symposium on internet technologies and systems, vol. 67. Seattle, WA, 2003.

[39] X. Xia, D. Lo, W. Qiu, X. Wang, and B. Zhou, "Automated configuration bug report prediction using text mining," in *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*. IEEE, 2014, pp. 107–116.

[40] B. Xu, D. Lo, X. Xia, A. Sureka, and S. Li, "Efspredictor: Predicting configuration bugs with ensemble feature selection," in *Software Engineering Conference (APSEC), 2015 Asia-Pacific*. IEEE, 2015, pp. 206–213.

[41] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, "Automatic misconfiguration troubleshooting with peerpressure." in *OSDI*, vol. 4, 2004, pp. 245–257.

[42] S. Zhang and M. D. Ernst, "Automated diagnosis of software configuration errors," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 312–321.

[43] T. Xu, J. Zhang, P. Huang, J. Zheng, T. Sheng, D. Yuan, Y. Zhou, and S. Pasupathy, "Do not blame users for misconfigurations," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 244–259.

[44] V. Bauer, L. Heinemann, and F. Deissenboeck, "A structured approach to assess third-party library usage," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 483–492.

[45] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, "An empirical analysis of bug reports and bug fixing in open source android apps," in *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*. IEEE, 2013, pp. 133–143.

[46] J. Viega, J.-T. Bloch, Y. Kohno, and G. McGraw, "Its4: A static vulnerability scanner for c and c++ code," in *Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference*. IEEE, 2000, pp. 257–267.

[47] D. A. Wheeler, "Flawfinder," https://www.dwheeler.com/flawfinder/, retrieved Nov. 2016.

[48] J. P. Near and D. Jackson, "Finding security bugs in web applications using a catalog of access control patterns," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 947–958.

[49] A. Nistor, T. Jiang, and L. Tan, "Discovering, reporting, and fixing performance bugs," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 237–246.

[50] G. Jin, L. Song, W. Zhang, S. Lu, and B. Liblit, "Automated atomicity-violation fixing," in *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '11. New York, NY, USA: ACM, 2011, pp. 389–400.

[51] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 3–13.

[52] G. Jin, W. Zhang, D. Deng, B. Liblit, and S. Lu, "Automated concurrency-bug fixing." in *OSDI*, vol. 12, 2012, pp. 221–236.

[53] C. Liu, J. Yang, L. Tan, and M. Hafiz, "R2fix: Automatically generating bug fixes from bug reports," in *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 282–291.

[54] X. Zhao, X. Xia, P. S. Kochhar, D. Lo, and S. Li, "An empirical study of bugs in build process," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM, 2014, pp. 1187–1189.

[55] C. B. Seaman, F. Shull, M. Regardie, D. Elbert, R. L. Feldmann, Y. Guo, and S. Godfrey, "Defect categorization: making use of a decade of widely varying historical data," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2008, pp. 149–157.

[56] X. Xia, X. Zhou, D. Lo, and X. Zhao, "An empirical study of bugs in software build systems," in *Quality Software (QSIC), 2013 13th International Conference on*. IEEE, 2013, pp. 200–203.

[57] N. Palix, G. Thomas, S. Saha, C. Calvès, J. Lawall, and G. Muller, "Faults in linux: Ten years later," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI. New York, NY, USA: ACM, 2011, pp. 305–318.

[58] S. K. Sahoo, J. Criswell, and V. Adve, "An empirical study of reported bugs in server software with implications for automated bug diagnosis," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1. IEEE, 2010, pp. 485–494.

[59] A. Ozment and S. E. Schechter, "Milk or wine: does software security improve with age?" in *Usenix Security*, 2006.

[60] F. Massacci, S. Neuhaus, and V. H. Nguyen, "After-life vulnerabilities: a study on firefox evolution, its vulnerabilities, and fixes," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2011, pp. 195–208.

[61] S. Neuhaus and T. Zimmermann, "Security trend analysis with cve topic models," in *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 2010, pp. 111–120.

[62] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: A comprehensive study on real world concurrency bug characteristics," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIII. New York, NY, USA: ACM, 2008, pp. 329–339.

[63] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang, "Elblocker: Predicting blocking bugs with ensemble imbalance learning," *Information and Software Technology*, vol. 61, pp. 93–106, 2015.

[64] X. Xia, D. Lo, E. Shihab, X. Wang, and B. Zhou, "Automatic, high accuracy prediction of reopened bugs," *Automated Software Engineering*, vol. 22, no. 1, pp. 75–109, 2015.

[65] X.-L. Yang, D. Lo, X. Xia, Q. Huang, and J.-L. Sun, "High-impact bug report identification with imbalanced learning strategies," *J. Comput. Sci. & Technol*, vol. 32, no. 1, 2017.

[66] X. Xia, D. Lo, X. Wang, and B. Zhou, "Automatic defect categorization based on fault triggering conditions," in *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*. IEEE, 2014, pp. 39–48.

[67] T. Aslam, I. Krsul, and E. H. Spafford, "Use of taxonomy of security faults," Purdue University, Department of Computer Science, Tech. Rep., 1996.

[68] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi, "A taxonomy of computer program security flaws," *ACM Computing Surveys (CSUR)*, vol. 26, no. 3, pp. 211–254, 1994.

[69] S. Weber, P. A. Karger, and A. Paradkar, "A software flaw taxonomy: aiming tools at security," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–7, 2005.

[70] A. P. Nikora and J. C. Munson, "Software evolution and the fault process," in *Proc. Twenty Third Annual Softw. Eng. Workshop, NASA/Goddard Space Flight Center*, 1998.

[71] A. P. Nikora, "Software system defect content prediction from development process and product characteristics," Ph.D. dissertation, University of Southern California, 1998.

[72] L. Ma and J. Tian, "Analyzing errors and referral pairs to characterize common problems and improve web reliability," in *International Conference on Web Engineering*. Springer, 2003, pp. 314–323.

[73] Z. Li and J. Tian, "Analyzing web logs to identify common errors and improve web reliability," in *Proceedings of the IADIS International Conference on e-Society, Lisbon, Portugal*, 2003, pp. 235–242.

[74] G. Vijayaraghavan and C. Kaner, "Bug taxonomies: Use them to generate better tests," *STAR EAST*, 2003.

[75] B. Freimut, C. Denger, and M. Ketterer, "An industrial case study of implementing and validating defect classification for process improvement and quality management," in *11th IEEE International Software Metrics Symposium (METRICS'05)*. IEEE, 2005, pp. 10–pp.

[76] M. Leszak, D. E. Perry, and D. Stoll, "Classification and evaluation of defects in a project retrospective," *Journal of Systems and Software*, vol. 61, no. 3, pp. 173–187, 2002.

has also been moved to abi/reflect.go) and checks whether the assignment was possible.
- **ethereum/go-ethereum #2255** After some investigations I found that SetReadDeadLine is a cause of these errors https://github.com/ethereum/go-ethereum/blob/develop/rpc/http.go#L115 it seems deadline prolonged only for OPTIONS request and GET & POST requests fall into timeout.
- **ethereum/cpp-ethereum #2156** The fallback function is not part of the external gas estimation. Added fallback function to gas estimation and fixed mix gas estimation.
- **AugurProject/augur #477** Market link doesn't work. load single market first if on market/report detail page.

2) Processing:

- **ethereum/go-ethereum #2194** The default gas price that was being used for transaction wasn't properly using the GPO.
- **bitcoin/bitcoin #6186** The use of x where 8 does not divide x was broken, due to a bit-order issue The use of e.g. 1.2.3.4/24 where the netmasked bits in the network are not 0 was broken. Fix this by explicitly normalizing the network according to the bitmask.
- **AugurProject/augur #710** ExtraInfo field not set for markets created through UI. Fixed extraInfo / detailsText field name; updated currentPeriod and currentPeriodProgress calculations; augur.js@1.8.12.
- **ripple/ripple-lib #204** When subscribing to orderbooks with 'model' event, the last offer created is not pushed correctly to the _offers array. In particular it misses some fields like: 'quality', 'bboknode', 'flags' , 'ledgerentrytype' and others.

3) Exception Handling:

- **ethereum/go-ethereum #869** cmd/geth, cmd/utils: improve interrupt handling. The new strategy for interrupts is to handle them explicitly.
- **bitcoin/bitcoin #6702** GetTempPath has unchecked error conditions. GetTempPath can fail, but its callers do not (all?) check for this failure.
- **steemit/steem #76** Steemd loss sync after a websocket client disconnected. Properly catch errors on async connection and release.
- **AugurProject/augur #718** Transactions Sub-System does properly fail on generate order book failure response. Market created successfully, but during order book generation, an error was returned (below), which ultimately stalls the generate order book method. The UI should reflect this error state, but it looks as though an onSuccess of case.

4) Missing Features:

- **ethereum/go-ethereum #1037** th_estimateGas requires to: ... to: is unknown in case you're creating a contract. Would it be possible to make this optional? I understand the RPC spec has been frozen, but do you think it can be changed to allow "to:" to be optional at this stage? Seems like a fairly minor alteration, but would make the function much more useful.
- **bitcoin/bitcoin #321** Wallet rescan doesn't work properly. rescan seems to work on linux but doesn't (always) work on windows.
- **steemit/steem #274** Deleted comments not updated in feeds.
- **AugurProject/augur #564** The UI isn't claiming a user's winnings after a market is closed atm. It should, and then should update realized profit and loss

5) Wrong Control Flow:

- **ethereum/go-ethereum #1037** There used to be a separate handler for the CLI and GUI versions of the protocol. Unfortunately it seems to have been "misplaced".
- **bitcoin/bitcoin #2178** Remove IsFromMe() call from bool CTxMemPool::accept().
- **ripple/ripple-lib #73** Seed.get_key method does not use account_id. Seed should use the account_id to determine which key to return.
- **AugurProject/augur #177** Update network info on block arrival.

6) Corner Cases:

- **ethereum/go-ethereum #1549** If a reply timeout fired (even just nanoseconds) before the deadline of the next pending reply, the timer was not rescheduled. The timer would've been rescheduled anyway once the next packet was sent, but there were cases where no next packet could ever be sent due to the locking issue above.
- **bitcoin/bitcoin #1902** Flood of retarget messages, if internal miner is used. I'm guessing this occurs (a) only on testnet, and (b) only at a difficulty retarget boundary.
- **AugurProject/augur #677** After some tracing looks to be related to the lack of trade volume (seems to be a corner case) with all prices being returned as 0 and thus causing the price percent change to be shown as 0.00

7) Output:

- **ethereum/go-ethereum #1438** geth log reports unsigned instead of signed transaction hash.
- **ethereum/cpp-ethereum #2217** Alethzero on OS X doesn't display new transactions / closed by the submitter.
- **AugurProject/augur #504** Creation Date sort backwards. Just created 2 new markets and they show up at the top of (oldest first).
- **dogecoin/dogecoin #282** Item "Show transaction of Dogechain" not translated at all. missing translation field.

8) Input:

- **ethereum/go-ethereum #41** Number fields are limited in characters Changed validators to regexp validators IntValidator limits to 32bit int (JavaScript limitation) and therefore the input fields are limited in length.
- **ethereum/cpp-ethereum #2473** Solidity: Exponential notation for constants. Literally "1e20" can't be converted to cpp_int directly, that's why it crashes. If we allow this, we might need to evaluate to cpp_float first and then convert to cpp_int. Check whether a literal is a valid literal before using it.
- **AugurProject/augur #387** Creating markets with negative liquidity. It allowed me to create a market with -20000 initial liquidity.

C. Environment and Configuration Bugs

Environment and configuration bugs correspond to the bugs that lie in third-party libraries, underlying operating systems, or non-code parts (e.g. configuration files). The representative *environment and configuration* bug samples include:

- **ethereum/go-ethereum #1818** Just returning the current path is working for me. Hope it will be a easy fix. It is definitely the `golang` bug as this bug report https://bugzilla.redhat.com/show_bug.cgi?id=1135152 says.
- **bitcoin/bitcoin #3274** You didn't give a –with-boost option to point to the one you compiled, so it's finding the ones on your system (which are presumably borked, which is why you're building your own). Use something like ./configure –with-boost=/usr/local, where /usr/local is the prefix where boost was installed.
- **dogecoin/dogecoin #277** Closing this for now as it's likely either a problem with Berkley DB or caused by a hardware error or similar.
- **ethereum/cpp-ethereum #1982** We can't do anything for cards with less than 1G of memory. Actually cards with 2G of memory is the absolute minimum. This is how the PoW algorithms works. Closing this issue.
- **AugurProject/augur #225** The underlying issue turned out to be a bug in js-ipfs-api. I made an issue for this on the js-ipfs-api Github (ipfs/js-ipfs-api#212) and walled off the bug in AugurProject/ramble@27b6cb7 / 09b1ab3 so that the site will now load on iOS. The IPFS-based features (comments and metadata) will not work until this bug is fixed, but in the meantime the rest of the site should work now.

TABLE III: Subcategories of *semantic* bugs.

Category	Subcategory	Description
***Semantic* Bug**	Missing Features	A feature is supposed to be but is not implemented.
	Missing Cases	A case in a functionality is not implemented.
	Corner Cases	Some boundary cases are considered incorrectly or ignored.
	Wrong Control Flow	The control flow is incorrectly implemented.
	Exception Handling	Does not have proper exception handling.
	Input	Input handling or validation is incorrect or ignored.
	Output	Output displays incorrectly.
	Processing	Data processing such as evaluation of expressions and equations is incorrect.
	Other Wrong Functionality Implementation	Any other *semantic* bug that does not meet the design requirement.

TABLE IV: Interpretation of Kappa values.

Kappa Value	Interpretation
< 0	poor agreement
[0.01, 0.20]	slight agreement
[0.21, 0.40]	fair agreement
[0.41, 0.60]	moderate agreement
[0.61, 0.80]	substantial agreement
[0.81, 1.00]	almost perfect agreement

er two Ethereum projects, ethereum/mist and ethereum/cpp-ethereum, and encounter new bug types. Thus we create two new categories *GUI* and *build* as described in TABLE II. Then we look into the bug reports in the AugurProject/augur and steemit/steem projects. We further create two new categories *compatibility* and *hard fork*.

Step 2: Labeling. The first author and a graduate student independently label 1,108 bug reports of the eight blockchain systems. We use Fleiss Kappa [20] to measure the agreement between the two labelers. The interpretation of Kappa values is shown in TABLE IV. The overall Kappa value between the two labelers on all bug reports is 0.65, which indicates substantial agreement between the labelers. After completing the manual labeling process, the two labelers discussed their disagreements to reach a common decision. While many bug reports are well-described, some with cannot be sorted into sets due to insufficient detail or uncertain root causes. We mark such bug reports as the *unknown*. Meanwhile, we identify the bug reports of *duplicate*, *reopen* and *not-a-bug* types. Those bug reports are excluded from our classification. Note that a *duplicate* or *reopen* bug report is often described by comments or marked by tags, and referred to the related bug reports.

III. Bug Categories

A. Overview

This section answers RQ1. We totally identify 946 unique bugs from the 1,108 bug reports. The overall distribution of bugs based on the 10 categories is shown in TABLE V.

We find that most of the bugs are labeled as *semantic* (67.23%). The number is lower than 70.1% - 87.0% reported in a previous study [14]. Programmers could easily introduce *semantic* bugs due to the lack of a thorough understanding of the system. Additionally, since *semantic* bugs are application-specific, it is hard to automatically detect *semantic* bugs. In

TABLE V: Bug types.

Type	Count	Percentage
Semantic	636	67.23%
Missing Cases	*152*	*16.07%*
Processing	*133*	*14.06%*
Other Wrong Implementations	*80*	*8.46%*
Exception Handling	*70*	*7.40%*
Missing Features	*58*	*6.13%*
Wrong Control Flow	*48*	*5.07%*
Corner Cases	*44*	*4.65%*
Output	*35*	*3.70%*
Input	*16*	*1.69%*
Environment and Configuration	108	11.42%
GUI	66	6.98%
Concurrency	42	4.44%
Build	39	4.12%
Security	18	1.90%
Memory	15	1.59%
Performance	14	1.48%
Compatibility	7	0.74%
Hard Fork	1	0.11%

order to understand what causes *semantic* bugs, we further break down the *semantic* bugs into subcategories as shown in Table III. 23.9% *semantic* bugs are caused by *missing cases*. *Processing* bugs also account for a large portion of *semantic* bugs. The results are consistent with the previous study [21].

Environment and configuration bugs have the second highest number of occurrence (11.42%). This might be the case that blockchain systems are often used within various environments by numerous end users. Different environments may reside on various hardware and operating systems with various versions of libraries installed. Even if same versions of libraries are installed, end users could have customized configurations. The bugs in libraries, library version mismatching, as well as incorrect configurations, would lead to *environment and configuration* bugs in blockchain systems.

In the following sections, for each bug type, we would also present representative bug samples. To do so, we first manually extract keywords from the title, body, and discussion of bug reports, pull requests, and commit logs. Next, for each bug type, we group the bug reports with similar keywords together. Finally we randomly choose a bug report from each group.

Of the 1,108 closed bug reports from the studied systems, 946 unique bugs could be classified into 10 categories. Semantic *bugs are the dominant runtime bug type. 23.9%* semantic *bugs are caused by* missing cases. Environment and configuration *bugs have the second highest number of occurrence.*

B. Semantic Bugs

Semantic bugs correspond to inconsistencies with the requirements or the programmers' intention. We find that *semantic* bugs are the dominant bug types in studied projects. The finding is in line with previous work [14] in open source software. The representative *semantic* bug samples include:

1) Missing Cases:

- **ethereum/go-ethereum #2519** Previously it was assumed that when-even type `[]interface{}` was given that the interface was empty. The abigen rightfully assumed that interface slices which already have pre-allocated variable sets to be assigned. This PR fixes that by checking that the given `[]interface{}` is larger than zero and assigns each value using the generic set function (this function

cases bugs in blockchain systems, future work could be developing rule mining tools for Go and JavaScript languages, leveraging more precise static analysis, and designing scalable graph mining algorithms.

Exception handling bugs comprise around 11% of the *semantic* bugs. Exception handling code is crucial for a robust system. Incorrect or missing exception handling in security-sensitive code often causes severe security vulnerabilities (e.g., CVE-2014-0092, CVE-2015-0208, CVE-2015-0288, CVE-2015-0285, and CVE-2015-0292) [36]. In addition, systems are expected to handle exception conditions and take necessary recovery actions such as releasing resources. Failing to release resource may cause performance degradation and lead to other critical issues [37]. Thus automated detection and fixing tools for *exception handling* bugs would significantly improve robustness of blockchain systems.

***Environment and configuration* bugs are one of the major types.** TABLE V shows that *environment and configuration* bugs account for 11.42% of the reported bugs in the studied systems. *Configuration* bugs are one of the major causes for the downtime of large-scale enterprise systems [38]. Previous studies have proposed ideas to predict, detect, diagnose, and fix *configuration* bugs [39–43]. In our studied systems, many *configuration* bugs are due to wrong compiler options or wrong configuration of external libraries. The aforementioned research directions could benefit from our bug characteristics study of blockchain systems. Moreover, understanding the characteristics of *configuration* bugs in blockchain systems may help developers to better design configuration logic and requirements, and could thereby reduce the likelihood of *configuration* mistakes by users.

On the other hand, many *environment* bugs are caused by issues in external libraries. For example, bug #1818 in the ethereum/go-ethereum project is caused by a bug in Go language. Libraries have a significant impact on the stability of studied systems. Therefore, software engineering of blockchain systems should not neglect this important factor. Further studies could be conducted on usage analysis, architecture analysis and software quality assessment [44] of third-party libraries.

***Memory* bugs only account for a small portion.** TABLE V shows that *memory* bugs account for a small fraction (1.59%) of studied bugs. This percentage is much lower than the 11.8% - 16.3% reported in previous work [14]. One possible reason for this reduction could be the adoption of programming languages with managed memory type (i.e., JavaScript and Go) [15], as well as the use of more advanced debugging tools during the development process in recent years [14]. The causes include NULL pointer dereference, use-after-free, memory allocation, memory corruption, heap overflow, and double deallocation. Most of *memory* bugs result in a crash. Therefore, future studies could be conducted to understand if existing *memory* bug detection tools are capable of addressing issues in blockchain systems (e.g., improving the capability of *memory* bug detection tools, designing *memory* bug detection tools for Go programming language, and reducing false positives).

TABLE VII: Numbers of bugs fixed within various durations.

Type	Duration	Count	Proportion
Semantic	within a month	373	58.65%
	winthin a year	199	31.29%
	more than a year	64	10.06%
Env and Conf	within a month	59	54.63%
	winthin a year	44	40.74%
	more than a year	5	4.63%
GUI	within a month	28	42.42%
	winthin a year	32	48.48%
	more than a year	6	9.09%
Concurrency	within a month	19	45.24%
	winthin a year	16	38.10%
	more than a year	7	16.67%
Build	within a month	23	58.97%
	winthin a year	12	30.77%
	more than a year	4	10.26%
Security	within a month	5	27.78%
	winthin a year	12	66.67%
	more than a year	1	5.56%
Memory	within a month	9	60.00%
	winthin a year	5	33.33%
	more than a year	1	6.67%
Performance	within a month	3	21.43%
	winthin a year	6	42.86%
	more than a year	5	**35.71%**
Compatibility	within a month	4	57.14%
	winthin a year	3	42.86%
	more than a year	0	0.00%
Hard Fork	within a month	1	100.00%
	winthin a year	0	0.00%
	more than a year	0	0.00%

***Security* bugs take the longest median time to be fixed.** We notice that *security* bugs get fixed slower than non-security bugs. The finding is similar to that of Bhattacharya et al.'s study [45]. Some *security* bugs are generic and cross-projects, i.e., buffer overflow vulnerabilities. We believe blockchain systems would benefit from adopting automatic vulnerability detection tools for buffer overflow vulnerabilities [46, 47]. In addition, application-specific *security* bugs exist in studied blockchain systems, e.g., timing attacks, application-layer denial of service, and missing security checks. Those *security* bugs receive little attention [48]. Further research could be conducted to design detection tools for application-specific *security* bugs in blockchain systems.

35.71% *performance* bugs are fixed in more than one year; *performance* bugs take the longest average time to be fixed. Note that 35.71% of *performance* bugs take more than one year to fix (see TABLE VII). The percentage is much higher than that of non-performance bugs. The average time necessary to fix *performance* bugs is about 280 days (see TABLE VI), which is longer than that of non-performance bugs. These numbers indicate that *performance* bugs are probably more difficult to fix than non-performance bugs. The finding is consistent with the previous study [49]. Current effort on helping developers fix bugs focuses on non-performance bugs [50–53]. Thus more research is needed on automatic repair tools of *performance* bugs.

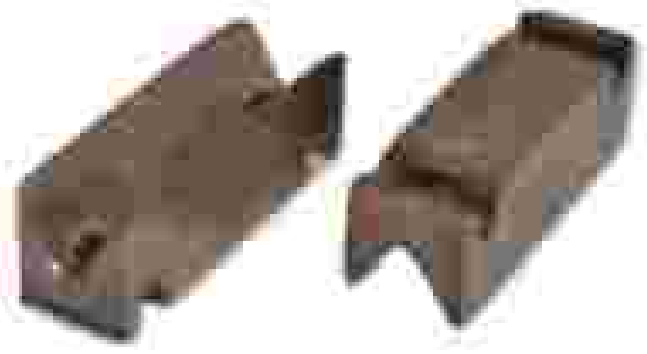
B. Threats to Validity

Construct validity. We rely on the tag "bug" from the GitHub issue repositories to identify closed bug reports. Thus we may miss some bug reports that are not tagged with "bug" due to ignorance of project contributors. In addition, some "closed"

Tezos, Stellar Lumens, DOGE aims to eliminate black Thursday losses.
The BigONE Exchange will open DOGE recharge trading activities. From 12 o'clock on April 10th, recharging DOGE can split 1 million DOGE by share within 7 days, and trading DOGE/USDT within 2 days has the opportunity to split 4 million DOGE, of which the first prize is 1.2 million DOGE.
Dynamic . . . The Dogecoin creator posted a script on Twitter to block XRP fans.
DOGE price soars to 16.67% after series of tweets from Tesla CEO
Fire Coin has announced that it will list three trading pairs for Dogecoin: DOGE/BTC, DOGE/ETH, and DOGE/USDT. Perhaps many people didn't think that the cryptocurrencies that had seen the biggest price increases during April Fool's Day were dog coins, the cryptocurrencies created by Jackson Palmer in 2014.
Decred, Dogecoin, Litecoin: Failed attempt to break up.
The Fire Coin Global Station announced the opening of Dogecoin (DOGE) recharge operations today at 9:00 pm, the DOGE/USDT, DOGE/BTC, DOGE/ETH transactions, and the simultaneous opening of Dogecoin (DOGE) withdrawals.
ACCORDING to OKEx's official announcement, OKEx launched DOGEcoin (DOGE)

on July 17, Hong Kong time, and opened DOGE recharge at 17:00 on the same day. In addition, the platform will be launched on July 18 at 17:00 DOGE/USDK, DOGE/USDT trading pairs. It is understood that OKEx is the world's leading digital asset trading platform, with French currency, coins, contracts and other trading sections. For more information on the announcement, please visit OKEx.com.
Dogecoin explodes in TikTok, price jumps 20%
The entire screen can be seen with a picture of the dog, with one post saying "Doge is the heart and soul of 'Murica'" meaning Doge is the heart and soul of Murica.
Dear LBank Users: LBank will launch DOGE on January 8, 2020 at 16:00 (UTC 8), as follows: Open Trading Pair: Doge/USDT On Top-up Time: January 6, 2020 at 14:00 (UTC 8).
A good currency, must have a good cultural background, in the background of tip culture, dog coin as an electronic currency, was born on December 12, 2013, Bitcoin is now relatively high price, so cheaper Dogecoin is popular. Just a week after it went online, it became the second-largest tip currency. They're hoping that Facebook will accept Dogecoin so that your friends can not onl

y like it, but also give you a tip by the way. Palmer also notes that Dogecoin is not like Bitcoin, where people don't get involved for the sake of speculation, a way of expressing sharing and gratitude.
DOGE / BTC 1 Week Chart on Poloniex (TradingView)
IOTA (MIOTA), Nebulas (NAS), Dogecoin (DOGE)
ACCORDING to OKEx's official announcement, OKEx launched DOGEcoin (DOGE) on July 17, Hong Kong time, and opened DOGE recharge at 17:00 on the same day. In addition, the platform will be launched on July 18 at 17:00 DOGE/USDK, DOGE/USDT trading pairs.
October 25th, Coin Security U.S. 24th announced that from October 24th at 21:00 EST, open dog money (DOGE) to the USDT trading pair of transactions, users can first start to fill doGE, ready to trade, pending the start of trading, can draw money.
OKEx will launch DOGE leveraged trading, residual currency and DOGE/USDT per permanent contracts.
According to the data on the chain, the person who sent the DOGE was much earlier than a month and a half ago. For some strange reason, the number of people using DOGE in global crises is increasing.
Metal (MTL)'s Metal Pay has integrated Dogecoin (DOGE) coins.



bugs cannot be sorted into sets due to insufficient detail or uncertain root causes. We omit such bug reports and do not categorize them.

Internal validity. Our card sorting of bug reports is subjected to interpreter's bias. We minimize the subjectivity in manual inspection through double verification: each bug report is examined at least twice by two different people independently. If they disagree, they would discuss and reach a consensus.

External validity. Similar to any characteristics study, our findings should be considered together with our methodology. The eight studied systems are open-source projects written in C++, Go, and JavaScript. Thus our results may not generalize to commercial blockchain systems or systems written in other programming languages.

VII. Related Work

In this section, we briefly review related studies. We first review some previous empirical studies on bugs. Next, we describe studies on bug categorization.

Empirical Study on Bugs. Prior work on bug empirical studies has examined the bug reports of both open source and proprietary software [8, 11, 13, 17, 54–58]. Chou et al. investigate bugs in Linux and OpenBSD kernels [8]. A similar study is performed a decade later by Palix et al. [57]. Maji et al. study defects in mobile operating systems, including Android and Symbian [11]. Sahoo et al. examine reported bugs in server software to facilitate bug diagnosis [58]. Li et al. analyze bugs in Mozilla and Apache Web server [17]. They categorize bugs in three dimensions - root cause, impact, and software component. Seaman et al. investigate bugs in NASA projects and categorize bugs depending on where these defects occur [55]. Thung et al. analyze bugs in three machine learning systems - Apache Mahout, Lucene, and OpenNLP [13]. Different from these studies, we analyze the bugs that appear in open source blockchain systems.

Other work focuses on specific bug categories. Zaman et al. analyze security and performance bugs in Firefox [12] and understand the difference between the two bug categories. They answer questions such as how fast bugs are fixed, who fix bugs, and what characteristics the bug fixes have. Ozment and Schechter [59] measure the rate at which code is introduced and the rate at which security vulnerabilities are reported in the code base of OpenBSD operating system. Then they determine whether OpenBSD's security increases over time. Massacci et al. [60] study reported security vulnerabilities in six major versions of Mozilla Firefox. Neuhaus and Zimmermann [61] study the security vulnerability reports in Common Vulnerability and Exposures (CVE) to find the trend of security vulnerabilities. Lu et al. conduct a detailed empirical study on concurrency bugs in MySQL, Apache Web server, Mozilla, and OpenOffice [62]. They analyze the root causes and types of concurrency bugs, and the number of threads and variables involved in the bugs. Tan et al. [14] study the concurrency bugs in the Linux kernel, Mozilla and Apache. They find that the Linux kernel has more concurrency bugs than the other two non-OS software. In contrast to these studies, we analyze a wide variety of bugs in blockchain systems and analyze how these bugs affect blockchain systems.

Bug Categorization. Previous studies have developed various bug taxonomies for different objectives [7, 10, 14, 21, 55, 63–76]. Some of the classifications are system-specific, the others are generic thus can become the basis for a more specific classification. Vijayaraghavan and Kaner summarize bug taxonomies and their objectives in [74]. Chillarege et al. [21] analyze the symptoms and causes of the defects, and categorize the defects into 5 types in terms of their causes. Later Chillarege et al. [7] present the most widely used classification Orthogonal Defect Classification (ODC). They extend the 5 types in their previous work to 8 types. A number of researchers use ODC as a starting point and develop new taxonomies for specific purposes. For instance, based on ODC framework, Leszak et al. [76] analyze root causes and classify defects by relating them with their underlying causes; Freimut et al. [75] build and validate a defect categorization scheme. In addition, Ma et al. [10, 72, 73] conduct a series of studies on categorization of Web applications faults. Nikora et al. [71] develop a new fault classification, which differs from others in that "it does not seek to identify the root cause of the fault. Rather, it is based on the types of changes made to the software to repair the faults" [70]. A number of taxonomies have been developed specifically for security concerns [67–69]). Seaman et al. [55] aggregate historical defect data using different categorization schemes to guide future development. We use card sorting to identify bug groups in blockchain systems, and give each group a name by referring to [14].

VIII. Conclusion and Future Work

This paper studies the bug characteristics in eight open source blockchain systems. We manually examine 1,108 bug reports in blockchain systems and leverage card sorting to categorize bugs. We further investigate the frequency distribution of bug types across projects and programming languages, and relationship between types and bug fixing time.

In order to reduce the manual effort in categorizing bugs, we plan to use text mining and machine learning techniques to automatically classify hundreds of thousands of bugs. We would like to investigate more blockchain systems and bugs. We also plan to investigate the approaches that could help developers to avoid *semantic* bugs, and the effectiveness of existing tools on *environment and configuration* bugs. Furthermore, we would like to investigate how the distribution of bug types evolves over time as the projects become more stable, and examine bug characteristics by considering bug severity levels.

Acknowledgment

We thank Mengfan Zhu for classifying some bug reports and early discussion. We also thank the anonymous reviewers for their insightful comments. This research is supported by NSFC Program (No. 61602403 and 61572426), and National Key Technology R&D Program of the Ministry of Science and Technology of China (No. 2015BAH17F01).

6-2017

Bug characteristics in blockchain systems: A large-scale empirical study

Zhiyuan WAN

David LO
Singapore Management University, davidlo@smu.edu.sg

Xin XIA

Liang CAI