A Trading Market to Incentivize Secure Software

Malvika Rao
Incentives Research
Ottawa, ON, Canada
malvikar@gmail.com

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

Don Marti
Mozilla
Mountain View, CA, USA
dmarti@mozilla.com

Andy Leak
Mountain View Smart Contracts
Mountain View, CA, USA
andy@r210.com

Rich Bodo
Mountain View Smart Contracts
Mountain View, CA, USA
richbodo@gmail.com

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 paper 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 paper, which contributes to peer production, software development practices, and incentives design.

CCS CONCEPTS

• **Security and privacy → Economics of security and privacy**; • **Human-centered computing → Open source software**; • **Software and its engineering → Open source model**;

KEYWORDS

Open Source, Futures Market, New Collaboration Model, Economics of Vulnerabilities

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

WEIS2018, June 18-19, 2018, Innsbruck, Austria

© 2018 Copyright held by the owner/author(s).

1 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 U.S. economy of software errors alone at approximately $59.5 billion [41]. More recent figures on cyber risk paint a graver picture. Global costs of cyber crime have been estimate to lie between $799 billion and $22.5 trillion (1.1% to 32.4% of global GDP) [18]. 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 [42]. Anderson argues that information insecurity is partly due to a failure in the design of incentives [1].

Open source software forms much of our digital infrastructure and has enabled the boom in startups [19]. 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 [9]. Peer production has successfully tackled complex, uncertain projects, underlying billions of dollars in open source software production [43]. A recent study, however, points out that this digital infrastructure is increasingly under strain [19]. Escalating demand and a lack of adequate resources has resulted in security breaches and service errors [19]. Earlier economics research foresaw these types of real-world problems. Kooths *et al.* [30] assert that the absence of price signals in open source development means that users' valuations remain unknown. 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 paper we make progress towards answering 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 so forth.

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 an inconvenience on desktop, become security risks when ImageMagick is integrated into a web application [26]. 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. 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 [13]), crowdsourcing contests (e.g., Topcoder [52]), and crowdfunding sites (e.g., Kickstarter [29]). However these market mechanisms have limitations that we address with a novel market design based on trading futures contracts. This is the first work to apply the concepts of futures trading to a software market setting.

[1]A recent audit found open source components in 96% of applications scanned [10].

2 RELATED WORK

Research on software economies has advocated a market-based approach where supply and demand determine the allocation of work and influence the evolution of the system [6, 7]. An equilibrium in the software economy is one where all issues for which there is enough value have been addressed. Rao et al. [46] 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 [45]. The present paper 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 in incentivizing vulnerability-reporting have been considered. Schechter [49] presents a vulnerability market where the first person to disclose a particular security flaw is offered a reward. The reward increases in value if no one steps forward to claim it. The product can be considered secure enough to protect information worth the combined value of all such rewards offered at a particular moment in time. Ozment [44] maps this type of vulnerability market to an open first price ascending auction. Although vulnerability markets as well as existing bug bounty programs (e.g., the Mozilla security bug bounty, and the "Hack the Pentagon" bug bounty program [54]) provide incentives to report flaws, users' valuations for fixes are not captured in these systems.

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 [2, 3, 16, 28, 33, 36, 50]. However, these works do not consider the design of a futures trading market. Hosseini et al. [23] 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 [4, 15, 17, 38, 39, 53]. 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 [20, 22, 32, 40, 48]. 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 [11, 31], and the design and modularity of software [8, 35].

Much software development is organized in peer production communities which have their own economic rules. Benkler [9] 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 various creative 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, including intrinsic as well as extrinsic rewards, because a clear measure of effort can no longer be determined.

Lerner and Tirole [34] study four cases of peer production and propose economic models to explain the motivations of open source contributors. Johnson [27] 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. The paper establishes technical conditions that characterize when a developer will choose to contribute. Athey and Ellison [5] 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, depending on the model parameters, the dynamic system can reach one of two steady-states from almost any starting conditions: a zero quality and zero altruism state and another state where the zero quality, zero altruism as well as positive quality, positive altruism states can exist.

Current market-based platforms

Examples of market-based platforms for software development include Bountysource, which is a funding platform for open-source software [13]. Here, users post bounties or rewards on issues they want 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 two 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 [29].

Topcoder is based on crowdsourcing contests and has over a million active members [52]. 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 [12] 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 [14] 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 Hackerone [21]. 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 [37]. Our approach addresses the same issues in a new way as we show in the next section.

3 OUR APPROACH

Recall that the research question that we consider in this paper is:

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

To this end, we propose a futures trading market for eliciting information and incentivizing tasks. The next example illustrates a simple case of how such a market might work.

Example 3.1. 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 two 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 two 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, 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 which can be verified in an issue tracker. On maturation, the contract pays out to the owner of the position corresponding to the issue's status in the issue tracker.

Comparison to existing markets

Although inspired by existing market mechanisms, such as open source bounty systems and prediction markets, 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 [24]. Further, 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 also have extra transaction costs of 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 (for example, 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.

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.

Why do we call it a futures trading market? Because the contracts have similarities to futures contracts. 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 closed or vulnerability found) or receives his deposit and that of the counter-party.

Design features

Before describing implementation details we discuss some key features in our design of a futures trading market.

Partial work. Trading behavior allowed in a futures market supports collaborative work, as the next example shows.

Example 3.2. 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.

Many problems can only be solved by drawing on different areas of expertise. For example, a bug may require

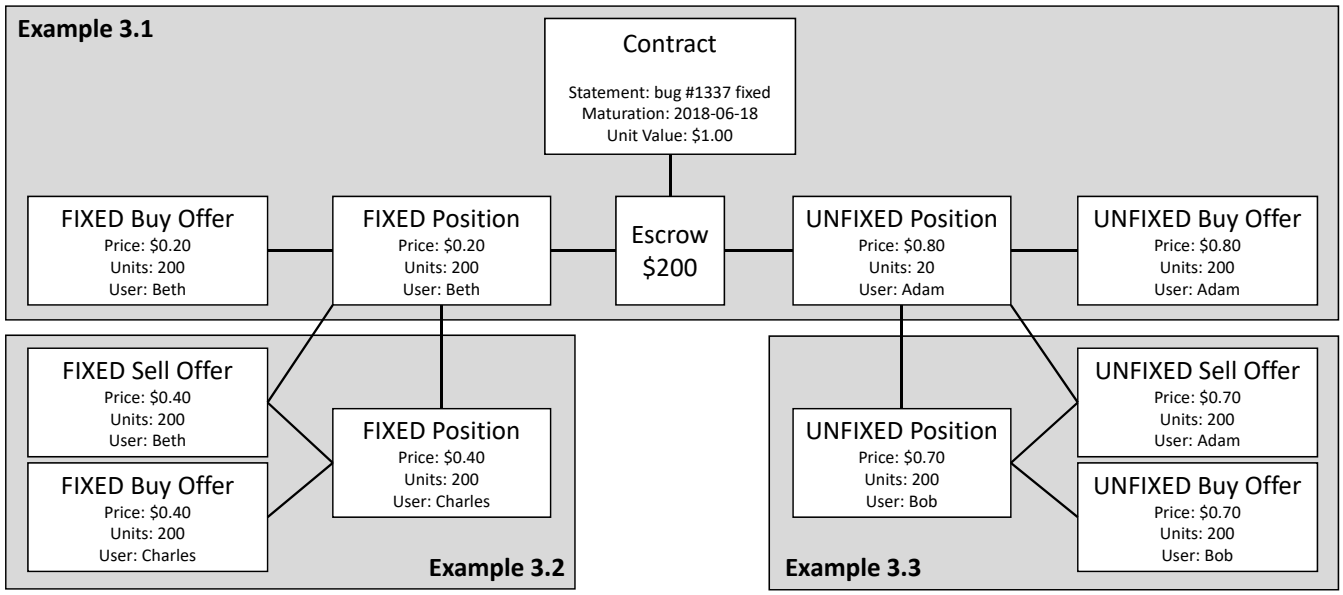

Figure 1: Graphical display of a futures contract, demonstrating how the elements of the contract fit together and evolve over time. A contract with its statement, maturation, and unit value make up the root. Underneath it, many escrows and positions can be created. Money is paid into escrow when fixed 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 below defines the contract elements in detail.

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 [24]. The design of our trading platform captures these types of dynamics. Because the market allows developers to earn credit for partial work, it incentivizes information sharing and facilitates collaboration.

The next example describes a case where selling happens for other reasons, such as if a user no longer wished to invest in the contract.

Example 3.3. 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. 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.

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 himself) might then create a new offer to pay for the completion of task *B*. Because he might not finish task *A* otherwise and lose his deposit, the new contract would have a maturation date no later than the date for task *A*.

Multiple contracts. A project maintainer might need different features coded for a single project. He may create a separate contract for each feature, and 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 person 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 escrow 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 the issue might already be marked as fixed in the issue tracker. 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. Further experiments are needed to understand the behavior of the market in this scenario.

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.

Indemnity. Our design allows for the possibility of an indemnity being paid if a request is not fulfilled. This is not unlike 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 [51].

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. 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 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 is fueling the community processes can be rewarded.

Moreover, the decoupling can lead to new workplaces where an entrepreneur hires developers to work on issues and get them closed. The entrepreneur trades on the futures market, following price signals, and assigns developers to complete the work. The developers are employed and have the benefits of a regular income; The entrepreneur organizes the human resources to the issues that are valued the most on the market place to maximize profit.

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.

Anonymity. Market participants are anonymous. Issues must be identified in an issue tracker. The market must be able to transfer payments between escrow and participants' accounts. The identities of participants are not needed.

Enhancing existing open source practices. 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. This has several implications.

First, open source project maintainers have full autonomy of work and make decisions on whether to close a bug or not. The reputation and reliability of a maintainer may be a factor for trading decisions on the marketplace.

Second, reopening bugs is a natural process in open source development. The status of the bug for payout on the futures market only matters at the time of contract maturation. If a contract is closed but reopened later, the worker receives the payout. If the bug is reopened before the contract matures and not closed again in time, then the funder receives the payout.

Third, 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, regardless of the marketplace. However, the futures market can increase incentives for collaborating and resolving disputes before contracts mature.

Proof-of-concept implementation of the futures trading platform

We substantiate our design of a futures trading platform through a proof-of-concept implementation [25], which we

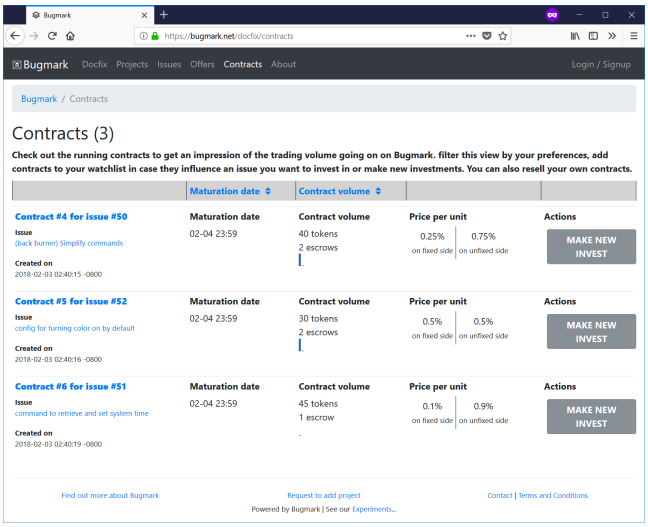

Figure 2: Screenshot of the Bugmark user interface.

call *Bugmark*. We use the Ethereum blockchain because it is the most mature for smart contracts. The benefits of the blockchain is anonymity of market participants, an immutable, decentralized, and audit-able record of transactions for trust, and automation through smart contracts. The user interface is written in HTML, CSS, and JavaScript (see Figure 2). The backend is written in Ruby and uses a PostgreSQL database for a local copy of events. We use an event stream architecture which 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 Figure 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

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*.

Table 1: Elements of a Bugmark futures contract.

contract the users want to buy and is ultimately recorded in positions. In other words, the volume is equal 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.

Figure 3: (a) The top diagram "Offer Prices & Volumes" shows over time that buy offers are created with varying prices in the range from 0.00 to 1.00 (green dots) and volumes in the range from 30 to 50 units (orange dots). (b) The bottom diagram "Market Overview" shows the number of open offers, number of contracts, and escrows over time. Contracts can have multiple escrows and the contract volume can expand over time as more escrows are added. The number of open offers decreases when offers get matched to form a new escrow.

'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.

Simulation

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 [47].

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.

For our first simulation[2], we demonstrate that our proof-of-concept implementation—Bugmark—works and that we are able to collect data to evaluate design changes. In this simulation, agents have no knowledge of the environment and randomly choose to create buy offers for fixed and unfixed positions (see Figure 3a). The agents use the full price range from 0.00 to 1.00 and choose volumes between 30 and 50 units. With these parameters, we have a good chance that offers match and enter into a contract to form positions. As positions are formed, the number of open contracts decreases (green line and orange line in Figure 3b). The blue line levels off at five contracts because the agents trade on only five issues with the same maturation date. As more offers match, however, the volume of positions on each contract increases, which is shown in the increasing orange line. With the infrastructure for agent-based-modeling built, we can, in future work, simulate varying agent behavior and investigate how incentives can impact the market behavior overall. Several questions we will investigate are described in the next section.

[2]The simulation code is available under an open source license online: https://github.com/bugmark/bmx_bots

4 RESEARCH DIRECTIONS

The research initiated in this paper 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 paper. 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? How can we turn those who might manipulate the market into constructive contributors?

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 paper. 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 be used to improve the performance of the market mechanism?

Security impact of the model. We hope to further investigate the application of our market mechanism to software security. How would our trading market interact with current secure software engineering practices? What problems in software security might this market address? Consequently, what types of security issues might require a different mechanism?

5 CONCLUSION

The main contribution of this paper is a novel market design to incentivize secure software development in peer production communities, based on futures trading markets. The market connects users who are willing to pay directly to workers who are 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. By enabling developers to earn credit for partial work, our market incentivizes information sharing and facilitates collaboration. 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.[3] Preliminary simulation results demonstrate that the implementation works as expected and can be used for future experiments. The present paper 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.

ACKNOWLEDGMENTS

We thank Zvi Boshernitzan, Matt Germonprez, and the anonymous reviewers for valuable feedback.

ORCID

Georg J.P. Link orcid.org/0000-0001-6769-7867

[3] https://github.com/bugmark/exchange

REFERENCES

[1] Ross Anderson. 2001. Why Information Security is Hard – An Economic Perspective. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*.
[2] Ross Anderson, Rainer Bohme, Richard Clayton, and Tyler Moore. 2008. Security Economics and the Internal Market. In *Technical Report, European Network and Information Security Agency (ENISA)*.
[3] Ross Anderson and Tyler Moore. 2006. The Economics of Information Security. *Science* 314 (2006), 610–613.
[4] Nikolay Archak and Arun Sundararajan. 2009. Optimal Design of Crowdsourcing Contests. In *International Conference on Information Systems (ICIS)*.
[5] Susan Athey and Glenn Ellison. 2014. Dynamics of Open Source Movements. *Journal of Economics and Management Strategy* 23, 2 (2014), 294–316.
[6] David F. Bacon, Eric Bokelberg, Yiling Chen, Ian A. Kash, David C. Parkes, Malvika Rao, and Manu Sridharan. 2010. Software Economies. In *Proc. FSE/SDP Workshop on Future of Software Engineering Research*.
[7] David F. Bacon, Yiling Chen, David C. Parkes, and Malvika Rao. 2009. A Market-Based Approach to Software Evolution. In *Proc. 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA '09)*.
[8] Carliss Y. Baldwin and Kim B. Clark. 2000. *The Power of Modularity. Vol. 1, Design Rules.* MIT Press.
[9] Yochai Benkler. 2002. Coase's penguin, or, "Linux and The Nature of the Firm". *Yale Law Journal* 112, 3 (2002), 369–446.
[10] Black Duck 2018. Open Source Security and Risk Analysis. *https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2018-ossra.pdf*. (2018).
[11] Kevin J. Boudreau, Nicola Lacetera, and Karim R. Lakhani. 2008. *Parallel Search, Incentives and Problem Type: Revisiting the Competition and Innovation Link. Working Paper, No. 09-041.* Technical Report. Harvard Business School.
[12] Bountify Inc. 2012. Bountify Inc. website. *https://bountify.co*. (2012).
[13] Bountysource Inc. 2013. Bountysource Inc. website. *https://www.bountysource.com*. (2013).
[14] BugCrowd Inc. 2012. BugCrowd Inc. website. *https://bugcrowd.com*. (2012).
[15] Shuchi Chawla, Jason D. Hartline, and Balasubramanian Sivan. 2012. Optimal Crowdsourcing Contests. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*.
[16] Ignacio Cofone. 2015. The Value of Privacy: Keeping the Money Where the Mouth is. In *14th Annual Workshop on the Economics of Information Security (WEIS)*.
[17] Dominic DiPalantino and Milan Vojnovic. 2009. Crowdsourcing and All-pay Auctions. In *Proceedings of the 10th ACM Conference on Electronic Commerce (EC '09)*.
[18] Paul Dreyer, Therese Jones, Kelly Klima, Jenny Oberholtzer, Aaron Strong, Jonathan William Welburn, and Zev Winkelman. 2018. *Estimating the Global Cost of Cyber Risk: Methodology and Examples.* RAND Corporation, Santa Monica, CA.
[19] Nadia Eghbal. 2016. *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure.* Ford Foundation.
[20] T. Gneiting and A. E. Raftery. 2007. Strictly proper scoring rules, prediction, and estimation. *J. Am. Stat. Assoc.* 102, 477 (2007), 359–378.
[21] Hackerone 2012. Hackerone website. *https://www.hackerone.com*. (2012).
[22] R. D. Hanson. 2007. Logarithmic market scoring rules for modular combinatorial information aggregation. *Journal of Prediction Markets* 1, 1 (2007), 1–15.

[23] Hadi Hosseini, Raymond Nguyen, and Michael W. Godfrey. 2012. A Market-Based Bug Allocation Mechanism Using Predictive Bug Life-Time. In *Proc. 16th European Conference on Software Maintenance and Re-engineering (CSMR)*.

[24] James Howison and Kevin Crowston. 2014. Collaboration Through Open Superposition: A Theory of the Open Source Way. *MIS Quarterly* 38, 1 (2014), 29–50.

[25] Scott E. Hudson and Jennifer Mankoff. 2014. Concepts, Values, and Methods for Technical Human Computer Interaction Research. In *Ways of Knowing in HCI*, Judith S. Olson and Wendy A. Kellogg (Eds.). Springer, New York, NY, 69–93. DOI: 10.1007/978-1-4939-0378-8_4.

[26] ImageTragick 2016. ImageMagick Vulnerability. *https://imagetragick.com/*. (2016).

[27] Justin Johnson. 2002. Open Source Software: Private Provision of a Public Good. *Journal of Economics and Management Strategy* 11, 4 (2002), 637–662.

[28] Karthik Kannan and Rahul Telang. 2005. Market for Software Vulnerabilities? Think Again. *Management Science* 51, 5 (2005), 726–740.

[29] Kickstarter 2009. Kickstarter website. *https://www.kickstarter.com*. (2009).

[30] Stefan Kooths, Markus Langenfurth, and Nadine Kalwey. 2003. Open-Source Software - An Economic Assessment. *MICE Economic Research Studies* 4 (2003).

[31] Karim R. Lakhani and Jill A. Panetta. 2007. The Principles of Distributed Innovation. *Innovations: Technology, Governance, Globalization* 2, 3 (2007), 97–112.

[32] N. Lambert, D. M. Pennock, and Y. Shoham. 2008. Eliciting properties of probability distributions. In *Proceedings of the ACM Conference on Electronic Commerce (EC'08)*. 129–138.

[33] Stefan Laube and Rainer Bohme. 2015. The Economics of Mandatory Security Breach Reporting to Authorities. In *14th Annual Workshop on the Economics of Information Security (WEIS)*.

[34] Josh Lerner and Jean Tirole. 2002. Some Simple Economics of Open Source. *The Journal of Industrial Economics* 50, 2 (2002), 197–234. http://www.jstor.org/stable/3569837

[35] Alan MacCormack, John Rusnak, and Carliss Y. Baldwin. 2006. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. *Management Science* 52, 7 (2006).

[36] Thomas Maillart, Mingyi Zhao, Jens Grossklags, and John Chuang. 2017. Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty programs. *Journal of Cybersecurity* 3, 2 (June 2017), 81–90. https://doi.org/10.1093/cybsec/tyx008

[37] Ke Mao, Licia Capra, Mark Harman, and Yue Jia. 2015. A Survey of the Use of Crowdsourcing in Software Engineering. *RN* 15 (2015).

[38] Benny Moldovanu and Aner Sela. 2001. The Optimal Allocation of Prizes in Contests. *American Economic Review* 91, 3 (2001), 542–558.

[39] Benny Moldovanu and Aner Sela. 2006. Contest Architecture. *Journal of Economic Theory* 126, 1 (2006), 70–97.

[40] A. H. Murphy and R. L. Winkler. 1984. Probability forecasting in meteorology. *J. Am. Stat. Assoc.* 79, 387 (1984), 489–500.

[41] NIST. 2002. *The economic impacts of inadequate infrastructure for software testing. Planning report 02-3*. http://www.nist.gov/director/planning/upload/report02-3.pdf.

[42] Dmitri Nizovtsev and Marie Thursby. 2005. Economic Analysis of Incentives to Disclose Software Vulnerabilities. In *In Fourth Workshop on the Economics of Information Security*.

[43] Open Source Press Release 2015. The Linux Foundation Releases First-Ever Value of Collaborative Development Report. https://www.linuxfoundation.org/press-release/the-linux-foundation-releases-first-ever-value-of-collaborative-development-report/. (2015).

[44] Andy Ozment. 2004. Bug Auctions: Vulnerability Markets Reconsidered. In *Third Workshop on the Economics of Information Security*.

[45] Malvika Rao. 2015. *Incentives Design in the Presence of Externalities*. PhD Dissertation, Harvard University.

[46] Malvika Rao, David C. Parkes, Margo Seltzer, and David F. Bacon. 2014. A Framework for Incentivizing Deep Fixes. In *Proc. AAAI Workshop on Incentives and Trust in E-Communities*.

[47] Yuqing Ren and Robert E. Kraut. 2014. Agent Based Modeling to Inform the Design of Multiuser Systems. In *Ways of Knowing in HCI*, Judith S. Olson and Wendy A. Kellogg (Eds.). Springer, New York, NY, 395–419. DOI: 10.1007/978-1-4939-0378-8_16.

[48] L. J. Savage. 1971. Elicitation of personal probabilities and expectations. *J. Am. Stat. Assoc.* 66, 336 (1971), 783–801.

[49] Stuart E. Schechter. 2002. How to Buy Better Testing: using competition to get the most security and robustness for your dollar. In *In Infrastructure Security Conference*.

[50] Stuart E. Schechter. 2005. Toward Econometric Models of the Security Risk from Remote Attack. *IEEE security and privacy* 1 (2005), 40–44.

[51] Alexander Tabarrok. 1998. The private provision of public goods via dominant assurance contracts. *Public Choice* 96 (1998), 345–362.

[52] TopCoder Inc. 2001. TopCoder Inc. website. *https://www.topcoder.com*. (2001).

[53] Gordon Tullock. 2001. Efficient Rent Seeking. *In James M. Buchanan, Robert D. Tollison, Gordon Tullock (eds.) Towards a Theory of the Rent Seeking Society* (2001), 97–112.

[54] U.S. DoD News Release 2016. Statement by Pentagon Press Secretary Peter Cook on DoD's Partnership with HackerOne on the Hack the Pentagon Security Initiative. http://www.defense.gov/News/News-Releases/News-Release-View/Article/709818/statement-by-pentagon-press-secretary-peter-cook-on-dods-partnership-with-hacke. (2016).