Toward Sustainable Communities with a Community Currency – A Study in Car Sharing

Keitaro Nakasai*, Yoshiharu Ikutani*, Daiki Takata*, Hideaki Hata*, Kenichi Matsumoto*
*Nara Institute of Science and Technology, Japan
{nakasai.keitaro.nc8, ikutani.yoshiharu.ip8, takata.daiki.ta3, hata, matumoto}@is.naist.jp

***Abstract*—We consider Free/libre and open source software (FLOSS) as a common pool resource (CPR). In economics, CPRs are frequently associated with markets, and it is reported that without appropriate agreement, monitoring and sanction, the resource will be overused. Toward building sustainable communities in FLOSS development, we first study our car-sharing experiment at NAIST, as a common pool resource management. We report the details of the car uses in our experiment, and describe the design of our new system to make better CPR management.**

I. Introduction

Free/libre and open source software (FLOSS) is characterized as a privately produced public good and a common pool resource (non-exclusive) [1]. Based on various internal and external motivations, developers engage in FLOSS [2], and FLOSS projects have a tremendous impact on our daily lives.

However, because of the nature of free and open source, and the lack of the view of the whole picture, there are several problems. *(i) Broken library dependencies*. Even though third-party library reuse is common practice, updating libraries is not as common. From the study of library migration in more than 4,600 `GitHub` software projects, Kula et al. reported that 81.5% of the studied software used outdated libraries [3]. From the survey, they also found that 69% of the answered developers claimed to be unaware of their vulnerable dependencies. *(ii) License violations*. From the study of the Android application market `F-Droid`, Mlouki et al. found license violations by developers in 17 out of 857 applications [4]. They also reported that many files are not licensed in their first release. *(iii) Ad hoc code reuses*. Copying software components and then maintaining them by a new owner is one type of code reuse, which is known as *clone-and-own* [5], [6]. In software product line engineering, the usage of this clone-and-own approach is discouraged, since it makes maintaining multiple product lines difficult. If changes are made in the original or copied components, they are not easily propagated. This is because of the ad hoc nature of component reusing. Developers working in different product lines do not know when clone-and-own operations occurred and where the cloned components are located. Although there are disadvantages, the clone-and-own approach is adopted in FLOSS projects as well as industrial software product lines due to its benefits, such as simplicity, availability, and independence of developers [5].

"The Tragedy of the Commons" is a well-known theory describing that, if all individuals maximize their gains from non-renewable resources, these resources will diminish [7]. Are the abovmentioned problems in FLOSS the results of the tragedy of the commons, which cannot be avoided? In this context, we consider human resources or effort as resources. It is known that such a dismal conclusion is not always true. It would be true only if the individuals using the commons are norm-free maximizers of immediate gains, who will not cooperate to overcome the common dilemmas they face [8].

Regarding common pool resources (CPR), field and experimental studies provided support for the proposition: *without some form of coordination or organization to enable individuals to agree upon, monitor, and sanction the patterns of appropriation by individuals from a CPR, the resource will be overused* [9]. Potential causes of several problems found in FLOSS development can be the lack of appropriate coordination and organization mechanisms or systems in FLOSS ecosystems.

Based on the extensive evidence from field studies, Ostrom and Walker clarified that CPRs are frequently associated with markets [9]. They also summarized design principles illustrated by long-enduring CPR institutions, such as, 1) clearly defined boundaries, 2) congruence between appropriation and provision rules and local conditions, 3) collective choice arrangements, 4) monitoring, 5) graduated sanctions, 6) conflict resolution mechanisms, and 7) minimal recognition of rights to organize [9].

There are some studies related to monetary systems or markets in FLOSS development. A number of companies and FLOSS projects rely on external parties to perform the security assessment of their software for reward called bug bounty program. It is reported that there are non-project-specific bug bounty hunters and they are different from traditional contributors who work on specific projects [10]. Nakasai et al. studied the impact of budges donors will get and reported that monetary contributions have impact on software development processes [11]. Although monetary systems are known to work in specific aspects in software development, currently there is no community currency working in the whole aspects of software development.

We are interested in building sustainable FLOSS communities as sustainable CPRs, with a community currency. However, FLOSS development communities are too hard to be experimental fields at our early stage, because of their complex internal and external systems and environments. Hence, we will first try an experiment with car sharing as an alternative

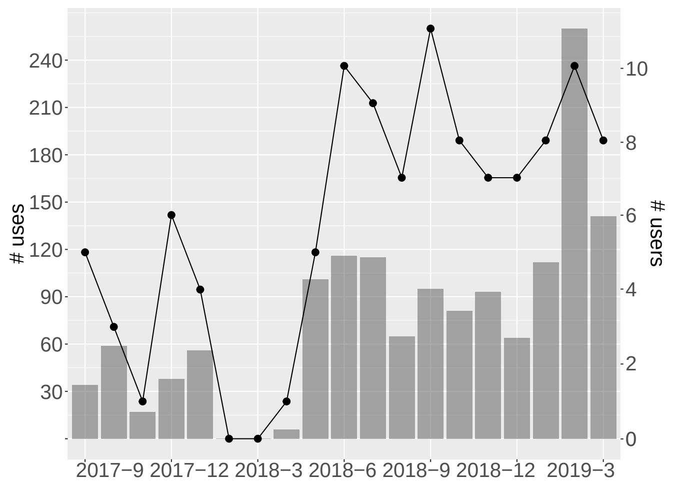


Fig. 1. The numbers of car uses (bar) and users (line) in months.

CPR, since we have started our car-sharing experiment at NAIST [12].

II. Car Sharing at NAIST

As it is known that common pool resources (CPRs) are frequently associated with markets [9], we study a community currency within a CPR. As a field of a CPR, we study our car-sharing experiment at NAIST [12].

In this car-sharing experiment, there are two electric cars (Mitsubishi i-MiEV) and two stations at NAIST and a near station (Gakken Nara-Tomigaoka Station), in which chargers exist. The two stations are in a walkable distance. Only the registered users can use cars by reserving a car in the predetermined time with their points, which can be considered to be a currency. In the current car-sharing setting, no specific mechanism to coordinate, monitor, and sanction the patterns of appropriation by users. According to the above-mentioned proposition [9], the overusing of the cars can be expected.

In this study, we analyze the records of our experiment during September 2017 to March 2019. Although we have approximately 20 registered users, some of them have left NAIST as well as the experiment. Figure 1 presents the number of car uses as a bar chart and the number of users as a line chart, per month. After the early phase and car maintenance period (no use), there had been around eight users and more than 60 uses in a month. Figure 2 shows the distribution of using time for one use. We found that short time uses of 30 minutes and one hour are the majorities in all uses.

Figure 3 illustrates car uses in different time slots in a week. We first observe that car uses decrease in weekends compared with weekday uses. Although the number of car uses in midnight (00:00-04:00) is smaller, cars had been used in most time slots. Figure 4 presents the distributions of starting times for different car directions. We see that the direction from NAIST to NAIST is the majority of uses. Especially in the time slot 16:00-20:00 there are many uses from NAIST,

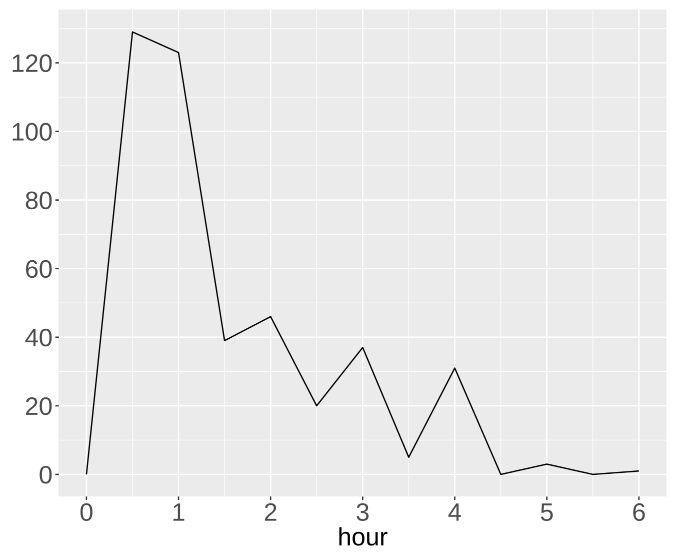


Fig. 2. The distribution of using time for one use.

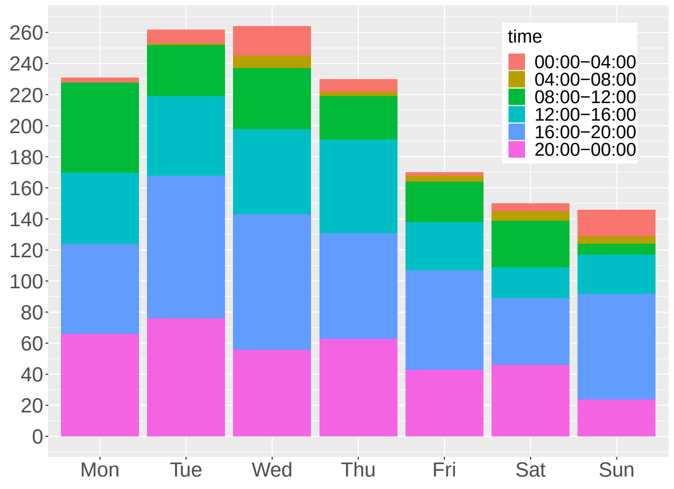


Fig. 3. Car uses in different time slots for seven days.

and many uses in 08:00-12:00 from the station. We can infer that there are typical uses from the station in the morning and from the NAIST in the evening.

Figure 5 is a chord diagram representing the volume of car uses from departing parking and time slots (right side) and arriving parking and time slots (left side). We see many uses (then bars are wide) (i) from the station to NAIST in 08:00-12:00, (ii) from NAIST to the station in 16:00-20:00, and (iii) from NAIST in 16:00-20:00 to NAIST in 20:00-24:00.

Figure 6 presents statuses of car reservations. More than 90% of reservations had been successfully executed. We think this is because of the small number of users (around eight as seen in Figure 1). Although the number is not large, there are some reservations that had not been executed. Figure 7 is a chord diagram representing those failed reservations. Two chord diagrams of succeeded and failed reservations look similar, which means conflicts of reservations hinder efficient

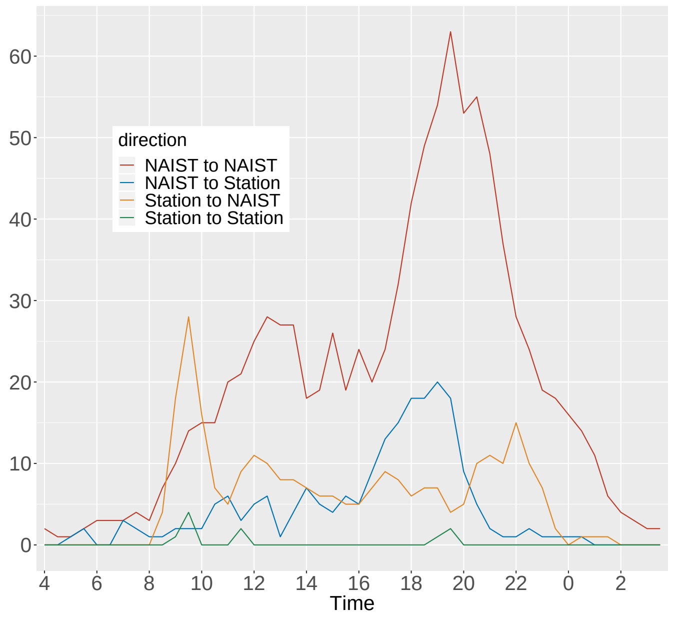


Fig. 4. The distributions of starting times for different car directions.

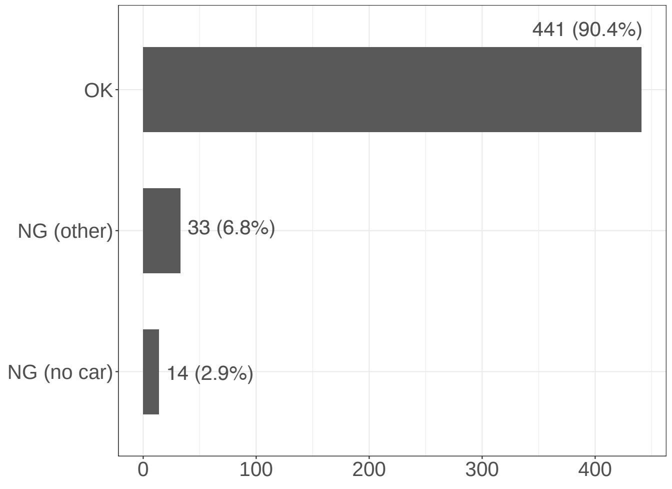


Fig. 6. Statuses of car reservations. OK for succeeded car uses, NG because there is no car, and NG because of other reasons.

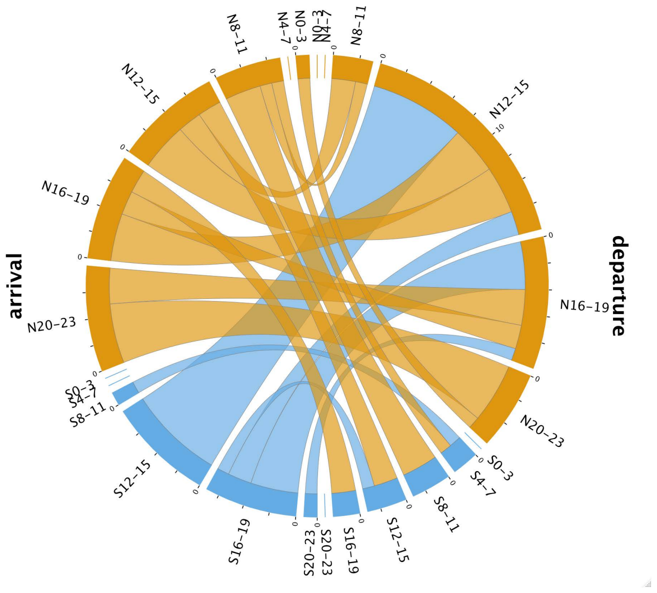


Fig. 7. Chord diagram of reserved but not used reservations. Right side is departure time slots and left side is arrival time slots. N stands for NAIST and S stands for the station. Orange bars represent car uses arriving at NAIST and blue bars represent car uses arriving at the station.

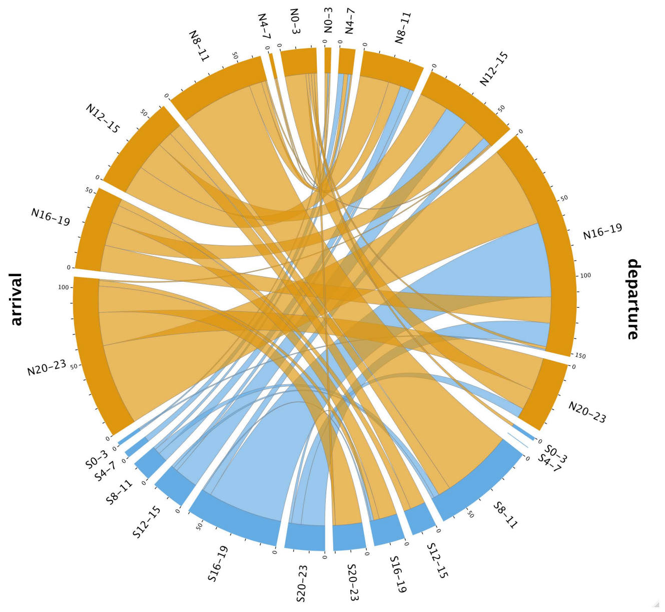


Fig. 5. Chord diagram of succeeded car uses. Right side is departure time slots and left side is arrival time slots. N stands for NAIST and S stands for the station. Orange bars represent car uses arriving at NAIST and blue bars represent car uses arriving at the station.

car uses.

Although there is no specific mechanism to coordinate, monitor, and sanction the patterns of appropriation by users, we think most car reservations executed effectively. Since we limit the number of users for the current preliminary experiment, cars are more underused than overused. However, as seen in Figure 2, some users appropriate cars for longer time, and as seen in Figure 6, some users cannot use cars, which may be caused by not efficient coordination. Note that we did not know when users gave up reservations although they would like to use cars. Considering such unknown intentions, there can be much more insufficient car uses. If we increase the

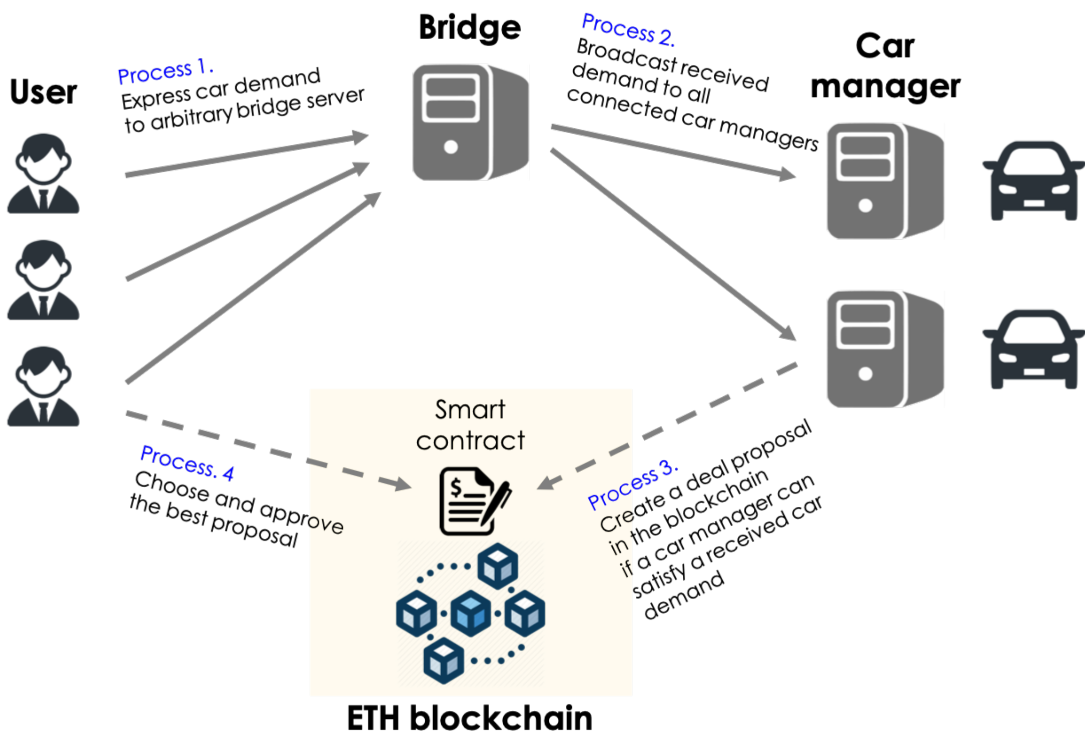

User
Process 1.
Express car demand to arbitrary bridge server
Bridge
Process 2.
Broadcast received demand to all connected car managers
Car manager
Smart contract
Process 3.
Create a deal proposal in the blockchain if a car manager can satisfy a received car demand
Process. 4
Choose and approve the best proposal
ETH blockchain

Fig. 8. Overview of our new car-sharing system.

number of users for our further experiment, it is expected that more failure reservations because of the overuses will increase.

III. Blockchain-based System

Currently we are planning to build a new car-sharing system to support "some form of coordination or organization to enable individuals to agree upon, monitor, and sanction the patterns of appropriation by individuals from a CPR". We intend to make a decentralized system instead of a centralized system in order to enable participants manage the system. To this aim, we are building a blockchain-based system. In this section, we briefly introduce some key concepts and the overview of our system design.

A. Blockchain

A blockchain is a chronologically irreversible distributed ledger that records history of transactions between nodes in a community. Blockchain was originally proposed as a solution to the double-spending problem of electronic payments by Satoshi Nakamoto in 2008 [13].

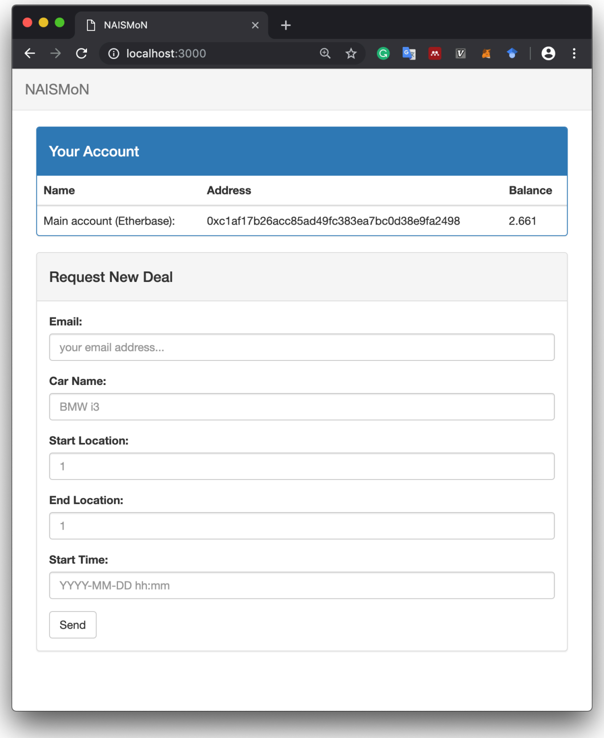
To elucidate the characteristics of blockchain, we can compare it with a traditional client-server model: (i) each node behaves as a predefined roll, client or server, (ii) all requests from clients are executed through a server, (iii) all history of transactions is stored only in a server. In contrast, a blockchain is based a peer-to-peer (P2P) model, which has the different characteristics: (i) every node is equivalent to each other and have a fair authority, (ii) every transaction is directly executed between two peers, (iii) all history of transactions is stored in a distributed ledger that all nodes share.

B. Smart Contracts

Nick Szabo introduced the concept of a smart contract as *a computerized transaction protocol that executes the terms of a contract* [14]. Within the blockchain context, smart contracts are scripts stored on the blockchain [15]. By addressing a transaction, we can trigger a smart contract. It then executes independently and automatically in a prescribed manner on every node in the blockchain network. Smart contracts operate as autonomous actors, whose behaviors are completely predictable.

The most prominent framework for smart contracts is considered to be Ethereum [16], a blockchain platform that remotely executes software on a distributed computer system called the Ethereum Virtual Machine. Ethereum smart contracts generally serve four purposes [17].

- Maintain a data store. For example, simulating a currency and recording membership in a particular organization.
- Manage an ongoing contract or relationship between multiple users. One example of this is a contract that automatically pays a bounty to whoever submits a valid solution to some mathematical problem
- Resend incoming messages to some desired destination only if certain conditions are met. This is called a forwarding contract.
- Provide functions to other contracts like a software library.



Your Account

Name	Address	Balance
Main account (Etherbase):	0xc1af17b26acc85ad49fc383ea7bc0d38e9fa2498	2.661

Request New Deal

Email: your email address...

Car Name: BMW i3

Start Location: 1

End Location: 1

Start Time: YYYY-MM-DD hh:mm

Send

Fig. 9. A prototype of the interface of expressing a car demand.

C. *Distributed Autonomous Organizations*

Based on smart contracts, there is a concept of "decentralized autonomous organizations" (DAOs) [15]. The organization members follow rules written in smart contracts, and those rules can be modified based on the voting of members. With our concept of global systems, all participants can be regarded as members of a DAO. Everyone, including smart agents, share information, follow rules and manage rules, based on autonomous and reliable mechanisms.

D. *Overview of a New System*

Figure 8 presents the design overview of our new car-sharing system. When a user express a requirement of a car use to a bridge server (1), it broadcasts its requirement to each car manager server (2). If a car can satisfy the requirement, its car manager server will create a smart contract on the Ethereum blockchain (3). After the user agree the contract (4), s/he can use the car with the condition described in the contract. Performed contracts can be publicly visible.

With this system, individual users can agree upon the predetermined car usages and monitor other car uses. Instead of enabling sanction to appropriation by individuals, we plan to impose extra fees for nuisance car uses, such as longer uses and not considerate car returns. These features and mechanism can mitigate the overuses of cars as common pool resources.

Figure 9 shows the prototype of the interface of expressing a car demand. Each user has its Ethereum address, which will be recorded in agreed smart contracts. Users hold their Ethereum tokes, which is prepared as the community currency of this car sharing. In the further study, we plan to empirically study the impact of the community currency to learn findings toward diverting this car-sharing currency system to software development ecosystems.

IV. Conclusion

This paper discusses the problem of community management from common pool resource management. We report several statistics of car uses in our car-sharing experiment as CPR management. Without preferable agreement, monitoring and sanction in the current setting, the results indicate the issue of overuses of the common pool resources. By taking the findings in economics into account, we are building a blockchain-based CPR management system.

Acknowledgment

This work has been supported by JSPS KAKENHI (Grant Number 16H05857 and 17H00731).

References

[1] S. O'Mahony, "Guarding the commons: how community managed software projects protect their work," *Research Policy*, vol. 32, no. 7, pp. 1179 – 1198, 2003, open Source Software Development. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0048733303000489

[2] J. Coelho, M. T. Valente, L. L. Silva, and A. Hora, "Why we engage in floss: Answers from core developers," in *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '18. New York, NY, USA: ACM, 2018, pp. 114–121. [Online]. Available: http://doi.acm.org/10.1145/3195836.3195848

[3] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" *Empirical Software Engineering*, May 2017. [Online]. Available: https://doi.org/10.1007/s10664-017-9521-5

[4] O. Mlouki, F. Khomh, and G. Antoniol, "On the detection of licenses violations in the android ecosystem," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 382–392.

[5] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," in *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, ser. CSMR '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 25–34. [Online]. Available: http://dx.doi.org/10.1109/CSMR.2013.13

[6] T. Ishio, Y. Sakaguchi, K. Ito, and K. Inoue, "Source file set search for clone-and-own reuse analysis," in *Proceedings of the 14th International Conference on Mining Software Repositories*, ser. MSR '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 257–268. [Online]. Available: http://doi.org/10.1109/MSR.2017.19

[7] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, no. 3859, pp. 1243–1248, 1968. [Online]. Available: https://science.sciencemag.org/content/162/3859/1243

[8] E. Ostrom, "Coping with tragedies of the commons," *Annual Review of Political Science*, vol. 2, no. 1, pp. 493–535, 1999.

[9] E. Ostrom and J. Walker, *Neither markets nor states: Linking transformation processes in collective action arenas*. Cambridge University Press, 1996, p. 3572.

[10] H. Hata, M. Guo, and M. A. Babar, “Understanding the heterogeneity of contributors in bug bounty programs,” in *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM ’17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 223–228. [Online]. Available: https://doi.org/10.1109/ESEM.2017.34

[11] K. Nakasai, H. Hata, and K. Matsumoto, “Are donation badges appealing?: A case study of developer responses to eclipse bug reports,” *IEEE Software*, vol. 36, no. 3, pp. 22–27, May 2019.

[12] Y. Arakawa, K. Yasumoto, K. Matsumoto, H. Hata, H. Suwa, A. Ihara, and M. Fujimoto, “Project is3: Incentive-based intelligent intervention for smart and sustainable society,” in *Proceedings of 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, July 2016, pp. 215–218.

[13] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[14] N. Szabo, “Smart contracts,” http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html, 1994, [Online; accessed Oct-2017].

[15] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[16] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on ethereum smart contracts sok,” in *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*. New York, NY, USA: Springer-Verlag New York, Inc., 2017, pp. 164–186. [Online]. Available: https://doi.org/10.1007/978-3-662-54455-6_8

[17] ethereum/wiki, “Ethereum development tutorial,” https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial, 2014–2017, [Online; accessed Oct-2017].