open source development. There is evidence for both the strive for simple designs allowing for change, and also refactoring, if not actually using this term. Raymond describes how he changed to a new development base ('...I'd be throwing away the coding I'd already done in exchange for a better development base...the first thing I did was reorganize and simplify...'). It is also evident in the saying 'Plan to throw one away; you will, anyhow', originally by Fred Brooks, adopted by the open source community. This of course also hints at prototyping and feedback from users. Another embodiment of refactoring can be found in the principles 'Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.' and 'Good programmers know what to write. Great programmers know what to rewrite (and reuse).' from Raymond. The strive for a simple design is obvious in the saying 'Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.' As the data from the Sourceforge repository suggests the usage of design patterns, these could also be used as a target for refactoring later in the life cycle.

5 Conclusion

As both agile and open source development have been hotly debated, claim some of the same benefits, and face some identical criticism, a comparison seems a logical step. Using their main principles, we have discussed whether open source development can be seen as an agile form of development. In several areas, amazing similarities have been found, for example the emphasis on highly skilled individuals or 'craftsmen' at the center of a self-organizing development team, the acceptance and embrace of change by using short feedback loops with frequent releases of working code, and the close integration and collaboration with the customers and users. For these points, empirical indications were found that at least partially confirmed the presence of these agile principles in open source projects. For example, empirical data suggests that an open source project has a relatively small inner circle of highly skilled and productive developers, around which a larger number of participants and users cluster in a highly efficient self-organization. On the other hand, one major difference showed up, the team co-location and personal contact demanded by agile development, which is not seen as a precondition in open source development. In addition, while Boehm and Turner see small product size as agility homeground [4], open source projects have undoubtedly been able to realize quite large products, with similar comments applying for safety criticality, as quality assurance is often cited as a main

benefit of open source development. Overall, the question whether the 'hackers' and 'cowboy coders' mentioned by the proponents of agile development and others are indeed the open source software developers of the world may therefore not be so easy to answer as it might seem at first glance. Additional research into the real workings of both agile and open source projects is in order, both to compare them to more plan-driven methods and also among each other.

References

1. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading, Mass. (1999)
2. Beck, K., Boehm, B.: Agility through Discipline: A Debate. IEEE Computer **36** (2003) 44–46
3. Boehm, B.: Get Ready for Agile Methods, with Care. IEEE Computer **35** (2002) 64–69
4. Boehm, B., Turner, R.: Using Risk to Balance Agile and Plan-Driven Methods. IEEE Computer **36** (2003) 57–66
5. Bollinger, T., Nelson, R., Self, K.M., Turnbull, S.J.: Open-Source Methods: Peering through the Clutter. IEEE Software **16** (1999) 8–11
6. Cockburn, A., Highsmith, J.: Agile Software Development: The People Factor. IEEE Computer **34** (2001) 131–133
7. Dempsey, B.J., Weiss, D., Jones, P., Greenberg, J.: Who is an Open Source Software Developer? Communications of the ACM **45** (2002) 67–72
8. Highsmith, J., Cockburn, A.: Agile Software Development: The Business of Innovation. IEEE Computer **34** (2001) 120–122
9. Jorgensen, N.: Putting it All in the Trunk: Incremental Software Sevelopment in the FreeBSD Open Source Project. Information Systems Journal **11** (2001) 321–336
10. Koch, S., Schneider, G.: Effort, Cooperation and Coordination in an Open Source Software Project: Gnome. Information Systems Journal **12** (2002) 27–42
11. McConnell, S.: Open-Source Methodology: Ready for Prime Time? IEEE Software **16** (1999) 6–8
12. Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology **11** (2002) 309–346
13. Rakitin, S.R.: Manifesto Elicits Cynicism. IEEE Computer **34** (2001) 4
14. Raymond, E.S.: The Cathedral and the Bazaar. O'Reilly, Cambridge, Mass. (1999)
15. Reifer, D.J.: How Good Are Agile Methods? IEEE Software **19** (2002) 16–18
16. Williams, L.: The XP Programmer: The Few-Minutes Programmer. IEEE Software **20** (2003) 16–20
17. Wilson, G.: Is the Open-Source Community Setting a Bad Example? IEEE Software **16** (1999) 23–25