Harzl *Journal of Internet Services and Applications* (2017) 8:7
DOI 10.1186/s13174-017-0058-z

Journal of Internet Services and Applications

RESEARCH Open Access

Can FOSS projects benefit from integrating Kanban: a case study

Annemarie Harzl

Abstract

Free and Open Source Software (FOSS) and Agile Software Development (ASD) have both been recognized as important software development methods; they have many success stories and share some similarities. However, there is a lack of research regarding the comprehensive integration of the two. This work presents a single case of a hybrid student FOSS project where ASD and FOSS were combined and reports if this combination benefits the contributors. We conducted Action Research (AR) with one sub-team of a large hybrid student FOSS project, and performed three AR cycles based on the Kanban method. The Kanban practices *visualize the workflow, make policies explicit* and *implement feedback loops* were examined during the AR cycles. They are discussed in detail in this paper, which has two main contributions: first, it describes a real world situation, where Kanban is applied to a hybrid student FOSS project, and second, it determines that the combination is experienced as beneficial by contributors. Study participants report a positive effect on communication with other teams and stakeholders due to the use of the Kanban and regard their time acquiring knowledge about Kanban practices as well spent.

Keywords: FOSS, Free open source software, Kanban, Agile software development, Lean, Action research

1 Introduction

This paper examines if the use of an Agile Software Development (ASD) method, namely the Kanban method [3, 24], in the context of a hybrid student Free and Open Source Software (FOSS) project is experienced as beneficial by the project's contributors or not. A large body of research has been published on agile methods [12] and their adoption in distributed settings [5, 30] since the publication of the agile manifesto. Within the last fifteen years both, FOSS and ASD, increased in popularity and are nowadays established processes in software development [9, 12]. Some studies have been conducted about combining ASD and FOSS [15, 17]. In 2003 Warsta and Abrahamsson [40] and Koch [22] in 2004 already determined that the definition of ASD methods and FOSS development are related, but in 2009 Ågerfalk et al. [1] still identified agile development in the context of open source software as a future research area. A systematic literature review in 2013 showed that ASD methods can support FOSS development due to their shared concepts and principles. However, Gandomani et al. [15] did not find a case study comprehensively combining FOSS and ASD.

This paper is an extension of previous work [17], where we positively answered the question "Can FOSS and ASD be comprehensively combined?". This new work improves the conference paper in the following aspects:

- It describes three new Action Reseach (AR) cycles, discovering more details about combining Kanban and FOSS.
- It determines that combining Kanban and FOSS is not only possibly, but it also experienced as beneficial by contributors. In fact, so beneficial that some study participants use personal Kanban to manage all their tasks.
- It reports a positive change in communication and interaction to other teams due to the use of the Kanban method.
- It shows, that contributors regard their time learning about Kanban as worth it.

The project studied in this paper uses an agile approach using elements of eXtreme Programming (XP) and Kanban. The following methods are used to a greater or

Correspondence: aharzl@ist.tugraz.at
Graz University of Technology, Inffeldgasse 16b, 8010 Graz, Austria

lesser extent: *automated unit tests* (the project strives to use test driven development), *pair programming, refactoring, release planning* (occurs in irregular intervals), *short releases, continuous integration, coding standards, collective code ownership, simple design and regular meetings* (weekly), and a Kanban board. Developers of the project are mainly students, but there are many international contributors, who, e.g., provide translations for the user interface, bug reports or feature requests. With this work we focus on the experience of the contributors. Do they benefit enough from integrating new practices, so the additional learning is worth the effort? FOSS projects try to keep the entrance barrier as low as possible and you would not want to drive away possible contributors by imposing practices on them, which do not provide much benefit to them. Thus, this paper attempts to answer the following Research Question (RQ) "Can FOSS projects benefit from using agile methods like the Kanban Method?". To answer this question we will focus on different aspects and try to answer three sub-questions,

- *RQ1.1 Do FOSS contributors, who are coached in the Kanban Method, experience this knowledge as beneficial to their work or not?*
- *RQ1.2 Do interaction or communication during meetings change with the use of the Kanban Method?*
- *RQ1.3 Do FOSS contributors regard their time acquiring Kanban knowledge well spent or rather a waste of time?*

Because FOSS contributors have only very limited time to spend on FOSS work, it is not only interesting to know, if Kanban is experienced as beneficial(see RQ1.1), but also if contributors regard this combination as beneficial enough to their work, so they are willing to spend some of their time on acquiring knowledge about Kanban instead of programming. This work describes three AR cycles, our findings and the experiences gained. It evaluates the possible benefits of integrating an ASD method into a hybrid student FOSS project and if contributors consider the time learning Kanban methods and practices well spent.

2 Related work

Compared to research in the fields of FOSS and ASD in general - which is thriving - research about both methods combined is rather scarce. Most work merely focusses on a collaboration and not integration of these methods. Most of the time only one specific practice is applied to a FOSS project. For example, Turnu et al. [38] created an open source simulation model, added test driven development to the simulation and reviewed the simulation results. Deshpande and Riehle [10] investigated more than 5000 active FOSS projects, whether they used continuous integration. Düring [13] investigated the effects of sprint driven development (people meet for up to one week to work on the code base together) on the PyPy project. Ahmad et al. [2] describe the use of a Kanban board in their project-based software engineering courses. MacKellar et al. [25] describe the client-oriented free and open source development approach, where university software engineering courses collaborate with local non-profits as clients to build an open source project. All of them only apply a small subset of ASD practices, often only one, to a FOSS project or university course or use FOSS elements for university courses. However, to the best of our knowledge there exists no study about comprehensively combining ASD and FOSS, neither in an educational nor industrial context, apart from [17]. Gandomani et al. [15] confirm the lack of literature regarding the combination of both methods.

3 Background

This section describes details about the hybrid student FOSS project under study, my role as a researcher within the project, reasons for choosing Kanban as the method to be integrated, and reasons why this project was selected.

3.1 The FOSS project

The project under study is an umbrella project, consisting of various teams developing mobile applications for different platforms, e.g., Android and iOS. Those applications should enable users to create small creative projects without prior domain knowledge and are primarily targeted at teenagers. These projects can be shared, even between different platforms, e.g. a project created on Android should also work on iOS. Some applications have already been released to the public, others are still in development. The project is a hybrid student FOSS project, meaning that the majority of developers is doing one part of their studies within the project, e.g., their Bachelor thesis or Master project. The project fulfils all terms of the open source definition [29] and uses the GNU Affero General Public License. The project does not exactly adhere to the onion model as described in FOSS research [8, 26, 28, 37]. The layers are similar but outer layers are not larger by an order of magnitude. Most developers (around 130) are students, non-student contributors focus their efforts mostly on translating the applications (around 90 people), creating tutorials, example projects, Youtube videos and bug reporting (around 80 people created an account on the project's issue tracking system; numbers of reporters using other means of reporting are not tracked). Students usually stay between six month and approximately two years with breaks between their Bachelor thesis and Master project. Coding is sometimes done at the university, but most of the time developers contribute from all over the world. Developers change all the time and there are no core teams or core developers, who stay with the project

for multiple years and have all the tacit knowledge and experience, which often builds the backbone of a project. Contributions to the code are more evenly spread than in usual FOSS projects, where a small percentage of contributors develops a large percentage of code [19]. Developers know only a little or a fair amount about agile methods. Practical software development skills range from beginner to intermediate. Different teams work on the applications for the different platforms and are only loosely connected. But as all user-created project should work on every platform, teams have to discuss and agree how the user-created content is stored and exchanged. Teams have to cooperate on the design of the user interface as well, so that the applications feel familiar on every device while adhering to the platform-specific style guidelines at the same time. Some teams are sub-teams of a bigger team and are more closely connected to each other. Super- and sub-teams have to adhere to the same coding standards, they have to jointly decide how the code of subteams is integrated in the code structure and they have to ensure that their code is not interfering with each other. These teams also show the pattern of "enforced ownership" [28], whereas teams without sub-teams do not show such a clear ownership pattern. As software development method, an agile approach is used. It contains elements of XP and Kanban as already described in Section 1. The project shows the same main differences [28] to industrial settings as other FOSS projects: the software is developed by a large number of people, they are not assigned to teams, they self-select the team and topic they are willing to work on, there exists no system-level design, or detailed design [39], and there is no list of deliverables or a project plan. The only distinct roles within the teams are: coordinators, who are the main contact persons for other teams or other interested parties, senior members, who are able and allowed to merge code into the main repository, and developers. The team coordinators have a good overview who is performing which task in the team and make sure the whole team moves into the same direction. Coordinators and seniors volunteer for the position and the team jointly decides who is ready and suitable for the position. Only two people can be considered central management, one is responsible for organizational activities, e.g., managing infrastructure and accounts, and the other, the project head, is mainly responsible for the overall orientation of the umbrella project. The project head is also the project initiator, who had an idea to develop software for teenagers. So the project is not "scratching a developer's personal itch" [31] but instead implementing an educator's vision. Because developers are not the target group of the software a Usability and User Experience (UX) team helps other teams understand user needs and adjust the applications accordingly.

3.2 My role as a researcher within the project

Within the project I am responsible for organizational and supporting processes within the project. So although the study is designed as insider in collaboration with other insiders, power relations may play a part. I am not leading any of the teams, I do not contribute code, and I am not responsible for deliverables. I am on good terms with the project head, who is also the professor grading the student contributors. Therefore, although my role does not have formal hierarchical power, I may be seen as someone with informal power within the organization. Possible implications of this perceived power distance are discussed in Section 6.1. Within the AR team I assumed the role of a Kanban coach.

3.3 Justification of Kanban

The Kanban method [3, 24] strives to accomplish evolutionary change and establish a culture of kaizen (continuous improvement). It consists of four principles and six practices. The principles are *start where you are; pursue incremental, evolutionary change; respect the current processes, roles, responsibilities and titles; promote leadership at all levels*. The practices are called *visualize the workflow; make policies explicit; manage flow; limit Work In Progress (WIP); implement feedback loops; improve collaboratively, evolve experimentally (using models and the scientific method)*.

Kanban was mainly chosen for the following reasons: Many contributors of the FOSS project are inexperienced software developers and the project serves not only software development but teaching purposes as well. According to Ahmad et al. [2] Kanban is a good pedagogical tool and useful for teaching inexperienced software developers about software engineering. Their study participants comprehended Kanban as having a short learning curve and a low adoption threshold. Furthermore, Kanban helps students improve their team work skills, like collaboration and communication, which are both very important for FOSS communities and student teams, making it a lightweight and appropriate approach for FOSS development and teaching purposes. Due to the short learning curve and low adoption threshold Kanban keeps the project's entrance barrier low, in order to not scare off potential contributors.

According to Kniberg and Skarin [21], Kanban is the most adaptive method and allows for small evolutionary changes. Thus, it does not require positional power, which is not available in FOSS projects. Job titles and responsibilities do not have to be changed and no week-long expensive trainings are required. People actively participate in changing their working environment and are encouraged to use their own mind when doing so, instead of following rules imposed from above or outside the team. It is participative and therefore a good match for

AR as a research methodology, which is discussed in Section 4.

Jira Kanban boards[1] were already in use throughout the whole umbrella project and with choosing Kanban as method, the team did not have to change their existing tools.

3.4 Case selection

This hybrid student FOSS project was selected due to the following reasons:

- It allows direct personal contact. More detailed observations on group interaction can be made when one has direct personal contact to study participants. Written information, e.g., in mailing lists, only conveys a small fraction of human communication and interaction and is often misunderstood [34], if not used correctly. If you want to change a process, people affected by these changes need to trust you and trust is easier established through personal contact. It also makes it easier to receive feedback on multiple levels and on this basis to refine researcher skills and the research methodology. We do not see personal contact as prerequisite, but as facilitating the research process.
- Evaluations and experiments are an important part of AR. Students are often used to research and to experimenting with different approaches and willing to evaluate them. Non-student contributors may be more reluctant to do so. Although this setting with mainly student developers is rather unusual, it allows to conduct more questionnaires and evaluations, which is important for research purposes. Moreover, students work on many FOSS projects and are not atypical FOSS contributors per se.
- Reduced bonding time and easy access to team members and artefacts were other reasons to select this project. It can take a very long time to build a good reputation within a FOSS project and to gain enough trust to be allowed to change work processes. We already had a basic trusting relationship with project members hence the bonding period could be minimized and allowed us to conduct the AR within a reasonable time frame. If one has direct access to people and artefacts, e.g., whiteboards and flipcharts, discussions can be done in a shorter time as well and it is easier to acquire all material used in the discussion for later analysis.
- Many characteristics are the same or similar to other FOSS projects. Even the rather unusual face-to-face gatherings are not unprecedented in FOSS projects [13].

Therefore, we think this project is a good starting point to explore Kanban in the context of FOSS development. It should of course not remain the only case, due to its limitations, which will be discussed in Section 6.1. The actual sub-project was co-selected by the participants of the AR. The team coordinator asked me for help regarding the team's motivation and workflow. Probably because of my supporting role in the umbrella project, which is described in more detail in Section 3.2. The team did not know how to overcome their problems and agreed to participate in research to achieve practical outcomes, which would hopefully improve their situation. This resulted in a bias for action, which added to the decision to select AR as research methodology. Other reasons for selecting this sub-team were: commitment to AR to achieve practical outcomes, the team uses the same agile workflow as the other teams, and the team (six to eight people) has roughly the average size (six to twelve people) of teams in the umbrella project. All team members are students, the only non-student developer left the project before the AR started. The AR team is a sub-team of another team and implements a larger feature within the super-teams mobile application, so the team has to coordinate their efforts with the super-team. The part developed by the sub-team has not been released to the public, therefore end users are not part of the sub-team's workflow for now.

4 Research methodology

4.1 Justification of action research

AR is an iterative approach where past cycles inform later cycles, which allows for flexibility and responsiveness to a changing environment and unexpected outcomes, making AR an appropriate approach for a research area, where only little research has been done and theory about it is not fully developed yet [14, 20]. Little research has been done on integrating ASD and FOSS [15], so AR is an appropriate method. According to Dick [11] AR is also a well suited method, if you want to achieve scientific and practical outcomes at the same time. Practical outcomes were the major motivating factor for the participating team, so a research method ensuring not only scientific, but also practical outcomes, increased the commitment of the research participants. In AR researchers and practitioners collaboratively take theoretically informed actions to solve real world problems [16]. People affected by those actions have to commit to them, so outcomes can be achieved. Involvement in action planning and action taking is one way to ensure commitment [11]. AR offers various participatory methods [18] which foster involvement of all AR participants.

4.2 Action research cycles

A modified version of Susman and Evereds' approach [35] was used as research method. The cyclical model contains five stages: *diagnosing, action planning, action taking, evaluating,* and *specified learning*. We performed three AR

cycles consisting of action planning, action taking, evaluation and specified learning, diagnosing was performed continuously. All steps were discussed jointly and decisions were made in consensus. We used a participatory AR approach and wanted to involve participants as much as possible. To provide participants with some theoretical background and information about Kanban principles and practices, they were provided with a video [23] and the opportunity to discuss Kanban with the coach during meetings. We provided this information to foster understanding of Kanban principles and to enable them to decide how to integrate Kanban practices into their workflow in a way, which would allow them to benefit from these changes.

4.3 Data sources

The following types of data sources were used as empirical basis: questionnaires, meeting notes from the weekly team meetings written by the participants and notes written by the researcher, the artefacts (pictures of the whiteboards) produced during the workflow meeting and the feedback meetings, the team's Kanban board, and the team's Cumulative Flow Diagram (CFD). Seven people participated in the AR.

4.4 Data analysis

All questionnaires were statistically analyzed, all notes taken during meetings and discussions were examined for issues of interest to the research and recurring problems or topics. Therefore, statements were coded [7] and grouped together, if they had a common theme, e.g., communication, interaction. These themes then inspired topics for new AR cycles.

5 Action research

This section shortly explains the three cycles and their phases as described in Subsection 4.2. The following Kanban practices were examined during these cycles, *visualize the workflow; make policies explicit; implement feedback loops*.

5.1 Diagnosing

Diagnosing was not a distinct phase before these cycles, but we were constantly searching for clues which could lead us to improvements of the work process. For this purpose we looked at data from the online ticket system Jira and analyzed surveys and meeting notes. Results of the continuous diagnosing will be discussed in the related cycles.

5.2 First cycle

Every team member watched the earlier mentioned Kanban video before starting the cycle. *Implement feedback loops* was introduced to the team during this AR cycle. We discussed different forms of feedback and how we could improve the team's work processes.

5.2.1 Action planning and action taking

For our first feedback cycle we decided to focus on two aspects. First, we fed information about a new Jira workflow (created during [17]) back to the super-team and the UX team. Previously the board was partly a push system, this new workflow implemented a pull system. To inform the other teams, we held an informal meeting with the coordinators of both teams and explained the changes and their rationales. We also asked them for feedback about the new Jira workflow.

Second, the team wanted to eliminate superfluous activities. Were there any activities in their work process, which did not deliver value to the team, their stakeholders or users? To explore this topic, the following topics were explored with the team: is there anything in their work process which they deem unnecessary or hindering. What could be improved in the work process in the next iteration? All topics were collected on a whiteboard and after that every team member voted for his or her most important one. The topic with the most votes was *estimating Jira issues*, therefore, we decided to work on that. First, we analyzed why estimation was identified as an issue, second, we searched for alternatives and third, we compared the initial approach with a possible new method.

5.2.2 Evaluating and specified learning

Inter-team feedback

Both coordinators, from the super-team and the UX team, reacted positively to the new workflow. The super-team coordinator even showed some interest in adopting it for his team. The new workflow also seemed to trigger a thought and discussion process for the coordinators, as both approached the researcher about additional changes to the workflow. The UX coordinator had suggestions to better integrate the UX team into the workflow and the super-team coordinator suggested new workflow states for the code review part. These conversations determined the theme for the second cycle, in which we worked on *visualize the workflow*.

Estimating Jira issues

In ASD effort of issues is usually estimated by the team to acquire some knowledge about the size and complexity of an issue. These estimates are then often used to determine a team's velocity, to predict if a goal can be achieved within the next iteration and to trade work items of similar size in and out of an iteration, if a time-boxed ASD approach is used. There exist different estimating techniques, e.g., story point estimation, Ideal Days (ID) estimation or T-shirt size estimation.

Kanban teams often do not estimate issues, because estimates do not deliver customer value and therefore do

not fulfill a purpose. However, in this project story point estimation is used. "A story-point estimate is an amalgamation of the amount of effort involved in developing the feature, the complexity of developing it, the risk inherent in it, and so on." [6]. Typical scales are derived from the Fibonacci sequence and contain values between one and ten, 21 or 100.

Advantages of story point estimation are:

- It is more accurate than other estimation methods.
- It is independent of time units.
- One can calculate team velocity directly (without needing to convert sizes to numerical values).
- A more fine-grained scale offers more detailed information.

Disadvantages of story point estimation are:

- Many articles about agile estimating specifically mention that joint experience as a team is a key factor to accurately determine story point estimates.
- New teams often struggle with estimating stories effectively, and it takes some iterations until a team's velocity becomes stable.
- In the beginning estimates vary too much and velocity is therefore unstable as well.
- If you cannot determine velocity, you can not derive duration.

The story point scale used in this hybrid student FOSS project contains the values 1, 2, 5, 10, 20, 50, 100, 200, and 500. The story points are not used to determine velocity, to determine the amount of issues for one iteration, or anything of this nature. In fact, right now it is not possible to calculate an average team velocity because iterations are very different in the amount of features implemented and time duration. Hence estimates do not fulfil one of the usual purposes. Team members only use estimates to determine, if an issue is small enough to fit into their schedule.

The major problems for the team with this kind of estimating were:

- The concept of story points is unknown to most contributors and this will probably not change, due to transient team members. Estimating by story points has to be explained before every planning meeting, which consumes quite some time and still this concept stays too abstract for some.
- The scale is too far reaching, making it very difficult to compare the efforts, i.e., what does it mean, if an issue is 200 or 500 times the effort of another issue? Humans have difficulties comparing values over more than one order of magnitude [27, 32].
- Issues and their effort were only compared within one iteration. The values were assigned more or less arbitrary and there was no consistency between planning meetings. This reduced the informative value of the effort estimation to a minimum and team members were unable to use it to select an issue which fitted into their schedule, because the meaning of the different values changed distinctively between two iterations.
- Thus, story points delivered no value to the team or anyone else. Effort estimation just "had to be done".
- Effort estimation did not provide the information team members wanted. For them the most important information concerning effort is connected to the time necessary to resolve an issue. However, relative sizing with story points is not intended to convey information about time. The duration of an iteration is only derived from the total amount of story points divided by the team's velocity.

The team could not determine velocity and therefore could not derive duration and, thus, not determine how long an issue would take. As a result team members saw no meaning in estimating issues and regarded it as unnecessary. To give estimating a meaning for the team we investigated alternatives. Two of them will be shortly explained here, because they motivated the new estimation scale.

First T-shirt sizing, issues are assigned a T-shirt size between, e.g., XS and XXL. Other methods relate size to cars or dogs, but they share the same principle, removing the implied precision of numbers. Theses sizes (T-shirts, cars, dogs) can be related to story point values, e.g., M = 5, L = 10, which can be used in metrics.

Advantages of T-shirt sizing:

- It can expedite the voting processes, because it provides fewer options.
- It does not suggest precision because of a non-numerical score.
- It allows to think in a more abstract way about effort. It even allows for creativity and fun during estimation, if more unconventional sizes are used.
- It can also be beneficial to start with a simpler approach than story points for new teams, e.g., T-shirt sizes, dog sizes etc. and to slowly move towards a numerical scale, when a team has some experience with estimating.

Disadvantages of T-shirt sizing:

- The accuracy of velocity estimates might be reduced, because the estimation scale is less detailed.
- There is no clear mathematical correlation between the different sizes.
- If you want to track effort and velocity over time, you need to convert the sizes to numerical values.

Another method is estimating in units of time. If you approximate time, you can either estimate in elapsed days or in ID. Elapsed days contain all interruptions, which might occur, while working on an issue, whereas ID contain only the amount of time an issue will take, without any interruptions, organizational overhead etc. Since one can never anticipate all possible interruptions estimating in ID is easier than in elapsed days. Because ID only consider the time necessary to finish an issue, they are also an estimate of size, but less strictly than story points [6].

The team liked the idea of estimating in time, because this information is important to them. But they did not want to estimate an exact number of ID, because this would suggest that their estimates are more precise than they actually are. They preferred a less detailed classification and wanted the estimate to convey some information about time duration. Thus, we decided to combine both methods and created a proposal for a future scale containing three values, S, M, and L. Similar to story points, for every team these sizes can mean something different, e.g., S = one to three ID, or S = up to one ID etc. This could lead to misunderstandings between teams, when they collaborate on larger features, but this is true for all team-specific estimation techniques and because there are only three sizes, differences between teams should be negligible.

In the context of this team ID usually are not successional, e.g., if an issue is estimated to take two ID, they can in fact stretch over a few weeks, as people usually do not work full time on the project. One ID can contain several smaller units, which take place on different week days in different weeks, whenever people have time to work on the project.

The team decided to use the following units in the beginning and to adapt them in the future, if necessary: S = up to one ID, M = two to three ID, and L = four to five ID. Larger issues would be divided into smaller issues before working on them. The team decided that this scale should be detailed enough for their purposes. A more in depth classification would not be needed and would have unnecessarily prolonged time expenditure for estimating issues. Although, the direct correlation of issue size to time is not ideal, it is a starting point and provides some information about time duration, which is currently the only information used by team members. We compared this proposal with the story points method. Table 1 shows the results of this comparison.

The story points approach was used for several iterations in the past. For future iterations the team switched their estimating scale to T-shirt sizes, because they deemed this approach more appropriate for them, because it reduces the time needed for voting due to the reduced number of options. After one iteration with estimating in T-shirt sizes, including planning meeting, estimating issues and code development, the team filled in a questionnaire about their experiences in this iteration with T-shirt sizes, and their experiences with story points in the iterations before the last. The questionnaire was statistically analyzed. In one question they were asked to rate how many of their issues they estimated on a scale of 0 to 100% (adapted from Shodan Input Metric Survey [41] with 0% being never, 10% hardly ever, 20% rarely, 30% sometimes, 40% common, 50% half & half, 60% usually, 70% often, 80% regular, 90% always and 100% fanatic), and how many of their tickets they would like to have estimated (desired value). For the story points method the actual mean value was 48% (Standard Deviation (SD)=36%) and for the desired value it was 64% (SD=32%). For the T-shirt size method the actual average value was 92% (SD=4%) and the desired value was 93% (SD=5%). Regarding the usefulness of both methods team members had to select between *not useful at all = 1; hardly useful = 2; neutral/undecided = 3; a bit useful = 4;* and *very useful=5*. Ratings for the story points method were: *hardly useful* and *not useful at all* were selected by 50%, 33% selected *a bit useful* and *very useful*, and 17% were *neutral/undecided* (Mean Value (MV)=2.8, SD=1.5). Ratings for the T-shirt sizes method were: 50% were *neutral/undecided* and 50% selected *a bit useful* and *very useful* (MV=3.8, SD=1). Thus, the use of estimation has improved and team members regard estimation as more useful than before. This is also supported by the analysis of the Jira issues. With story point estimation 63% issues were not estimated or had the default value. "Not estimated" was added as default option only shortly before switching from story point estimation. With T-shirt size estimation only 3% of issues are not estimated. We can not report on the accuracy of the estimations, because team members do not log their working hours for issues.

Despite the advantages of story points and the shortcomings of T-shirt sizes, for this hybrid student FOSS team a time related estimation seems to be more desirable. And because story points and velocity are not used in the value chain the team could switch to a simpler approach. It is more efficient for them, because the scale is less detailed, more intuitive to them, because they are more used to estimating in units of time, and it provides all information the team requires, i.e., if an issue fits into their schedule. Hence, in the context of FOSS projects using simpler methods can sometimes be more useful, than intending to use more elaborate methods and have people not use them, because they are perceived as too complicated, too time-consuming, and as to not deliver value.

An additional result of this questionnaire was, that planning meetings were conducted too rarely. Some new team members had not taken part in one up to this point. This finding called for a closer investigation of that topic. Through analysis of previous planning meetings, we

Table 1 Characteristics of story points and T-shirt sizes as estimating units as seen by team members

Criteria	Story points	T-shirt sizes
Simplicity/Complexity	Estimation depends on the reference issue(s), which has to stay the same over time. Otherwise estimations are not meaningful.	Simpler, intuitive, not interdependent, relates to working time, easier. Everybody can do it.
Level of detail	More exact. If it is not correctly done, it is less meaningful. It is more complicated to assign issues a correct value.	It is less detailed. Estimation and reality are closer to each other.
Usefulness	Useful if it is done correctly. More experience needed to estimate correctly. If there are too many different units, estimating becomes impossible. Usefulness depends on project and team size. Our estimation procedure delivered useless estimates.	Always useful. A rough estimation is always helpful. It is sufficient if a more detailed planning is not necessary.
Informative value	It is more exact but also more error-prone. One needs to know the estimation system. A finish date can be determined more exactly.	Less granular.
Makes use of learning effects	Estimates become more accurate with experience.	No.
Time expenditure	It is much more time-consuming, especially in the beginning.	It is faster.

discovered that they were conducted only twice a year and lasted three hours on average. The long duration made it difficult in the past to set a date, where everybody could participate. Thus, the planning was sometimes conducted in the late evening, after students' working hours in their day jobs and various university courses, making it difficult to concentrate the whole time and the whole planning meeting strenuous. Therefore, team members were not looking forward to planning meetings. A new approach was proposed to the team. They could consider their average number of finished issues per month and plan iterations accordingly, which in their case would mean to plan fewer issues. It was also proposed to plan smaller iterations, which would yield two positive effects. They would become more experienced regarding planning and the meeting would be shorter, making it easier to fit it in everbody's schedule. The team tried it and the following meeting took only one and a half hours, all necessary information for the next iteration was gathered and afterwards one member said "This was fun, can we do it more often."

5.3 Second cycle

Based on the feedback from the super-team and UX team we received during the first cycle we worked on the Kanban practices *visualize the workflow* and *make policies explicit*.

5.3.1 Action planning and action taking

During a regular meeting previously created policies were reviewed and discussed by the team, if they were still valid, if some became obsolete or if they should add new ones.

To revise the Jira workflow we invited all coordinators from all teams within the umbrella project. The coordinator of the super-team and the UX team because they are directly connected to the team and the others because they could deliver different points of view on the matter and at the same time were informed about possible workflow changes. We held a meeting and used whiteboards to draft the new workflow. We collected and discussed all possible states within a new workflow. Afterwards, we put them in order and discussed which transitions were needed, who should be allowed to use the transitions, e.g., code review transitions should only be done by experienced team members, etc. Finally, we sketched the whole workflow on a whiteboard and asked everyone if they could think of any more improvements or if they approved of it this way, until consensus was reached. The whiteboard drawing can be found in the Additional file 1.

5.3.2 Evaluating and specified learning

Only a few policies had to be changed, but reviewing them refreshed everybody's memory. The team realized that they should reconsider them on a regular basis to keep the policies alive and present.

The new workflow (see Fig. 1) is more elaborate than the previous one and allows for a more thorough division of labor and a more detailed view of the current work items. The Issues Pool is a mixture of user feature requests and bug reports, issues created by the project head and the team. Between the Issues Pool and Ready for Development issues are sorted through, e.g. for duplicates, and requirements are defined. Code reviews are split into two parts, one being the actual review and the other one being the actual merge into the master branch. Previously code review and merging code were combined in on state. Because the project requires the code to be tested, before it is merged (ideally a test driven approach is used), this also means that code *and* tests have to be reviewed. As it turned out contributors sometimes are

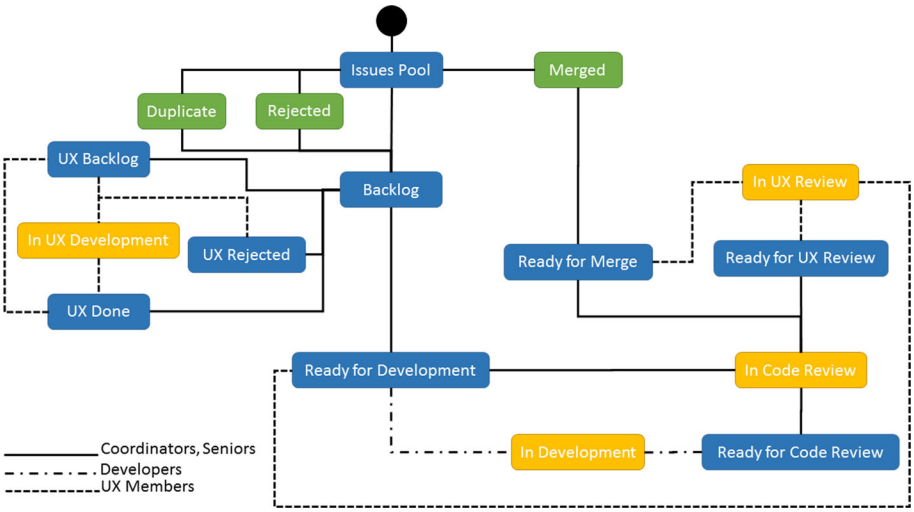

Fig. 1 The new workflow for the whole umbrella project

afraid of the responsibility which comes with merging new code into the master branch, because they are not very experienced in using Git, and because of that, were reluctant to conduct code reviews at all. Now they can start with reviewing code and slowly step up to also merging code. This separation of review and merge also provides the opportunity to include UX reviews of new features neatly into the workflow after the code has been approved technically and before the code is merged into the master branch. This was an important request from the UX team. Before this change, developers often forgot to ask them for feedback before merging code, because the UX review was not represented in the workflow. As a result sometimes new features were merged, that did not meet UX specifications and had to be repaired. Now if an issue received UX input in the beginning, before developing it, it has to undergo UX review as well before it can be merged. Transitions between UX specific states can only be used by UX members. Some transitions are restricted to seniors members, because the actions connected to those transitions require a certain amount of knowledge about the project. Actions necessary for a release are only executed by a handful of persons. These steps are not represented in the Jira workflow.

The new workflow was initially intended for the AR team and only meant to be distributed to other teams if it proved beneficial, but many coordinators wanted to test it with their teams as well, so we decided to adopt the new Jira workflow throughout the whole umbrella project. Some teams are already considering introducing WIP limits and one team even started its own improvement initiative. It is about improving code reviews, not introducing Kanban, but it is a first step into actively improving their work process. Thus, we can observe that the effects of the Kanban initiative have spread past the original team and triggered changes in other teams as well.

5.4 Third cycle

The third cycle was again about *implement feedback loops* but this time it was similar to a retrospective. What did the team achieve so far? Do they recognize improvements or not? Has something changed for the worse? What could be next steps?

5.4.1 Action planning and action taking

The team's CFD, a questionnaire and a feedback meeting were used to answer these questions. The questionnaire included the topics communication, feedback about Kanban, and the importance of a designated coach role. During the feedback meeting the team should collect issues which they would like to improve in their work process. Arising topics were discussed. Team members gave examples and reasons why things should change. After the major topics were collected, the team decided in consensus on concerns to work on.

5.4.2 Evaluating and specified learning

Jira Board and CFD

The analysis of the Jira board and CFD (see Additional file 1) showed, that the team was not developing more issues than before, only the distribution was a bit more evenly. Before the Kanban coaching the team sometimes binge-worked or updated their Jira board irregularly.

Questionnaires

In the questionnaire about Kanban (see Additional file 1) AR participants had to decide if they experience their Kanban knowledge as *very beneficial = 1; a bit beneficial = 2; undecided = 3; a bit useless = 4; very useless = 5* to their

work within the project and outside the project. The same choices were given to rate the usefulness of a coach. 29% rated Kanban knowledge as *very beneficial* for their work within the project and 71% rated it *beneficial* (MV=1.7, SD=0.48). 57% rated it as *very beneficial* and 43% as *beneficial* for their work outside the project (MV=1.4, SD=0.53). Interestingly, more people rated Kanban knowledge as *very beneficial* for their work outside the project (57%) than within the project (29%). The following answers may explain this discrepancy. 28% created a personal Kanban board [4], which contains tasks concerning their whole life and not only the project, which makes Kanban outside the project more important for them than within the project. Another reason was that sometimes the time spent on Kanban practices reduced the available time for writing code. Team members often work in companies while studying and therefore have only a very limited amount of hours to work on the project and sometimes spent more time on Kanban practices than coding. One team member responded: "I think Kanban is generally very beneficial to the team. However, I think the programming vs. Kanban work ratio is not perfect right now". Additionally, he talked to his boss at his workplace to introduce Kanban to the company. He spends more time working at the company than in the FOSS project, so he rated the usefulness of Kanban knowledge outside the project higher than within the project.

The presence of a Kanban coach was rated *very beneficial* by all team members. On the one hand team members appreciate the role of a coach and on the other hand are aware of the risks a (missing) coach entails: "The board is now up-to-date but in my opinion the reason is more the presence of the coach than Kanban", "Without a coach this could end in chaos", "Without a coach there is a greater risk of leaving important things out or to ignore them because it's easier", "You can become dependent on the coach, so you don't achieve anything without him or the coach could take on a leading role instead of a coaching role or he might be seen that way by some team members", "The team would have never dealt this intensely with the topic without a coach", "Feedback from outside the team is great, because inside the team you often develop a tunnel vision", "The coach points out possibilities for improvement we may have not discovered otherwise", "Whenever you try to change something, you are easily tempted to fall back into old habits. In these moments someone who puts you back on the right path is very precious".

In general, team members seem to experience the use of the Kanban method as beneficial to their work inside and even outside the project, which answers RQ 1.1.

Regarding the communication, AR participants had to rate their communication with stakeholders and within the team on the following scale *very positive change = 1; a bit positive change = 2; unchanged/undecided = 3; a bit negative change = 4; very negative change = 5*. The communication to stakeholders outside the team changed *positively* or *very positively* according to 71%, and 29% said it *did not change* or they are *undecided* (MV=2.1, SD=0.69). As most important changes, communication with the super-team, the project head, the UX team and users were mentioned. Team members are aware that there is still plenty of room for improvement, but thanks to the stakeholder analysis in cycle zero they now know who they need to talk to and which relations they need to cultivate more. Within the team itself changes were not that distinct. 43% said it changed *positively* or *very positively* and 57% said it *did not change* or they were *undecided* (MV=2.3, SD=0.95). Text comments showed that team members consider the communication within the team generally as very good, so it is not so surprising, that changes were not as evident. Some team members noticed that meetings were held more regularly and attendance improved, although they could not say if it was due to Kanban or the coach's presence. In general, team members recognized a positive change regarding communication and interaction within the team and to other teams. Therefore, we are confident that the communications behavior changed and we confirm RQ 1.2. Concerning the interaction during meetings, the most visible change is, that the team now almost always uses their Jira board, when discussing issues. Previously they only seldom looked at it.

Regarding RQ 1.3 team members had to answer if they regard the benefit from using Kanban, as worth the time for all the coaching sessions and Kanban meetings. Possible answers were *yes, definitively = 1; yes = 2; undecided = 3; no = 4; no, definitively not = 5* and *other*. 29% answered *yes, definitively* and 71% answered *yes* (MV=1.7, SD=0.48), so this team regards their time well spent learning about Kanban.

Feedback meeting

During the meeting several concerns arose, e.g., scope of issues is too extensive, the most important ones which were selected to be worked on will be discussed here. A major issue was that team members wanted to be more proactive when dealing with problems and bottlenecks. Although they identified a major bottleneck in their workflow, they ignored it for a while and waited for someone else to resolve it for them. This was mentioned by team members as both an advantage and disadvantage of Kanban. On the pro side problems and bottlenecks become visible quite early, but on the con side, if nobody feels responsible and the bottleneck is not resolved, the WIP limit stops the whole development process. We talked about this and it was explained to them, that this is one purpose of WIP limits, to make it impossible to ignore problems and bottlenecks so they have to be resolved and not linger on for all eternity. If they still choose to ignore it,

they will soon see the consequences. They understood this explanation and consequently they made a time-phased plan how to resolve this bottleneck and immediately took the first steps. Another effect of this situation was, that they want to give the work on the project more priority, so this "Waiting for Godot" will not happen again. As this is a more vague resolution we plan on examining this strategy in the future.

6 Results and discussion

This study shows some promising outcomes regarding the possible benefits of integrating Kanban practices into FOSS development and also some possible challenges.

RQ1.1 asks if FOSS contributors experience Kanban knowledge as beneficial to their work and all study participants answered yes. Interestingly, some study participants even use personal Kanban after the AR and one wants to introduce it to his working environment. In this case the use of Kanban transferred over into other parts of the participants' lives.

RQ 1.2 raises the question whether interaction or communication during meetings change with the use of the Kanban method. The majority observed a positive change in interaction and communication especially with other teams and stakeholders. Changes within the team were not experienced as that distinctive.

RQ 1.3 asks if contributors see their time learning about Kanban as well spent or not. All of them regard their time as well spent.

Interesting to note is the role of the coach. It was rated as *very beneficial* by all study participants, which begs the questions, if integration of an ASD method can be accomplished in other FOSS settings where a coach is not available to the team and if the observed benefits are stemming more from the coach's presence than the use of Kanban practices. Reliance on a coach could also become a problem, if a team depends on him or her too much, as one participant stated in Section 5.4.2. Another challenge may be, that a team decides to ignore problems, despite all Kanban practices, and one has to figure out how to overcome such blockades. Adding a new role to already established FOSS roles [36] could be one way to solve both problems. Someone with interest in the topic could gather teaching materials, e.g., videos, which are already available online, and could remind contributors to pay attention to the WIP limits and flow etc., very similar to inspecting code and giving feedback on the code. This role could also be mindful of possible blockades and speak out, if he or she discovers one.

Although these results do not provide comprehensive proof that all FOSS projects can profit from using agile methods, they show a case where a project profited from integrating agile methods. The introduction of new methods most probably takes longer than in companies due to the limited amount of time, contributors can spend on a project per week, so one needs patience and endurance to introduce Kanban. The team will probably experience some problems with fall-backs into old habits, e.g., trying to sit problems out instead of resolving them quickly, before people develop a sense of kaizen.

6.1 Limitations

This study has very limited external validity because it is limited to one hybrid student FOSS project. The results are not generalizable to other teams or FOSS projects without further research.

Response bias, might have led to a more positive feedback about Kanban and its possible benefits, because team members know the researcher personally.

Another limitation could be the researcher's positionality [18] in the setting. Herr and Anderson [18] describe positionality as asking the question, "Who am I in relation to my participants and my setting?". In Subsection 3.2 power distance was already identified as a possible limitation. Participants perceiving the researcher as someone with informal power, may result in research bias, since suggestions the researcher makes could be accepted due to this perceived power distance and not only because team members agree with suggestions. If possible two or more alternatives for future steps were proposed to the team, before they decided, so to hopefully counterbalance this bias a bit.

The special setting of a hybrid student FOSS project may be seen as a limiting factor as well, because some characteristics differ from traditional FOSS projects. Developers want to earn course credits and not only earn reputation among other developers, contribute to a bigger cause or "scratch a personal itch" [31]. There exists no group of core developers, which in typical FOSS projects consists of 10 to 20% of a team, and which creates around 80% of the source code [22], and student members change regularly. This could be a future area of research, determining if and how these different characteristics impact a FOSS project.

Students as main developers of this project may also be considered as a limiting factor, because they have not finished their studies. However, many people without a formal education in software engineering and from various backgrounds are FOSS contributors, regardless of their formal education. According to Salman et al. [33] when a development approach is new to both groups, students and professionals, show similar performances in carefully scoped software engineering experiments.

7 Conclusion

Studying the integration of ASD and FOSS can further our understanding of both worlds. In this paper we were able to show, that a specific student FOSS project benefited

from integrating Kanban. Future work should cover more AR cycles to see if improvements are permanent. The cycles so far should be conducted with other teams and within other FOSS settings to assure validity of the findings so far. Another interesting aspect would be to examine the importance of a coach more closely and to figure out if an additional FOSS role could replace an on-site coach.

Endnote

[1] https://www.atlassian.com/software/jira

Additional file

Additional file 1: Supplementary material. (ZIP 674 kb)

Acknowledgements
The author would like to thank Wolfgang Slany for his invaluable contribution to this study.

Competing interests
The author declares that she has no competing interests.

Publisher's Note

Received: 14 August 2016 Accepted: 17 May 2017
Published online: 06 June 2017

References

1. Ågerfalk PJ, Fitzgerald B, Slaughter S. Introduction to the special issue - flexible and distributed information systems development: State of the art and research challenges. Inf Syst Res. 2009;20(3):317–28.
2. Ahmad MO, Liukkunen K, Markkula J. Student perceptions and attitudes towards the software factory as a learning environment; 2014.
3. Anderson D. Kanban - Successful Evolutionary Change for Your Technology Business. Seattle: Blue Hole Press; 2010.
4. Benson J, DeMaria Barry T. Personal Kanban: Mapping Work, Navigating Life. Seattle: Modus Cooperandi Press; 2011.
5. Boland D, Fitzgerald B. Transitioning from a co-located to a globally-distributed software development team: A case study at Analog Devices, Inc. In: Proceedings of 3rd Workshop on Global Software Development. Edinburgh; 2004. p. 4–7. http://dx.doi.org/10.1049/ic:20040303.
6. Cohn M. Agile Estimating and Planning. Upper Saddle River: Pearson Education; 2005.
7. Corbin J, Strauss A. Basics of Qualitative Research: Grounded Theory Procedures and Techniques (2nd Edition). Thousand Oaks: Sage publications; 1998.
8. Crowston K, Howison J. The social structure of free and open source software development. First Monday. 2005;10(2). http://firstmonday.org/ojs/index.php/fm/article/view/1207.
9. Crowston K, Wei K, Howison J, Wiggins A. Free/libre open-source software development: What we know and what we do not know. ACM Comput Surv. 2012;44(2):7.
10. Deshpande A, Riehle D. Continuous integration in open source software development In: Russo B, Damiani E, Hissam SA, Lundell B, Succi G, editors. Open Source Development, Communities and Quality, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, OSS 2008, September 7-10, 2008, Milano, Italy, *IFIP*, vol. 275. Cham: Springer; 2008. p. 273–80.
11. Dick B. A beginner's guide to action research. 2000. http://www.aral.com.au/resources/guide.html. Accessed 19 May 2017.
12. Dingsøyr T, Nerur SP, Balijepally V, Moe NB. A decade of agile methodologies: Towards explaining agile software development. J Syst Softw. 2012;85(6):1213–21.
13. Düring B. Sprint driven development: Agile methodologies in a distributed open source project (PyPy). In: Abrahamsson P, Marchesi M, Succi G, editors. Extreme Programming and Agile Processes in Software Engineering, 7th International Conference, XP 2006, Oulu, Finland, June 17-22, 2006, Proceedings, *Lecture Notes in Computer Science*, vol. 4044. Berlin Heidelberg: Springer; 2006. p. 191–5.
14. Edmondson AC, McManus SE. Methodological fit in management field research. Acad Manag Rev. 2007;32(4).
15. Gandomani TJ, Zulzalil H, Ghani AAA, Sultan ABM. A systematic literature review on relationship between agile methods and open source software development methodology. CoRR. 2013;abs/1302.2748:1602–1607.
16. Greenwood DJ, Levin M. Introduction to Action Research: Social Research for Social Change: SAGE Publications; 2007.
17. Harzl A. Combining FOSS and Kanban: An Action Research. In: Open Source Systems: Integrating Communities - 12th IFIP WG 2.13 International Conference, OSS 2016, Gothenburg, Sweden, May 30 - June 2, 2016, Proceedings; 2016. p. 71–84. doi:10.1007/978-3-319-39225-7_6. http://dx.doi.org/10.1007/978-3-319-39225-7_6.
18. Herr K, Anderson G. The Action Research Dissertation - A Guide for Students and Faculty 2nd Edition. Thousand Oaks: SAGE; 2015.
19. Kagdi HH, Hammad M, Maletic JI. Who can help me with this source code change? In: 24th IEEE International Conference on Software Maintenance (ICSM 2008), September 28 - October 4, 2008. Beijing, China: IEEE Computer Society; 2008. p. 157–66.
20. Kampenes VB, Anda B, Dybå T. Flexibility in research designs in empirical software engineering. In: Visaggio G, Baldassarre MT, Linkman SG, Turner M, editors. 12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008, University of Bari, Italy, 26-27 June 2008, Workshops in Computing. BCS; 2008. http://ewic.bcs.org/category/16334.
21. Kniberg H, Skarin M. Kanban and Scrum - making the most of both. In: IEEE Computer Society. USA: C4Media; 2010.
22. Koch S. Agile principles and open source software development: A theoretical and empirical discussion. In: Eckstein J, Baumeister H, editors. Extreme Programming and Agile Processes in Software Engineering, 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004, Proceedings, *Lecture Notes in Computer Science*, vol. 3092. Berlin Heidelberg: Springer-Verlag; 2004. p. 85–93. doi:10.1007/b98150. http://dx.doi.org/10.1007/b98150.
23. Leopold K. Kanban im schnelldurchlauf. https://youtu.be/6nOUa6E0250. Accessed 19 May 2017.
24. Leopold K, Kaltenecker S. Kanban in der IT - Eine Kultur der kontinuierlichen Verbesserung schaffen. Hanser. 2013.
25. MacKellar B, Sabin M, Tucker A. Bridging the academia-industry gap in software engineering: A client-oriented open source software projects course. In: Open Source Technology: Concepts, Methodologies, Tools, and Applications, chap. 99. Hershey: IGI Global; 2015. p. 1927–50.
26. Masmoudi H, den Besten M, de Loupy C, Dalle J. peeling the onion. In: Boldyreff C, Crowston K, Lundell B, Wasserman AI, editors. Open, Source Ecosystems: Diverse Communities Interacting, 5th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2009, Skövde, Sweden, June 3-6, 2009. *Proceedings, IFIP Advances in Information and Communication Technology*, vol. 299. Springer; 2009. p. 284–97.
27. Miranda E. Improving subjective estimates using paired comparisons. IEEE Softw. 2001;18(1):87–91.
28. Mockus A, Fielding RT, Herbsleb JD. Two case studies of open source software development: Apache and mozilla. ACM Trans Softw Eng Methodol. 2002;11(3):309–46.
29. Perens B, et al. The open source definition. Open sources: voices from the open source revolution. 1999;1:171–88.
30. Ramesh B, Cao L, Mohan K, Xu P. Can distributed software development be agile? Commun ACM. 2006;49(10):41–6.
31. Raymond ES. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. Sebastopol, CA, USA: O'Reilly & Associates, Inc; 2001.
32. Saaty T. Multicriteria Decision Making: The Analytic Hierarchy Process. Analytic Hierarchy Process Series: R W S Publications; 1996.
33. Salman I, Misirli AT, Juzgado NJ. Are students representatives of professionals in software engineering experiments? In: 37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1. Piscataway: IEEE Press; 2015. p. 666–76.

34. Schafer S. Office E-mail: It's fast, easy and all too often misunderstood. New York: Int Herald Tribune; 2000.
35. Susman GI, Evered RD. An Assessment of the Scientific Merits of Action Research. Adm Sci Q. 1978;23(4):582–603.
36. Tatham E. Roles in open source projects. 2010. http://oss-watch.ac.uk/resources/rolesinopensource. Accessed 19 May 2017.
37. Teixeira J, Robles G, González-Barahona JM. Lessons learned from applying social network analysis on an industrial free/libre/open source software ecosystem. J Internet Serv Appl. 2015;6(1):14:1–14:27.
38. Turnu I, Melis M, Cau A, Marchesi M, Setzu A. Introducing tdd on a free libre open source software project: A simulation experiment. In: Proceedings of the 2004 Workshop on Quantitative Techniques for Software Agile Process, QUTE-SWAP '04. New York, NY, USA: ACM; 2004. p. 59–65.
39. Vixie P. Open Sources: Voices from the Open Source Revolution. Sebastopol: O'Reilly; 1999, pp. 91–100. http://www.oreilly.com/openbook/opensources/book/vixie.html.
40. Warsta J, Abrahamsson P. Is open source software development essentially an agile method? In: Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering. Portland, Oregon: IEEE Computer Society; 2003. p. 143–147.
41. Williams L, Krebs W, Layman L. Extreme Programming Evaluation Framework for Object-Oriented Languages – Version 1.4. 2004;TR-2004-18:. http://www.realsearchgroup.org/realsearch/publications/extreme-programming-evaluation-framework-for-object-oriented-languages/.