A market for trading software issues

Malvika Rao
Incentives Research

Georg J.P. Link
University of Nebraska at Omaha, glink@unomaha.edu

Don Marti
Mozilla

Andy Leak
Mountain View Smart Contracts

Rich Bodo
Mountain View Smart Contracts

Recommended Citation

Malvika Rao, Georg J P Link, Don Marti, Andy Leak, Rich Bodo, A market for trading software issues, Journal of Cybersecurity, Volume 5, Issue 1, 2019, tyz011, https://doi.org/10.1093/cybsec/tyz011

Journal of Cybersecurity, 2019, 1–14
doi: 10.1093/cybsec/tyz011
Research paper

Research paper

A market for trading software issues

Malvika Rao [1,*], Georg J. P. Link [2], Don Marti[3], Andy Leak[4] and Rich Bodo[4]

[1]Incentives Research, Ottawa, ON, Canada; [2]University of Nebraska at Omaha, Omaha, NE, USA; [3]Mozilla, Mountain View, CA, USA; [4]Mountain View Smart Contracts, Mountain View, CA, USA

*Correspondence address. Incentives Research, 221 Lyon Street North, Apt. 2007, Ottawa, ON, Canada K1R 7X5. E-mail: malvikar@gmail.com

Received 15 October 2018; revised 15 April 2019; accepted 20 August 2019

Abstract

The security of software is becoming increasingly important. Open source software forms much of our digital infrastructure. It, however, contains vulnerabilities which have been exploited, attracted public attention, and caused large financial damages. This article proposes a solution to shortcomings in the current economic situation of open source software development. The main idea is to introduce price signals into the peer production of software. This is achieved through a trading market for futures contracts on the status of software issues. Users, who value secure software, gain the possibility to predict outcomes and incentivize work, strengthening collaboration and information sharing in open source software development. The design of such a trading market is discussed and a prototype introduced. The feasibility of the trading market design is corroborated in a proof-of-concept implementation and simulation. Preliminary results show that the implementation works and can be used for future experiments. Several directions for future research result from this article, which contributes to peer production, software development practices, and incentives design.

Key words incentives; market design; peer production; software market

Introduction

In today's world, software is ubiquitous and will become even more so with the advent of the Internet of things. The importance of software security cannot be overstated.

Software systems have evolved to be large, decentralized, dynamic systems where the model of computation is one of continuous interaction with other large systems and with the world. This rapid increase in software size and complexity has given rise to massive inefficiencies and errors. A 2002 study estimated the annual cost to the US economy of software errors alone at approximately \$59.5 billion [1]. More recent figures on cyber risk paint a graver picture. Global costs of cybercrime have been estimated to lie between \$799 billion and \$22.5 trillion (1.1–32.4% of global GDP) [2]. Insecure software can be traced back to the incentives firms face to release software early and achieve network effects. After all, firms can release security updates later. It has also been shown that software producers tend to release security updates later than is socially optimal [3]. Information insecurity is partly due to a failure in the design of incentives [4].

Open source software forms much of our digital infrastructure and has enabled the boom in start-ups [5]. Peer production, the mechanism behind the development of open source software is an organizational innovation where individuals, in a diverse and distributed community, self-match to the tasks best suited for them [6]. Peer production has successfully tackled complex, uncertain projects, underlying billions of dollars in open source software production [7]. A recent study, however, points out that this digital infrastructure is increasingly under strain [5]. Escalating demand and a lack of adequate resources has resulted in security breaches and service errors [5]. Earlier economics research foresaw these types of real-world problems. The absence of price signals in open source development means that users' valuations remain unknown [8]. Therefore the supply of and demand for open source software goods do not fully align.

In fact, software quality today seems to be below the level preferred by users and developers alike. That is, users are willing to pay to avoid the risks of using software that is broken or missing functionality, and developers are willing to fix and upgrade software if compensated. Thus, there appears to be a failure of market design, and an opportunity to design better market mechanisms to incentivize a higher quality equilibrium in software production. This leads to the research question that motivates our work:

> How can a market design incorporate price signals into peer production, facilitate information sharing, and promote quality?

In this article, we make progress toward addressing this question. We present our vision and preliminary work on a "futures trading market" for finding and fixing software issues. Participants in our market can create futures contracts to predict whether these issues will be addressed, hedge the risks to which they are exposed by using defective software, and incentivize work. Users, developers, testers, project maintainers, investors, and others in the software ecosystem can trade on questions such as: will a bug be fixed? or will a vulnerability be found? Our design incentivizes the discovery and resolution of software vulnerabilities and bugs, but is not restricted to these. Rather, our platform is broad in scope and can be used for various events and tasks in the software ecosystem, including design, documentation, code reviews, and testing.

Issues of software quality (proper design and execution) and security (preventing unauthorized access to data and systems) are both classified as software defects. With increasing use of open source components in software,[1] a quality issue "upstream" in the software development process can imply a security issue "downstream." For example, flaws in the ImageMagick image processing software that are a usability inconvenience on desktop, become security risks when ImageMagick is integrated into a web application where unauthorized users can exploit the flaw to execute remote code [10]. In the absence of any signals, the maintainers of the upstream project may not know that the integration of their components (including any bugs residing therein) may put the downstream project's users at risk.

Bug bounty programs have existed since 1995 and are the state of the art for reporting vulnerabilities (e.g. the Mozilla security bug bounty, and the "Hack the Pentagon" bug bounty program [11]). Open source bounty systems offer rewards for reporting and fixing known bugs. Marketplaces for software tasks include crowdsourcing platforms for open source bounties (e.g. Bountysource [12]), crowdsourcing contests (e.g. Topcoder [13]), and crowdfunding sites (e.g. Kickstarter [14]). However, these market mechanisms have limitations that we address with a novel market design based on trading futures contracts, inspired by instruments used in financial markets.

Related work

Research on software economies has advocated a market-based approach where the supply and demand determine the allocation of work and influence the evolution of a system [15, 16]. An equilibrium in the software economy is one where all issues for which there is enough value have been addressed. Rao *et al.* [17] consider the problem of how to incentivize deep fixes in a public software economy. They design market mechanisms that use externally observable information only in determining outcomes and payments. A mean field equilibrium methodology is used to evaluate the performance of the mechanisms in simulation. A theoretical analysis of the model establishes the existence of an equilibrium [18]. The present article extends this line of research but has a different focus. Here, the objective is to introduce price signals in a peer production market while facilitating information sharing and collaboration.

Market-based approaches to incentivizing vulnerability reporting have been considered. Closely related to our work are "exploit derivatives," proposed by Bohme [19, 20]. Where we apply the principles of futures contracts to the design of a market for software tasks, exploit derivatives apply the concepts of binary options, also taken from the theory of financial markets, to address security. A market failure happens because vendors supply lower levels of security than appropriate and users demand less security than what would be in their best interests. Exploit derivative contracts pay out a certain amount if a security event takes place by a given date, and are issued as a bundle with the inverse contract (that pays out if the event does not take place). In this respect, the binary option property corresponds to the "positions" that a trader can hold, in our contract structure, that also function in pairs (see Example 3.1). However our contract design incorporates many additional elements as described in the next section.

Like exploit derivatives, our market mechanism meets criteria desirable in a vulnerability market: prices provide signaling information on the quality or security of a product, financial incentives motivate addressing software flaws, users can trade to hedge against exposure to software risk, and the market promotes efficiency (e.g. low transaction costs to engage in the market, transparency of trades, and accountability of traders) [19, 20]. That said, our market is further structured to facilitate collaboration and information sharing in peer production settings. By focusing on the contract design as a whole, which includes holding positions, we are able to incentivize various forms of work, such as partial work.

Other vulnerability markets include that proposed by Schechter [21], where the first person to disclose a security flaw is offered a reward. The reward increases in value until claimed, and the product can be considered secure enough to protect information worth the combined value of all rewards. Ozment [22] maps this type of vulnerability market to an open first-price ascending auction. Although this type of vulnerability market provides incentives to report flaws, the price is set by the software producer, users' valuations for fixes are not captured, and there is no mechanism to hedge against risk.

The role of incentives with respect to different aspects of information security and privacy has been studied, and various policies have been proposed for the improvement of these aspects [23–28]. However, these works do not consider the design of a futures trading market. Hosseini *et al.* [29] consider the problem of efficient bug assignment. In their model, the bug triager is an auctioneer and programmers are bidding agents in a first-price sealed-bid auction. Bug triaging is but one of several software tasks that a futures trading market can address.

The characteristics of various contest architectures have been examined [30–35]. Contest architectures have been used in the design of software marketplaces such as Topcoder, but function unlike futures markets. There is a large body of work on prediction markets [36–40]. However, our futures market differs from prediction markets in several ways as we explain in Section 3. Other papers have considered software from the perspective of technological innovation [41, 42], and the design and modularity of software [43, 44].

1 A recent audit, released in 2018, found open source components in 96% of applications scanned [9].

Much software development is organized in peer production communities which have their own economic rules. Benkler [6] describes the emergence of what he refers to as commons-based peer production. He explains that in a networked world, information may be produced and exchanged cheaply and efficiently. Unlike in firms, where the human talent that can be harnessed is mainly limited to its employees, peer production is able to take advantage of a diverse and distributed community, with a variety of skills. Individuals, with private information regarding their skills, match themselves to the tasks best suited for them. Benkler argues that as projects become increasingly more complex and uncertain, it becomes more important to harness diverse motivations because a clear measure of effort can no longer be determined.

The present article belongs to a line of work that considers the economics of open source software, an example of peer production. Lerner and Tirole [45] study four cases of peer production and propose economic models to explain the motivations of open source contributors. Johnson [46] describes a model of open source software as a public good, where developers incur a private cost to contribute and obtain a private valuation whenever any improvement is made. He establishes technical conditions that characterize when a developer will choose to contribute. Athey and Ellison [47] present a model to capture the dynamics of open source contribution. They assume that programmers are motivated by both their own need to use the software as well as altruistic feelings involving the benefits to the community. At any point in time, the open source software meets some subset of the total set of needs, and the measure of this subset is termed the "quality" of the software. The authors show that the dynamic system can reach one of two steady-states: a zero quality and zero altruism state, and a state where the zero as well as positive quality and altruism states can exist.

There is a rich literature on market design for settings as diverse as the labor market for physicians [48], electricity markets [49], the incentive auctions for wireless spectrum [50], high-frequency trading [51], and an electronic marketplace for agricultural goods [52]. Designing a marketplace involves formulating the rules that specify what types of market activities are possible and creating the infrastructure to support these activities [53]. Market design must take into consideration the context for which the market is needed. Thus different situations may necessitate vastly different solutions. Our work addresses market design for software tasks in the context of peer production, and has a practical focus. Accordingly, we devise market rules and infrastructure, presented in section "Our approach."

This manuscript is an extended version of a paper presented at the 17th Annual Workshop on the Economics of Information Security (WEIS) 2018 [54].

Current market-based platforms

There is a variety of market-based platform models, implemented with their own unique flavors. We present here a few samples that focus on software development. Examples include Bountysource, which is a funding platform for open source software [12]. Here, users post bounties or rewards on issues they want to be addressed while developers create solutions and claim rewards. Once a reward is made available, a developer picks the issue he wants to tackle and begins work. The developer submits a claim once the work is done. There is a 2-week verification period during which backers vote to accept or reject the claim. If the claim is accepted the developer receives the reward. Otherwise the bounty is refunded. Bountysource also organizes fundraisers for costly new features requiring a significant investment of time and effort. Kickstarter is a global crowdfunding platform that collects money from the public to fund various projects, including software projects [14].

Topcoder is based on crowdsourcing contests and has over a million active members [13]. Companies with software needs are matched to a global community of programmers who compete in a contest with cash awards to provide the best solution that can address a client request. The Topcoder community works on a variety of tasks from bug fixes and features to design and analytics.

Bountify [55] is another platform based on crowdsourcing contests. It focuses on coding tasks. A client posts the task and the associated reward. Programmers must submit solutions before the reward expires. The client decides which is the best solution and awards the reward to the winner. Interestingly, the client is not refunded if none of the submitted solutions is acceptable. Instead, the reward is donated to charity.

Rather than supplying coding solutions, Bugcrowd [56] is a bug bounty platform for security vulnerabilities that has a crowd of workers at its disposal. The crowd tries to uncover vulnerabilities in a client's software. The client only rewards workers whose vulnerabilities are judged to be valid, regardless of the effort expended. A similar platform is of Hackerone [57]. In addition to supplying a crowd of hackers to uncover vulnerabilities, Hackerone assists organizations to deploy and manage bug bounty programs.

For a survey of crowdsourcing for software development and the related literature, see Mao [58]. Our approach addresses similar issues in a new way as we show in the next section.

Our approach

We propose a futures trading market for eliciting information and incentivizing tasks. Example 3.1 (Fig. 1) illustrates a simple case of how such a market might work.

Example 3.1. (Base Case: Fixing Issue): User Adam finds a software bug. Adam has heard of a futures trading market where he can get the bug fixed for a price. Adam documents the bug in an issue tracker (now identified as bug #1337) and goes to the trading market. Adam creates an offer with a maturation date in 2 weeks for a payout of \$200. Adam buys 200 units – \$1 potential payout each – at a unit price of \$0.8, paying \$160, by depositing the money into escrow. Developer Beth sees the offer, has time to fix bug #1337 within 2 weeks, and decides to accept the offer. Beth buys the 200 units at a unit price of \$0.2, paying \$40, by depositing the money into escrow. The contract is now formed: Adam owns the *UNFIXED* position and Beth owns the *FIXED* position. Two weeks pass by, during which Beth is working on the fix and Adam is waiting to receive it. On maturation there are two possible outcomes: bug #1337 has been fixed or remains unfixed. If bug #1337 is fixed then the issue is closed: Beth earns the reward of \$160 and gets her \$40 deposit back. If bug #1337 is unfixed then the issue remains open: Beth loses her \$40 deposit while Adam receives his and Beth's deposits, earning \$40.

In the same way that a user can offer to pay for a fix (Example 3.1), a user can offer to pay if a vulnerability is found in a specified software project by the maturation date. That is, the user would create a *FOUND* offer. The possible outcomes would be that the vulnerability is *FOUND* or *UNFOUND*.

The basic idea of the market is that participants can create contracts to predict outcomes and incentivize work. A contract is associated with outcomes that can be verified in an issue tracker. On

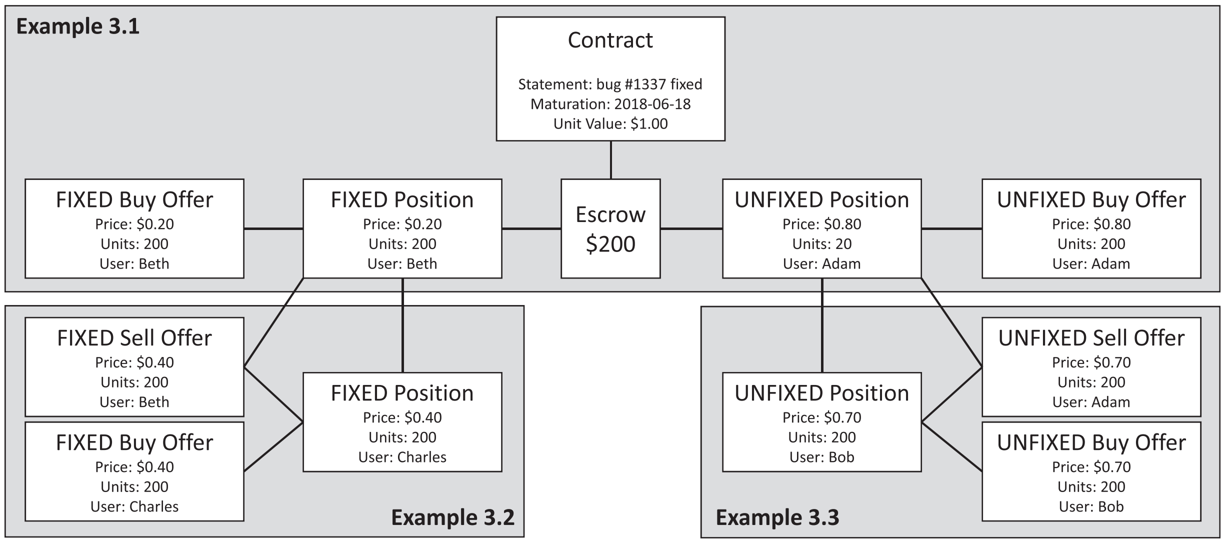


Figure 1. Graphical display of a futures contract, demonstrating how the elements of the contract fit together and evolve over time. A contract with its statement, maturation, and unit value make up the root. Underneath it, many escrows and positions can be created. Money is paid into escrow when fixed and unfixed buy offers match. Positions record ownership and can be resold independent of the original contract partner. Examples 3.1, 3.2, and 3.3 provide the rational for how the contract evolves. Table 1 defines the contract elements in detail.

maturation, the contract pays out to the owner of the position corresponding to the issue's status in the issue tracker. To this end we make a simplifying assumption in the design of the marketplace: the contract must provide clear criteria, including verification tests that specify what must be achieved for the issue to be resolved as fixed.

Design features

A market mechanism for trading software tasks must take into consideration important design criteria. How can we design a platform that balances the incentives of markets with the dynamics of peer production? In this section we describe some key features of our market design.

Partial work

Many problems can only be solved by drawing on different areas of expertise. For example, a bug may require both database and cryptography knowledge, which a single developer may not have. In this instance, the bug would be solved most efficiently if individuals with the needed types of expertise were to collaborate and share information. Furthermore, it has been shown that collaborative work in open source development is often done through a process of "superpositioning," where new layers of work build on existing layers [59]. The design of our trading platform captures these types of dynamics by allowing developers to earn credit for partial work. Thus, trading behavior fosters information sharing and collaborative work, as the next example shows.

Example 3.2. (Partial Work): As before (Example 3.1), Adam ($160 for the *UNFIXED* position) and Beth ($40 for *FIXED* position) enter into a futures contract worth a payout of $200 in two weeks depending on the status of bug #1337. One week later, Beth realizes that she does not have the expertise to fully fix bug #1337. Beth decides to submit a partial fix and sell her *FIXED* position. Charles buys the 200 units of the *FIXED* position from Beth at a unit price of $0.4, paying $80 to Beth. Beth had paid $40 and receives $80, earning $40 for her partial work. Another week passes, during which Charles is working on the fix. On maturation there are two possible outcomes: Bug #1337 has been fixed or remains unfixed. If bug #1337 is fixed then the issue is closed: Charles earns the reward of $200 for a net gain of $120. If bug #1337 is unfixed then the issue remains open: Charles receives nothing but loses the $80 paid to Beth, while Adam receives his and Beth's deposits from escrow, earning $40. In both cases, Beth earned $40 for her partial work.

Another instance of collaborative work supported by the market is when a task consists of several subtasks that can be completed in parallel. Suppose instead that in Example 3.1, Beth buys 100 units and Charles buys the remaining 100 units at the same price. Then each worker would work on a specified fraction comprising the total work required by the contract. On maturation, if the issue is resolved as *FIXED* then both workers are paid. If not, neither worker receives any payment and the funder collects the indemnity.

Dependencies

The completion of task *A* may depend on task *B* being completed first. A developer who has bought the *FIXED* position of a contract on task *A* (with the intention to do the work herself) might then create a new offer to pay for the completion of task *B*. Because she might not finish task *A* otherwise and lose her deposit, the new contract would have a maturation date no later than the date for task *A*.

Flexibility

Example 3.2 demonstrates that developer Beth has flexibility in the market by reselling her side of a contract, after partial work, without affecting funder Adam. Beth might also choose to resell her side of the contract if, after completing the work in its entirety, she needed the funds earlier than the maturation date. Thus, Charles would pay Beth for her work and then be reimbursed on contract maturation. A fair market place should allow both sides of a contract to sell their position and be relieved of their obligations. A funder, like Adam, may decide that the feature is not wanted anymore or that he wants to pay a different developer to complete the work. Flexibility is a core dynamic in open source projects where coordination is based on self-selection and voluntary task execution [6]. The following Example 3.3 demonstrates how the market can provide flexibility in a case where selling happens for other reasons, such as if a user no longer wished to invest in the contract.

Example 3.3. (Selling Position): As before (Example 3.1), Adam ($160 for the *UNFIXED* position) and Beth ($40 for *FIXED* position) enter into a futures contract worth a payout of $200 in 2 weeks depending on the status of bug #1337. After 2 days Adam decides he wants his money back. He sells his *UNFIXED* position of the contract to Bob. Bob buys the 200 units of the *UNFIXED* position at a unit price of $0.7, paying $140 to Adam. Adam has paid $20 for 2 days of development. Bob pays $140 for the remaining days until maturation. Beth is unaffected and continues developing.

Multiple contracts

A project maintainer might need different features for a single project. He may create separate issues and contracts for each feature, such that all contracts have the same maturation date. It is also possible to create multiple contracts for the same issue (e.g. multiple contracts that all offer to pay for a fix for bug #1337), where the maturation dates and payment amounts may be different. The *FIXED* positions on these contracts may be owned by the same developer in order to accumulate rewards for doing the work. Alternately, they may be owned by different people. Because payout at maturation depends solely on the status of the issue in an issue tracker (e.g. has bug #1337 been marked fixed?), this scenario might give rise to free-riding. Consider the contract with the earliest maturation date and suppose Beth owns the *FIXED* position. Clearly, Beth is incentivized to fix the issue or else she will forfeit her deposit. However, those owning *FIXED* positions on the contracts with later maturation dates may get paid on maturation without having to do any work because of Beth's work, which ensures that the issue is already marked as fixed in the issue tracker. On the other hand, they may help Beth to successfully complete her task because their own payout depends on the issue being fixed. To what extent free-riding may occur is influenced by many factors including the beliefs held by participants in the market regarding the ability and willingness of others to perform tasks. Whether free-riding is a serious problem will have to be tested in future research studies.

Collaborative funding

Our marketplace allows collaboration on the *UNFIXED* side. When multiple funders fund the same contract, this gives rise to an instance of "collaborative funding." This does not require the funders to know each other because they can independently post their buy *UNFIXED* offers on the marketplace and still collaborate. The marketplace pools individually created offers on the same issue to create larger pools of funding for developers.

Collaborative funding can take on different flavors. Consider Example 3.1. Suppose instead that Adam buys 100 out of the 200 total units at a unit price of $0.8, paying $80. The remaining 100 units are purchased by Bob at the same price. Beth's situation is unchanged and the contract proceeds as before. Since both Adam and Bob hold *UNFIXED* positions, the indemnity would be proportionally[2] split between them if the outcome on maturation is *UNFIXED*. This is an example of a single offer with multiple funders who may or may not know each other. Alternately, multiple people can pool their funds to create a single offer via a single funding entity (e.g. Bob gives $80 to Adam who creates the offer as described in Example 3.1).

Multiple offers by different people for the same issue and maturation date provide a larger pool for developers to tap into. If more users feel that a fix for an issue is important they can post additional offers, leading to an increasing aggregate price signal for that issue. Furthermore, this option allows multiple developers to take up offers on the same contract if a single developer chooses not to accept all the offers. This may result in several fixes for the same issue and the project maintainer might decide which fix to retain. Regardless, all developers who accepted offers will be paid if the issue is marked *FIXED* by the contract maturation date.

Multiple offers for the same issue can have different maturation dates and different prices. Thus funders can incentivize work by increasing the price on a new offer for the issue, or by allocating more time for the work to be completed. From the developer's perspective, the faster the developer works, the more offers he can cash in on and the slower he works the fewer offers he can benefit from.

Funders can also collaborate by funding different issues that belong to the same project. This results in a shared responsibility by the various stakeholders of a project.

Indemnity

Our design allows for the possibility of an indemnity being paid if an issue is not fixed. This is inspired by dominant assurance contracts where a project owner pays agents, who accept to contribute to the production of a public good, some amount if the project fails to secure a minimum number of contributing agents [60].

Consider Example 3.1. Because Adam contributes $160, Beth must contribute $40 in order to form the contract which has a total payout of $200. If the outcome is that bug #1337 is unfixed then Beth forfeits her $40 to Adam. In this instance the amount that Beth must pay indicates her confidence in being able to fulfill the contract.

Now suppose Adam believes that bug #1337 cannot be fixed and would like to profit from this knowledge. Because Adam is confident that he will recover his payment at maturation, he is willing to contribute a larger fraction of the total contract payout thereby making it easier for a developer to accept his *FIXED* offer. Thus Adam contributes $195 and Beth accepts his offer contributing just $5. At maturation Adam may be proven right or he may be the means of funding a rare solution to a hard problem. With this market design, a belief that something cannot happen is handled the same way as an incentive to make it happen. The attention from speculators, who may buy contracts without having to participate in software development, and the additional money in the market is an advantage. Speculators who speculate that an issue cannot be fixed create an incentive for developers to produce a fix. Aggregating all such contracts, the market potentially creates a pool of wealth that can be captured by innovators.

Decoupling funding from work

The payout from a futures contract is solely dependent on the status of the issue in the issue tracker. At maturation the owner of the position corresponding to the issue's status (e.g. fixed or unfixed) is paid, regardless of who may have done the work to resolve the issue. This decoupling can give rise to interesting scenarios. For instance, an investor in a software project may put up the capital to buy a *FIXED* position and hire a team of talented developers to

2 Each funder would receive an indemnity of $(\frac{100}{200} * 40)$ which is $20.

collaborate and do the work. This also enables someone who does bug triaging to buy *FIXED* positions and find developers who can close the issues and sell them the positions at a profit. Thus, work on open source that fuels community processes can be rewarded.

Moreover, the decoupling can lead to new workplaces. An entrepreneur can trade on the futures market, follow price signals, and assign developers to complete the work. The developers are employed and have the benefits of an income; the entrepreneur organizes resources to the issues that are valued the most on the marketplace to maximize profit. The open source project continues to be coordinated by maintainers who accept solutions to issues based on their quality.

The decoupling may also result in cases where the worker is not rewarded. Suppose users offer to pay for a fix to a particular bug, and a trader realizes that a fix is already underway. The trader quickly buys the *FIXED* position and the market gets information on the likelihood of the bug being fixed, but the worker responsible for the fix does not profit from it. If this happens repeatedly and a developer becomes aware of it, he may post an offer on the market place and only submit his solution if he gets paid. The speculator may be out of his money if the developer decides to withhold solutions until the undesired trading behavior stops.

Anonymity

Anonymity is a desirable property of transactions in many settings. For example, in auctions, it is important that winner determination and payment decisions are made taking into account bidders' bids and not their identities [61]. Anonymity can deter marketplace collusion and manipulation, and provide privacy. However, it is not without drawbacks. For instance, it can be challenging to build trust amongst anonymous parties to a transaction. Online platforms have addressed this problem by allowing buyers and sellers to be selectively anonymous (i.e. to certain parties only) or to have pseudonymous identities, and by instating reputation systems.

Participants in our trading market need to estimate how long it may take a worker to complete a task, how much indemnity is appropriate given workers' skill levels, whether a task is doable at all, and more. These assessments require knowledge of the software as well as the people in the ecosystem. Our market also links to issue trackers, where identities may not be anonymous. Moreover, open source project members typically have working relationships and reputations, which determine the roles they can play. Established and respected members are more likely to become maintainers who decide which software features are accepted and which issues are closed [62].

Participants in our market have the option to select usernames that may be pseudonymous or reflect their true identities. The market must be able to transfer payments between escrow and participants' accounts. No other personal information is revealed to other market participants.

Comparison with existing markets

The set of features presented above work together to create a system that aligns incentives with the natural dynamics of peer production. Continuing, we compare our market mechanism to other mechanisms and argue that our design is suitable to the context.

Although inspired by existing market mechanisms, our design departs from these in several ways. Bug bounty programs, open source bounty systems, and other crowdsourcing approaches to software work typically reward the reporter of a vulnerability or the submitter of a fix. However, software development is often done in a collaborative fashion, where a final solution builds upon the input of others [59]. Furthermore, a task may require different areas of expertise and necessitate the input of different individuals. In these approaches there does not seem to be a way to assign credit to all contributors of a final submitted report or solution. As a result, contributors may be less motivated to collaborate and share information. This limitation is addressed by a futures market because a participant may do partial work on a contract and resell his position (see Example 3.2).

Open source bounty systems have transaction costs for claiming bounties. Bounties must be resolved independently of the fixed or unfixed status of the underlying bug in order to determine whether the work done by the bounty claimer is relevant. In contrast, participants in our futures trading market invest in outcomes which are determined by the status of issues in an issue tracker. Participants may create more expressive contracts (e.g. entailing dependencies) to satisfy their requirements. Moreover, open source bounty systems fail to incentivize meta-work as rewards must be explicitly divided among multiple testers, bug triagers, and developers instead of letting the system handle it. Bug bounty programs that reward the discovery and disclosure of vulnerabilities do not capture the valuations of users for fixes.

Our market resembles a prediction market in regards to the contract structure, but differs in other aspects of the design. Prediction markets tend to have a small number of questions but a large number of participants ("wisdom of the crowd"). Here we have a large number of futures contracts but a small number of participants per future contract ("wisdom of individuals" revealed to crowds of software users). Participants in a prediction market typically cannot influence the outcome whereas bug futures draw participants who have information about that bug and thus may affect the outcome. Prediction markets aggregate information whereas bug futures additionally incentivize tasks.

Auctions have been used for resource allocation in a range of settings and come in a variety of forms. Objects sold at auctions typically require the discovery of the appropriate prices. Bids from participants are compared and a winning bid is selected according to the rules of the auction [63]. In procurement auctions, bidders compete for the right to sell certain goods to a buyer or work on specific contracts, and the buyer's objective is to minimize costs given the requirements. The auction can be combinatorial, that is, bidders can bid on bundles of items, and winner determination might include more than one bidder to satisfy all requirements. This splits a large deliverable into smaller ones, thereby diversifying the competition [64]. Multi-attribute bids, where a bid can include price as well as other parameters such as delivery times, payment conditions, and product quality, are also used [65]. Furthermore, the auction can be run in iterations, providing information feedback to the bidders and permitting them to revise bids accordingly [65].

Our market design retains some of the above properties but is motivated by different conditions. The goal is to introduce prices into the process of peer production in such a way as to leverage and strengthen its successful qualities. Our contract language includes multiple attributes (statement, payout, maturation date) and can be made more expressive if needed. Our market permits multiple contracts on the same issue as well as counteroffers from workers who may submit requests to work on different combinations of work items. Hence funders and workers can compare the set of offers in the market and choose which offers to take up. Once a contract is formed in our market, however, the terms of the contract cannot be revised.

When a contract is formed in our market, the owner of the *FIXED* position need not be the entity who will work to fulfill the contract terms. Decoupling funding from work in our market design means that the funders of a contract bet on outcomes as determined by the status of the work in an issue tracker. Thus, the actual work itself can be carried out by any individual or group of individuals who self-match to the task, as would be the case in a peer production setting. In contrast, a procurement auction typically awards the contract to the winner and all others are excluded from working on the contract.

Group work on a contract can be explicitly undertaken in a few different ways in our trading market. For instance, the owner of the *FIXED* position can hire a group of workers with a set of requisite skills. Alternately, a funder can fund multiple contracts (buy *UNFIXED* positions), each addressing a subtask of the overall task. Group work may also be done by subcontracting dependencies, or by completing partial work on a contract and then reselling the *FIXED* side of the contract to the next worker. In addition, the partial work feature allows market participants to collaborate when an uncertainty in completion of the task appears after a contract is formed. Such situations are typically not addressed by auctions.

Turning to financial markets, there exists a variety of instruments with different characteristics. However, not all of these are applicable for what we want to accomplish with our marketplace. Broadly, tradable financial assets include stocks, bonds, and derivatives. Stocks permit a way to divide and share ownership of an entity and stock owners are entitled to a fraction of the earnings of the entity. Open source projects are not incorporated entities and no one owns open source code. Therefore this type of instrument is not relevant to our situation. A bond is a debt security. A bond issuer issues bonds to borrow funds from the bond holder, with regular interest payments and a promise to repay the principal at the bond's maturation. A bond market does not solve the financing issue in open source because if a worker issues a bond to get money, they would have to pay it back and find other ways to earn money.

A derivative's value stems from the value or performance of an underlying asset. There are several different types of derivatives, including options, swaps, forwards, and futures [66]. In general, the holder of an option has the right, but not the obligation, to trade an underlying asset at a predetermined price and future date (i.e. exercise the option). The dynamics with options do not provide guarantees to market participants that issues will get funded and that work will be completed by a given date. Binary options form the basis for exploit derivatives [19, 20]. As explained in section "Related work," binary options alone are insufficient for our purposes. A swap is a derivative by which two parties exchange the cash flows or liabilities of two different financial products. Swaps are not suitable because we do not have financial products with cash flows to exchange. Forwards are not traded on markets but futures, their standardized version, are, which is discussed in next section.

Futures market

Futures are standardized forward contracts to buy or sell an asset at a specified future time, and at a price determined in the present. They are commonly used in commodity markets (e.g. tulip, rice, cotton, corn, onions, gold, and oil futures) to hedge against price fluctuations and other uncertainties [67]. We apply principles from the design of futures markets to the design of a market for software tasks.

First, there is a quoted price on the market place for the expectation that an issue will (not) be closed at a specified future date. Second, the price of entering a contract is equal to zero but a deposit into escrow is required for the maximum possible loss from the futures contract. Third, at any time before the specified future date, the owner of a contract can leave the contract and receives the difference in price since he entered the contract and the deposit back. Fourth, at the specified future date, the owner of a contract pays with his deposit for the expected outcome (e.g. issue fixed or vulnerability found) or receives his deposit and that of the counterparty.

Futures markets typically reduce transaction costs associated with uncertainty about contract partners by introducing a clearing organization. The trusted clearing organization is buyer and seller to all other market participants. In our market design, we collect the total potential loss up-front and deposit it in an escrow account. By doing this, we establish a direct connection between market participants, but eliminate the risk that one side will not honor the contract, since the market platform controls the money for all contract payouts. We therefore do not need a clearing organization but build this function into the market place. A clearing organization also masks market participants from other market participants, ensuring anonymity, which we emulate through the design of our market platform.

Commodities have characteristics that make them suited for trading on a futures market [67]. Our futures trading market trades contracts on issues, which deviate from these characteristics. We consider five characteristics in turn and reflect on the challenges in applying futures market principles to open source issues:

1. Commodities are traded on futures markets to reduce uncertainty about future prices and availability. An open source issue has uncertainty, including how much users are willing to pay for its fixing, how many users are affected, how much effort is required for fixing an issue, how much a developer would have to be paid for fixing it, how much damage not fixing an issue will cause, and when developers will fix an issue.
2. Futures contracts standardize commodity grade and location, which is important for physical commodities where differences in quality and location (due to transportation costs, tariffs, and taxes) result in varying prices. By standardizing commodities in different futures markets (e.g. corn traded in Chicago and Paris), price differentials emerge that are beneficial for traders, but prices ultimately correlate across futures markets for the same commodity. Open source issues vary in nature, difficulty, and skills required to fix them, which eludes standardization. However, contracts traded on the same issue will be comparable. Because open source projects only have one location (i.e. Internet), price differentials will result only from using different market places, not inherent differences of the issue that is traded. Due to network effects, a tendency is expected for all traders of open source issues on one project to converge on a single market place.
3. A large number of interested participants are required to establish an attractive futures market. A sparsely traded market encounters difficulties in finding counterparties for trading. Open source issues have a potentially large number of users who might be willing to fund it, but typically only a few developers engage in fixing an issue. In such a market, users compete across issues for the time and attention of developers and are expected to have to pay a premium, which is exactly what is desired to fund an open source development work.
4. A greater value of total trades means that risk reduction in the future market has more value. Larger trading volumes incentivize risk-averse people to hedge more on futures markets.

Speculators are attracted by large trading volumes where incentives exist to profitably trade if they believe they have superior information than other market participants. Open source issues might not attract as much trading volume where issues are highly specialized. On the other hand, some issues, similar to those that have received large bug bounty payouts or those that have resulted in billions of dollars in damage (e.g. Heartbleed), might attract large trading volumes.

5. Another criterion for futures markets to develop for a commodity is the free development of prices and absence of regulation. When prices are influenced by government regulation or controlled by a single firm, a futures market is unlikely to exist. This is the case in open source where many successful and valuable projects have a broad diversity of organizations, and wages for developers are not regulated by governments.

Because open source issues are not traded like commodities, a futures market cannot "predict" future prices or hedge against price volatilities. We are inspired by prediction markets, where the payout is determined by an event and the price to enter a prediction represents the beliefs of the market participants about which outcome is most likely. We adopt this premise and standardize contract payouts to $1. The market thus has two prices, one for the fixed side and one for the unfixed side, which together have to sum to $1. In Example 3.1, Adam pays 80% and Beth pays 20% of the contract payout.

Achieving large trading volumes in our market is likely to be a challenge. Strategies to ensure liquidity have to increase the possibility that offers match and result in trades. One option is to define a limited set of maturation dates (i.e. many commodity futures have one date per month, such that soy beans are traded once in August, September, October,...). It is a well-observed phenomenon that futures markets have more trades on short-term contracts than long-term contracts. By avoiding offers that are spread out in time, chances are higher that offers match and result in trades. Another strategy is to allow offers with "variable parameters," such as price limits and maturation date ranges. By introducing variability in offers, they exist in a larger space for potential other offers to match. Futures markets often enlist a "market maker" who buys and sells at all times, thereby ensuring that there is always a trading partner. This approach would have to be adapted to suit the needs of our market.

Proof-of-concept implementation of the futures trading platform

We substantiate our design of a futures trading platform through a proof-of-concept implementation [68], which we call *Bugmark*. The event-driven architecture is inspired by blockchain technology, which can power a decentralized market place. By storing events on a blockchain, such as Ethereum, transactions are stored in an immutable and distributed way, which allows the detection of manipulations of platform operators. The user interface is written in HTML, CSS, and JavaScript (see Fig. 2). The backend is written in Ruby and uses a PostgreSQL database for a local copy of events. We use an event stream architecture that allows updating the blockchain and local database – key elements to a decentralized market design. The implementation confirms that our innovation works.

Futures contract design

An important aspect of the implementation is in regards to the futures contract design. We describe the elements of a Bugmark futures contract (see Fig. 1 and Table 1) by walking through the life of such a contract from creation to maturation (i.e. payout). This description reflects the process in Example 3.1 to introduce the contract elements and more complex use cases are possible.

A new contract gets formed when two buy offers match. In matching offers for creating contracts, a bin-packing algorithm is used to generate the largest possible trading volume. The two buy offers for opposite sides – one for a *FIXED* position, the other for an *UNFIXED* position – have to match in price, volume, statement, and maturation date. Unit prices of the buy offers must together equal to $1 per unit. For example, respective unit prices of $0.20 for a *FIXED* position and $0.80 for an *UNFIXED* position. This is important because we standardize the contract payout to $1 per unit and the two buy offers together have to pay that amount of money into escrow. The unit price determines which share of the $1 payout each buy offer pays into escrow at the time of forming the contract. Volume is how many units of this contract the users want to buy and is ultimately recorded in positions. In other words, the volume is equal to the amount of money both buy offers together must pay into escrow, which is then available for payout upon maturation. The unit price multiplied by the volume determines the amount each user has to pay into escrow, i.e. the total price.

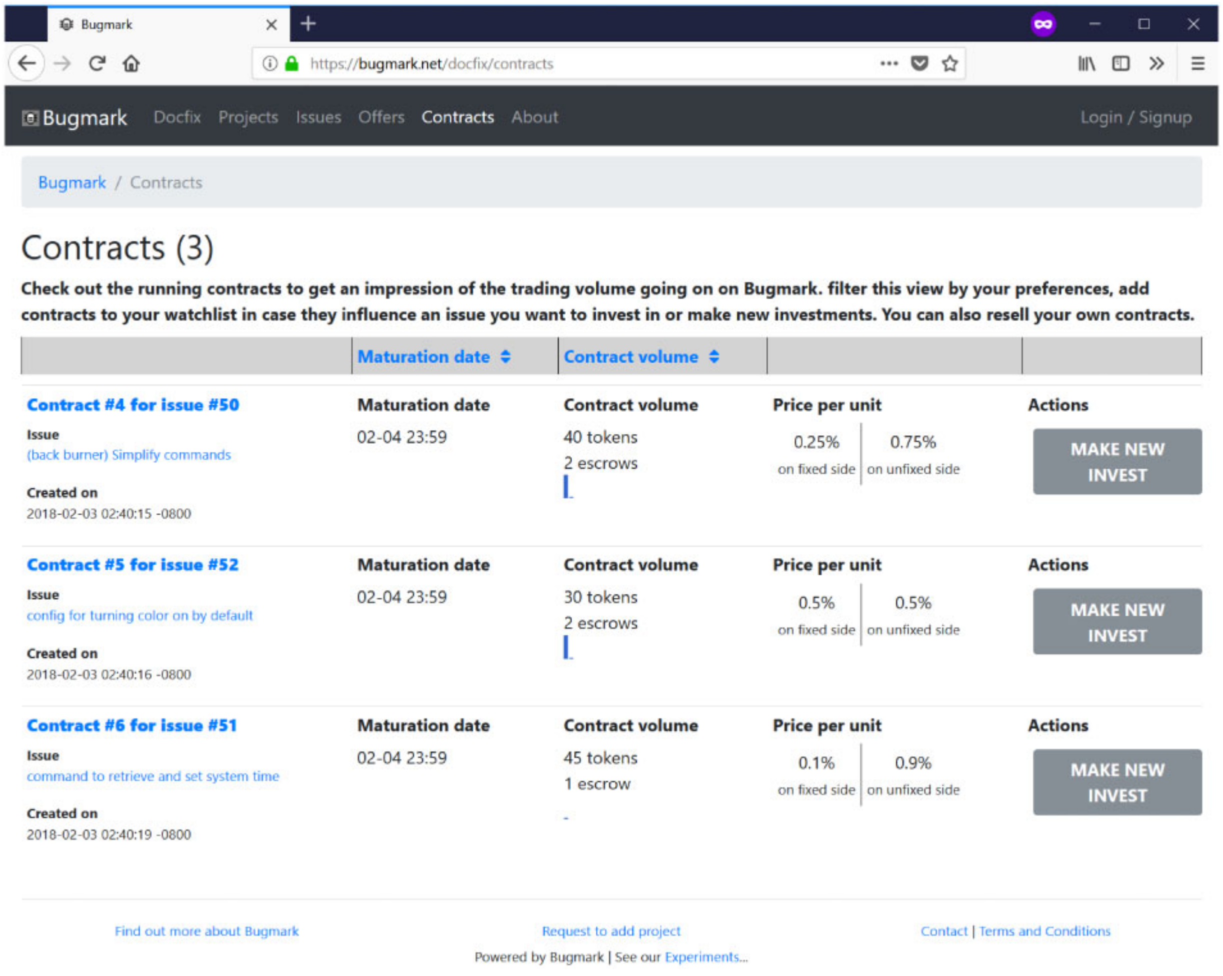
A contract is primarily defined by the statement and maturation date. For each contract, any number of escrows, positions, and offers can exist. This design allows users to trade any number of units (full or partial positions) as long as they belong to the same contract. This design enables several funders to pay for the fix or discovery of a bug and thus pool resources necessary to incentivize difficult tasks.

The *FIXED* and *UNFIXED* buy offers result in positions of the same side: a user posting a *FIXED* buy offer will receive a *FIXED* position; a user posting an *UNFIXED* buy offer will receive an *UNFIXED* position. A position embodies the right to a payout when the statement is evaluated. The statement, e.g. 'bug #1337 is fixed', is the core of a Bugmark futures contract and will be evaluated on the maturation date. The owner of the *FIXED* position gets the payout when the statement evaluates to true; *UNFIXED* positions get paid when the statement is false. At this time, the user that holds the winning position receives the payout from the escrow account.

Multiple buy offers on both *FIXED* and *UNFIXED* side can exist for a single contract. In our design, these offers will result in multiple escrows and positions on the same contract. We acknowledge, that this may appear as an instance of multiple contracts because several developers and funders can have matching offers to create new escrows and positions without affecting already existing escrows and positions. What appears to users as multiple contracts is implemented as a single contract in our design with several positions.

Evaluating the proof-of-concept

Large multi-user systems, such as Bugmark, are challenging to design. The system is non-deterministic and evolutionary because the user experience depends on the decisions of other users. Agent-based modeling is a method to evaluate design decisions in such a system through computer simulation [69]. The major benefit of simulation is that system-level behavior can be observed without a need to recruit a large number of human users, especially for testing small changes to the design. In designing agents, we make assumptions about how they make decisions. Our simulation results show that the proof-of-concept implementation works as expected and the market place provides the desired features. Details are in the Online Appendix.


Bugmark
https://bugmark.net/docfix/contracts
Bugmark Docfix Projects Issues Offers Contracts About
Login / Signup
Bugmark / Contracts
MAKE NEW INVEST

Contracts (3)

Check out the running contracts to get an impression of the trading volume going on on Bugmark. filter this view by your preferences, add contracts to your watchlist in case they influence an issue you want to invest in or make new investments. You can also resell your own contracts.

	Maturation date	Contract volume	Price per unit	Actions
Contract #4 for issue #50 Issue (back burner) Simplify commands Created on 2018-02-03 02:40:15 -0800	Maturation date 02-04 23:59	Contract volume 40 tokens 2 escrows	Price per unit 0.25% on fixed side 0.75% on unfixed side	Actions MAKE NEW INVEST
Contract #5 for issue #52 Issue config for turning color on by default Created on 2018-02-03 02:40:16 -0800	Maturation date 02-04 23:59	Contract volume 30 tokens 2 escrows	Price per unit 0.5% on fixed side 0.5% on unfixed side	Actions MAKE NEW INVEST
Contract #6 for issue #51 Issue command to retrieve and set system time Created on 2018-02-03 02:40:19 -0800	Maturation date 02-04 23:59	Contract volume 45 tokens 1 escrow	Price per unit 0.1% on fixed side 0.9% on unfixed side	Actions MAKE NEW INVEST

Find out more about Bugmark Request to add project Contact | Terms and Conditions
Powered by Bugmark | See our Experiments...

Figure 2. Screenshot of the Bugmark user interface.

Table 1. Elements of a Bugmark futures contract

Element	Description
Statement	A truth statement about whether work will be done. For example: "Bug #1337 is fixed" or "A high security vulnerability was found."
Maturation	Date on which the oracle evaluates the Statement and determines contract payout.
Escrow	Account with deposited money collected at the time of contract formation.
Unit value	Standardized size of a contract: $1.
Unit price	Price for a unit ranges from $0.00 to $1.00.
Position	Record of contract ownership with the right to the payout amount from escrow when the statement gets evaluated. The volume of a position is the number of units the owner bought.
Buy offer	A user indicating willingness to form a new position or take over an existing position for a specified upper price limit and with a specified volume.
Sell offer	A user indicating willingness to sell a specified number of existing positions for a specified lower price limit.
Side	*FIXED (FOUND)* and *UNFIXED (UNFOUND)*: Offers have to specify which side they are for. On Maturation, owner of the *FIXED* position gets payout on a *true* statement and owner of *UNFIXED* on *false*.

The proposed market design has the potential to change the dynamics of open source software production. Before this impact can be realized, more research is needed to better understand the incentives created by the market place, the way users would interact with it, and how adjusting features can impact market behavior. To this end, we ran a live usability study to understand how users might perceive the platform and how trading develops. We designed an experiment that leads participants through a three-part process that increases in complexity of trading behavior. We found no significant barriers to adoption provided the trading rules are carefully explained. A full description of the usability study is outside the scope of this article.

Building on this pilot study, we are currently working on a controlled lab experiment to evaluate how participants in the market place use price signals in their decision-making process. Participants will be provided with different signals, including aggregate market information and open source project health metrics. We would like to know how signals from the market place compare to competing signals in guiding decisions regarding what projects and issues to address.

Discussion

Peer production is one of three ways of organizing human economic activity alongside markets and firms [6]. The trading market introduced in this article sits in the space between peer production and markets. Our work has several implications for market design, software security, and the dynamics of peer production. Ultimately, a real-world deployment would provide the data and feedback to evolve the design and functionality of our platform. In what follows, we discuss these implications, highlight the challenges ahead in applying this work to a more practical context, and describe directions for future research.

Design considerations

We assume that workers, taking the *FIXED* side of a contract, can estimate their time and cost for completing a task. However, estimating the effort required to complete a task is a nontrivial problem. Various factors can cause prior estimates to be false and require an update as new information about the task comes to light. An example is when the code base of a software changes from the time a worker accepts a *FIXED* offer to the time the worker proposes a solution. This may result in unintended consequences such as merge conflicts that are hard to resolve, the proposed fix no longer satisfying all test cases, or the fix breaking down altogether. What recourse does the market permit in these cases? One option is to decide that the additional cost is too high and forego the indemnity. A second option is to renegotiate the price by posting a new offer to do work for a higher price. A third option is to resell the *FIXED* position on the contract to someone else who might be able to fix it in time.

A related question is what steps can the funder of a contract take if he realizes that the worker will not be able to complete the task by the contract maturation date. If so, the funder will receive the indemnity at maturation as compensation for not receiving a fix. Nonetheless, delivery of work on time is important in many settings. For instance, the fix might be for a time-critical vulnerability or the funder might need this fix for another investment. The funder might try to attract other talent to complete the work by creating a new contract (offer to buy *UNFIXED*), possibly with a higher payout. The funder always has the option of bypassing the marketplace and simply investing his own time or directing an employee to assist in timely completion.

The design of the trading market relies on checking the status of issues in an issue tracker on the contract maturation date. In current practice, an issue's status is determined (e.g., fixed or unfixed) once reviewers have approved the fix and the maintainers of the open source repository merge this new code into the main code base. How do we then incentivize and compensate reviewers and maintainers? How do we handle collusion? This is work in progress and we are considering a few different solutions, such as paying a percentage of each contract to reviewers and maintainers, or to make review and maintenance a task for which contracts can be opened.

A further issue is the quality of the review and maintenance work. All open source project pipelines must contend with reviewers and maintainers and, therefore, are necessarily affected by the skill and speed with which they complete tasks. Our market design does not change the open source process – rather, it proposes a new mechanism that works within the same parameters. If reviewers and maintainers consistently fail to review issues within an expected time window, this might affect the price for issues on this project. A norm may emerge that developers allow for enough time for reviews. Creating the right incentive and reputation mechanisms for reviewers and maintainers and allowing them to partake of market forces seem to be important considerations for the smooth running of the platform.

An interesting question is how to verify partial work when one party wants to resell their position to another? At present we have no verification mechanisms built in for partial work. In Example 3.2, ascertaining the value of partial work is a transaction cost for Charles when he takes over from Beth. Charles must evaluate what he is buying and how much to pay for it.

In designing this market, we must consider whether it introduces adverse incentives. What types of market manipulations might arise, for example, might a developer introduce a bug with the intention to make money from fixing it? Although this scenario seems plausible, only a real-world deployment can ascertain whether such behavior might become widespread. On the other hand, repeated manipulations by the same developer would likely be noticed by the peer production community. Developers' reputations are important signals and can influence trading terms. Adding a reputation system seems like a natural next step worth exploring.

The nature of software is that it is in constant evolution as new technologies emerge, licenses expire, use cases change, which all require ongoing maintenance. That said, it is indeed the case that fixing bugs can introduce new bugs. Our market design is flexible and can be coupled with other mechanisms that address incentivization problems in software economies [17]. A reputation system can also alleviate this issue. If a developer is known to repeatedly introduce regressions, they could lose their standing within the project community and thus may not have their work accepted in the future. Open source has developed several methods for addressing regressions, including automated unit testing and continuous integration. Price signals, project metrics, and other data, that are a by-product of market activity, can reflect the quality of work on the platform thereby guiding users, investors, and other market participants to respond accordingly. Designing incentives to prevent the introduction of new bugs within our platform involves understanding how project processes interact with market processes. This remains an interesting open problem.

Note that the marketplace permits certain types of speculative activity and treats this as a feature. As mentioned in section "Indemnity," a prediction that something cannot happen is handled the same way as an incentive to make it happen. Speculators who want to profit from "inside" knowledge of the software ecosystem without contributing to development inject additional funds into the market, creating a pool of wealth that may enable a breakthrough.

A challenge of the contract design is its versatile application. The marketplace relies on the status of issues in an issue tracker to determine payout. We make a simplifying assumption – that the contract specifies well-defined criteria that must be met for the issue to be considered fixed. In practice, however, different applications carry different levels of uncertainty and well-defined tests may not exist for certain types of tasks. For example, fixing an identified bug is a well-defined task. Bug descriptions include the steps to reproduce the bug and can also include failing tests that any fix must pass. A fix might introduce other bugs or affect other components. While there may be variance in the quality of a potential fix, the basic criteria for a fix to be considered correct are clear. The development of new features carries greater uncertainty. Verification of the correctness of a feature will have more dimensions to satisfy. Not all new features may be tradable because they may not be sufficiently well-defined for traders to understand how the corresponding issues will be resolved. The volume of trading on a feature could be an interesting measure of how well understood the feature is.

Of still greater uncertainty is the discovery of vulnerabilities or bugs. An offer that rewards the discovery of a vulnerability in a software component will need to determine price as well as criteria for payout. This may not be possible because, until the vulnerability is disclosed (or exploited), it is hard to know what impact it may have. There are a few ways this situation could be handled on our platform. With a more expressive bidding language, a funder might make a tiered offer that pays out different amounts based on different standards being met. These standards might classify externally observable outcomes in terms of levels of importance (e.g. a defect that spontaneously deletes all data is relatively more serious than a defect that disrupts workflow).

Alternately, an anonymous hacker could take the *FOUND* position and place an offer on the marketplace, seeking a funder to take the *UNFOUND* position. The contract would specify some guarantees of the vulnerability's impact, with payment being conditional on these criteria being met once the vulnerability is disclosed. Further, by having a large indemnity the hacker signals that they are certain to find such a vulnerability by the maturation date. In addition, the futures market, in the case of an undiscovered vulnerability or bug, could function more like a prediction market on whether a particular bug bounty program will pay out within a certain date range. The bounty program might take the *FOUND* position to hedge and be able to pay out more to the bug finder(s), effectively subsidizing the program with money from the funders holding the *UNFOUND* position.

In building the user interface for a pilot study, we realized that a universal user interface would only be useful to experts and confuse other user groups. For user groups who have only one reason for trading on the market, we concluded that specialized user interfaces are needed. For example, a developer who wants to find an issue that would pay well, would only use a subset of all features and further would benefit from a design that supports the search for valuable issues while hiding excess features. Similarly, a project manager who wants to incentivize developers to work on specific issues uses a different subset of the contact design features and uses a different workflow, which would best be supported by a specialized user interface.

Security considerations

Agile development and continuous integration processes have found widespread adoption. The open source development model is in line with the agile approach, which has several security considerations. The incremental and iterative nature of the agile approach makes incorporating secure software engineering practices into the continuous development cycle challenging [70, 71]. Techniques to integrate security into agile practices include implementing user stories, which may involve changes to existing components and their security assurances, and modeling threats based on developers' perception of the likelihood of a threat and customers' perception of its impact [70]. Results of a study of security practices in public organizations suggest that risk perception and analysis must be improved and stakeholders in a project must exercise greater cooperation [71]. A concern regarding the continuous development paradigm is focusing attention on features that are used the most, which requires "feature analytics" to provide information for decision makers [72]. Fitzgerald and Stol [72] describe a quote from a manager of a software product who stated that "he knew that half of the features being offered were not used by any customers, but the trouble was that he did not know *which* half" [72, p. 185, emphasis in original].

We argue that a trading market for software tasks can complement software security practices in several key ways (see Fig. 3). The market mechanism allows us to elicit and aggregate continuous and real-time information on signaling, event likelihood, and usage. Prices reveal user valuations and help to focus attention on issues that are most important to users, thereby enabling more efficient allocation of development resources. Our platform is designed for collaboration on both sides, allowing for collaborative funding as well as collaborative work. Contracts opened and closed on the market reveal the most compelling user stories. This facilitates cooperation and shared investment among the various stakeholders. Trading behavior on the market can reveal users' perception of potential risks and can provide a means to address or otherwise hedge against these risks.

The marketplace distributes wealth among all participants in the software ecosystem. It can be used to incentivize different types of software tasks, including tasks that typically get less attention, such as testing, triage, and documentation. As a result the overall quality of software development is improved. Quality issues, if left untended, can "become" security issues further in the pipeline (see "Introduction" section). Current costs of cybersecurity run into the billions of US dollars. It would seem that bug bounty and other programs that operate downstream in the software development process (typically once the software is deployed to users) would incur higher cost than a platform upstream that can catch issues early while the software is still being built. Furthermore, by capturing a portion of the spending on cybersecurity, it may be possible to fund open source development that is currently not funded.

The marketplace can also highlight those software projects that adhere to existing best practices. For instance, projects that follow secure software development practices (e.g. earned a CII Best Practices Badge[3]) may have a different price equilibrium from non-secure software projects. Market participants may be willing to pay more for open source projects that are known to follow these best practices and are thus more likely to be secure.

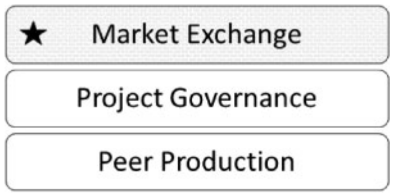


Figure 3. Security of Open Source Software can be implemented in all layers. At the lowest layer are the tools of generic peer production, e.g. with GitHub's warning for outdated dependencies. At the next layer are governance decisions where secure engineering practices are enforced, e.g. through the CII Best Practices Badge. Our market place sits on top of that and provides additional incentives to guide attention of developers and users.

3 The CII Best Practices badge provides a checklist with secure practices. Open source projects earn a badge by verifying that they are following these best practices. The badge serves as a stamp of approval of completed audit. https://bestpractices.coreinfrastructure.org/en.

Open source considerations

Our goal is to incentivize behavior without changing the development workflow or forcing open source projects to adopt a different system for doing work. We integrate our marketplace with existing issue trackers. Our market is designed to support collaboration. This has several implications.

Our market design maintains the autonomy of open source project members, who can ignore the market place or respond to it at their own choice. Open source project maintainers make decisions on whether to close an issue or not. The reputation and reliability of a maintainer may be a factor for trading decisions on the marketplace. Traders, in reducing their risk, may avoid projects with maintainers who might have garnered a low reputation.

Reopening issues is a natural process in open source development. The status of the issue for payout on the futures market only matters at the time of contract maturation. If an issue is closed but reopened after contract maturation, the worker who already received the payout will keep it. If the issue is reopened before the contract matures and not closed again in time, then the funder receives the payout.

Disputes amongst market participants (if they reveal their identities) can flow over into open source projects. Similarly, disputes in open source projects will have an influence on trading behavior in the marketplace. We take a hands-off approach to disputes. Maintainers and developers have to figure out the technical details and solve any issues, not unlike the current practice in open source development and regardless of the marketplace. However, the futures market can increase incentives for collaborating and resolving disputes before contracts mature.

Workers who feel they might not be able to complete the work before the contract maturation date can sell their side of the contract, get compensated for partial work done, and exit the contract in a transparent way. Alternately, if an issue is not fixed satisfactorily, it will not be closed and the funder gets the contract payout at maturation. The funder then has the option to create a new offer and provide potentially different and more favorable terms, such as greater time to complete the task or a higher reward. The simplifying assumption in section "Our approach," requiring contracts to specify clear verification tests, preempts disputes where the correctness or quality of work comes into doubt after the worker has been paid.

In general, a maintainer who operates fairly, in keeping with industry norms, cultivates a good reputation and environment that attracts and retains contributors. This results in greater adoption and growth of the project. Poorly managed, contentious projects reflect on the reputation of the maintainer and drive away the contribution and resources needed for a project to thrive. Developers have the recourse of forking their own copy of the open source code and evolving that copy, if disputes become excessive. However, fragmenting projects is not a desirable outcome because it can create competing forks with divided resources [73]. In practice, the marketplace will need to contend with various types of disputes. Future work might study dispute resolution mechanisms such as incentivizing testing and incorporating arbitrators.

A market place that can connect users who are willing to pay directly to developers who are willing to work has the potential to allow users greater influence over software development. Further, it can enable more demographics to participate in open source. Participants can earn an income for working in open source without having to be employed by a company. This could improve the diversity within open source projects. The market place has the potential to distribute the wealth created in open source projects differently.

The introduction of external rewards into open source is opposed by some [74]. The fear is that the community spirit will be lost when intrinsic motivation is replaced by paid work. Price signals could lead contributors to compare the market value of different projects and leave projects that are perceived to be worth less to join higher valued projects. However, a similar trend of developers moving to more popular projects already exists [75]. Many open source projects are not actively under development because developers have lost interest. Our market place could reinvigorate these projects that would otherwise have little or no attention from developers or organizations.

Financial and nonfinancial incentives currently co-exist in open source software production. Major open source projects benefit from corporate sponsorship. Many open source developers are paid employees of companies that use open source in their software and encourage their employees to contribute to open source development. Moreover, open source bounty platforms have existed for some time. Nonetheless, how intrinsic motivation is affected by the introduction of external rewards would be useful to understand. Another interesting question is how communities might react when they perceive that someone else is earning money from their work, when this is more directly exposed in a market.

Future work

The research initiated in this article raises several interesting questions that we plan to investigate going forward. A practical question is how to ensure market liquidity and avoid a thinly traded market. A usability question is how to make the market design intuitive for open source developers who have no background in futures markets, which is important for widespread adoption. A social question is how to ensure that everyone gets fairly compensated, including reviewers of code, who do not have the information advantage that developers have because the review comes late in the development life-cycle. Other promising lines of research that stem from this work are as follows:

Characteristics of the model: An insightful direction for future work is to characterize the futures trading model presented in this article. How does our model compare to other market models? Does a trading market enable new types of tasks that were otherwise not addressed by existing platforms? Alternately, does it allow us to perform old tasks in new ways? What is a measure of "task complexity" and how does it help to characterize different models of accomplishing work?

Consequences of the model: Our platform introduces price signals into the peer production of software. It would be interesting to study the consequences of introducing price signals in peer production and explore the connection between markets and peer production. How does this market design impact information sharing and collaboration? Does this lead to more efficient resource allocation, better quality output, and result in higher utility for all participants? How are equilibria in this market characterized? What opportunities for manipulation exist and how can they be addressed?

Extensions of the model: In future work, we plan to explore the futures market features and mechanisms in more detail. We would also like to consider extensions to the contract design presented in this article. For instance, how can the expressive power of the contract language be improved? What kind of information might be inferred from the dynamics of contract creation and resolution? How might techniques from machine learning and financial theory be used to improve the performance of the market mechanism?

Security impact of the model: We hope to further investigate the interaction of our market mechanism with software security practices. What problems in software security might this market address?

Consequently, what types of security issues might require a different mechanism?

Conclusion

The main contribution of this article is a novel market design to incentivize secure software development in peer production communities, inspired by futures trading markets. The market connects users who are willing to pay directly to workers and willing to work through price signals. The core of our innovation lies in introducing price signals in such a way as to leverage and strengthen the successful qualities of peer production. Our design facilitates collaboration on both sides of the platform – thus multiple stakeholders can pool resources and share investment in collaborative funding, and multiple workers can draw on a diverse set of skills in collaborative work. By enabling developers to earn credit for partial work, our market incentivizes information sharing. The market treats a prediction that something cannot happen in the same way as an incentive to make it happen. Thus, in aggregate, the market creates a pool of wealth that can be captured by innovators.

We further contribute a proof-of-concept implementation of our innovation, which confirms the practicality of the trading market. The source code is publicly available under an open source license.[4] Preliminary simulation results demonstrate that the implementation works as expected and can be used for future experiments. The present article lays a foundation upon which future research may build. In ongoing work, we are designing experiments to work with real software project groups. Our goal is to run a series of experiments, simulated and real-world, to test various hypotheses about the characteristics of the system and to arrive at a deeper understanding of the impact of the incentives design.

Acknowledgements

We thank Daniel Arce, Rainer Bohme, Zvi Boshernitzan, Matt Germonprez, Margo Seltzer, the participants of the 17th Annual Workshop on the Economics of Information Security (WEIS) 2018, and the anonymous reviewers for valuable feedback.

Supplementary Data

Supplementary data is available at *Journal of Cybersecurity* online.

Conflict of interest statement. None declared.

Funding

This work was supported by Mozilla through the open source experiments program. The project was also supported by the Alfred P. Sloan Foundation Digital Technology grant on Open Source Health and Sustainability (https://sloan.org/grant-detail/8434). Funding for open access publishing was provided by the University of Nebraska at Omaha Libraries' Open Access Fund.

References

[1]. NIST. The economic impacts of inadequate infrastructure for software testing. Planning report 02-3, 2002. http://www.nist.gov/director/planning/upload/report02-3.pdf (13 September 2019, date last accessed).

[2]. Dreyer P, Jones T, Klima K, *et al. Estimating the Global Cost of Cyber Risk: Methodology and Examples*. Santa Monica, CA: RAND Corporation, 2018.

[3]. Nizovtsev D, Thursby M. Economic analysis of incentives to disclose software vulnerabilities. In: *Fourth Workshop on the Economics of Information Security*. Cambridge, MA, USA, 2005.

[4]. Anderson R. Why information security is hard – an economic perspective. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC), New Orleans, LA, 2001*. Washington, DC: IEEE Computer Society, 2001.

[5]. Eghbal N. *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure*. Ford Foundation, 2016.

[6]. Benkler Y. Coase's penguin, or, "linux and the nature of the firm". *Yale LJ* 2002;**112**:369–446.

[7]. Open Source Press Release. The linux foundation releases first-ever value of collaborative development report, 2015. https://www.linuxfoundation.org/press-release/the-linux-foundation-releases-first-ever-value-of-collaborative-development-report/.

[8]. Kooths S, Langenfurth M, Kalwey N. Open-source software - an economic assessment. *MICE Econ Res Stud* 2003;**4**:1–95.

[9]. Black Duck. *Open source security and risk analysis*, 2018 https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2018-ossra.pdf (13 September 2019, date last accessed).

[10]. ImageTragick. *Imagemagick vulnerability*, 2016. https://imagetragick.com/.

[11]. U.S. DoD News Release. Statement by pentagon press secretary peter cook on dod's partnership with hackerone on the hack the pentagon security initiative, 2016 http://www.defense.gov/News/News-Releases/News-Release-View/Article/709818/statement-by-pentagon-press-secretary-peter-cook-on-dods-partnership-with-hacke.

[12]. Bountysource Inc. Bountysource inc. website, 2013. https://www.bountysource.com.

[13]. TopCoder Inc. Topcoder inc. website, 2001. https://www.topcoder.com.

[14]. Kickstarter. Kickstarter website, 2009. https://www.kickstarter.com.

[15]. Bacon DF, Chen Y, Parkes DC *et al.* A market-based approach to software evolution. In *Proc. 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, Orlando, Florida, USA*. OOPSLA '09, New York: ACM Press, 2009, 973–80.

[16]. Bacon DF, Bokelberg E, Chen Y, *et al.* Software economies. In: *Proc. FSE/SDP Workshop on Future of Software Engineering Research, Santa Fe, New Mexico, USA*. New York, NY: ACM, 2010.

[17]. Rao M, Bacon David F, Parkes David C, Seltzer M, Incentivizing deep fixes in software economies. *IEEE Transactions on Software Engineering*, 2018a. DOI: 10.1109/TSE.2018.2842188.

[18]. Malvika R. Incentives design in the presence of externalities. Ph.D. Dissertation, Harvard University, 2015.

[19]. Rainer B. Vulnerability markets - what is the economic value of a zero-day exploit? In: *Proc. of 22C3: Private Investigations, Berlin, Germany*, 2005.

[20]. Bohme R. A comparison of market approaches to software vulnerability disclosure. In: Muller G (ed.), *Emerging Trends in Information and Communication Security, LNCS 3995*, Berlin Heidelberg: Springer-Verlag, 2006, 298–311.

[21]. Stuart ES. How to buy better testing: using competition to get the most security and robustness for your dollar. In Davida G, Frankel Y, Rees O (eds), *Infrastructure Security. InfraSec 2002, Bristol, UK. Lecture Notes in Computer Science*, vol 2437. Berlin, Heidelberg: Springer, 2002.

[22]. Ozment A. Bug auctions: vulnerability markets reconsidered. In: *Third Workshop on the Economics of Information Security*. Minneapolis, MN, USA, 2004.

[23]. Anderson R, Moore T. The economics of information security. *Science* 2006;**314**:610–13.

[24]. Cofone I. The value of privacy: keeping the money where the mouth is. In: *14th Annual Workshop on the Economics of Information Security (WEIS)*. The Netherlands: Delft University of Technology, 2015.

[25]. Kannan K, Telang R. Market for software vulnerabilities? Think again. *Manag Sci* 2005;**51**:726–40.

4 See https://github.com/bugmark/exchange.

[26]. Laube S, Bohme R. The economics of mandatory security breach reporting to authorities. In: *14th Annual Workshop on the Economics of Information Security (WEIS)*. The Netherlands: Delft University of Technology, 2015.

[27]. Maillart T, Zhao M, Grossklags J. *et al*. Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty programs. *J Cybersecur* 2017;**3**:81–90.

[28]. Schechter SE. Toward econometric models of the security risk from remote attack. *IEEE Security Privacy* 2005;**1**:40–44.

[29]. Hosseini H, Nguyen R, Godfrey MW, A market-based bug allocation mechanism using predictive bug lifetime. In *Proc. 16th European Conference on Software Maintenance and Re-engineering (CSMR), Szeged, Hungary, 2012*. Washington, DC: IEEE Computer Society, 2012.

[30]. Moldovanu B, Sela A. The optimal allocation of prizes in contests. *Am Econ Rev* 2001;**91**:542–58.

[31]. Moldovanu B, Sela A. Contest architecture. *J Econ Theory* 2006;**126**: 70–97.

[32]. Tullock G. Efficient rent seeking. In: Buchanan JM, Tollison RD, Tullock G (eds), *Towards a Theory of the Rent Seeking Society*. Texas: A&M University Press, 2001, 97–112.

[33]. Archak N, Sundararajan A. Optimal design of crowdsourcing contests. In: *Proceedings of the 30th International Conference on Information Systems (ICIS)*. Phoenix, Arizona, USA, 2009.

[34]. Chawla S, Hartline JD, Sivan B. Optimal crowdsourcing contests. In: *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms, Kyoto, Japan, 2012*. SODA '12, Philadelphia, PA: Society for Industrial and Applied Mathematics, 2012.

[35]. DiPalantino D, Vojnovic M. Crowdsourcing and all-pay auctions. In: *Proceedings of the 10th ACM Conference on Electronic Commerce, Stanford, CA, USA, 2009*. EC '09, New York, NY: ACM, 2009, 119–28.

[36]. Gneiting T, Raftery AE. Strictly proper scoring rules, prediction, and estimation. *J Am Stat Assoc* 2007;**102**:359–78.

[37]. Hanson RD. Logarithmic market scoring rules for modular combinatorial information aggregation. *J Predict Markets* 2007;**1**:1–15.

[38]. Lambert N, Pennock DM, Shoham Y. Eliciting properties of probability distributions. In: *Proceedings of the ACM Conference on Electronic Commerce, Chicago, IL, USA, 2008*. EC'08, New York, NY: ACM, 2008, 129–38.

[39]. Murphy AH., Winkler RL. Probability forecasting in meteorology. *J Am Stat Assoc* 1984;**79**:489–500.

[40]. Savage LJ. Elicitation of personal probabilities and expectations. *J Am Stat Assoc* 1971;**66**:783–801.

[41]. Boudreau KJ, Lacetera N, Lakhani KR.. Parallel search, incentives and problem type: revisiting the competition and innovation link. Working paper, no. 09-041. Technical report. Harvard Business School, *September* 2008.

[42]. Lakhani KR, Panetta JA. The principles of distributed innovation. *Innov Technol Gov Glob* 2007;**2**:97–112.

[43]. Baldwin CY, Clark KB. *The Power of Modularity. Vol.* **1**, *Design Rules*. Cambridge, MA: MIT Press, 2000.

[44]. MacCormack A, Rusnak J, Baldwin CY. Exploring the structure of complex software designs: an empirical study of open source and proprietary code. *Manag Sci* 2006;**52**:1015.

[45]. Lerner J, Tirole J. Some simple economics of open source. *J Ind Econ* 2003;**50**:197–234.

[46]. Johnson J. Open source software: private provision of a public good. *J Econ Manag Strategy* 2002;**11**:637–62.

[47]. Athey S, Ellison G. Dynamics of open source movements. *J Econ Manag Strategy* 2014;**23**:294–316.

[48]. Roth AE, Peranson E. The redesign of the matching market for American physicians: some engineering aspects of economic design. *Am Econ Rev* 1999;**89**:748–80.

[49]. Wilson R. Architecture of power markets. *Econometrica* 2002;**70**: 1299–340.

[50]. Leyton-Brown K, Milgrom P, Segal I. Economics and computer science of a radio spectrum reallocation. *Proc Natl Acad Sci* 2017;**114**:7202–09.

[51]. Budish E, Cramton P, Shim J. The high-frequency trading arms race: frequent batch auctions as a market design response. *Q J Econ* 2015;**130**: 1547–621.

[52]. Immorlica N, Bergquist L, Lucier B, Quinn J, McIntosh C, Newman N, Leyton-Brown K, Ssekibuule R. Designing and evolving an electronic agricultural marketplace in Uganda. In: *Proceedings of the ACM SIGCAS Conference on Computing and Sustainable Societies (COMPASS), Menlo Park and San Jose, CA, 2018*. New York, NY: ACM, 2018.

[53]. Kominers SD, Teytelboym A, Crawford VP. An invitation to market design. *Oxf Rev Econ Policy* 2017;**33**:541–71.

[54]. Rao Georg M, Link JP, Marti D, Leak A, *et al*. A trading market to incentivize secure software. In: *17th Annual Workshop on the Economics of Information Security (WEIS) Proceedings*, Innsbruck, Austria, 2018b. https://weis2018.econinfosec.org/wp-content/uploads/sites/5/2016/09/WEIS_2018_paper_27.pdf.

[55]. Bountify Inc. Bountify inc. website, 2012. https://bountify.co.

[56]. BugCrowd Inc. Bugcrowd inc. website, 2012. https://bugcrowd.com.

[57]. Hackerone. Hackerone website, 2012. https://www.hackerone.com.

[58]. Mao K, Capra L, Harman M, *et al*. A survey of the use of crowdsourcing in software engineering. *J Syst Software* 2017;**126**:57–84.

[59]. Howison J, Crowston K. Collaboration through open superposition: a theory of the open source way. *MIS Q* 2014;**38**:29–50.

[60]. Tabarrok A. The private provision of public goods via dominant assurance contracts. *Public Choice* 1998;**96**:345–62.

[61]. Krishna V. *Auction Theory*, 2nd edn. Academic Press, 2010.

[62]. Faraj S., Kudaravalli S., Wasko M. Leading collaboration in online communities. *MIS Q* 2015;**39**:393–412.

[63]. Milgrom PR. Auction theory. In: Bewley T. (ed.), *Advances in Economic Theory: Fifth World Congress*. London: Cambridge University Press, 1987, 1–32.

[64]. Kalagnanam J, Parkes DC. Auctions, bidding and exchange design. In: Simchi-Levi D, Wu SD, Shen Z. (eds), *Handbook of Quantitative Supply Chain Analysis: Modeling in the E-Business Era*. Boston: Kluwer, 2004, 143–212.

[65]. Bichler M, Davenport A, Hohner G. *et al*. Industrial procurement auctions. In: Cramton P., Shoham Y., Steinberg R. (eds), *Combinatorial Auctions*. Cambridge, MA: The MIT Press, 2005, 593–612.

[66]. Allen JL. Derivatives clearinghouses and systemic risk: a bankruptcy and Dodd-Frank analysis. *Stanford L Rev* 2012;**64**:1079–108.

[67]. Carlton DW. Futures markets: their purpose, their history, their growth, their successes and failures. *J Futures Markets (Pre-1986); New York* 1984;**4**:237.

[68]. Hudson SE, Mankoff J. Concepts, values, and methods for technical human computer interaction research. In: Olson JS, Kellogg WA (eds), *Ways of Knowing in HCI*. New York, NY: Springer, 2014, 69–93.

[69]. Ren Y, Kraut RE. Agent based modeling to inform the design of multi-user systems. In: Olson JS, Kellogg WA. (eds), *Ways of Knowing in HCI*. New York, NY: Springer, 2014, 395–419.

[70]. Othmane LB, Angin P, Weffers H. *et al*. Extending the agile development process to develop acceptably secure software. *IEEE Trans Depend Secure Comput* 2014;**11**:497–509.

[71]. Tondel IA, Jaatun MG, Soares Cruzes D. *et al*. Risk centric activities in secure software development in public organisations. *Int J Secure Softw Eng* 2017;**8**:1–30.

[72]. Fitzgerald B, Stol K-J. Continuous software engineering: a roadmap and agenda. *J Syst Softw* 2017;**123**:176–89.

[73]. Karl, F. *Producing Open Source Software: How to Run a Successful Free Software Project*, 2015. https://producingoss.com/ (13 September 2019, date last accessed).

[74]. Hansson DH, *The perils of mixing open source and money (DHH)*, 2013. http://david.heinemeierhansson.com/2013/the-perils-of-mixing-open-source-and-money.html.

[75]. Beecher K, Capiluppi A, Boldyreff C. Identifying exogenous drivers and evolutionary stages in FLOSS projects. *J Syst Softw* 2009;**82**:739–50.