Spicing Up Open Source Development with a Touch of Crowdsourcing

Terhi Kilamo*, Jurka Rahikkala†, and Tommi Mikkonen*
* Department of Pervasive Computing, Tampere University of Technology
Korkeakoulunkatu 10, FI-33720 Tampere, Finland
Email: terhi.kilamo@tut.fi, tommi.mikkonen@tut.fi
†Vaadin ltd.
Ruukinkatu 2-4, FI-20540 Turku, Finland
Email: jurka@vaadin.com

Abstract—**Leveraging the work and innovation of third party developers has risen as a viable business model for software companies. Most obviously, open source software has become an opportune ecosystem for creating innovative products with minimum number of paid developers. Then, having a company core where most of the development is done in-house by developers employed by the company can lead to a situation where the community contributions are not smoothly integrated into the code base of the open source product. Similarly, during the last decade, the use of the specialized workforce available online – so-called crowdsourcing – has received a lot of attention. While tapping into the unknown group of experts differs from the open source community-driven approach, they share certain similarities as well. In this paper, we present results of an initial study on how adopting and utilizing elements from crowdsourcing can help to boost community contributions in company lead development of an open source software product. We further discuss how such activity can be supported by an in-house development model where all contributions whether done by the developers of the company or community participants enter a common, automated integration pipeline.**

Keywords*—*Open source software, crowdsourcing, software ecosystems

I. Introduction

A key lesson learned from the expansive thinking of the 1990s is that it is no longer practical for every company to produce and own every aspect of its business [1]. While many – if not all – companies rely on software and even produce it, not all of it is an integral part of their business output. Instead, leveraging the work and innovation of third party developers can be a part of a viable business model even for software intensive companies themselves. In a similar fashion, software ecosystems [2] where businesses work as a unit, co-operating and competing together to produce new innovations and products, and to satisfy their customers' needs, have emerged as a way to make business today.

In such contexts, open source systems have demonstrated the strength of the community-driven development model [1]. Slogans such as "*given enough eyeballs, all bugs are shallow*" (http://en.wikipedia.org/wiki/Linus's_Law) and "*scratch your own itch*" (http://en.wikipedia.org/wiki/The_Cathedral_and_the_Bazaar) have shaped how software companies work, making open source software development a viable business model and giving a voice for developers participating in the development.

Unfortunately, balancing between company and volunteer participation is difficult. In particular when the company behind a piece of software is relying on employees forming the core of the company, business interests and the corporate way of working easily take over the control. This results in complications when integrating community contributions into development actions that have been planned with business interests in mind. This in turn is frustrating for volunteers, who can create bug reports, included in a joint bug database, but cannot still easily contribute fixes to those bugs all the way up to the product source code, if screening the community contributions requires time-consuming manual work inside the company. Overlooking the proposed fixes then prevents the community from scratching its own itch, and leads to unsatisfactory inclusion of community contributions to the upstream code base. In addition, it can lead to the community contributions coming to a halt.

Crowdsourcing [3] is one modern way to tap into the group of volunteers accessible over the Internet. In 2006 Howe defined crowdsourcing as "*the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call*" [4]. While Howe further in his blog goes into defining the so-called "Soundbyte Version" of crowdsourcing as "*the application of Open Source principles to fields outside of software*" [5] there are in fact subtle differences when comparing Open Source and crowdsourcing. Crowdsourcing is more about outsourcing company specific tasks to an unknown, heterogenous online workforce who self-select to perform such tasks to the benefit of the company. OSS in turn is based in commons-based peer-production [6] and the notion of openness. The community in open source is similarly self-selecting but in addition they have more significant amount of control on what gets done [3]. In addition, their contributions are open and available to the benefit of the entire community.

In this paper, we propose a solution where approaches commonplace in crowdsourcing are included into an Open Source project with a company core to motivate more code contributions from the community. To foster such activity, the in-house development model has been partially revisited to reduce manual work needed for accepting contributions. The changes culminate in continuous integration and test automation, which allows frequent builds, with small changes between them, and in minimizing program code handovers during the build process, which enables developers to run tests independently.

This study, a single case study in a software intensive company with an Open Source product, aims to answer how crowdsourcing can answer to a company-led Open Source project's need for timely delivery of specific contributions. In particular, in this paper we seek to investigate, how the selected approach handles the challenges of crowdsourcing [7], such as task allocation, communication, scheduling, quality management, IPR, and motivation and what were the benefits gained in this case constext.

The rest of this paper is structured as follows. In Section II, we discuss the basic principles of open source development and crowdsourcing. In Section III, we introduce the context of this study, Vaadin, which is an open source company developing a Java-based framework for web applications. There, we also provide insight to the experiment the company has executed to simplify the inclusion of community contributions to the main code base and draw contributions in the form of an open call. In Section IV, the research approach is discussed. In Section V, we introduce our main results, and in Section VI, we provide an extended discussion regarding the outcome. Finally, towards the end of the paper, in Section VII, we draw some final conclusions.

II. Background

The rapid change in the field of software development has revolutionized the way software is composed. Traditionally, software has been composed module by module, which have been integrated into complete systems manually; today, numerous software organizations operate so that whenever new features are introduced, they are immediately automatically incorporated into the final system using techniques referred to as Continuous Integration [8], [9], Continuous Deployment, and as the most latest trend DevOps [10], where software developers and operators responsible for the information systems collaborate in close communication. The promise is that the tighter cooperation offers rapid continuous development and thus benefits such as a faster time to market, timely addressment of user needs, shortened lead times and a better quality of new releases.

This change from manual to automatic integration, compilation, testing, and deployment of software introduces also changes the fashion software can be developed. In contrast to the traditional model, where manual work was needed for integration, the new model allows rapid introduction of new features, which can be automatically tested. If tests show that the change is desirable, the resulting system can be immediately deployed and taken to use, which in the traditional model was usually cumbersome and error-prone. Moreover, as manual work is minimized, massive parallel development becomes a viable option. To benefit from this option, Open Source Software and crowdsourcing introduce two plausible models for inclusion of features and fixes from third party developers. While the two share some commonalities and are often mixed, there are certain unique characteristics for both of these approaches. In the following, we introduce open source software and crowdsourcing in more detail to clarify what we mean with these terms in this paper.

A. Open Source Software

Open source software (OSS[1]) incorporates aspects from open access to the source code and the ability to develop it further to a complete business model. One of the most distinctive aspects of OSS is its software development approach – community driven development.

Communities enable open collaboration of keen participants who are distributed all over the globe still maintaining the power to develop high-quality products [11]. This claim emerges from the two often cited fortes of open source [12]: self-motivation and high product quality. First, Raymond's "*scratching your own itch*" indicates the inherent motivation for a developer to focus on things that are close to their own heart. Second, Linus' "*given enough eyeballs, all bugs are shallow*" law, indicates that open access to the source code together with a plentitude of users and use contexts make a lot of the possible shortcomings in the code apparent. The third benefit of the open source development model is Raymond's "*release early, release often*". Rapid release cycles provide all users a quick access to the latest improvements. Moreover, taking into account also the often omitted "*and listen to your customers*" open source makes it possible to address specific customer needs while allowing the entire community to benefit from the results.

Open source software has traditionally been visualized with an onion model [13], where the developers with most influence to the code are at the core of the onion and the users and other less code-centric roles are on the outer rims. In OSS business contexts the onion tends to take a shape where most of the development is done by the company whose business depends on the OSS product. A company core onion tends to shape according to Figure 1, which is drawn based on the case company discussed in more detail in Subsection III-A.

B. Crowdsourcing

Defined by Jeff Howe in his Wired article in 2006 [4], the concept of crowdsourcing is relatively new. While crowdsourcing can be – and has been [14]–[17] – used in many domains the concepts of microwork and micro-tasking is heavily associated with it. A micro-task is a small, repetitive task that requires little cognitive effort. One of the best-known general crowdsourcing platforms, Amazon's Mechanical Turk [18], is built on the concept of microtasking. A typical microtask can be word recognition, image processing or similar task that is easy for a human to complete. Remuneration for a single task is also typically minute. Still, there are successful examples crowdsourcing of exceedingly complex tasks as well, including cases such as the development of a mobile phone or participation in innovation processes [19], [20].

The benefits of crowdsourcing include cost-efficiency, faster development, innovation and improved quality of work [21]–[23]. While in the context of software engineering, recent study shows that these are not rendered by crowdsourcing [7] research also shows that there is promise in combining

[1]The authors acknowledge the philosophical differences between the terms Free Software and Open Source Software. In this paper, the focus is in the development model with a business model perspective. Thus Open Source Software is used throughout the paper.

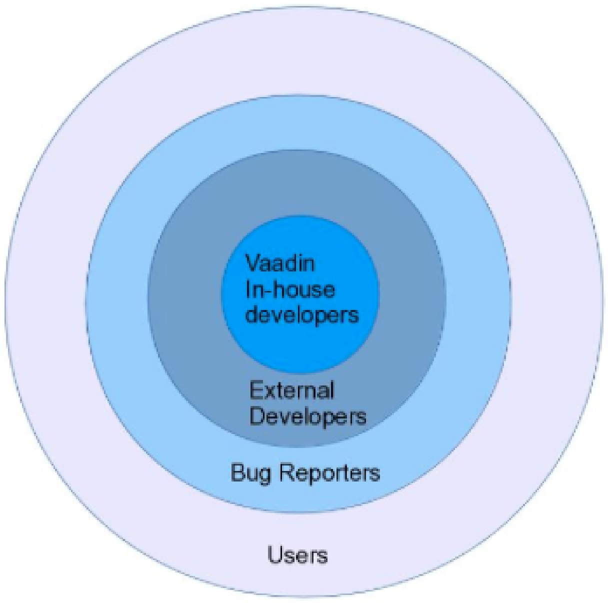


Fig. 1. Company-core Community Structure. Most of the heavy lifting is done by company employed developers and the community contributes bug reports and fixes that rise out of their use contexts – issues that can be hard to detect through internal testing.

crowdsourcing and software development methods [22], [24], [25].

Based on the above, while the concepts of Open Source Software and crowdsourcing are definitely related, the differences between them are many. These include policy regarding intellectual property rights, access to the outcome of the work, promise on rewarding contributions, and so on. However, there is no fundamental reason why crowdsourcing could not be used in connection with open source software, when the above issues are treated in a suitable fashion.

C. Motivation to Participate

The reasons people participate in online communities vary. In general, motivational aspects are divided into intrinsic and extrinsic elements. In open source development the reasons for participating are typically assumed to be largely intrinsic; developers are volunteering to development activities to scratching their own itch [12], [26], [27]. Still extrinsic elements such as remuneration exist. In crowdsourcing the motivation is often similar but more towards the extrinsic [28], [29]; participants are motivated by a rewarding system, for instance small monetary reward for an accepted contribution.

In practice, when considering real projects and companies, it is common that there is some gray area between the two extremes. For instance, on one hand a developer can be paid for participating in the activities of an open source community by a company that uses the outcome of the community's work in its business, and on the other hand individuals can for example choose to participate in crowdsourcing activities that promote their favourite games to make them more attractive for themselves. Consequently, the difference between open source development and crowdsourcing sometimes lies only in the eye of the beholder and in fundamental motivation for participation, not in the contribution itself nor the licensing model that is used to govern the contributions.

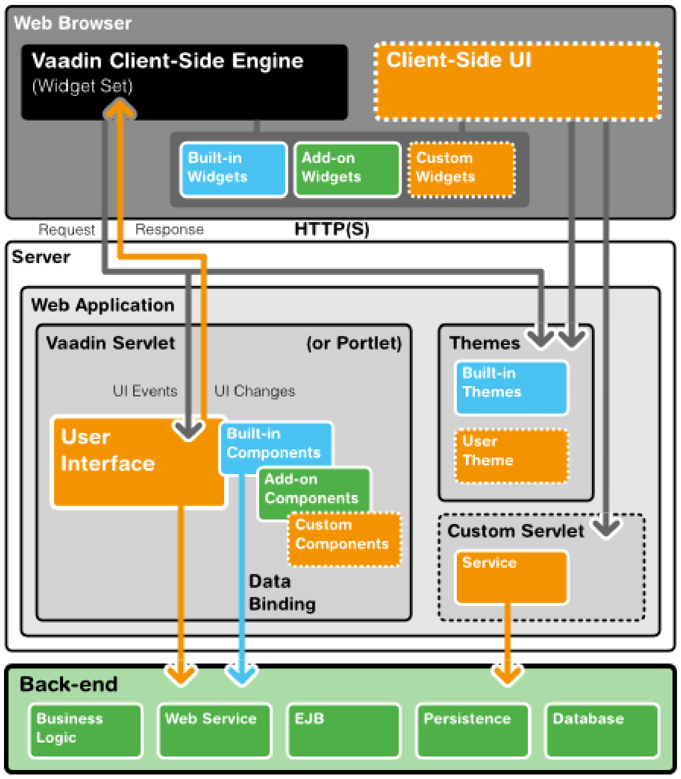


Fig. 2. Vaadin Runtime Architecture [31]

III. Research Context

We present here the results of a case study on a Finnish software intensive company to use crowdsourcing to draw contributions from external developers – an internal pilot study done in the company. The case company Vaadin ltd. (http://www.vaadin.com) has been developing their main Open Source product, the Vaadin Framework, and its predecessors since 2000. Their customer base is global ranging across various industries who build applications of their own on top the framework. Vaadin's business model is to offer a set of services from planning and implementation to sub-contracting to complement the OSS product.

A. Vaadin Framework

Vaadin, Vaadin ltd's OSS product which shares the name of the company, is a Java framework for building rich internet applications. Vaadin is licensed under a liberal Apache 2 [30] license. What sets Vaadin apart from other approaches such as Javascript libraries and browser-plugin based solutions is that it supports two programming models: server-side and client-side (see Figure 2 for the runtime architecture). The framework also comes with a large collection of user interface components on both sides.

Vaadin has an active community of over 100.000 members. What is noteworthy of the community is that Vaadin users are also developers themselves using Vaadin to produce their own products and services. In addition to the framework

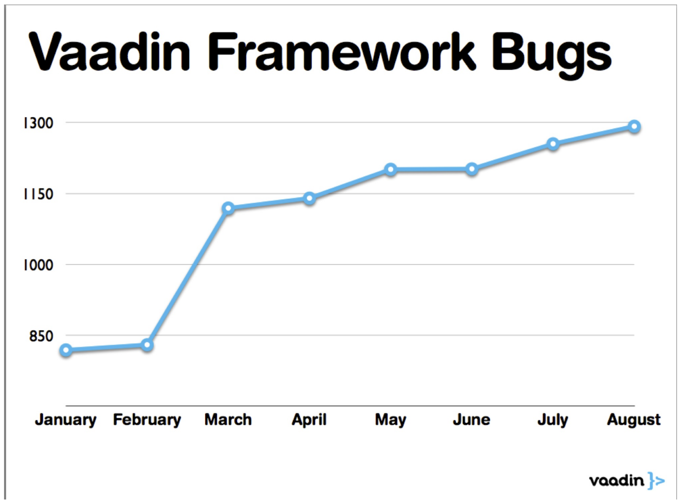
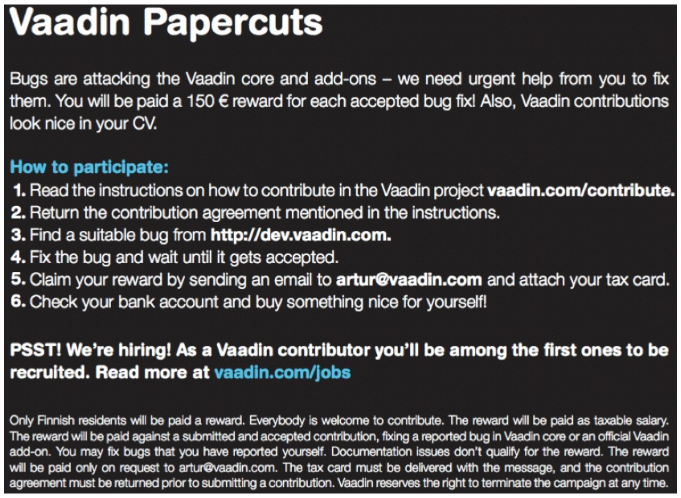


Fig. 3. Amount of bugs in Vaadin over the time period Jan-Aug 2014



Vaadin Papercuts

Bugs are attacking the Vaadin core and add-ons – we need urgent help from you to fix them. You will be paid a 150 € reward for each accepted bug fix! Also, Vaadin contributions look nice in your CV.

How to participate:

1. Read the instructions on how to contribute in the Vaadin project **vaadin.com/contribute.**
2. Return the contribution agreement mentioned in the instructions.
3. Find a suitable bug from **http://dev.vaadin.com.**
4. Fix the bug and wait until it gets accepted.
5. Claim your reward by sending an email to **artur@vaadin.com** and attach your tax card.
6. Check your bank account and buy something nice for yourself!

PSST! We're hiring! As a Vaadin contributor you'll be among the first ones to be recruited. Read more at vaadin.com/jobs

Only Finnish residents will be paid a reward. Everybody is welcome to contribute. The reward will be paid as taxable salary. The reward will be paid against a submitted and accepted contribution, fixing a reported bug in Vaadin core or an official Vaadin add-on. You may fix bugs that you have reported yourself. Documentation issues don't qualify for the reward. The reward will be paid only on request to artur@vaadin.com. The tax card must be delivered with the message, and the contribution agreement must be returned prior to submitting a contribution. Vaadin reserves the right to terminate the campaign at any time.

Fig. 4. Vaadin Open Call to Bug Fixers

community, Vaadin has an active add-on developer community in the Vaadin Directory [32] where there currently are 470 add-ons publicly available, which are mostly implemented by outside contributors to add more features to the existing core framework. While there are some actively contributing developers in the community, the development has been quite focused on in-house activities. In addition to bug reporting, a significant community contribution is typically discussion and helping other community members with their problems. Contributing code has been rather cumbersome for third party developers and thus there haven't been many coming in despite the promise of an engaged community.

An unfortunate rise in the number of bugs reported for the Vaadin Framework has been recognized. The company had four developers dedicated to working on bug fixes yet the number of bugs kept growing. This trend is illustrated in Figure 3. In addition, fixing one bug was rather costly to the company, estimated 750€ per bug. As the product is open source and as such already has an engaged community, the company started looking for ways to encourage community contribution in fixing known bugs. To do this, an idea grew to add approaches known from crowdsourcing in order to create a faster, more fluent way to get bug fixes from the community. Additionally the approach aimed at adding motivational aspects to making contributions while reducing the cost per fixed bug at the same time.

Taking stock of the situation led Vaadin further to the conclusion that the in-house processes were not supportive enough to allow the company to accept bug fixes from the outside developer community. Vaadin has an active community. Thus, there is clear potential of gaining high-quality contributions from them. Still, the company-led processes need to be efficient enough to incorporate community contributions fluently. The decision was made to focus on automated testing, integration and deployment in order to help also community contributions to get through more efficiently and faster.

Vaadin decided to experiment with spicing up the development model with an open call to contribute in order to motivate third party developers to contribute – a crowdsourcing approach was included. An open call to the community and a remunerations plan for contributions was set. Despite the remuneration for contribution the aim was that the improved processes would lead to more cost efficient bug fixing scheme overall.

B. *Experiment: Vaadin Papercuts*

To try out the crowd approach, an experiment entitled Vaadin Papercuts[2] was run in the last quarter of 2014. A call (see Figure 4) with a monetary reward per squished bug was issued. The idea was to tap into the community already working with Vaadin and motivate them with the crowdsourcing approach through an open call to focus on tasks that had significant value to the company without at the same time moving away from the OSS community-driven development model.

The contributors were to select the issues to work on from the entire bug base of the framework. In addition contributors would be paid 150 squish reward per bug fixed. Additional motivators mentioned were the opportunity to showcase development skills through the fixes and the opportunity to improve the framework where the fix would have relevance to the developers themselves.

The call in this pilot experiment was company internal with an intention to take the experiences to a full call to third party developers. However, the contributors were expected to act as any community member would and self-select the bugs to fix from their own viewpoints and make the contributions as their own activity outside working hours.

For quality assurance in addition to the automated testing, reviewing the contributions needed attention. Vaadin dedicated a team working in DevOps mode for the bug fixes. This so-called Master Team is responsible for all code integration to the code base whether internal or external as a contribution.

IV. Research Approach

The case study context is a company internal crowdsourcing experiment that investigates the suitability of a crowdsourc-

[2]The name refers to a simple UI/UX bug http://en.wikipedia.org/wiki/Paper_cut_bug. Currently the contributions are given under a new name – Bounty.

ing call together with their new continuous integration pipeline to combine community contributions to the company's main product. Our study falls under the customer perspective of crowdsourcing software engineering as proposed by Stol and Fitzgerald [33].

A. *Method*

The study conducted is a descriptive and exploratory single case study research [34]. Case study research is a suitable approach for the domain of software engineering research in general [35] as the real world issues as a whole are complex and non-deterministic and the boundaries between the studied phenomenon and its context are not clearly separable.

Here, the case study investigates the introduction of a new approach for handling code contributions to the main code branch of Vaadin. Especially the addition of crowdsourcing in conjunction to the contribution handling is at focus. The research questions we aim at answering are the following:

> **RQ1**: What were the benefits gained in combining crowdsourcing flavors to open source software development?
>
> **RQ2**: How the selected approach handles the challenges of crowdsourcing?

To gain insight on the suitability of the approach as a whole, a company internal experiment [36] was run. The case study context and the data collected from it are introduced next.

B. *Data Collection and Limitations*

Data for the study was collected through six open-ended, semi-structured interviews held at the company's premises. All interviewees were employed by Vaadin: two were interviewed in their role in the company, one represented both the Master Team and had acted as a contributor and three interviewee's were contributors to the pilot. The interviews of the company representatives covered background information on the role of OSS product and the internal processes, a description of the new approach and the aims for the pilot. The contributors were asked about their overall experience and motivation. In addition they were given the opportunity to give feedback on the crowdsourcing pilot. One researcher conducted the interviews, which were recorded. The researcher also wrote down the interviewees answers as notes. In addition to interview data, the company provided data on the bug amounts and costs. Vaadin's openly available community information and blogs were also used as supportive data.

The obvious limitation to the study is that it is a single case study and at an initial stage supported only by the internal case. While this means more validation as the process becomes more mature is needed, we believe the case presented here does provide valuable insight onto the idea and illustrates the benefits of the DevOps process as well as the promise of adding crowdsourcing flavours to company lead OSS.

All the interviewees were employed by Vaadin although some of them were interviewed in the role of a volunteer contributor. While the participants were Vaadin insiders they here had the same interest to the product that we can take any active third party developer to have. Thus their experiences with the case can be considered realistic. Still, naturally some of the motivation to participate in the first place comes from the daily work. We will get more detailed results on the difference as more outsider data comes available.

V. Results

The result of the case pilot was roughly 40 contributions in one month. The cost of the bugs fixed was 8000 € in total for the patches which lead to a drop in the cost per fixed bug by 40% to 450 euros instead of the 750 euros which was the initial bug fixing cost per bug.

A. *Master Team*

A new team responsible for the new contribution flow was established in September 2014 working in the spirit of DevOps. The tickets are available in Vaadin Trac (http://dev.vaadin.com). All patches to tickets are handled in the same way through the concept of a Master Team which enables handling outside contributions – a significant improvement to the earlier process that lagged in the ability to include third party contributions fluently. For outside developers the company uses a contributor license agreement to manage the IPR. Inside the company, the Master Team is responsible for verifying incoming tickets, reviewing the contributions and is joinable by anybody in the company. Each patch needs to be incorporated with a set of tests that verify the patch. The flow of a ticket is shown in Figure 5. With the introduction of the Master Team, Vaadin has also moved to rapid two-week release cycles.

In the review process, anybody can contribute, but only the Master Team can accept contributions. Patches can be given +2 to -2 review points where +2 is reserved for the company core. +2 indicates that provided the tests pass, the patch can be included into the master branch. The basic rule is that at least two people need to be satisfied with the patch for it to pass.

Based on the interviews with the master team the speed of the team varies on the contribution. A ticket marked "In Progress" can iterate over development and review several times. The automated tests provide the first feedback and due to the continuous process that is available within hours. Once the tests pass, the Master Team gets notification on it and can start the review process. If the tests fail the situation can be tricky for a third party developer as there is no support process in place for that. The time span for one patch flowing through the process is typically days but due to a limited amount of eyeballs on the code, can extend to weeks.

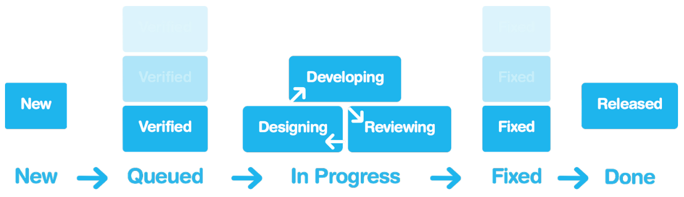


Fig. 5. The flow of a Ticket in the new bug fixing process [37]

The price of the bug was set to an lump sum regardless of the ticket as the work needed for estimating the tickets would require unnecessary effort. The amount per bug may seem smallish. Despite this, the interviewee working daily with the contributions reported there are a lot of tickets where the amount of work is in par with the offered compensation. The remuneration offered can be seen as a motivating compensation for such tickets. The amount was also seen as motivating to keep contributing after getting the first fix through.

B. Contribution Experiences

One out of the four interviewed contributors had completed their first bug fix, one 7-8, and the others reported to have contributed 20-30 small fixes. The bugs to fix were chosen based on the required effort and familiarity. If the developer had met the bug in their daily work or the buggy component was well-known to them, they were inclined to fix the bug as well. The contributors also looked for ”easy-fix” bugs, i.e. tickets that were easy to understand and not too complex or effort heavy. If there was a lot coding, refactoring or the workload turned out to be more significant than initially estimated, the developers reported to have dropped the fix. A typical timely effort per bug was a couple of hours of work.

Reasons to participate were both internal and external in motivation. Two mentioned that they are coding any way on their free time and the opportunity to get a bit of money out of the effort helped to gear participation. Not getting enough coding into the workday was also mentioned as a motivator. Remuneration was also acknowledged as an initial motivator that then resulted in much more fixed bugs as the work turned out to be fun and fruitful. The possibility to be included as one of the core contributors to the product ranked with the contributor who had completed the first patch. When asked directly about the role of the remuneration, all interviewed developers stated that it did motivate to contribute. The monetary reward was also an added motivator that pushed earlier thoughts to contribute into action. The reward was considered sufficient to the effort. Without the reward three of the contributors doubted they would have participated on their own, one saw it plausible if a nice workshop or similar were to take place. One saw it unlikely yet plausible. The reward was seen as important but not necessarily required to be monetary by one.

The biggest obstacle seen was the review process. It was also seen as an integral point in the process. There were several iterations of rework to the patches that each could raise different issues in the contributed patch. Getting back to the code after a while was hard. The first contribution was seen as the most laborious, going through the entire process once made the next patches more fluent to get through. One developer had received feedback faster than others maybe due to the nature of the patches they had contributed.

VI. Discussion

Based on the experiences gained so far, the approach shows promise. There was a rise in bug fixing activity as developers already involved with the project got engaged with fixing bugs. More bugs fixed improves the overall quality of the product and the two-week release cycle makes the fixes available to all fast.

The overall cost of the remuneration is seen reasonable and the price per bug is actually decreased as several small bugs got fixed through the approach. The size of the remuneration gears the contributions to tickets that can be fixed with reasonable effort in regard to the compensation. It also seems that the money acted as a motivator to start contributing and once started the contributors felt they wanted to keep contributing in the future.

Typical concerns in crowdsourcing software development include quality assurance, knowledge and intellectual property, and motivation and remuneration. While the initiated approach seems to tackle these, there are still points to address and develop further. The biggest bottleneck is still the human eyes. Quality assurance requires people to look through the contributions even though testing is an automated process. The lags in reviews are due to insufficient number of Master Team’s hours dedicated for reviewing contributions. This lag can be frustrating especially to an outside developer as revisiting code after a while can be challenging and is reported by the participants as demotivating. The long review period can also create the need for further iterations as the main branch evolves as well.

Research on motivation to participate in OSS shows that remuneration is not rating that high with the community. However, learning more and the possibility to show expertise do act as motivators and that is also something incorporated into the call for contributions. Considering the fact, the software development requires specialized skills, the need for remuneration goes beyond microwork. This is also supported in [7]. Here, monetary reward played a key role in both the type of contribution made and in getting people engaged. Consequently, instead of plain monetary reward, it would be interesting to conduct another study where remuneration mechanisms geared towards community appreciation would be used as the motivating factor. In particular, this would include the present developer community as a whole.

In our case study, the crowd consisted of Vaadin employees. Consequently, although a goal was to make the participant act as a real crowd, there are numerous aspects that are overlooked or at very least play a smaller role than in a more open experiment. Company internals have a lot of prior knowledge on the product and the development processes. They can get started with development more easily and evaluate the bugs differently. There is also greater affinity to helping the product. Having prior knwoledge came up in the interviews as an addition support that helped the developers keep going. They also had knowledge of coding conventions and quality requirements. In particular, when allowing contributions in global scale, there will be a need for added special screening for e.g. intellectual property rights related issues, taking into account factors such as cultural differences, and even legal aspects such as the procedure for establishing a monetary transaction for paying a reward, including for instance taxation. Therefore, scaling the approach of this particular study to truly open, community-supported development work still requires some considerations. For the company, getting patches is valued. The size of the bug fixed is considered irrelevant.

Since all the contributions were treated equally, there are numerous opportunities for cherry-picking. The most obvious way to perform this is to seek for easy and simple bug fixes and to focus on those. This was clearly evident in the case data. The developers chose bugs that they estimated worth the monetary compensation in effort. However, while one can label such activity harmful, similar options are available in many other settings as well. Consequently, we do not consider this a major problem, in particular as monetary reward was small. This also reflects the underlying view of the company – treating all contributions similarly means that no estimates are needed. Estimating the effort needed for implementing different features would be a tedious task, and in any case developers could identify designs where the effort would deviate from the estimates. From a purely quality perspective, getting any bug fixed is valuable to the overall quality of the product.

VII. Conclusions

Different ways of leveraging activities carried outside companies play an important role in various software companies. Practices such as open source software and crowdsourcing have gained wide interest, as they promise rapid advances in development capabilities with small monetary investment. In this paper, we have been examining how an open source software company could introduce crowdsourcing in its processes. As a concrete outcome, we addressed the fashion contributions have been rewarded and how the necessary technical infrastructure within the company was improved to support the increasing number of contributions. Previously, the lack of such infrastructure has introduced a major obstacle for accepting new contributions, whereas now, the ability to scale practices related to contributions makes their acceptance feasible.

The next steps are to go live with the approach with the full community, not with a selected set of friendly and trusted developers only. The plan to go live is the following. First, the company issues a call for contributions to university students in a number of partner universities. Based on lessons learned from this stage and the domestic pilots process, improvements can be made in the contribution process. The next step is to go live domestically in Finland, and finally take the approach to the global scale.

Acknowledgment

The authors wish to thank all the interviewees for their input, time and effort.

This work is a part of the Digile Need for Speed project (http://www.n4s.fi/en), which is partly funded by the Finnish Funding Agency for Innovation Tekes (http://www.tekes.fi/en/tekes).

References

[1] R. Goldman and R. P. Gabriel, *Innovation happens elsewhere: open source as business strategy*. Morgan Kaufmann, 2005.

[2] D. G. Messerschmitt and C. Szyperski, "Software ecosystem: understanding an indispensable technology and industry," *MIT Press Books*, vol. 1, 2005.

[3] D. C. Brabham, *Crowdsourcing*. MIT Press Essential Knowledge, 2013.

[4] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[5] ——. (2015, Feb.) Crowdsourcing: A Definition. [Online]. Available: http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html

[6] Y. Benkler, "Coase's penguin, or, linux and" the nature of the firm"," *Yale Law Journal*, pp. 369–446, 2002.

[7] K.-J. Stol and B. Fitzgerald, "Two's company, three's a crowd: a case study of crowdsourcing software development," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 187–198.

[8] D. Ståhl and J. Bosch, "Modeling continuous integration practice differences in industry software development," *Journal of Systems and Software*, vol. 87, pp. 48–59, 2014.

[9] M. Fowler, "Continuous integration," http://martinfowler.com/bliki/ContinuousDelivery.html, retrieved: November 2014.

[10] P. Debois, "Devops: A software revolution in the making," *The Journal of Information Technology Management*, vol. 24, no. 8, pp. 3–5, 2001.

[11] M. Aberdour, "Achieving quality in open-source software," *Software, IEEE*, vol. 24, no. 1, pp. 58–64, 2007.

[12] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.

[13] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *Proceedings of the international workshop on Principles of software evolution*. ACM, 2002, pp. 76–85.

[14] (2015, Feb.) Threadless art community. [Online]. Available: https://www.threadless.com

[15] (2015, Feb.) Open street map. [Online]. Available: http://www.openstreetmap.org

[16] (2015, Feb.) iStockphoto. [Online]. Available: http://www.istockphoto.com

[17] (2015, Feb.) The Noun Project. [Online]. Available: http://thenounproject.com

[18] (2015, Feb.) Amazon mechanical turk. [Online]. Available: https://www.mturk.com/mturk/welcome

[19] (2015, Feb.) Jolla Community. [Online]. Available: https://together.jolla.com/

[20] (2015, Feb.) Innocentive. [Online]. Available: http://www.innocentive.com

[21] T. D. LaToza, W. B. Towne, A. Van Der Hoek, and J. D. Herbsleb, "Crowd development," in *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*. IEEE, 2013, pp. 85–88.

[22] K. Lakhani, D. A. Garvin, and E. Lonstein, "Topcoder (a): Developing software through crowdsourcing," *Harvard Business School General Management Unit Case*, no. 610-032, 2010.

[23] E. Schenk and C. Guittard, "Crowdsourcing: What can be outsourced to the crowd, and why," in *Workshop on Open Source Innovation, Strasbourg, France*, 2009.

[24] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. van der Hoek, "Microtask programming: Building software with a crowd," in *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 2014, pp. 43–54.

[25] F. Pastore, L. Mariani, and G. Fraser, "Crowdoracles: can the crowd solve the oracle problem?" in *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 342–351.

[26] K. Lakhani and R. G. Wolf, "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," 2003.

[27] J. Bitzer, W. Schrettl, and P. J. Schröder, "Intrinsic motivation in open source software development," *Journal of Comparative Economics*, vol. 35, no. 1, pp. 160–169, 2007.

[28] M. J. Antikainen and H. K. Vaataja, "Rewarding in open innovation communities–how to motivate members," *International Journal of Entrepreneurship and Innovation Management*, vol. 11, no. 4, pp. 440–456, 2010.

[29] N. Kaufmann, T. Schulze, and D. Veit, "More than fun and money. worker motivation in crowdsourcing-a study on mechanical turk." in *AMCIS*, vol. 11, 2011, pp. 1–11.

[30] (2015, Feb.) Apache 2.0 licence. [Online]. Available: http://www.apache.org/licenses/LICENSE-2.0

[31] M. Grönroos. (2015, Feb.) The Book of Vaadin. [Online]. Available: https://vaadin.com/book

[32] (2015, Feb.) Vaadin Directory. [Online]. Available: https://vaadin.com/directory

[33] K.-J. Stol and B. Fitzgerald, "Researching crowdsourcing software development: perspectives and concerns," in *Proceedings of the 1st International Workshop on CrowdSourcing in Software Engineering*. ACM, 2014, pp. 7–10.

[34] R. K. Yin, *Case study research: Design and methods*. Sage publications, 2014.

[35] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[36] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *Software Engineering, IEEE Transactions on*, no. 7, pp. 733–743, 1986.

[37] (2015, Feb.) Fixing bugs in Vaadin. [Online]. Available: https://vaadin.com/blog/-/blogs/fixing-bugs-at-vaadin