Why Does Code Review Work for Open Source Software Communities?

Adam Alami
IT University of Copenhagen
Denmark

Marisa Leavitt Cohn
IT University of Copenhagen
Denmark

Andrzej Wasowski
IT University of Copenhagen
Denmark

***Abstract*—Open source software communities have demonstrated that they can produce high quality results. The overall success of peer code review, commonly used in open source projects, has likely contributed strongly to this success. Code review is an emotionally loaded practice, with public exposure of reputation and ample opportunities for conflict. We set off to ask why code review works for open source communities, despite this inherent challenge. We interviewed 21 open source contributors from four communities and participated in meetings of ROS community devoted to implementation of the code review process.**

It appears that the hacker ethic is a key reason behind the success of code review in FOSS communities. It is built around the ethic of passion and the ethic of caring. Furthermore, we observed that tasks of code review are performed with strong intrinsic motivation, supported by many non-material extrinsic motivation mechanisms, such as desire to learn, to grow reputation, or to improve one's positioning on the job market.

In the paper, we describe the study design, analyze the collected data and formulate 20 proposals for how what we know about hacker ethics and human and social aspects of code review, could be exploited to improve the effectiveness of the practice in software projects.

***Index Terms*—Open Source, Code Review, Motivation**

I. Introduction

Code review is an established software engineering practice, that ensures good quality of source code, lowers bug frequency, and enforces coding standards [1]. During code review, reviewers (software engineers other than the author) read code in order to point out mistakes, shortcomings, and convention violations that had been overlooked during programming. The practice has evolved over the years from simple inspections of sections of code to formalized techniques that give immediate feedback. Reviews are performed in various forms, such as pair programming, informal walkthroughs, and mandatory approvals before code merging. A variety of tools have been developed to help reviewers scrutinize and check the viability and functionality of code during these reviews, and to formulate feedback.

Code review is particularly successful and cherished in free and open source software (FOSS) communities [1]–[4]. Some would go as far as to say that code review is the *raison d'être* of FOSS: *"code review . . . is the reason behind open source. If anyone could contribute to a project, there could be chaos."* It is a *"wall that separates bad code from good code."*[1] When asked about relative importance of code review and testing, code review is often ranked clearly above testing by FOSS engineers.

[1]Statements by FOSS engineers interviewed in this study.

The effectiveness of code review depends on the level of participation, the size of the changes made, and the reviewer experience and expertise. Thus it is not an entirely obvious practice to implement. Now, that code review is also widely used in the industry [4], [5], it is particularly relevant to understand why and how it works. As, the standard of peer reviewing in the open source environment remains a beacon of best practice, we turn our attention to FOSS projects for insight. We investigate the practice from the perspective of the main participants, their motives and behaviors. We ask:

RQ: *Why does code review work for FOSS communities?*

We want to understand how contributors deal with the inherent negative feedback; what motivates them; and what values lead them to first contribute code of high quality, then produce high quality feedback, and finally to diligently consider the feedback to improve the contributions. We ask this question to (i) learn from FOSS communities to translate the experience to closed-source environment, (ii) to help other projects in implementing the practice successfully. We formulate observations based on data and then speculate how project and community managers can incorporate these results into their work culture. We find that:

- FOSS contributors experience *rejection and negative feedback* regularly. Communities do not eliminate this negative experience, as this seems to be the core improvement mechanism of code review.
- Our subjects develop *mature attitude to negative feedback*, taking it as an opportunity to learn, to improve, and ultimately excel in their job.
- The *ethic of passion* drives the contributors. They are passionate about all aspects of the project, including the reviews. The passion allows them to invest themselves into code review, and it also makes them resilient to negative interactions.
- Community members develop a working *ethic of care*, showing commitment and care toward the project, the community, and other developers. The code review is a gate keeping practice, an implementation of the care for quality of the project. Thus care is a strong motivator for both for performing the code review, and for diligent execution.
- *Intrinsic motivation:* Altruism and enjoyment are key intrinsic motivators of FOSS code reviewers. Even

paid-for code reviewers are effectively volunteers, who choose tasks following intrinsic interests.

- FOSS communities have developed a *gift economy*, centered around a range of non-monetary non-material *extrinsic motivators*: reciprocity of contributions, sharing success, displaying status and reputation, transfer of FOSS reputation to professional career, continuous learning and development, all the way to punishment for under-performance. These economy "pays" for code review effort.

In the final part of the paper, we reinterpret these observations to extract ideas for improving the code review process in existing projects and communities. We hypothesize that practices described by our subjects (for instance, communicating guidelines, or establishing mentorship) are the reasons behinds their thriving during code review. We thus formulate them as imperative suggestions, that, we hope, can be used for further investigation in action research projects.

We start by characterizing the studied communities in this paper, using them also to cast some light on the practice of FOSS code review (Section II). Section III describes the study design. Section IV is devoted to the analysis and interpretation of the data. We aggregate the actionable hypotheses in Section V, discuss related work in Section VI, and conclude in Section VII.

II. Subject Communities

We begin by characterizing the five studied FOSS communities and how do they use code review. Throughout the paper we use the terms developer, programmer, software engineer, and hacker interchangeably to refer to FOSS contributors.

Robot Operating System (ROS) is a popular open-source middleware for robotics. It provides standard communication and coordination features, and bundles implementations of essential robotics-specific functionality with software drivers for popular hardware components. A ROS-based application is created by complementing selected ROS components with application specific code, typically in Python or in C++. The project originated in 2007 at the Stanford Artificial Intelligence Laboratory. In 2008, it transferred to Willow Garage, a robotics start-up. Since 2013, it has been stewarded by the Open Robotics foundation.

Since its inception, the community enforced reviews for every pull request, even for changes from the core team. Common code inspection meetings were also organized. Over time the review practices have deteriorated and are abandoned due to lack of resources. This development partly motivates our study—we work with the community to re-energize the practice through an action research style intervention. As a result the community is initiating a pilot project using standard GitHub tools for code review and a policy that encourages code contributors submitting pull requests to review their peers' contributions.

Apache Allura is a hosting platform. Allura integrates key development tools (version control repositories, issue tracker, wikis, blogs, etc.) for developers. It is an open source project, under the umbrella of Apache foundation, implemented in Python. Since 2012 it is used by SourceForge as the main platform.

The official community code guidelines[2] require that each contribution is tested in a local fork prior to submission. Each pull request must be accompanied with the necessary test cases to be added to the automated test suite of the project. As a precondition to the submission, the contribution must pass all existing tests. Contributions with "open" status are reviewed and discussed in the Git Merge Request discussion forum or in the issue tracker. Violations of the community code review are highlighted, architectural and programming decisions are debated. Good contributions are praised. A contribution could be rejected for minor code guidelines violations. Any member of the community can review code; however, ultimately it is the decision of the maintainer whether to merge a contribution or not.

The Comprehensive Knowledge Archive Network (CKAN) is an open source project developing a web-based storage and distribution platform for data, mostly used by public institutions joining the open data movement. CKAN is implemented primarily in Python and JavaScript. The project is overseen by the Open Knowledge International association.

The CKAN community uses GitHub for pull requests management and code review. Each contribution must be accompanied with the relevant tests and updated documentation. Once a pull request is submitted, it is subject to review by one to four other community members. The community specifies coding standards for all used technologies (i.e. Python, CSS, etc.). Architectural and programming decisions are debated and minor code guideline violations are pointed out. The official review guideline recognizes that besides standards, reviewer's judgment is a key factor in accepting or rejecting contributions. [3]

FOSSASIA is a community centered in Southeastern Asia developing software for social change since 2009. FOSSASIA projects are not limited to software applications, but include also hardware and design.

FOSSASIA focuses "*on the code quality more than on managing pull request ethics*,"[4] emphasizing the actual goal of the code review work. Its best practices are documented in programing and commit style guidelines. Pull requests are reviewed using standard GitHub facilities. Up to eight reviewers participate in a single discussion. Acceptance requires reaching a consensus between reviewers. It is the responsibility of the core team to review code. Reviewers in this community can be pedantic, even grammatical errors in the code comments are pointed out. Still, FOSSASIA manages to maintain a friendly atmosphere. Reviewers offer help to fix errors if required.

The Linux Kernel is the most popular and versatile operating system kernel on the planet, used on super computers and webservers, powering up cloud infrastructure, and controlling lots of mobile and embedded devices (including all Android devices).

[2]https://forge-allura.apache.org/p/allura/wiki/Contributing%20Code/
[3]http://docs.ckan.org/en/2.8/contributing/reviewing.html
[4]https://github.com/fossasia/susi_server

Since its inception in 1991, the project is a success story, especially in terms of developing a sustainable community.

Unlike other communities, Linux is not using GitHub for code review, but communicates changes and performs the review using mailing lists. The code style guideline and pull request submission documentation are thoroughly detailed. Even the size of the email is specified. Contributors are encouraged to prepare patches that are concise and logically atomic. Contributors submit their patches to the relevant subsystem mailing list, where they are reviewed by volunteers. Criticism and other comments are exchanged very openly in the mailing list. Code is pedantically reviewed; there are known cases when a patch went through 20 review iterations. Acceptance of a patch is subject to a community consensus, but the ultimate decision resides with the maintainer.

III. Method

Recall that we are asking *Why does code review work for FOSS communities?* To answer this question we want to dig deeply into the mindset of participants in this inherently human process that relies heavily on communication skills; that involves feedback, critique and rejection on daily basis; and that exposes power and decision hierarchies in communities. For this reason, we choose a qualitative research method that is suitable for exposing participants' experiences and motivations. We have conducted 21 semi-structured interviews with members of the communities that are using code review (Apache Allura, CKAN, FOSSASIA, Linux). To this we add observation of three meetings of the ROS community (10–16 participants) devoted to implementing a new code review process. While the interviews explain mostly the experience with the existing process, the meetings are more likely to review the expectations, wishes, and concerns with the process, as during the meetings people reflect about consequences of instituting it.

A. Interviews

While a structured interview has a rigorous set of questions, which does not allow to divert, a semi-structured interview is open, allowing new ideas to be brought on top of a predefined question framework. The flexibility is used to enhance the depth of interviewee's statements. This makes semi-structured interview suitable for capturing rich qualitative data and obtaining deep insights into people's believes and behaviors.

The questions in our interview framework fall into three categories: introductory, core, and probing (Tbl. I). The introductory questions were designed to warm up the conversation. The core question related directly to our research question. The probing questions aimed at making the conversation detailed and concrete. We also asked other questions (not in the table) about quality assurance practices in FOSS. These questions sometimes revealed the relation of code review to other practices, as shown in the statements reported in Sect. I.

TABLE I
KEY PARTS OF THE INTERVIEW FRAMEWORK

intro	Can you talk to me about your community?
	What first motivated you to participate in this community?
core	Can you describe the code review process in your community?
probing	What makes you adhere to best practices?
	How do you cope with the feedback?
	What makes you want to participate in code review?
	Can you share with me an example of good feedback you received, a part of the code review, and how did you feel about it?
	Can you share with me an example of negative feedback you received, part of the code review, and how did you feel about it?

B. ROS Community Meetings on Code Reviews

The interviews were executed with members of the communities that successfully use code review. In contrast, the ROS community has failed to sustain the practice in the past, and now, having grown substantially, it attempts to reboot it. As part of our action research involvement with the ROS community we facilitated community meetings, where QA practices are being discussed and implemented. Re-instituting the code review was one of the highest prioritized activities of this group (within the top three of the sixteen prioritized actions).

Three one hour long community meetings were dedicated to the implementation and the logistics required to implement a code review process in the community. The meetings had an open structure. Ideas to implement, and details of implementations were proposed and voted by participants (and not by the facilitators, the authors). It is important that the meeting group takes the execution of decisions on themselves, so they have to bear in mind the cost of the implementation. In the meetings, we also observe reflections on the prior attempt to implement code review in the community, and reflections on this. The meeting data complements the interview data in following ways: we collect opinions of much more contributors quickly, we can see how differing opinions are confronted, we see how ROS contributors discuss the failure of code review, and how they anticipate the practice in a community that is not yet converted to use it.

C. Subject Selection

We have selected the five FOSS communities (Sect.II), to achieve a deep understanding of the phenomena under study. We interviewed 21 participants from Allura, CKAN, FOSSASIA, and Linux communities. We searched for contributors on LinkedIn, using community name and terms "contributor"/"developer". We contacted random entries from the search results and used snowballing to increase the sample. We had no prior relationships to any of the subjects. Table II summarizes the demographics of the population. The role labels are self-selected by participants (on LinkedIn profiles). The majority of the participants contribute to FOSS as part of their professional employment and are paid for their contributions. Two of the participants were students. Twenty participants were males and one was a female.

The participants for ROS community meetings on quality assurance have been recruited using two methods: directly inviting community members to participate (eight individuals) and via an announcement in the community public forum (15 individuals). The group counts predominately developers, but also some directors, project managers and CTOs of companies using ROS are amongst the group members. Not all 23 participants joined all three meetings; attendance was in average 16 participants.

D. Data Collection

Due to geographical distribution of subjects, all interviews were conducted remotely, using Google Hangouts. Each interview lasted 40-60min and generated on average 14 pages of verbatim. The ROS QA group meetings used an electronic meeting platform, GoToMeeting. We transcribed the recorded audio, generating 16 pages of verbatim per meeting on average.

E. Analysis

We analyzed the material from interviews and meetings following the guidelines of Robson and McCartan [9] and of Miles and coauthors [10]. The analysis was iterative, started in early stages of data collection, and continued throughout the study. First, the open coding enabled us to retrieve and compare the text that has been linked to a particular theme. We devised the codes by examining the data line-by-line using the following questions as a lens: What is this saying? What does it represent? What is happening in here? What is she trying to convey? What is the process being described?

Then we searched for patterns in statements and ideas, formulating themes. A theme is a concept, an implied topic that organizes a group of repeating ideas that help to answer the study question [9]. We used *analytical memos* to formulate and work with the themes: the first author compiled a number of memos based on the coding that summarized and aggregated observations. These where used as discussion material between the authors. Table III captures the identified themes, examples of verbatim, and argues why a particular theme was selected.

TABLE II
DESCRIPTION OF THE STUDY POPULATION

Participant	Community	Role	Experience [Y]	Country
1	Allura	student	2	India
2	Allura	software developer	12	USA
3	Allura	senior software developer	14	USA
4	CKAN	software developer	10	Slovenia
5	CKAN	software engineer	12	UK
6	CKAN	student	2	Slovenia
7	CKAN	senior software engineer	8	UK
8	FOSSASIA	software developer	2	India
9	FOSSASIA	software engineer	8	India
10	FOSSASIA	software engineer	10	India
11	FOSSASIA	software developer	8	India
12	FOSSASIA	software developer	7	India
13	FOSSASIA	software engineer	13	India
14	Kernel	Linux kernel engineer	18	Denmark
15	Kernel	Linux kernel hacker	10	Denmark
16	Kernel	principal engineer	23	Brazil
17	Kernel	embedded Linux engineer	5	Spain
18	Kernel	embedded Linux engineer	7	USA
19	Kernel	Linux kernel engineer	10	USA
20	Kernel	Linux kernel engineer	12	USA
21	Kernel	senior project manager	30	USA

IV. Findings

Code review is an emotionally loaded practice, with lots of exposure of reputation and ample opportunities for conflict. In the following we present our findings, which explain how the successful communities deal with these issues. In a nutshell, we stipulate that the underlying reason behind code review success in FOSS is hacker ethics. Himanen [11] argues that the hacker ethics are more about moral virtues, in contrast to the protestant work ethic, which stresses diligent hard effort. Hacker's values, according to Himanen include but are not limited to passion, caring, creativity and joy in creating software.

A. Rejections & Negative Feedback

A publicly communicated rejection of a contribution is a common experience in FOSS code review, across all studied communities. The reports from the Linux kernel community are most pronounced, where some even speak of *"rejection by default,"* assuming the rejection as the initial position (Participant 16). An examination of the Linux Kernel mailing list archives shows that the language used can be intimidating.[5] The Linux kernel community uses frequent rejections and the harsh language deliberately as a *"congestion control"* mechanism (Participant 13) that limits the overflow of contributions.

How do contributors handle rejections and negative feedback in code review? Given the vulnerability of contributors in the code review process, one would expect that FOSS communities would be decaying. Yet, the communities, we studied, are flourishing. It appears that ability to deal with rejection is *sine qua non* for succeeding in open source: *"for someone to succeed he needs to be able to handle rejections, rudeness, and jarring-to-the-senses language"* (Participant 14). In the remainder of this paper we investigate the mechanisms that we observed at work, that, among others, minimize the negative effects of receiving critical feedback: learning from rejection, the ethics of passion, the ethics of care, and reputation.

Implications. Crucially, none of the subject communities attempt to eliminate rejection and negative feedback from their development process. This is clearly not a route to deal with the issues of code review. Anybody implementing a code review process should institute an environment where rejections are common, accepted, and normal. Mentoring and training should be considered to support newcomers to the practice in learning how to handle rejections.

Observation 1. *Contributors are subject to frequent rejections in code review. Communities neither reduce nor eliminate the negative feedback, as they believe it is core to the practice.*

[5] https://lkml.org/lkml/2018/8/3/621

TABLE III
THEMES: EXAMPLES, DEFINITIONS, AND WHY THEY WERE CHOSEN

Theme	Definition	The theme in our data	Example verbatim
Rejection	An action taken by someone of not accepting, trusting, or considering a contribution of another community member. In the context of code review, this refers plainly to the refusal to include the contributed code in the main project.	Rejection is an inherent and dominating characteristic of the code review process, thus it appears in our data naturally. Rejections were discussed frequently, both directly and indirectly.	*I was completely depressed. It was some feedback I got on Friday. I have a patch which has gone through 5 revisions … I had these 5 revisions and this has not happened! … I got these two guys which they are anti-social. They picked a piece of my code and say why do you do this? This is crap, it will hurt performance … They never say something nice.* Participant 14
Iterative improvement	In code review, a cycle of repeated review, rejection, and improvement (in response to criticism) of the same contribution.	Iterative improvement (our name) was brought up by several participants, both in negative and positive sense. Some participant perceive it as ”depressing“ other see it as an opportunity to learn and grow.	*Only a certain type of people can handle this. It's not very healthy on the mental state. You have to be able to handle this and very persistent. The last patch I got in has gone through 7 revisions before it got in.* Participant 14.
Passion	A strong inclination toward a significant activity in one's life. Passion is often self-defining, pertinent to one's identity. It is a necessary component in reaching the highest level of achievement, and contributes to creativity. It affects autonomy, competence, and relatedness [6].	Passion occurs directly numerous times in data. Subjects also talk about their community and work passionately. They speak with certainty, in higher pitched and faster voice that demands attention, with positive and assertive body language.	*… down the line, you always get to be attached to the project and get the passion of contributing and getting it out to the world so yes, it is one of the reasons why people contribute.* Participant 8
Caring	A relationship where the “caring” person acts in response to a perceived need from the “cared-for.” A caring relationship is a basic human instinct, a universal virtue. The caring party engages in helping the cared-for [7].	Caring repeatedly appears as a core value in the interviews and discussions, with symptomatic phrases like *“I care,” “we care”*. We see both care for abstract entities (the project, community, quality) and for community members.	*In FOSSASIA we care. It's not like an average job. We do it because we want to do it and we care about the quality.* Participant 10
Intrinsic motivation	An internal desire to perform an activity. Self-applied. Arises from a direct relationship between the individual and the circumstances. The reward is intangible—a sense of achievement or satisfaction. Intrinsic behavior springs from the human need for competence and self-determination, directly derived from the emotions of interest and enjoyment [8].	While talking about their tasks, participants repeatedly used phrases like ”makes me happy“ or ”feels good/nice.“	*It feels nice doing something for the community. There is satisfaction, especially when the PR is merged. It feels nice!* Participant 10
Extrinsic motivation	Inspiration to act to gain some external reward [8], valued by goal-oriented individuals. An extrinsic reward is tangible or physically given to award one's participation. Extrinsic rewards are easier to exploit in project management than intrinsic ones.	This theme emerged as participants talked repeatedly about the significance of reputation to themselves or to their peers.	*It's a great piece of software and a success story. Everybody wants to be part of it. Not only that, having a reputation in the community also counts.* Participant 16

B. Iterative Improvement

Code review in the studied communities is iterative. Typically, after a rejection, or in response to negative feedback, the contributors implement necessary improvements, and ask for another review. A pull request may go through 20 iterations of review in the Linux community (Participant 15). In FOSSASIA, the range is 1–4 iterations (Participant 12). This amount of iterative scrutiny may appear intimidating at first. One could be tempted to conclude that the comfort of the contributors is sacrificed in the name of quality. However this is not the view shown by our subjects: Iterative improvement is a mechanism to turn the negative feedback into a positive experience—they can advance their technical excellence in the process.

How learning affects code review? Ghosh and coauthors observe that knowledge is a salient motive for participation in FOSS [12], [13]. Lakhani and Wolf report that 45% of the survey's participants join a FOSS community to improve their skills [14]. FOSS creates a positive environment for learning [13]. Our subjects concur. They use terms such as *“opportunity to learn”* (Participant 15) and *“self-growth”* (Participant 21) when referring to processing feedback. The rejection and negative feedback are rationalized from an issue to a reason for individuals to join the process.

Observation 2: *The iterative improvement cycle in code review turns negative feedback into a positive opportunity for learning and technical-growth by contributors. Receiving feedback may even become a reason for participation.*

Implications. Reacting to feedback in mature ways can be learned. Organizations and FOSS communities should consider using, for instance, coaches and mentors to help the participants in code review to develop a constructive attitude to feedback.

C. Ethic of Passion

Passion is a strong inclination or desire toward an activity that one likes or even loves, that one finds important, and one invests time and energy in [15]. Passion is a necessary component in reaching the highest level of achievement, and contributes to creativity [15]. FOSS contributors review code that does not concern them in any way out of their passion for programming, passion for the community's project, and passion for excellence. *"In open source or at least in the Linux Kernel community it's not a job like in a company, even though most people are getting paid now. People have passion about this. I think the most important thing is how they perceive the criticism when they are passionate about the work. I think we don't see it as criticism"* (Participant 20). Participants see passion also as a help in coping with negative feedback. They speak with passion and a sense of purpose.

How does passion shape the execution of code review? Vallerand et al. propose a dualistic model of passion, distinguishing obsessive and harmonious passion [15]. *Obsessive passion* is tied to a person's self-esteem, ego. The person's identity is defined by the passion, thus she is compelled to engage in the activity. Such person engages in an activity rigidly. Obsessive passion generates strong negative effect when the person is unable to be involved in the activity. Our data does not show any strong links between the activity of code review and the participants' identity, and we do not observe any negative effect regarding inability to perform code reviews. Hence, we conclude that passion of the studied subjects is of the *harmonious* type. Harmonious passion results from an autonomous internalization of acceptance of the activity [15]. The desire to participate in the activity is significant, but not overpowering. Harmonious passion generates greater positive effect than obsessive passion. Individuals subject to harmonious passion demonstrate better concentration, better flow, and flexible persistence.

According to Vallerand's model, passion should positively affect the quality of the performed code reviews, and our subjects concur, for instance: *"passion is the force behind the quality of the work and people contributions"* (Participant 11). Interestingly, Bonneville-Roussy et al. [16] analyze the types of goals that people subject to harmonious and obsessive passion set. They note that only harmonious passion generates so called mastery goals, which is consistent with our subjects being very passionate about technical excellence, and the participation in code reviews to achieve the excellence (see Observation 2).

Observation 3: *The ethic of passion motivates FOSS contributors. Consequently, they dedicate effort to code review, deliver high quality, and are more resilient to rejection.*

Implications. It is definitely difficult to operationalize passion. Passion is innate, but it can be developed and nurtured like any other value. It should be nurtured in software engineering environments. If code review is important for an organization, passion for the practice should clearly be key for selecting the project members (as opposed to training skeptics to perform code review). Since passion is contagious, organizations and projects should strive to recruit passionate managers and team members, and encourage them not to be shy about their passion.

D. Ethic of Caring

Toombs et al. [17] argue that hacker communities demonstrate a nonliberal ethos, prizing self-determination, technological expertise, independence, freedom from government, and suspicion of authorities. However, for these communities to function, care values are also important, those values of collaboration, cooperation, and support of one another. For FOSS contributors, work is about the power of human reunion, about working together and caring for each other, for the community, and for the project. *"People in the community care about the work. They care about the community, about the product and its quality. Everybody cares. This caring together makes a difference. You don't feel [like if you were] working for a company."* (Participant 11) Caring, as articulated by the subjects, is the feeling and the display of concern and attaching importance to the community work and its products.

How does care shape the execution of code review? Subjects believe in a positive correlation between caring ethic, work satisfaction and performance. Community members appear to naturally care for the project and do not want anything that would hamper its effectiveness and efficiency. Code review is a way to execute that care for reviewers, to enforce the quality requirements. Asked how people cope with a heavy review load in the Linux community, Participant 16 stated *"People care about quality in this community. Not only that. People are passionate about the project."* Participant 3 puts code reviews and care as the two most important factors behind the success of FOSS: *"number one is peer review, so that is one of the main practices. Number two, we try to do it with care. Number three, we try to have as many tests as possible and we try to have at most 80-90% code coverage."*

Paradoxically, caring is the motivating factor for some subjects, also when they are reviewees. Care shown by others may help to deal with negative aspects of code review. Participant 1 names the caring attitude of his community mentors as the main motivator behind his persistence in the early days. Others talk about reviewers who not only criticize contributions, but also offer help to improve them.

Observation 4: *The ethic of care drives our subjects. They use the gate of code review to exercise care for quality. Care also helps them to control the negative feedback.*

Implications. Care is easier to habituate than passion. Thus companies and projects should find it easier to exploit it, by instituting an ethic of care. A caring attitude can be rewarded and encouraged. It appears that a successful implementation of code review would be helped if leaders cared about how contributors cope with feedback, and if they themselves shown care for code review. Note that in many FOSS communities the project leader is actually the most active code reviewer.

E. Intrinsic Motivation

Intrinsic refers to the innate, the natural, the part of a whole that cannot be removed from the whole nor the whole from it. Intrinsic motivation derives from internal satisfaction. Some subjects directly name natural affinity for programming behind their dedication to work (including code review), adherence to best practice and assuring the overall quality of the tasks.

Altruism and enjoyment were observed to be the main intrinsic motivation amongst our participants, for instance: *"The*

TABLE IV
EXTRINSIC MOTIVATORS IN THE STUDIED COMMUNITIES

Motivator	Description
Reciprocity	Known in the literature as the gift economy. Other community members respond positively to your contributions, through mechanism like reviewing code for each other, offering help, public appreciation, etc.
Participation in success	Success is attractive, and contributors find it rewarding to be part of a successful project, and being able to help, or given responsible roles in it (such as code reviewer).
Status, reputation	Work experience in a FOSS project accumulates technical expertise and social capital elevating the contributor status. Since the management hierarchy is much less important than in commercial organizations, the status matters more here. Reputation and status might be intangible, or expressed using a metric system (like Karma points, forks, likes, etc.)
Career building	Both intangible and "tangible" reputation accumulated in successful open source projects, translate into carrier opportunities for engineers: interesting job offers, presenting at conferences, etc.
Learning	Code review, like any feedback process, provides peer-learning and development opportunities. The reviewers act like masters in relationship to apprentices (contributors). These roles can of course swap for the same individuals.
Be amongst the best	Code-review introduces a high acceptance bar, deterring mediocre submissions. Contributors' apprehension of critique is an extrinsic motivator that leads to better initial submissions and more diligent improvement to feedback. Unlike the previous five, this is the only negative extrinsic motivator (exercised by code reviewers) that we have seen in our study.

feeling that your code is going to be used, maybe, for future. Maybe its going to help some people and that's something I really like" (Participant 7). The FOSS motivation literature suggest that enjoyment is a key motivator for contributors [14], [18], [19]. Subjects are *"happy"* (Participant 7) or experience *"nice feelings"* (Participant 21) when performing code review.

How does intrinsic motivation influence the execution of code review? Intrinsically motivated employees perform well, behave effectively, and remain loyal to the organization [20]. Rogstadius and coauthors determined that increased pay does increase worker's willingness to accept a task and faster completion, but pay does not affect the quality of the work [21]. They claim that, intrinsic motivators can lead to higher quality work, in fact, higher than extrinsic rewards.

Our subjects agree with these findings, Participant 11 states *"It's a great feeling. I don't know how to describe it, but feels nice. It drives me always to do more and better."* The intrinsic quality in the execution of code review starts from the selection process. Contributors voluntarily select the patch they are comfortable to review. This is different from many closed-source environments, where reviewers are assigned to review code.

Observation 5: *Altruism, and enjoyment are key intrinsic motivators for our subjects. Open source reviewers are effectively volunteers (even if paid) and can choose review tasks following intrinsic interests.*

Implications. The very nature of intrinsic motivation is that it cannot be easily controlled externally. Perhaps, the only way to exploit it, is to watch for the symptoms (altruistic behaviours, enjoyment), and take them as indicators of good code reviewers. Furthermore, increasing the freedom of choice for reviewers might increase their effectiveness.

F. Extrinsic Motivation

Perhaps the most interesting are the extrinsic motivators that affect the code review, as they are the most controllable mechanism in place. We analyze them in more detail than the previous findings. Table IV summarizes the six extrinsic motivators identified in our data. We devote a paragraph to each below. We close each of them with a hypothesis on how it could be exploited by project managers implementing code review.

Reciprocity. In contrast to our present economy system, which is based on quantifiable and measurable exchange transactions using money as unit for measurement, the gift economy is much more flexible. When giving, the contributor is owed [22]. There is an implicit moral obligation to reciprocate the gesture of giving [23], but more subtle than just give-and-take. The exchange does not require quantification and measurement, or at least not explicitly. A FOSS community gives you learning, expertise and the sense of togetherness. Your respond helping others to experience the same. *"I feel satisfied when I'm able to give back something that I have"* (Participant 6).

How shall one operationalize this motivator? First, the situation where learning, expertise, and the sense of community is offered to contributors and reviewers is easy to mirror in other projects (both closed and open source). Second, our data shows that it may be worth to make the software engineering environment a relational (where relations grow through reciprocal exchanges), not only transactional (where time and effort is exchanged for a paycheck).

Participation in success. Success is attractive to hackers. A successful project earns the respect of contributors, and attracts more contributors. *"It's a great piece of software and a success story. Everybody wants to be part of it"* (Participant 16). Success is a general motivator, also for parties during code review, which is seen as a key practice contributing to the success in FOSS communities. It is hard to make a project successful, however successful projects can exploit it to attract community members, and can expect members to be more motivated to contribute.

Status in the community. Reputation. Reputation is the most pronounced extrinsic motivator in our data, out of all listed in Table IV. This is in line with the rich literature about desire for reputation as a motivation to participate in FOSS communities overall [14], [24]–[26]. Some subjects partake in code review as a way to gain reputation and recognition. Some go as far as to name the status an *"award"* for code review. Some perform it particularly diligently, as not to loose the hard earned reputation. Work experience in an open source project accumulates technical expertise and social capital elevating the contributor's status. Since the management hierarchy is much less important than in commercial organizations, the status matters more here.

Reputation is motivating even for junior contributors. Participant 1 reports how he felt having received praise from his mentor for suggesting an alternative architectural solution in a code review. The mentor recognized his technical expertise: *"I felt like I [was] made for this, I had to do this* *more and more … four to five hours continuously, I coded for the second issue to get the same feedback and I'm still doing this continuously because that energizes me."*

Paradoxically, code reviewing earns reputation, but then it may diminish the effect of code review on one's contributions. Some highly recognized senior contributors admit that they are treated more respectfully in code review, because of their reputation, or even that they are able to commit code without review.

How to exploit reputation and status in project organization to boost code review? One simple idea is to use gamification. Another idea is to let the code review practice contribute to building a meritocratic structure in the project.

Today reputation is often gamified through some kind of points system (karma), computed by the project development platform. ROS community members, who work on implementing the new code review process, believe that performing code reviews should be reflected by rewards in such a point system.

Accruing and recognizing status based on technical contributions leads to construction of meritocracy as a backbone social hierarchy in the project. The senior members of ROS community believe that meritocracy should be designed and blended into the process of code review. So code review should become an instrument into organizing the community—then, as a side effect, the code review will work better itself.

Career building. Subjects recognize that active participation in FOSS projects is a differentiating factor on the job market, a tangible sign of expertise that can be leveraged in career development. *"It's nice to build up a CV, it's nice for an intellectual perspective because once you get some code accepted, it means that you are reaching some level of, you know some stuff"* (Participant 17). A company director active in the ROS community meetings stated that his company benchmarks candidates using community profiles and reputation.

Learning. Participants of code reviews learn from each other. Subjects agree that is beneficial for many; for junior project members, but also for the reviewers. *"that's how they [reviewers] learn, maybe they will see in the code something that they didn't know about and it's interesting to them so they will ask about it, or they discuss it"* (Participant 4). *"Review is making us better programmers. We learn when we review others code and when our code is being reviewed. You learn from other code and how they code and you learn from the feedback"* (Participant 14). This is the self-applying motivator of code review: people participate for the direct educational benefit of it. Project managers should remember that for many subjects not the mundane and simple, but the stimulating and developing tasks are motivating, and maintain the corresponding allocation of tasks.

Be amongst the best. Mediocrity is prosecuted with a harsh language and strong tone in the Linux Kernel community. This attitude aims at filtering the best and shields the project from average contributions. *"I think if you get just the best people, perhaps the contributions are then the best. There are many ways of interacting that requires this high touch."* (Participant 16). While widely criticized online,[6] even the critiques admit that code review communication needs to be harsh. The problem they criticize is not harshness, but lack of respect: *"I need communication that is technically brutal but personally respectful"* (*idem.*). This is the only negative motivator observed in the study; the main form of punishment for under-performing used in FOSS communities.

Observation 6: *An established reciprocal gift culture, sharing in the fame of success, reputation, public visibility of status for employers, learning opportunities, and punishment for not performing ultimately the best are the key extrinsic motivators behind work and code review of our subjects. These are all non-monetary motivators that can be used to improve code review.*

G. *Trustworthiness of the findings*

The validity of qualitative research is achieved through trustworthiness [27], [28]. Four constructs are used to establish trustworthiness: credibility, transferability, dependability, and confirmability.

Credibility establishes internal validity, which rivals hypotheses exclusion [27], [29], [30]. It ensures the proposed theory is reliable and representative of the raw data [31]. We used peer debriefs and participant checks. One author conducted the coding the other authors confirmed the emerging theory and categories from the collected data. Participant checks have been used for narrative accuracy and interpretive validity [27], [29], [32]. Participants were asked to validate the authenticity of the verbatim transcripts. They were also asked to comment on the analytical interpretation. Their comments served as a check on the viability of the coding.

Transferability refers to the extent to which the findings of qualitative research, either partially or completely, can be generalized or applied to similar settings [27]. We believe that we meet the transferability requirements by providing evidence that the research findings could be applicable to other similar contexts (i.e., free and open source communities). An audit trail is available and detailed enough to allow other researchers to replicate a similar inquiry in similar communities [33]. Sikolia et al. [28] suggest that researchers can ensure transferability by describing the research clearly, explaining the diverse experiences of the participants, implementing methodology, interpreting the results, and adding contributions from debriefing.

Dependability is synonymous with reliability in the traditional quantitative research. It is concerned with the ability of the research to reach the same conclusions if replicated in the same setting and conditions [27]. It measures replicability or repeatability. This is done by a peer researcher who audits and confirms that the research procedures are followed and authentic. Shenton [27] suggests that the research report should include discussions on dependability and that researchers should comprehensively explain the research design and the data gathering methods.

Confirmability refers to real objectivity in the study. It is improved by triangulation of the study data and findings [27]. The study should reflect the preferences of the participants and

[6]For example: https://sage.thesharps.us/2015/10/05/closing-a-door/

not the researchers. Unlike quantitative studies, the direction of a qualitative study is created by the participants and not the researchers. The reflective discussion of the researcher promotes the reality that the data indeed reflects the participants' views and not the researcher's. An audit trail should also be discussed in the study report as another tool to improve confirmability in the study [27], [29]. This audit establishes confirmability [28]. An audit trail is when a detailed process of data collection, data analysis, and interpretation of the data has been provided.

V. Discussion

Our analysis shows that the human aspect of code review is definitely not to be ignored. Human constructs such as handling rejection, coping with close scrutiny, ethic of passion, ethic of care, intrinsic and extrinsic motivations shape the execution of code review in important and mostly positive ways. We aggregate the most actionable consequences of the study, along with proposals of actionable interventions in Table V.

Rejections are an inherent aspect of the code review process (Observation 1). Software engineering environments should institute a culture where rejections are embraced. A rejection culture implies understanding and communicating that rejection is not a failure. Otherwise it is very difficult to use code-review to improve quality. Instead of being eliminated, rejection and the negative experiences need to be sublimated into a learning opportunity. Fortunately, this can be rationalized and trained and many organizations use internal reviewing successfully as way to raise quality of products.

According to Burke and Fiksenbaum passion enhances mental and psychological well-being of the employees [34]. FOSS contributors (Observation 3) show as the most passionate workers in engineering, and, as such, an excellent subject to study this phenomenon further. Project and community managers should definitely not ignore but cherish and support this virtue of software teams.

While the ethic of care is not really associated with a stereotypical antisocial programmer in public perception, the FOSS contributors clearly exhibit traits of caring, at least in the limited scope of their project and community (Observation 4). That care is apparently developed through existing mechanisms in the FOSS communities that can also be used by others: the contagious care of project leaders, mentoring arrangements, and a sense of shared ownership of project's design, goals, and ways of working.

It is difficult to directly implement exploitation of intrinsic motivators in code review. Self-determination theory attempts to differentiate factors that facilitate and that undermine intrinsic motivation [35]. A sub-theory, the cognitive evaluation theory, maintains that interpersonal events that lead to feelings of competence enhance intrinsic motivation when they are accompanied by a sense of autonomy [35], [36]. Autonomy is described as the leeway that is given to the employee to complete their job tasks [37].

Scientists do agree that satisfaction and performance increases intrinsic motivation. Kraiger, et al. [38] argue that a positive effect increases people's enjoyment and interest. Erez et, al. [39] found that a positive effect increases the intrinsic attractiveness (i.e. goodness) of moderately desirable rewards. It also affects satisfaction and performance during the activity [39], [40]. Thus it would be extremely valuable to explore more action oriented research involving intrinsic motivation in code review, and collaborative software development in general.

The richness of extrinsic motivators is visible in the studied communities (Observation 6). There are known results that extrinsic motivators do correlate with performance and that they synergize with intrinsic motivations [41], [42]. We list some ideas on how to exploit them to improve code review in the bottom most part of Table V.

VI. Related Work

The subject of code review has been investigated from various angles, yet, the human and social aspects of the process received little attention. Bachelli and Bird [43] found that the top motivation for code reviewers is finding defects, but in fact, defect-related communication is proportionally small. Instead, the reviewers are concerned with knowledge transfer, increasing team awareness, and creating alternative solutions. The fabric of code review is communication, knowledge transfer, praise and critique. Code review is primarily a social activity.

Lussier [44] describes the experience of first rejection of a contribution to the Wine project, an open source implementation of the Windows API. While the team was initially resentful, after three more rejections, the code was accepted. Meanwhile the team developed a real sense of ownership and pride in their work. Lussier recounts that the passion of contributors is one of the reasons for the high quality of code. These findings are in accordance with our conclusions.

Asynchronous reviews support team discussions and find the same number of defects as collocated meetings [1]. They provide passive listeners with learning experience; focus better on the ideal solution, not on the defects, but still find defects earlier than the scheduled reviews. Most FOSS reviews begin within hours of submitting the change and are completed within one or two days. It is key that the reviewed changes are small, independent, and complete; typically 11—32 lines of code. Rigby et al. point out that communities allow expert developers to self-select submissions to review [1]. Selecting tasks can allow experts to stay vested in a project. However, their study misses the human and social factors of code review. We show that these are important. They sway the execution positively and steer the outcome to higher quality. They also appear to be exploitable in project organization.

Votta [45] suggests that face-to-face code review meetings of the whole team should be replaced with depositions. A deposition is a three-person team: an author, a moderator, and a reviewer. To save costs, the moderator can be eliminated. The results of such meetings are just as effective as full-team meetings. Our study indicates that human and social aspects of code review can counteract the impersonality of asynchronous meetings, and they can still allow for many experts to interact.

German and colleagues [46] studied OpenStack. They found twenty-four percent of participants subject to reviews stated that

TABLE V
AN INTERPRETATION OF THE STUDY: SUGGESTIONS OF ACTIONABLE INTERVENTIONS

Consequence of Observations	**Examples of consequent actions/interventions for projects**
A working environment valuing code review *embraces rejection*, as rejection is inherent and valuable in code review. (cf. Observation 1)	– Provide code review guidelines on how to communicate constructive feedback and how to interpret feedback. – Include handling rejections into training for new engineers. Rejection is not failure. – Establish an in-house counseling, mentoring or coaching function for engineers that maintains understanding of the positive value of code review.
Iterative improvement is an important method of programming work, besides the pervasive striving for perfection. (cf. Observation 2)	– Promote code review practice as a knowledge sharing and learning opportunities. – Reward and appreciate learning, and striving for excellence both in teams and individuals. – Democratize the code review process. Allow reviewers to select what they want to review.
The *ethic of passion* should be nurtured in project teams and communities. (cf. Observation 3)	– Recruit passionate developers if possible. Seek passion for code review in particular. Appoint passionate individuals to lead software engineers. – Eliminate toxic sources that deplete passion from engineers. – Reduce tasks that do not coincide with passions. – Include passion and self-determination as a parameter when allocating tasks. FOSS contributors choose what to work on, what to review, when to do it, etc.
The *ethic of care* is a positive contributor in a project team involved in code review, worthwhile cultivating. (cf. Observation 4)	– Care and commitment shall be demonstrated at every level of the software team. Leaders should review code diligently. Many FOSS project leaders are top code reviewers. – The FOSS we studied, nurture care by establishing mentor–contributor relationships that last up to five years. Developing relationships seems key to the ethic of care. – Develop a team as a community, especially regarding reviews. Develop *"our way"* of doing things and enforce it in reviews; emphasize common ownership, methods, designs and successes.
Nurturing engineers' *intrinsic motives*, altruism and enjoyment. (cf. Observation 5)	– Intrinsic motives are very difficult to exploit, and little known results of action research regarding that exist in software engineering. To the best of our knowledge, so far this is mostly an area for future research.
Many non-monetary exploitable *extrinsic motives* drive FOSS contributors to make code-review effective. (cf. Observation 6)	– Do not neglect non-monetary non-material rewards, but develop a relational environment in your team, where rewards come from collaboration. – If your project is successful, use it to attract motivated community members. – Build a meritocratic hierarchy, a key motivator for performing well in code reviews. – You might want to use a point system to gamify gaining reputation and status. – Acknowledge and reward individuals and group achievements regularly, with growth opportunities (promotions, conference talks, etc.). – Use code review as a key learning and development opportunity for engineers in your team.

they are treated unfairly occasionally and fifteen percent feel they are treated unfairly often. Reviewers who were questioned stated that they conduct reviews fairly (60%), but some stated that they conduct reviews unfairly occasionally (40%). They stated that contributions are prioritized for review by the developer's expertise, the importance of the patch, the author of the patch, the difficulty of the patch, or the freshness of the patch, which can affect consistency and perceptions of fairness. Newcomers are often treated with a negative bias, as well. These findings emphasize the importance of the human and social aspect of code review.

VII. Conclusion

We had set out to explore why code review works for open source software communities. We interviewed 21 open source developers from four different successful communities and collected data from meetings of a community debating introduction of code review. Having analyzed the data, we find that, besides the well known project management and QA reasons for success of the code review practice, a number of human and social aspects are key—they create a psychological and social environment that is friendly for development of successful code review interactions.

In order to encourage future work, we examined the data, and extracted patterns of management, behavior, and other elements of the FOSS work environment pertaining to code review that appear to be replaceable in other contexts (Table V). We hope that these can be a useful inspiration for project and community managers. Fore-mostly, we hope that they can inspire action research interventions into both closed and open source projects, and that this way we can obtain a better understanding how to proactively control the quality of code review.

Acknowledgment: Supported by the ROSIN project under the European Union's Horizon 2020 research and innovation programme, grant agreement No 732287. We would like to thank the interviewees for their participation and making this research possible.

REFERENCES

[1] P. Rigby, B. Cleary, F. Painchaud, M.-A. Storey, and D. German, "Contemporary peer review in action: Lessons from open source development," *IEEE software*, vol. 29, no. 6, pp. 56–61, 2012.

[2] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: a case study of the apache server," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 541–550.

[3] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proceedings of the 11th working conference on mining software repositories*. ACM, 2014, pp. 202–211.

[4] A. Aurum, H. Petersson, and C. Wohlin, "State-of-the-art: software inspections after 25 years," *Software Testing, Verification and Reliability*, vol. 12, no. 3, pp. 133–154, 2002.

[5] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Foundations of empirical software engineering: the legacy of Victor R. Basili*, vol. 426, no. 37, 2005.

[6] R. J. Vallerand, C. Blanchard, G. A. Mageau, R. Koestner, C. Ratelle, M. Léonard, M. Gagné, and J. Marsolais, "Les passions de l'ame: on obsessive and harmonious passion." *Journal of personality and social psychology*, vol. 85, no. 4, p. 756, 2003.

[7] N. Noddings, *Caring: A relational approach to ethics and moral education*. Univ of California Press, 2013.

[8] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: Classic definitions and new directions," *Contemporary educational psychology*, vol. 25, no. 1, pp. 54–67, 2000.

[9] C. Robson and K. McCartan, *Real world research*. John Wiley & Sons, 2016.

[10] M. B. Miles, A. M. Huberman, and J. Saldana, "Qualitative data analysis: A method sourcebook," *CA, US: Sage Publications*, 2014.

[11] P. Himanen, *The hacker ethic*. Random House, 2010.

[12] R. A. Ghosh, R. Glott, B. Krieger, and G. Robles, "Free/libre and open source software: Survey and study," 2002.

[13] R. A. Ghosh, "Understanding free software developers: Findings from the FLOSS study," *Perspectives on free and open source software*, pp. 23–46, 2005.

[14] K. R. Lakhani, R. G. Wolf, and Others, "Why hackers do what they do: Understanding motivation and effort in free/open source software projects," *Perspectives on free and open source software*, vol. 1, pp. 3–22, 2005.

[15] R. J. Vallerand, S.-J. Salvy, G. A. Mageau, A. J. Elliot, P. L. Denis, F. M. E. Grouzet, and C. Blanchard, "On the role of passion in performance," *Journal of personality*, vol. 75, no. 3, pp. 505–534, 2007.

[16] A. Bonneville-Roussy, G. L. Lavigne, and R. J. Vallerand, "When passion leads to excellence: The case of musicians," *Psychology of Music*, vol. 39, no. 1, pp. 123–138, 2011.

[17] A. L. Toombs, S. Bardzell, and J. Bardzell, "The proper care and feeding of hackerspaces: Care ethics and cultures of making," in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. ACM, 2015, pp. 629–638.

[18] K. R. Lakhani and E. Von Hippel, "How open source software works: " free" user-to-user assistance," *Research policy*, vol. 32, no. 6, pp. 923–943, 2003.

[19] B. Luthiger and C. Jungwirth, "The Chase for OSS Quality: The Meaning of Member Roles, Motivations, and Business Models," in *Emerging Free and Open Source Software Practices*. IGI Global, 2007, pp. 147–168.

[20] R. Q. Danish, M. K. Khan, A. U. Shahid, I. Raza, and A. A. Humayon, "Effect of intrinsic rewards on task performance of employees: Mediating role of motivation." *International Journal of Organizational Leadership*, vol. 4, no. 1, 2015.

[21] J. Rogstadius, V. Kostakos, A. Kittur, B. Smus, J. Laredo, and M. Vukovic, "An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets." *ICWSM*, vol. 11, pp. 17–21, 2011.

[22] E. S. Raymond, "The Cathedral and the Bazaar," 1998.

[23] D. Zeitlyn, "Gift economies in the development of open source software: anthropological reflections," *Research policy*, vol. 32, no. 7, pp. 1287–1291, 2003.

[24] J. Hahn, J. Y. Moon, and C. Zhang, "Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties," *Information Systems Research*, vol. 19, no. 3, pp. 369–391, 2008.

[25] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel," *Research policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[26] J. Lerner and J. Tirole, "Some simple economics of open source," *The journal of industrial economics*, vol. 50, no. 2, pp. 197–234, 2002.

[27] A. K. Shenton, "Strategies for ensuring trustworthiness in qualitative research projects," *Education for information*, vol. 22, no. 2, pp. 63–75, 2004.

[28] D. Sikolia, D. Biros, M. Mason, and M. Weiser, "Trustworthiness of grounded theory methodology research in information systems," 2013.

[29] S. C. Brown, R. A. Stevens, P. F. Troiano, and M. K. Schneider, "Exploring complex phenomena: Grounded theory in student affairs research," *Journal of college student development*, vol. 43, no. 2, pp. 173–183, 2002.

[30] G. Rolfe, "Validity, trustworthiness and rigour: quality and the idea of qualitative research," *Journal of advanced nursing*, vol. 53, no. 3, pp. 304–310, 2006.

[31] D. Straub, M.-C. Boudreau, and D. Gefen, "Validation guidelines for IS positivist research," *The Communications of the Association for Information Systems*, vol. 13, no. 1, p. 63, 2004.

[32] M. Carcary, "The Research Audit Trial: Enhancing Trustworthiness in Qualitative Inquiry." *Electronic Journal of Business Research Methods*, vol. 7, no. 1, 2009.

[33] A. Cooney, "Rigour and grounded theory," *Nurse researcher*, vol. 18, no. 4, pp. 17–22, 2011.

[34] R. J. Burke and L. Fiksenbaum, "Work motivations, satisfactions, and health among managers: Passion versus addiction," *Cross-Cultural Research*, vol. 43, no. 4, pp. 349–365, 2009.

[35] R. M. Ryan and E. L. Deci, "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being." *American psychologist*, vol. 55, no. 1, p. 68, 2000.

[36] F. P. Morgeson, K. Delaney-Klinger, and M. A. Hemingway, "The importance of job autonomy, cognitive ability, and job-related skill for predicting role breadth and job performance." *Journal of applied psychology*, vol. 90, no. 2, p. 399, 2005.

[37] C. J. Fornaciari and K. L. Dean, "Experiencing organizational work design: Beyond hackman and oldham," *Journal of Management Education*, vol. 29, no. 4, pp. 631–653, 2005.

[38] K. Kraiger, R. S. Billings, and A. M. Isen, "The influence of positive affective states on task perceptions and satisfaction," *Organizational Behavior and Human Decision Processes*, vol. 44, no. 1, pp. 12–25, 1989.

[39] A. Erez and A. M. Isen, "The influence of positive affect on the components of expectancy motivation." *Journal of Applied psychology*, vol. 87, no. 6, p. 1055, 2002.

[40] B. M. Staw and S. G. Barsade, "Affect and managerial performance: A test of the sadder-but-wiser vs. happier-and-smarter hypotheses," *Administrative Science Quarterly*, pp. 304–331, 1993.

[41] J. Boiché, P. G. Sarrazin, F. M. Grouzet, L. G. Pelletier, and J. P. Chanal, "Students' motivational profiles and achievement outcomes in physical education: A self-determination perspective." *Journal of Educational Psychology*, vol. 100, no. 3, p. 688, 2008.

[42] L. G. Pelletier and P. Sarrazin, "Measurement issues in self-determination theory and sport." 2007.

[43] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013, pp. 712–721.

[44] S. Lussier, "New tricks: How open source changed the way my team works," *IEEE software*, vol. 21, no. 1, pp. 68–72, 2004.

[45] L. Votta, "Does the Modern Code Inspection Have Value," in *Presentation at the NRC Seminar on Measuring Success: Empirical Studies of Software Engineering*, 1999.

[46] D. M. German, G. Robles, G. Poo-Caamaño, X. Yang, H. Iida, and K. Inoue, "Was my contribution fairly reviewed?: a framework to study the perception of fairness in modern code reviews," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 523–534.