# Understanding the Software Development Practices of Blockchain Projects: A Survey

Partha Chakraborty[1], Rifat Shahriyar[1], Anindya Iqbal[1], and Amiangshu Bosu[2]
[1]Department of Computer Science & Engineering
Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
[2]Department of Computer Science, Wayne State University, Detroit, MI, USA
Email:shuvopartho@gmail.com,rifat@cse.buet.ac.bd,anindya@cse.buet.ac.bd,amiangshu.bosu@wayne.edu

## ABSTRACT

**Background:** The application of the blockchain technology has shown promises in various areas, such as smart-contracts, Internet of Things, land registry management, identity management, etc. Although Github currently hosts more than three thousand active blockchain software (BCS) projects, a few software engineering research has been conducted on their software engineering practices. **Aims:** To bridge this gap, we aim to carry out the first formal survey to explore the software engineering practices including requirement analysis, task assignment, testing, and verification of blockchain software projects. **Method:** We sent an online survey to 1,604 active BCS developers identified via mining the Github repositories of 145 popular BCS projects. The survey received 156 responses that met our criteria for analysis. **Results:** We found that code review and unit testing are the two most effective software development practices among BCS developers. The results suggest that the requirements of BCS projects are mostly identified and selected by community discussion and project owners which is different from requirement collection of general OSS projects. The results also reveal that the development tasks in BCS projects are primarily assigned on voluntary basis, which is the usual task assignment practice for OSS projects. **Conclusions:** Our findings indicate that standard software engineering methods including testing and security best practices need to be adapted with more seriousness to address unique characteristics of blockchain and mitigate potential threats.

## CCS CONCEPTS

• **Human-centered computing → Empirical studies in collaborative and social computing**;

## KEYWORDS

blockchain, cryptocurrency, survey, bitcoin, ethereum

**ACM Reference Format:**
Partha Chakraborty[1], Rifat Shahriyar[1], Anindya Iqbal[1], and Amiangshu Bosu[2]. 2018. Understanding the Software Development Practices of Blockchain Projects: A Survey. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '18), October 11–12, 2018, Oulu, Finland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3239235.3240298


*ESEM '18, October 11–12, 2018, Oulu, Finland*

ACM ISBN 978-1-4503-5823-1/18/10…$15.00
https://doi.org/10.1145/3239235.3240298

## 1 INTRODUCTION

Blockchain-centric software development is in its infancy in consideration of the timeline; however, the high demand caused its expansion at an unprecedented pace. Reaching the market cap of 147 billion USD for Bitcoin in May 2018 and 760 billion USD for all cryptocurrency [5], [23], it is evident that developers of all types are incentivized to create blockchain applications. As of January 2018, more than 3,000 developers are regularly contributing to around 2,087 blockchain repositories hosted on Github [1]. We also observed each of the top 25 blockchain projects hosted on Github receiving more than 1500 stars[2] , indicating substantial interests in software development community. However, with the rapidly changing ecosystem and tight deployment schedules, it is rare to design robust architecture, review codes, test functionality, performance, scalability by developers themselves or external experts. The scenario is correctly pointed out in [17] that "*.. the numerous software projects rapidly born and quickly developed around the various blockchain implementations is that of unruled and hurried software development. The scenario is that of a sort of competition on a first-come-first-served base which doesn't assure neither software quality, nor that all the basic concepts of software engineering are taken into account.*"

The high performance, scalability, and security requirements of such a large-scale decentralized system cannot be satisfied unless the system is well designed and thoroughly tested. Identifying the immediate need of software engineering tools and techniques along with testing approaches specially designed to address the novel features introduced by decentralized programming on blockchains, Destefanis *et al.* [6] urged to introduce and focus on Blockchain-Oriented Software Engineering (BOSE). Blockchain-native technology offers an attractive combination to the hackers with high value and low maturity. Poor choices in the architectural design and immature development tools imply that even security-conscious developers are susceptible to creating security loopholes with severe consequences. The unalterable nature of blockchain technology makes a recovery prohibitively difficult or effectively impossible if the vulnerability is detected after deployment. Therefore, the development team is recommended to adopt a forward-looking approach

[1]https://github.com/topics/blockchain
[2] Starring a repository on Github makes it easy to find the repository later and also shows a user's appreciation and interests to that repository

to software engineering best practices, secure development, and extensive testing for eliminating bugs before they enter into the system [23]. Destefanis *et al.* has shown in [6] that the infamous Parity attack that caused freezing of 162 Million USD [22] could be mitigated with proper adoption of software engineering best practices.

Other relevant software engineering components that are likely to have a critical impact on blockchain development are the requirement collection/analysis and task assignment among the project members. Since the market is very competitive, innovative products are coming to the fore almost every month; requirement specification on of attractive and new features is a significant challenge. The reason behind additional complexity in task assignment for blockchain-native projects (compared to regular software development) stem from the inherent nature of the development team which is mostly sparse geographically and loosely controlled since most of the projects are open source.

To provide the robustness blockchain applications demand, first, we have to concretely understand the current software engineering practices of BCS projects or lack thereof. The exact practices could be understood reliably from the developers themselves. To the best of our knowledge, there is no formal study on the software engineering methods followed by blockchain development projects that would reveal the facts about the concern of the research community. To provide the currently missing insight on blockchain-centric projects, we have set an objective *to carry out the first formal survey to explore the software engineering practices including requirement analysis, task assignment, testing, and verification.* Specifically, we are interested in the developers' opinion about several questions regarding blockchain-centric development practices. For example, i) *Which are the software development practices that BCS developers follow?* ii) *How do BCS developers identify and select the requirements for their projects?* iii) *How are development tasks assigned to the BCS project members?* iv) *How is the correctness of BCS projects code verified?* v) *How are BCS projects tested for security and scalability?* and vi) *What are the communication channels for the BCS developers?*

To conduct the study we sent an online survey to 1,604 BCS developers gathered via mining the Github repositories of 145 BCS projects. The survey is an ideal instrument for this study as current BCS developers have first-hand experiences of their challenges and needs. The survey received 156 responses from BCS developers that met our criteria for analysis. We adopted a systematic qualitative analysis approach to building a coding scheme for the open-ended responses. Using a qualitative analysis software, multiple coders independently assigned codes to each response and achieved a 'substantial' inter-rater reliability [4].

Our study finds some similarities of BCS applications with general OSS projects and also some interesting differences. For instance, the requirements of BCS projects are mostly identified and selected by community discussion and project owners, which is different from requirement collection of general OSS projects where the requirements are mostly selected by developers [15, 19]. On the other hand, the development tasks in BCS projects are primarily assigned on a voluntary basis, which is similar to the case of OSS projects [24]. Regarding software engineering practice, code review and unit testing are the two most popular ones among BCS developers, and they hardly prefer pair programming which is very popular among OSS developers. The lack of specialized tools for blockchain to automate integration, regression, and security testing demands attention from developers of SE tools and relevant researchers.

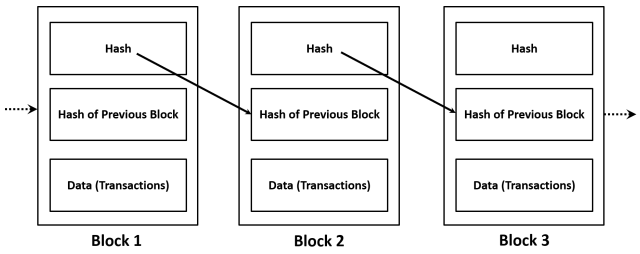



**Figure 1: Simplified diagram of a blockchain**

The remainder of the paper is organized as follows. Section 2 provides a background on blockchain. Section 3 introduces the research questions of this study. Section 4 describes our research methodology. Section 5 describes the demographics of the respondents and their projects. Section 6 presents the results of this study. Section 7 discusses the implications of our findings. Section 8 describes the threats to the validity of our results. Finally, Section 9 concludes the paper.

## 2 BACKGROUND

`Blockchain` is a decentralized, peer-to-peer, public, immutable, and append-only data storage. It keeps a permanent record of writes called `transactions`. Multiple transactions are grouped in `blocks`. Each block in a blockchain contains its hash computed using a well-known hashing or proof-of-work [10] algorithm (e.g., SHA256, ethash, and equihash) and the hash of the previous block called parent block (Figure 1). The first block in a chain is called the `genesis block`, which does not have any parent. Each block's hash is calculated based on its data, current timestamp and the hash of its parent block. Any change in a block's data causes alteration of its hash and invalidates all the subsequent blocks and the tampering becomes immediately evident to every member node of the chain. Hence, to compromise a blockchain, collusion of the majority of the network is required which is impractical in case of a large blockchain [14]. Therefore, blockchain is a chain of blocks where the blocks are irreversible and immutable.

All the nodes in a blockchain network participate simultaneously to find the next block to write. This process is called `mining`, where the nodes calculate a hash value by adding a nonce (i.e., a random value) to a list of transactions waiting to be added to the blockchain. To be eligible as the next block, the hash must be smaller than an agreed-upon value (known as `difficulty`), and the nodes continue calculations using different nonces until they find a nonce that generates a hash satisfying the 'difficulty'. The node finding a new block will broadcast it to all other nodes in the network to confirm the correctness of this new block. Once confirmed, the new block is added to the blockchain and each of the transactions contained in the block is considered `verified` [3]. The finder is usually rewarded with a predefined number of tokens, known as `block reward`. The difficulty of the next block is determined by the network with a predefined algorithm.

There is no central control over the operation of a blockchain. The underlying philosophy is that no single participant or group of

participants can control the infrastructure and all the participants in the network have an equal role to play. In the absence of a central controller, the transactions are mediated by the member nodes using a `consensus` protocol, which ensures that all the nodes have an identical copy of the blockchain. A new block is considered verified only after the majority of the member nodes vote it as true and trustworthy using the consensus protocol. A blockchain's security is based on the assumption that tampering would have to happen across majority of the nodes (aka `51% attack`) of a network in the same way simultaneously. So once a blockchain network achieves critical mass, altering a blockchain posthoc becomes infeasible.

In the context of blockchain, public key cryptography [7] ensures the integrity and authenticity of any message/transaction. Each node owns a pair of asymmetric encryption keys [20], where the `public key` is broadcast to all relevant nodes but the `private key` is kept secret. A sender signs messages with its own private key and a receiver verifies the integrity of the message by decrypting it with the sender's public key. In cryptocurrency applications, the public key of a user also acts as his/her `account address`. Therefore, a user must sign outgoing transactions using his/her private key. A miner node would verify an outgoing transaction from an account only when it can authenticate the transaction using the owner's public key.

One of the recent innovative applications of blockchain is `Smart-contracts`, which are self-executing contracts with the terms of the agreement between buyer(s) and seller(s) of transactions written using lines of code instead of a legal language. Smart contracts permit trusted transactions and agreements to be carried out among different anonymous parties without the need for a central authority, legal system, or external enforcement mechanism. Since a smart-contract, once deployed, lives on a distributed and decentralized blockchain network, it remains traceable, transparent, and irreversible.

Blockchain deals with financial and non-financial transactions usually managed through smart contracts. Hence, transaction testing of validity and integrity, and smart contract testing of specifications and compliances are essential for Blockchain. The smart contract may also need specific software tools. Blockchain-based systems may require new models for representation where traditional use case diagram, activity diagram, state diagram, etc. may not adequately represent the system. Existing programming languages may need enhanced support for testing and debugging as they are used for Blockchain development. New programming languages are also being created for Blockchain development. All of these requirements and needs call for Blockchain Oriented Software Engineering [6].

## 3 RESEARCH QUESTIONS

This study aims to understand the software development practices of BCS projects. This section introduces six research questions to achieve this goal with each question followed by a brief motivation.

**RQ1:** *Which are the software development practices that BCS developers follow?*

The Blockchain technology is changing rapidly with new protocols, innovations, and possibilities emerging every day. BCS projects are expected to follow standard software development practices to keep up with this rapid pace. Otherwise, they are at the risk of losing their market capitalization.

**RQ2:** *How do BCS developers verify the correctness of their software?*

An exploration of the ongoing practice to verify the correctness of codes of BCS projects is expected to expose the challenges and needs of this important SE practice and encourage research to overcome those challenges.

**RQ3:** *How do BCS developers test their software for security and scalability?*

The requirement of tools and techniques for security and scalability testing of BCS projects to address the special decentralized nature, need to be understood clearly. It will then lead to the development of required tools and relevant knowledge.

**RQ4:** *How do BCS developers identify and select the requirements for their projects?*

Understanding the requirement collection process for BCS projects was an objective of the study. Considering that many of BCS projects are open source in nature developed by community participation and project requirements are defined very quickly to meet the dynamic market demand, requirement collection process is likely to be different from that of traditional software projects.

**RQ5:** *What are the task assignment procedures among the BCS projects?*

With majority participation of volunteers working from different areas controlled by loosely connected management, the task assignment process has to be non-traditional and sometimes it might face challenges. Hence, the task assignment process of BCS projects is worth investigation.

**RQ6:** *What are the communication channels for the BCS developers?*

Since the majority of the BCS projects are open source, the developers need to communicate very frequently to discuss different new ideas and issues. Hence, the developers of BCS projects are likely use a wide range of different communication platforms.

## 4 RESEARCH METHODOLOGY

Since the six research questions of this study are geared towards gathering the opinions of BCS developers, we chose a survey as our research instrument. The remainder of this section describes the survey design, the participant selection criteria, pilot testing, data collection, and qualitative data analysis.

### 4.1 Survey

Our goal in designing the survey was to keep it as short as possible, while still gathering all of the relevant information. Our survey included questions to understand BCS developers' motivations, BCS software development practices, and challenges, and to compare BCS development with a non-BCS. For the current paper, we only consider a subset of the survey questions that that focus on the software development practices of BCS projects. Table 1 lists each survey question included in this paper, the research question that motivated its inclusion, and the answer choices provided. Note that questions indicated with a 'D,' rather than a 'RQ#' were included to gather demographics about the respondents.

**Table 1: Survey Questions**

| # | RQ* | Question Text | Answer Choices |
|---|---|---|---|
| Q1 | D | How many years of software development experiences do you have? | [Less than a year, between one to five years, between six to ten years, more than ten years] |
| Q2 | D | How many years have you been developing blockchain software? | [Less than a year, between one to two years, between three to five years, more than five years] |
| Q3 | D | What is your primary Blockchain software project (i.e. the project that you have spent most of your time)? | [#] |
| Q4 | D | What are your roles in your primary project (please check all appropriate roles)? | [Developer, Maintainer, requirement analysis, Testing, User support, Documentation, Social marketing] |
| Q5 | D | Approximately, how many pull requests have you submitted to your primary project? | [Less than 10, Between 11 to 30, More than 30] |
| Q6 | D | Approximately, how many hours on average do you spend per week on your primary project? | [Less than 5, between 6 to 10, Between 11 to 20, Between 21 to 35, I work full time] |
| | | *Please answer the following questions based on your own experiences with your primary project (i.e., as responded in Q3.)* | |
| Q7 | RQ6 | Which of the following channels do you use to communicate with the peers from your primary project? Please check all that apply. | [email, Slack, Discord, Github, instant messenger, Skype, mailing list, Reddit, Medium, others (please specify)] |
| Q8 | RQ1 | Which of the following software development practices do you follow (Please check all that apply)? | [peer code review, automated build ( aka one-click build using ANT, Maven, Gradle, CMake), continuous integration (e.g., Travis, Jenkins, Teamcity, Bamboo, Buildbot, or Cruisecontrol), pair programming, unit testing, automated testing, test driven development (aka test first development), performance testing, formal verification, others (please specify)] |
| Q9 | RQ1 | Please rank the following software development practices based on their effectiveness to improve the quality of your project (you can drag and drop to reorder the following list): | [formal verification, automated testing, pair programming, performance testing, test driven development (aka test first development), unit testing, continuous integration (e.g., Travis, Jenkins, Teamcity, Bamboo, Buildbot, or Cruisecontrol), peer code review, automated build( aka one-click build using ANT, Maven, Gradle, CMake)] |
| Q10 | RQ4 | How the requirements of your projects are identified and selected? | [#] |
| Q11 | RQ5 | How development tasks are assigned among the project members? | [#] |
| Q12 | RQ2 | How do you verify the correctness of your code? | [#] |
| Q13 | RQ3 | How do you test your software for security and scalability? | [#] |

**numbers refer to the research question that motivated the inclusion of the survey question, 'D' refers to demographic questions*

## 4.2 Participant Selection

To ensure valid results, we only surveyed BCS developers with sufficient experience. We identified 145 BCS projects based on following four criteria:

- Tagged under at least one of the following six 'topics'[3]: `blockchain`, `cryptocurrency`, `altcoin`, `ethereum`, `bitcoin`, and `smart-contracts`.
- 'Starred' by at least ten users.
- Have at least five distinct contributors.
- A manual verification of the repository confirmed it as a BCS project.

We used Github API[4] to identify 1,604 contributors, each of whom had submitted at least five changes to one of those 145 projects. We mine the Git commit logs of the identified 145 projects to gather the email addresses of those 1,604 active contributors.

## 4.3 Pilot Survey

To help ensure the understandability of the survey, we asked Computer Science professors and graduate students with experience in SE and experience in survey design to review the survey to ensure the questions were clear and complete. The feedback only suggested minor edits. The changes we made include: adding more

[3]https://blog.github.com/2017-01-31-introducing-topics/

[4]https://developer.github.com/v3/

answer choices to several questions and adding clarifying examples to three questions.

### 4.4 Data Collection

We got our research methodology (i.e., survey questions, participant selection, recruitment email, consent form, data collection, and data management) reviewed and approved by the SIU Institutional Review Board. On December 13, 2017, we sent each of the 1,604 BCS developers in our list a personalized email mentioning the BCS repository that we mined to obtain his/her email address with a link to the survey hosted on Qualtrics [21]. Approximately 62 of those emails bounced, leaving at most 1,542 potential participants, assuming all other emails reached their intended recipient. On December 21, 2017, we sent a reminder email. We closed the survey on January 5, 2018; after the response rate slowed to almost no response each day.

Data from the survey link created with Google's URL shortener showed a total 358 clicks on the survey URL (≈23% of the invitations). Out of those clicks, 200 people took the survey with a response rate of ≈13% (200/1542). As most of the questions were optional, many respondents skipped some of the questions. Only 115 respondents answered all the questions. After the exclusion of the 44 responses that did not answer either at least 75% of the questions or at least one open-ended question, we were left with 156 responses for analysis.

### 4.5 Qualitative Analysis Process

For the open-ended questions, we followed a systematic qualitative data analysis process. First, two of the authors independently extracted the general themes from the first 75 responses to each question. Using those themes, the authors had discussion sessions to develop an agreed-upon coding scheme for each question. Using this coding scheme, another author went through the remaining answers to determine any additional codes that need to be added.

With this scheme, two of the authors independently coded each response using the Coding Analysis Toolkit (CAT) [12] software. The coders could also add new codes, if necessary. We computed the level of inter-rater reliability of the manual coding process using Cohen's kappa [4], which was measured as 0.62. While there is no universally accepted 'good' kappa, values between 0.61 to 0.80 are generally recognized as 'substantial agreements' [9]. We used CAT to identify the discrepancies in coding and had discussion sessions to resolve all conflicts. Once we completed the coding process, we transferred the data into IBM SPSS for further analysis along with the quantitative data.

## 5 DEMOGRAPHICS

To provide a proper context for the results, this section describes the demographics of the projects represented by the respondents and of the respondents themselves.

### 5.1 Projects Represented

Table 2 provides the results to Q3 (Table 1) about respondents' primary projects. The number in parenthesis represents the number of respondents who listed that project. Our respondents represent 61 different BCS projects. The Coin Development Index [8], which tracks the top BCS projects, indicates our respondents representing 18 out of the top 25 projects. 37% of our respondents have come from the top ten projects which indicates a substantial participation of top BCS developers in our survey.

### 5.2 Respondents' Demographics

This section describes the demographics of the respondents in terms of their software development experience(Q1), BCS development experience(Q2), roles(Q4), number of total commits to BCS projects(Q5), and the average number of hours per week spent in BCS development(Q6).

In terms of roles (Figure 3) in the primary project, the majority of our respondents had multiple responsibilities for their primary project. 'Developer' was the most common role (93%) among our respondents followed by 'maintainer' (45%) and 'QA' (34%).

In terms of software development experiences (Figure 2(a)), 70.5% of our respondents have more than five years of development experiences with 42.3% having more than 10 years. However, in terms of BCS development experiences (Figure 2(b)), 81.4% of our respondents have less than 2 years of experiences with 37.8% having less than a year. These numbers indicate a large number of software developers, who are experienced in non-BCS development, have recently joined BCS projects.

In terms of the number of contributions to a BCS project (Figure 2(c)) 57.6% of our respondents have made more than 10 with 42.9% submitting more than 30. On the other hand, 42.7% of our respondents spend at least 20 hours a week on a BCS project with 32.7% working full time (Figure 2(d)). Combining our respondents' number of commits and number of hours per week spent in BCS projects, we conclude that our respondents include a sample of active BCS developers who are qualified to provide valuable insights for the goals of this study.

## 6 RESULTS

The following subsections describe the results of our survey by answering the six research questions introduced in Section 3. To help clarify the results, we also include excerpts from the qualitative responses to the open-ended questions. Each of the excerpts is followed by a number representing a unique identifier for the respondent who expressed that opinion. For example, [#5] indicates a response from respondent number 5.

As a result of the coding process (Section 4.5), each of the open-ended questions had a large number of detailed categories. For this presentation of the results, we abstracted the detailed categories into a smaller number of high-level categories. In a qualitative analysis, each open-ended response could match multiple codes. Therefore, the sum of the percentages can be greater than 100%.

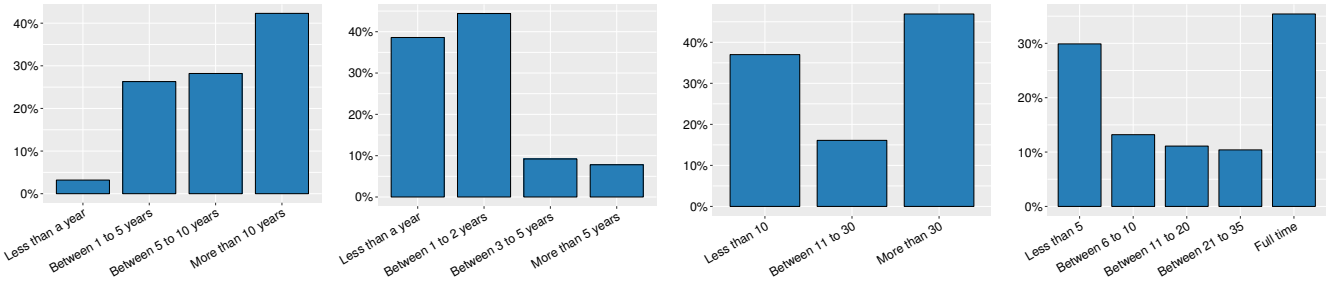
### 6.1 RQ1: Software Development Practices

Figure 4 shows different software engineering practices of BCS developers and Figure 5 ranks them based on their perceived effectiveness, which emerged from the answers to Q8 and Q9 (Table 1) of our survey.

It is evident from Figure 4 that code review is the most common practice. Continous integration and unit testing are also widely used by BCS developers. Figure 5 shows that according to the BCS

**Table 2: Projects represented**

| Multiple occurrences | | | Single occurrences | | | |
|---|---|---|---|---|---|---|
| Ethereum (22) | Bitcoin (9) | Bitshares (7) | Ambisafe | Basic Identity Token | Bytom | Cpuminer |
| Monero (6) | Sia (6) | Waves (5) | Dash | Distense | DNSChain | Ebets |
| Solidity (5) | Lbry (4) | Ripple (4) | ESKU | Etherbet | Etherplay | Fabric Labs |
| Nem (3) | Cardano (3) | Decred (3) | Golem | Haskoin | ZeroLink | Icofunding |
| EOS (3) | Hyperledger (3) | IOTA (3) | Ind | Iroha | JS Miner | Keyrun |
| Factom (2) | Feather coin (2) | Lisk (2) | Libsnark | LiteCoin | Payroll System | PHP-Mpos |
| Metamask (2) | Namecoin (2) | Neo (2) | Populus | Progmathon | Pycoin | Shapeshift |
| Remix IDE (2) | Stratis (2) | Trezor (2) | Snapcoin | Status | Steller | Storj |
| Zcash (2) | Undisclosed /Private (14) | | Swifty | Vandal | Vcash | Viper |




(a) Software development experience (b) BCS development experience (c) Total number of code commits in BCS projects (d) Average number of hours per week spent on BCS development

**Figure 2: Demographics of the respondents**

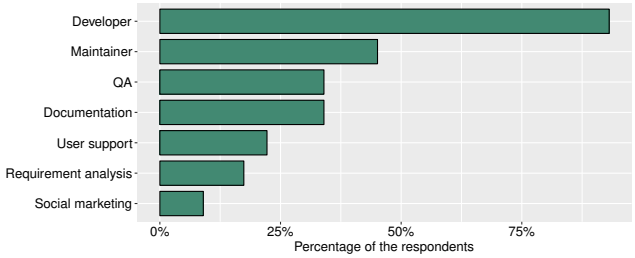



**Figure 3: Roles of the survey respondents**

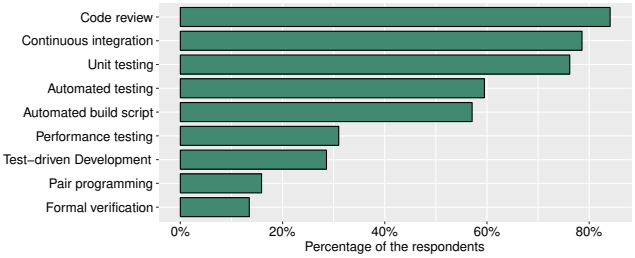



**Figure 4: SE practices of BCS developers**

developers code review is also the most effective one. As a young and immature ecosystem, the BCS domain lacks adequate testing tools and without that humans are the best bug finders in BCS projects. Although formal verification ranks sixth in terms of effectiveness, it ranks last among the frequency of regular practice

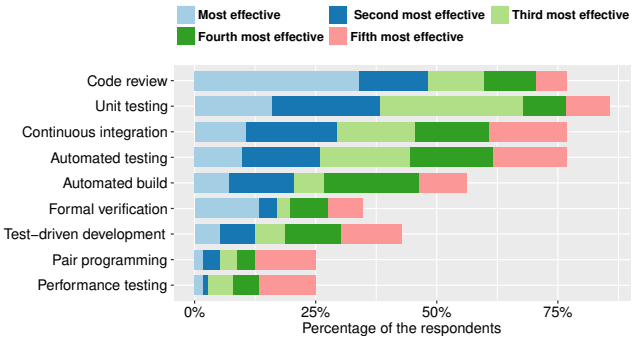



**Figure 5: Ranks of software development practices based on effectiveness**

indicating most of the developers have difficulties using it. Pair programming is very popular among non-BCS developers based on the *2018 Stack Overflow Annual Developer Survey* [16], but it is not the case for BCS developers.

## 6.2 RQ2: Verification and Validation

RQ1 addresses the software development practices generally followed by BCS developers and RQ2 addresses the specific tool/techniques they used to verify the correctness of their code. Figure 6 shows different techniques used to verify the correctness of BCS projects, which emerged from the answers to Q12 (Table 1) of our survey.

The figure suggests that code review and unit testing are the most used techniques by the BCS developers to verify their code correctness. Since blockchain is a new technology with a limited

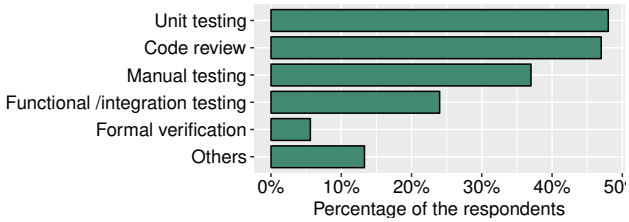



**Figure 6: Different Technique to Verify Correctness of BCS projects**

number of testing tools, many BCS developers indicate that they verify the correctness of their codes through manual testing. The following subsections examine these techniques in more detail.

*6.2.1 Unit testing.* The correctness of some BCS projects codes is verified using unit testing written either by the developer himself or by the separate QA team.

*Mostly with writing unit tests. [#39]*

*6.2.2 Code review.* Like other traditional software projects, code review by peers is used to verify the correctness of BCS projects codes.

*Code review and compliance with a test suite. [#42]*

*6.2.3 Manual testing.* The correctness of some BCS projects codes is verified by manual testing by the developer himself.

*Manually testing (but in my blockchain hobby project. [#38]*

*6.2.4 Functional and Integration testing.* The correctness of some BCS projects code is verified using functional testing that ensures that the project has all the required functionality that's specified within its functional requirements.

*Running any changes on a testnet before setting a new release tag. [#69]*

The correctness of some BCS projects code is verified using integration testing that integrates different internal modules together with external libraries and runs various automated tests of different categories.

*Automated testing and continuous integration (Travis, AppVeyor). [#149]*

*6.2.5 Formal verification.* The correctness of some BCS projects codes is verified using formal verification methods such as automated theorem proving, model checking, equivalence checker, property checker, etc.

*Security proofs / formal verification. [#59]*

*6.2.6 Others.* The correctness of some BCS projects code is verified using crash analysis that records and analyzes different runtime crash of the project.

*Unit-tests - crash-report analysis [#12]*

Like other traditional software projects, beta testing and corresponding feedback by the users is used to verify the correctness of BCS projects codes.

*We test the code and also do peer review plus get feedback from our users (through Github and through our support chat) [#67]*

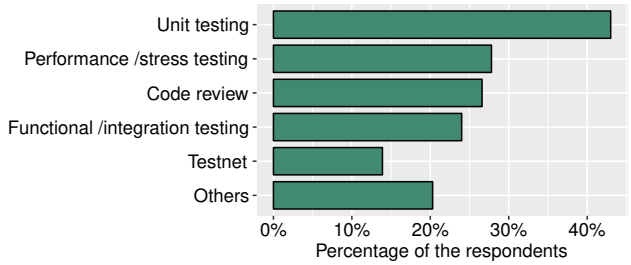



**Figure 7: Different Technique for Testing Security and Scalability of BCS projects**

## 6.3 RQ3: Security and Scalability

Figure 7 shows different techniques used to test the security and scalability of BCS projects, which emerged from the answers to Q13 (Table 1) of our survey.

The figure suggests that code review, stress testing, and unit testing are the most used techniques by the BCS developers to test the security and scalability of their projects. Some BCS developers use the Testnet, which is an alternative blockchain to be used for testing without worrying about breaking the main blockchain. Some BCS developers also use techniques like static program analysis, bug bounty, simulation and external audit.

*6.3.1 Unit test.* The security and scalability testing of some BCS projects are done through appropriate unit testing written either by the developer himself or by the separate QA team.

*Unit testing in a test network with a test block chain. Scalability is tested to a given level only. [#127]*

*6.3.2 Performance/Stress testing.* Blockchain based applications are usually large-scale and very much performance critical. Comprehensive performance testing is required by generating enormous load with appropriate tools on the system prior to actual deployment.

** security: testing and bounties * scalability: stress testing. [#33]*

*6.3.3 Code review.* Like other traditional software projects, code review by expert peers is used to test the security and scalability of BCS projects.

*As I understand these aspects may be focused during peer Code review. [#149]*

*6.3.4 Functional and Integration testing.* The security and scalability testing of some BCS projects are done through manual testing by the developer himself.

*Mostly manually using Test and Development networks. With help of various transactions generators. But we have few automated performance tests. [#119]*

Usually, blockchain applications are large in terms of features, and a good number of developers are assigned to develop individual components. Due to the inherent complexity of the system, it is quite common that well tested individual modules would not function properly when integrated. Hence, extensive integration testing is required whenever a new module is merged into the system.

*Fuzz testing, integration testing, unit testing, manual audit, etc. [#137]*

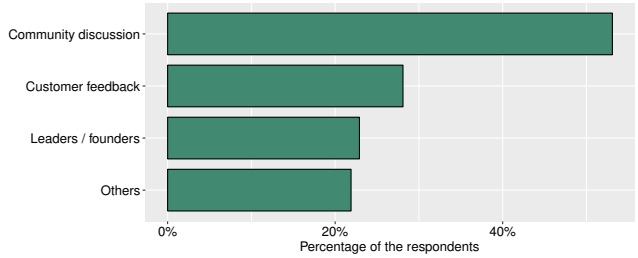



**Figure 8: Requirements collection**

*6.3.5 Deployment on a testnet.* The Testnet is an alternative blockchain to be used for testing. Testnet allows application developers to experiment without worrying about breaking the main blockchain. Testnet is used to test the security and scalability of some BCS projects.

*Deploy on the testnet and have different threat model to test its correctness. [#34]*

*6.3.6 Others.* The security and scalability testing of some BCS projects are done through executing in a simulated environment and analyzing the outcome.

*Simulation frameworks and live testing. [#84]*

The security and scalability testing of some BCS projects are carried out through static program analysis on the code with an automated tool without actually executing it.

*Unit tests, and static analysis. [#52]*

To remove potential vulnerabilities that might escape internal testing, external audit by experts outside the project team is a common practice and is highly recommended for complex and widespread systems such as blockchain.

*There are external security auditors who works on that. [#147]*

The bug bounty is a deal by which individuals can receive recognition and compensation for reporting bugs, especially those about exploits and vulnerabilities. It is used to test the security and scalability of BCS projects.

*Bug bounties and a dedicated security review team, rigorous academic peer review for scaling. [#56]*

## 6.4 RQ4: Requirement Analysis

Figure 8 shows different requirement selection processes for BCS projects, which emerged from the answers to Q10 (Table 1) of our survey.

The figure suggests that the requirements of BCS projects are mostly identified and selected by community discussion and project leaders. The majority of requirements in traditional OSS projects are selected by the developers [15, 19]. Requirements in OSS projects are also identified through customer feedback. However, in BCS projects the majority of requirements are identified through community discussion which is dissimilar to general OSS projects. This result is counter-intuitive as most of the BCS projects are also open source and believed to possess similar characteristics in these aspects. Moreover, project leaders or founders directly influence the selection of requirements in BCS projects, which is not the case for OSS projects [13]. It is worth mentioning that recently OSS projects are also collecting requirements through community discussion [18]. The following subsections examine these processes in more detail.

*6.4.1 Community Discussion.* Community discussion is the primary source of requirements for BCS projects. For most of the projects, requirements evolve through collective idea generation of the community members through both online and offline meetings and discussions.

*Community input. [#122]*

Ensuring the security, scalability, and reliability of a BCS project is an open research area. Many Community members engage in research and brainstorming to generate innovative concepts and ideas. They also look into academic research to determine potential new directions.

*By reading academic literature in the field to discover interesting research directions which have not previously been addressed [#66]*

Some BCS projects have specific project roadmap or high-level specifications that the community members have agreed on. That roadmap guides the requirement analysis for such projects (e.g., Ethereum, EOS).

*We have a roadmap, which is based upon the Namecoin community.[#52]*

*6.4.2 User / customer feedback.* Requirements for some BCS projects are identified by the feedback from user and customer on what next to incorporate into the project.

*The community requests features and these are evaluated and translated in technical requirements. [#127]*

Some BCS projects employ standard issue tracking system. The requirements for those projects are selected by the different features or improvements added by the users.

*Top issues (by demand) are upvoted and prioritized on the issue tracker.[#98]*

*6.4.3 Technical leaders/ founders.* Requirements for some BCS projects are identified and selected by project owners/founders, management/business team, lead technical persons.

*Inputs come from the community, the selection is made by founders. [#68]*

*6.4.4 Others.* Some BCS projects are managed by a single developer. The requirements for those projects are self-selected by that developer. In some BCS projects, the requirements are identified and selected by studying existing projects.

*Badly. Getting better as we get bigger and bigger projects trying to do concrete things. [#137]*

Some BCS projects have no defined process to identify requirements.

*As I go along, no process. [#5]*

## 6.5 RQ5: Task Assignment

Figure 9 shows different development task assignment processes for BCS projects, which emerged from the answers to Q11 (Table 1) of our survey.

The figure suggests that the development tasks are primarily assigned on voluntary basis. Ideally, the key element of any OSS project is voluntary participation and voluntary selection of tasks; i.e., each person is free to choose what he or she wishes to work on [24]. We see that the task assignment in BCS projects follows the similar task assignment strategy as OSS projects. Moreover, since blockchain is a new technology with a limited number of experts,

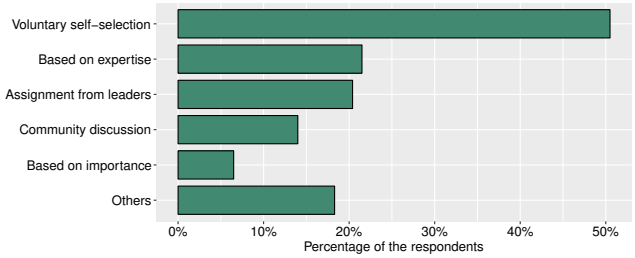



**Figure 9: Different Task Assignment Process for BCS projects**

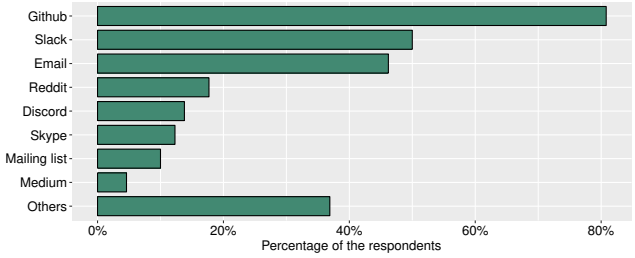



**Figure 10: Communication channels of BCS developers**

many BCS developers indicate that tasks are assigned for some projects based on the expertise of the developers. The following subsections examine these processes in more detail.

*6.5.1 Voluntary self-selection.* In some BCS projects, the development tasks are selected voluntarily by the team members. Team members are free to choose tasks they want to do.

*Whoever is most able to do them, and also available to do so, can offer to work on tasks. It's volunteer based, even when being compensated. [#52]*

*6.5.2 Assignment based on skill and interests.* In some BCS projects, the development tasks are assigned based on the interest of the developers as well as their skillsets. Team members can express their interest in doing any specific task, but the task will be assigned considering their technical skills.

*Based on individual skillsets and interests. [#66]*

*6.5.3 Assignment from project leads.* In some BCS projects, the development tasks are assigned by the owner, manager, technical lead. Team members need to do the tasks they are assigned.

*CTO assigns the tasks. [#166]*

*6.5.4 Assignment based on importance.* In some BCS projects, the development tasks are assigned based on the importance. The high priority tasks with scheduled deliverables are assigned first.

*Most critical first, then more popular requests. [#38]*

*6.5.5 Others.* Some BCS projects are managed by a single developer. In some BCS projects, the tasks are assigned either on first come first serve basis or on an ad-hoc basis. *First come first served. [#195]*

## 6.6 RQ6: Communication Channels

Figure 10 shows different communication channels used by the BCS developers, which emerged from the answers to Q7 (Table 1) of our survey.

The figure suggests that Github and Slack are the most used communication channels by the BCS developers. Github is very popular because developers extensively communicate through pull requests, code reviews, etc. directly in codes. Slack is very popular among BCS developers because some of the large and most popular cryptocurrency communities are on Slack, and the basic account is free. Medium is also popular among BCS developers because it is the largest repository of Blockchain related technical articles. Due to the intent of being anonymous, the use of mailing list is less frequent among BCS developers. In general, the communication channels for BCS developers are different from traditional OSS developers.

# 7 IMPLICATIONS

**Research:**

Our study identified several areas ready to receive an impactful contribution from the researchers of software engineering, security, and distributed systems. First, in the rapidly expanding and competitive field of blockchain native applications, innovative ideas are needed to design features of the applications to attract the users. At the same time, reliability and efficiency of the system must not be compromised. Hence, the developers are likely to be aided by collaboration with researchers who have the knack of thinking ahead of the technology of the future.

Next, BCS developers have to rely mostly on code review and unit testing practices for verification of correctness of the functionality of the applications. Besides, stress testing is conducted for security and scalability testing. Mostly manual efforts are reported due to lack of sufficient automation of testing addressing the decentralized nature of blockchain. The contribution from relevant research community will be very significant to make BCS robust and efficient.

**Practices and trends:**

First, this study finds some interesting practices of blockchain centric projects related to software engineering that vary substantially from that of traditional software development. For example, the requirements of BCS projects are mostly identified and selected by community discussion and sometimes by project owners. This is even different from the requirement collection of general OSS projects where requirements are predominantly identified by developers and through customer feedback. Since BCS is a complex technology, users or general people have little to input. It implies that requirement formulation and specification development is a task of technical intellect in case of BCS.

Secondly, the development tasks are primarily assigned on the voluntary basis as most of the developers are loosely controlled voluntarily working in decentralized projects. This resembles the task assignment practice for general OSS projects.

Finally, the survey testifies that regarding the highly important functional, scalability, and security testing, there is an absence of quality tools that can address BCS characteristics. Especially, the low response to integration or regression testing should be a

matter of concern considering the distributed development culture prevailing in most of the BCS projects. Also, lack of awareness about advanced and automated testing methods of the developers was evident from the response of the participants of the survey.

**Shortcomings and recommendations:**

The findings regarding requirement collection suggest that future sponsor of BCS projects should put special care on it and prepare a team of experts for that. Among the standard software engineering practices, as we see in figure 4, formal verification, pair programming, performance testing, test driven development are quite infrequent in case of BCS. However, these are widely accepted in traditional software development [16]. The lack of testing tools, especially for integration, regression, and security testing, which is the potential cause for the deviation from recommended practice, calls for attention on blockchain to the developers of SE tools and researchers. The researchers and academics should also respond to the demand of specialized software engineering guideline for blockchain based development as justifiably proposed in [6].

## 8 THREATS TO VALIDITY

The primary threats to validity are related to sample selection. The respondents of our survey may not adequately represent all BCS developers. While our respondents come from 61 different BCS projects, they primarily represent the top ones. Therefore, some of the opinions, especially, the software development practices used by BCS developers in smaller projects may be different from those included in this study. However, the software development practices mentioned by our respondents cover a broad range, so software development practices in smaller BCS projects may be similar to the practices our study revealed.

We selected five commits as a threshold for an invitation to this survey. A higher or lower threshold may have altered the results. However, none of the results of our research questions significantly differed based on the number of pull requests of a respondent (Q5). Therefore, this threat may be minimal.

Although the response rate of our survey is similar to prior SE surveys [2, 11], the response rate is only (≈13%). Therefore our results could suffer from a potential 'non-response bias' (i.e., the opinions of the respondents who chose to participate may be different from who did not) [1]. Even so, the 156 responses that we analyzed provide a rich source of data to reveal the insights described in this paper.

## 9 CONCLUSION

Although blockchain centric projects are sharply increasing in number and the market cap has become surprisingly high, empirical SE research exploring this area is rather inadequate. Consequently, the software engineering practices are not studied substantially and limitation of the practices remain unaddressed. This study has been designed to bridge this gap and unveils the details from the feedback of the developers themselves. Our results suggest that some of the software engineering practices are followed in case of BCS effectively, whereas scarcity of support is felt in some critical testing and security analysis requirements. As a young and immature ecosystem, BCS domain needs an array of tools, SE methods, and resources (as reported in this study) where SE research has scope to contribute.

## ACKNOWLEDGMENT

We thank the anonymous respondents of our survey.

## REFERENCES

[1] J Scott Armstrong and Terry S Overton. 1977. Estimating nonresponse bias in mail surveys. *Journal of Marketing Research* 4, 3 (1977), 396–402.
[2] Amiangshu Bosu, Jeffrey C Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. 2017. Process aspects and social dynamics of contemporary code review: insights from open source development and industrial practice at microsoft. *IEEE Transactions on Software Engineering* 43, 1 (2017), 56–75.
[3] David Lee Kuo Chuen. 2015. *Handbook of digital currency: Bitcoin, innovation, financial instruments, and big data.* Academic Press.
[4] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46.
[5] CoinMarketCap. 2018. Bitcoin Marketcap. https://coinmarketcap.com/currencies/bitcoin/. Accessed: 2018-05-13.
[6] Giuseppe Destefanis, Michele Marchesi, Marco Ortu, Roberto Tonelli, Andrea Bracciali, and Robert M. Hierons. 2018. Smart contracts vulnerabilities: a call for blockchain software engineering?. In *Proceedings of the 2018 International Workshop on Blockchain Oriented Software Engineering*. IEEE, Campobasso, Italy, 19–25.
[7] Simson L Garfinkel. 1996. Public key cryptography. *Computer* 29, 6 (1996), 101–104.
[8] Tor Gilbertson and Leo Vroegindewey. 2017. Larry's Cryptocoin Risk. https://www.coindevelopmentindex.com/
[9] Gary G. Koch J. Richard Landis. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174.
[10] Markus Jakobsson and Ari Juels. 1999. Proofs of work and bread pudding protocols. In *Proceedings of the Conference on Secure Information Networks: Communications and Multimedia Security*. Springer, Deventer, The Netherlands, 258–272.
[11] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. 2016. Code review quality: how developers see it. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE/ACM, Austin, TX, USA, 1028–1038.
[12] Chi-Jung Lu and Stuart W Shulman. 2008. Rigor and flexibility in computer-based qualitative research: Introducing the Coding Analysis Toolkit. *International Journal of Multiple Research Approaches* 2, 1 (2008), 105–117.
[13] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. 2002. Evolution Patterns of Open-source Software Systems and Communities. In *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE '02)*. ACM, Orlando, FL, USA, 76–85.
[14] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. 2016. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.* Princeton University Press.
[15] John Noll and Wei-Ming Liu. 2010. Requirements Elicitation in Open Source Software Development: A Case Study. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS '10)*. ACM, Cape Town, South Africa, 35–40.
[16] Stack Overflow. 2018. Stack Overflow Annual Developer Survey. https://insights.stackoverflow.com/survey/
[17] Simone Porru, Andrea Pinna, Michele Marchesi, and Roberto Tonelli. 2017. Blockchain-oriented software engineering: challenges and new directions. In *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, Buenos Aires, Argentina, 169–171.
[18] Arif Raza and Luiz Fernando Capretz. 2014. Addressing User Requirements in Open Source Software: The Role of Online Forums. *Journal of Computing Science and Engineering* 8, 1 (2014), 57–63.
[19] W. Scacchi. 2002. Understanding the requirements for developing open source software systems. *IEE Proceedings - Software* 149, 1 (2002), 24–39.
[20] Gustavus J Simmons. 1979. Symmetric and asymmetric encryption. *ACM Computing Surveys (CSUR)* 11, 4 (1979), 305–330.
[21] Jonathan Snow and M Mann. 2012. *Qualtrics Survey Software: Handbook for Research Professionals.* Qualtrics Labs, Incorporated.
[22] Parity Technologies. 2017. A Postmortem on the Parity Multi-Sig Library Self-Destruct. https://paritytech.io/a-postmortem-on-the-parity-multi-sig-library-self-destruct/.
[23] Synopsys technology. 2018. Blockchain software security best practices. https://www.synopsys.com/blogs/software-security/blockchain-software-security-best-practices/.
[24] Steven Weber. 2004. *The Success of Open Source.* Harvard University Press, Cambridge, MA, USA.