Powering Software Sustainability with Blockchain

Omar Badreddin
University of Texas
El Paso, TX, U.S.A
obbadreddin@utep.edu

ABSTRACT

Software sustainability is a systematic challenge that impacts broad segments of software systems. Software codebases must evolve overtime to address changing contexts and adapt to the flux in middlewares and platforms. In the process, it accumulates arbitrary complexities and its maintenance becomes progressively difficult.

Current sustainability approaches focus on the symptoms and tend to be reactive in nature, and ignore the fundamental incentive structures that drive decision-making processes. Moreover, contemporary approaches are insensitive to the uniqueness of each software project context and operate on the assumption that sustainability measurements are universally applicable to the majority of software systems.

This paper introduces a fundamentally novel peer-driven approach to managing software sustainability. The methodology ensures that software teams can define their own sustainability measures that adequately address the unique context of their project and its priorities. These measures are dynamically defined by the project peers to ensure their applicability as the project context evolves. Finally, the paper introduces Susereum, a blockchain platform that materializes the methodology and establishes novel incentive structures to systematically promote software sustainability throughout the project lifecycle.

CCS CONCEPTS

• **Software and its engineering~Software design engineering** • Software and its engineering~Maintaining software.

KEYWORDS

Software Sustainability, Blockchain, Crowd Computing, Distributed Consensus, Distributed Sovereignty, Scientific Software, maintainability, Code Smells, Design Smells.

1. Introduction

Software sustainability is a systematic challenge and not an individual, team, or organizational failings. It impacts many communities including professional software developers, open source communities, and scientists who develop their own research software. This challenge will most likely become even more prominent as software platforms continue this period of flux as evident by the emergence of new computing platforms such as Cyber-Physical Systems, mobile computing, high-performance platforms, and the Internet of Things. Moreover, and as the cost of data capturing and producing devices continue to fall, the size and velocity of the data will continue its exponential growth, along with the codebases necessary to manipulate and analyze such data. This will require advanced software codes and algorithms that are efficient and can execute on emerging and diverse platforms. As such, sustaining such significant intellectual investments in software and cyberinfrastructures will remain a priority for decades to come.

Software becomes unsustainable often due to deficiencies in its design or due to growing arbitrary complexities embedded in its algorithms and codes. Current methodologies often focus on the symptoms as evident by identifications of code smells and quantifications of technical debt. There are three fundamental deficiencies with such methodologies. First, they tend to be reactive; sustainability is only addressed after its symptoms become identifiable in the underlying codebase. Second, they are insensitive to the unique and changing context of software projects. For instance, a specific code smell may not be applicable to a certain codebase underdevelopment or may not be equally applicable to all of its modules. Moreover, such a code smell suitability and severity may change over time. Third, current methodologies ignore the fundamental incentive structures that drive decision making processes for individuals, teams, and organizations.

Another significant dimension is software security and reliability. Deficiencies in designs and the arbitrary complexities obscure software vulnerabilities making critical software elements unsecure in the face of growing cyber threats. Moreover, and as development practices continue to rely on code reuse, vulnerabilities obscured by such complexities can quickly become widely spread and quickly reach global scale. This is evident by the recent unprecedented denial-of-service IoT attack that was caused by large scale reuse of vulnerable code [23].

This paper proposes a fundamentally novel approach to managing software sustainability. Instead of relying on universal sustainability measures, the approach empowers project peers to continuously and dynamically redefine sustainability based on evolving project priorities and context. The approach addresses

CASCON'18, October, 2018, Markham, Ontario Canada
 978-1-4503-0000-0/18/06...$15.00
https://doi.org/10.1145/1234567890

the deficiencies in current incentive structures by creating permanent records of sustainable contributions in a public ledger. This approach proactively promotes sustainability by discouraging unsustainable code contributions.

The rest of the paper is organized as follows. In Section 2, we provide a background covering fundamental sustainability concepts, their deficiencies, and introduce the blockchain technology. In Section 3, we introduce the proposed peer-driven sustainability methodology. In Section 4, we present Susereum, the proposed blockchain sustainability platform. In Section 5, we discuss the potential impact of the proposed approach and platform. Open research questions are discussed in Section 6. We conclude the position paper in Section 7.

2. Background

In this background, we present a brief overview of the fundamentals of contemporary sustainability approaches and identify key deficiencies. We also provide a background on research software as a classical example of software sustainability. Finally, we present a background on Blockchain; the underlying technology for the proposed methodology and platform.

2.1. Contemporary Sustainability Approaches

Software sustainability is often assessed by means of quantifying its symptoms. One broadly recognized notion is ‘Code Smells’ [1][2]. Code Smells represent a surface observation that often suggest the presence of a deeper problem. God Class is an example of such code smell that refers to the presence of a code unit that gains too much control or influence over other elements [3]. Another example of code smells is Inappropriate Intimacy that refers to a Class that accesses internal fields and methods of another class [5]. A similar notion is the concept of ‘Design Smell’, which refers to structures in the design that indicate violation of fundamental principles that may negatively impact design quality and the sustainability of the software system [6]. Some of the common Design Smells include ‘Missing Abstraction’ where elements of data or computations are not assigned to appropriate abstractions [7]. Another related and important notion is Technical Debt (TD) [8]. Technical Debt refers to the effort required to refactor the code or design to remove the undesired characteristics or smells.

2.2. Fundamental Deficiencies

This paper argues that there are three fundamental deficiencies in the available methods to assess and measures software sustainability. First, prevalent sustainability measures are insensitive to the uniqueness of project contexts, platform, technologies, and evolving priorities. One may argue that existing Code Smells and Technical Debt methodologies should be used as guidelines and must be customized for each project. However, this is rarely done particularly for small and medium size software projects, and projects developed by non-professionals such as many segments of research software. For large projects, the measures and their priorities may not be applicable to all modules at all times. Moreover, such guidelines will require significant adaptation and modification to suit a particular domain and context. Even then, such measures will have to be continuously modified to address the evolving context and priorities as they change over time. In the face of this continuous change, such sustainability measures often lose their credibility and relevance, and are frequently, and sometimes intentionally, ignored by the software developers [9].

Second, current methodologies tend to be reactive; they address sustainability concerns after their symptoms become identifiable. For example, methodologies that propose code refactorings to reduce code smells and technical debt requires that such smells and debts exist and become identifiable before the refactorings can be applied. As such, these methodologies focus on the symptoms, and not the elements of the systematic challenge.

Third, Current methods ignore the fundamental incentive structures that drive decision making processes for individuals, teams, and organizations. Software Sustainability is a systematic challenge, and not an individual or team failings. Competing priorities and limited resources often influence decisions in favor of short-term objectives, which often become detrimental for the long-term sustainability of the codebase.

2.3. Sustainability of Research Codebases

Sizable and growing segments of research software begin life in research labs with the sole purpose of generating new research results. This software is often developed by untrained practitioners and transient graduate students and researchers who often lack formal software engineering education or training. Their software contributions are motivated by academic credit and recognition with little regard to the sustainability and evolution of the underlying codebase. Moreover, typical research careers are advanced by publications, with very little recognition to software contributions even if significant impact is achieved. Research software contributions are often not appropriately tracked, cited, or assessed, making it even harder to recognize and credit their authors. These challenges are further exacerbated by the prevalent funding structures that are typically between three to five years. As such, researchers have little incentive to invest effort in sustainability or maintenance beyond the immediate project scope. This context discourages upfront designing efforts and further exacerbates the problem.

Another challenge is Platform Dependence. We are in a period of flux in both hardware and middleware. In this environment, without platform independence and appropriate abstractions, software systems can become quickly obsolete. As such, research software code represents a classical example for software sustainability.

2.4. Blockchain Technology

A blockchain is a list of records organized in blocks. Blocks are linked to each other and are often secured by strong cryptography [10][11]. Each block contains a certain amount of information, or 'transactions', which are validated by consensus. Blockchain is typically managed by a peer-to-peer network and adhere to the blockchain protocol. Blockchains are open, secure, and are resistant to change by design. Once information is encoded in a block, it becomes extremely difficult to retroactively change the information without changing all subsequent blocks. Such a change will also require collusion from the majority of peers in the network. As such, the more peers involved in the network, the more reliable and secure the network becomes without the need for a central authority. Blockchain is the underlying technology of cryptocurrencies such as bitcoin and Ethereum [12]. The technology has demonstrated early potential in broad and diverse disciplines, including in many aspects of software engineering [13][14][15].

Blockchain protocols define all related mechanisms of the network, such as how blocks are created, how transactions are verified and recorded, etc. For this paper, two important concepts are particularly relevant; Proof of Stake (PoS) and Proof of Work (PoW). Both PoS and PoW are mechanisms to determine how to assign transaction validation and the creation of new blocks to individuals (or peers) in the network. PoS selects the individual using the amount of stake that individual has in the network. For example, in the case of cryptocurrencies, the stake is defined by the amount of coins an individual owns. PoW uses often computationally-heavy problems and requires that the problem be solved in order for a new block to be created. Many blockchain protocols define variations of PoS, PoW and/or add randomness to ensure balanced distribution of validation work amongst the network peers.

3. Proposed Peer-Driven Sustainability Methodology

We argue that the methodology to identify sustainability measures must be both dynamic and peer-driven; dynamic so it evolves as the project context and priorities change over time, and peer-driven to ensure that measures are perceived as valid and are trusted by the project peers. This enhances the likelihood that such measures are more sensitive to each project uniqueness and increases the likelihood of adoption and adherence by peers and project stakeholders. These measures, while unique to each project, must maintain a valid perception even by members not belonging to the immediate software project under development. In open source projects for example, it is common that contributors commit code fragments to multiple codebases. Sustainability measures must translate to uniform quantifications that are acceptable to the broader community of developers. This is also to ensure that software developers –regardless of the specific projects they contribute to– can build a reputation and claim credit for their sustainable contributions. This in fact is extremely important for many communities, take for example the research community. Researchers are often motivated by contributing research articles because such articles can represent permanent evidence of contribution to knowledge and science. However, scientists who contribute code do not receive any comparable credit, even if the code they contributed is broadly adopted and achieved significant impact. This is because it is much harder to track, evaluate, and assess such contributions. As such, creating a public record using uniform measures becomes crucial.

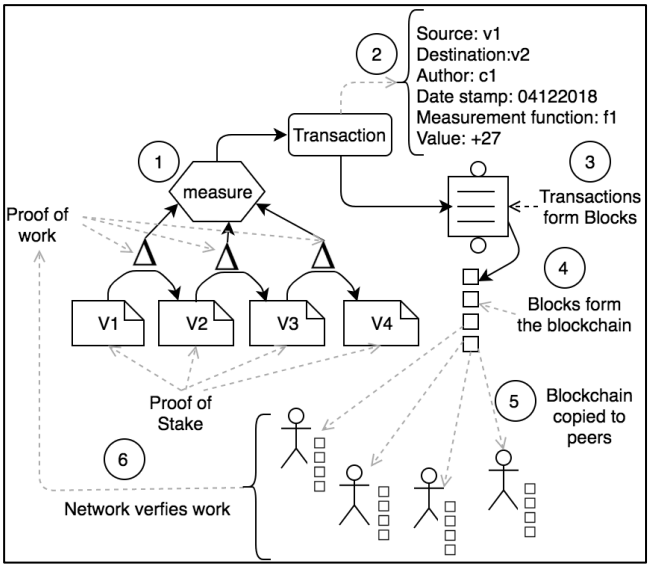

Figure 1: Overview of the technical approach

4. Approach Overview

To demonstrate the technical approach, assume an open source project that accepts external contributions. With each code modification or contribution, the repository version is incremented. For simplicity, we assume there are no code branches or merges, and that all contributions are sequential. Functional validity of contributions is managed by unit and system tests and are not within scope of this approach. Rather, the focus is to incentivize and quantify the health and sustainability of such contributions, and ultimately promote sustainable software development practices. Figure 1 above illustrates the proposed protocol's six steps.

Step One: Occurrence of a code modification. When the codebase is modified, the network applies health measures functions to estimate the improvement or declination of the overall health of the codebase as a result of this modification.

Step Two: Transaction formation. A transaction is identified by the source version, destination version, the author of the contribution, the date, the specific measurement function(s) that was applied to assess code health, and the resulting value; a positive value indicates health improvement and a negative value indicates declinations in overall health. Code contributors/developers can accumulate credit from one or more projects that they contribute to.

Step Three: Block Formation. Bocks contain a predefined number of transactions. The specific number of transactions is based on the size of the code and frequency of the contributions and code changes. Blocks are created using PoS and PoW as discussed earlier. Stake is defined by the size of the cumulative code contributions by the individual in the network. PoW is defined by the required computations to assess the resulting sustainability for the unit, module and the entire codebase. These computations may be significant based on the complexity of the sustainability measure and the size of the underlying codebase. As such, the required computations (PoW) results in useful information for the project under development. This is in contrast to the predominant PoW in cryptocurrencies where solving the crypto problem is required but provides only minimal value to the network.

Step Four: Chain Formation. The chain is formed by appending the newly created block to the chain. As with other blockchains, each block contains a link to the previous and next block in the chain. Blocks and their transactions are immutable.

Step Five: Chain Distribution. Individuals in the network maintain copies of the chain in similar fashion to their copies of the codebase. The same infrastructure repositories can be reused for storing the chain, such as GIT and SVN repositories.

Step Six: Verification. Individuals in the network with copies of the full chain can verify the work (assessments of code contributions sustainability). Individuals in the network can also propose new health measures. Through voting (discussed in the next Section), individuals can vote on adopting new or modifying existing health measures. This ensures that health measure functions are customizable to each project and its evolving context.

5. Proposed Protocol

The premise of the peer-driven sustainability approach is based on the notions that 1) peers who contribute code are uniquely equipped to define appropriate sustainability measures, and 2) the approach ensures that the measures are sensitive to each project context and are evolvable as the project code and context changes. As such, the methodology must support key aspects including a mechanism to allow peers to 1) propose new, modify, or reuse existing sustainability measures, 2) build consensus by voting on measure proposals, and 3) aggregate measures to form cohesive complete sustainability quantifications. The specific mechanism that facilitate proposals, voting, and aggregation are made

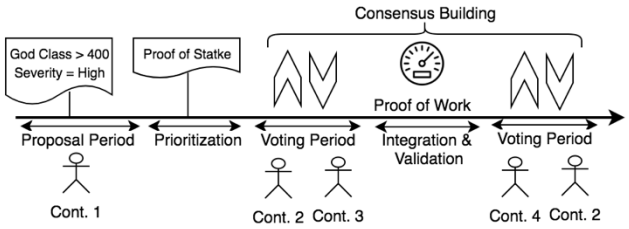

Figure 2: Peer Driven Sustainability Quantification Protocol

available by the underlying blockchain protocol and supporting platform. The following demonstrates the main outlines of the protocol as summarized in Figure 2.

The methodology is based on a continuous timeline covering the entire project lifecycle, from inception and including maintenance and sustenance phases. Any code contributor or stakeholder can propose new measure or propose a modification to an existing measure. This is followed by a period of voting by any other peer (excluding the peer that proposed the change). Peers can vote up or down, and their vote can be weighted based on their level of stake in the network or codebase. If the proposal is voted for inclusion, a single peer in the network will perform the work of integration and validation of the new measure, as well as performing the necessary calculations to assess current sustainability of the entire codebase. This task forms the Proof of Work (PoW) in the blockchain as discussed earlier. The protocol may allow multiple voting periods to ensure consensus by allowing peers to vote after the measure has been integrated. This protocol is supported by the underlying blockchain platform, as discussed in the next Section.

5.1 Managing Subjectivity and Uncertainty

Developing a sustainable software is both subjective and highly unpredictable process. First, engineers and software developers are not certain about how the software may evolve, what future platforms it may need to support, the nature of future technology it may adopt, or the new priorities that may emerge.

Second, and since peers can propose new or modify existing measures, engineers will have little knowledge as to how the measures may evolve over time. The peer-drive process as proposed is inherently unpredictable.

Third, there is a significant delay from the time a measure is proposed to the time the impact of that measures are reflected in current and future sustainability quantifications. As such, it is very likely that proposed and adopted measure to exaggerate software (un)sustainability. Given that the process is highly subjective, peers who may have voted for a measure may wish to reverse their vote after the hindsight knowledge.

To manage this subjectivity and uncertainty, the protocol incorporates the following features. First, it allows for unbounded voting sessions on any measure. This allows peers to revote on a measure after the quantification information becomes available. Second, the protocol is based on continuous time line. This means

that measures are likely to continuously change as the codebase evolves. Third, the protocol uses PoW in addition to PoS. Since any peer regardless of their stake can perform the required calculations for PoW, it means it is less likely that only a few privileged contributors may manipulate measure formulations. In essence, this ensure distributed sovereignty, rather than a concentrated one.

5.2 Fundamental Research Questions

To materialize this protocol, many fundamental research questions must be addressed, including whether such protocol could eliminate the need for an external third-party entity to manage the formalization, definition, and application of sustainability measures. How to aggregate unit measures to form holistic complete measures? and whether the resulting holistic measures result in adequate assessment of sustainability? Will such a process develop new measures or will peers be satisfied with reusing existing measures?

Another set of fundamental questions is related to the potential impact of implementing such a protocol. Will the protocol result in an overhead that discourages engineers from contributing code fragments? Or, will the protocol encourage contributions, particularly external contributions, since credit will be permanently awarded? Can this protocol function as the foundation for compensating contributors?

In large projects, there may be a need to prioritize proposals if more than one proposal becomes active at end of a voting period. As such, how to prioritize such proposals. Using the contributors' level of stake may centralize too much power within a handful of peers, possibly alienating others and ultimately reducing trust in validity of the process and the quantification measures. In the next Section, we introduce Susereum, the proposed platform that addresses some of these fundamental questions.

6. Susereum

Susereum [21], the proposed blockchain platform, aims at establishing novel incentives and rewards structure. Its name is derived from the words 'sustainability' and 'ether'. The unit of credit in this platform is 'Suse'. Susereum enables greater insights into the quality, sustainability, and impact of code contributions. It credits authors of sustainable code by providing a permanent record of their contributions. Consider the research and scientific sphere where academic articles serve as the primary evidence for contributions and are permanently credited to their authors. Susereum credits researchers of their code contributions in a similar fashion. Transactions in Susereum blocks are calculated based on the sustainability of the code contributions using measures that themselves are defined by the peers.

The primary goal of Susereum is to create and accumulate knowledge about various aspects of software sustainability and make the knowledge publicly available. Specifically, Susereum aims at: 1) establishing incentive structure to promote contributions of sustainable codes. 2) building trust within communities to reuse and extend (rather than create new) software codes. This is because engineers can refer to the sustainability rating as means to assess code quality. 3) promoting accountability so that publicly or privately funded research delivers accessible, usable, reusable, extendable, and sustainable research codes. This is because the funding agency can establish minimum sustainability requirements without having to specify how sustainability is measured; particularly since sustainability is both subjective and uncertain as discussed in Section 5.1.

6.1 Components of Susereum

Susereum is designed to be purely peer-to-peer distributed public ledger. Therefore, blockchain is identified to be the platform for development of Susereum. In crypto-currency applications, credit is mined by solving complex cryptographic problems. In Susereum, credit (Suse) is mined when contributions to a listed code repository result in improvement in the repository or module code sustainability as calculated by the sustainability measures that are active at the time of code contribution. Instead of using unproductive crypto-problem, miners in Susereum can demonstrate PoW by calculating sustainability of components and codebases using the referenced measures. Because transactions in a blockchain are permanent, code contributors are awarded credit even if a) their code contributions are modified or removed altogether as the software evolves, or b) the sustainability measures are modified or removed and may be no longer active.

Susereum platform is developed using Openchain [16] and Quorum [17]; both are open source blockchain development platforms. Openchain facilitates the development of the underlying distributed ledger, and Quorum provides facilities for enforcing smart contracts. For Susereum, the smart contract is encoded using the sustainability measures as discussed earlier.

6.2 Extensibility Points and Forking

Susereum supports two mechanisms to reuse or extend sustainability measures. For simple measures, peers may opt to reuse existing measures from the library of measures (discussed in next Section). Alternatively, the platform supports code injection; where users can inject new code (or modify the existing code) to implement these measures.

A second mechanism involves development of a simple Domain Specific Language (DSL) to allow for the specification of sustainability measures. The DSL will generate the code required for performing the calculations. The benefits of the development of the DSL is twofold. First, the DSL reduces barriers for external contributions; a contributor does not need to learn the base programming language to propose new measures. Second, the DSL makes measure specifications independent of the base language. As such, measures specification is uniform regardless of the underlying language and technology. All contributed DSLs and their revisions are stored as part of the sustainability measurement library.

Forking is another extensibility mechanism to ensure flexibility in the platform. Currently, each Susereum block contains a capped number of transactions that represent repository changes or code contributions. Susereum allows for hard-forking which may occur when a need emerges for changing the protocol or the block structure. We expect that hard-forking may occur more frequently at early stages (or to address unique needs in certain domains) and stabilizes as the community agrees on block structures and protocol.

6.3 Sustainability Measurement Library

Users of the platform can reuse existing or define new sustainability measures. The goal of Susereum sustainability library is to accumulate all such measures and organizes them in a way to make them available for reuse. The library also stores the voting and adoption data of such measures. This library is an important source of data for analysing how such measures are defined, how they evolve, their reuse patterns, and what characteristics that may affect their broad adoption.

The library includes a recommender module that can propose specific measures to contributors based on the characteristics of the underlying codebase. The module will leverage historical adoption data of past measures in the process.

7. Potential Impact

The potential impact of improving software sustainability on the cost and reliability of many software and software-intensive systems is significant. We discuss the potential impacts of the proposed approach and supporting platforms on Open Source, professional spheres, and scientific research software sphere, as well as the potential impacts beyond software sustainability.

7.1 Open Source and Professional Impacts

Many open source software emerges organically with little structure and with diverse contributors. Often, these projects experience high contributors’ turnover rates. As a result, ensuring high-quality codebase is a challenge. Imposing strict coding standards is often unpractical for many reasons. Diverse contributors’ base often has diverse experiences and expertise. Secondly, open source projects often aim at reducing barriers for external contributions; enforcing coding standards is undesirable.

Focusing on incentives means that contributors with high expertise will find the rewards for contributing high-quality sustainable code, without rejecting contributions from the broader pool of potential contributors who may not be seeing the credit. The new reward system will also incentivise contributors to refactor existing code elements to gain more credit.

Another important potential impact is promoting reuse, rather than creating new, code elements. Because the public ledger will make available information on software sustainability assessments, it will promote trust code fragments, and will increase the chances for code reuse.

In many professional software development cases, code is not made publicly available. The proposed methodology and platform do not require publicising the code segments, but rather, data related to their sustainability. Moreover, the blockchain can be hosted and stored in closed environment, without being shared publicly.

7.2 Scientific and Research Software Impacts

Scientific and research codes are a classical example for unsustainable codebases due to many factors, some of which have been discussed earlier. As a result, many public funding agencies have fundamentally changed their funding structures to promote software sustainability and its broader disseminations. For example, large funded projects are often mandated to support professional software engineer to ensure that deliverable software elements adhere to fundamental software engineering standards.

The proposed methodology and platform will enable funding agencies to mandate minimum levels of sustainability. Such a mandate is ‘outcome oriented’ as it does not mandate specific sustainability mechanisms or measures to quantify it. Ultimately, sustainable research software will enable further dissemination of research findings, results reproduction, and results extension by the broader community.

7.3 Discovering and Sharing of Novel Sustainability Measures

Current sustainability measures are insensitive to the codebase context, the development processes, expertise of the developers, the development technologies used, and the application domains. Contemporary measures assume universality. However, and as platforms and middlewares go through this period of flux, universality is questionable. As one example, software running on a small handheld device will necessarily have different characteristics and unique priorities to software running on a high-performance platform. As such, sustainability measures that are applicable to mobile applications with various power and processing constraints will most likely not be appropriate when such limitations are relaxed.

An important aspect of Susereum is accumulating data about appropriate sustainability measurements. As discussed in Section 6.3, Susereum includes a public library to store data about the developed sustainability measurements. As the community of Susereum users grows, it will become possible to understand what characteristics that makes some measures more appropriate than others. The library will also uncover patterns of use and reuse of sustainability measures. This will potentially unveil novel measures that emerges from the peer driven process.

7.4 Decentralized Sovereignty and Impacts Beyond Software Sustainability

The essence of the proposed approach and blockchain protocol is an empowerment of peers to define their own quality and sustainability metrics. In essence, this is a form of decentralization

of sovereignty. As argued in this paper, this novel methodology is potentially transformative to how software sustainability is assessed and the role project peers play in the identification and formulation of the sustainability metrics.

Beyond software sustainability and its metrics, project peers should be empowered to dynamically define their development processes, artefacts formalisms, versioning merging and conflict resolution of conflicting artefacts. Taken a step further, peers may be empowered to define project tasks priorities, scheduling, effort estimates, and task assignments. Such empowerment could mean that development of software system could become significantly more adaptable as the codebase size, context, and complexity continue to evolve throughout the project lifecycle.

In crowd-based software engineering [18][19][20], development tasks are delegated to broad members of the general public. this emerging development methodology has many promises, including significant reduction in costs and increased elasticity in development capacity. A key challenge in such approaches is managing incentives, rewards, and quality assurance. The proposed concept of decentralization of sovereignty could potentially include members of the crowd. In other terms, empowerment of the peers could extend to become empowerment of the crow. As such, tasks related to verification, validation and quality assurance could themselves be delegated to the crowd.

8. Fundamental Research and Open Questions

Materializing the proposed approach and platform opens the following research questions.

First, how to aggregate the unit sustainability measures contributed by the peers into wholistic measures that are applicable to the entire codebase. The unit sustainability measures are likely to focus on aspects related to a small subset of the codebase. Their aggregation will be challenging for two reasons. First, there is a chance that such measures are in conflict; their aggregation may result in inadequate sustainability assessments. Second, how to assess priorities for unit measures that are not in conflict. It is likely that peers will often view their unit measures to be more important than others.

Second, how to empower peers to propose and modify appropriate sustainability measures. In many cases, peers do not have a broader view of the entire codebase which may negatively impact their individual assessment. Moreover, how to enable peers to comprehend the potential impact of their proposed measures with the uncertainty of how these measures maybe aggregated and prioritized.

Another important set of questions are related to what types of measures that may emerge from the peer-driven process. In what ways will the expertise of the contributors affect the quality of the measures. Will such measures adequately reflect codebase quality? And, will such measures be project-specific, or will they become largely transferable to other projects. A related set of questions is how the peers and project stakeholders perceive the validity of such measures.

The proposed blockchain protocol and structure is largely arbitrary. There are open questions to identify the most appropriate protocol and structure, and whether a single protocol/structure maybe sufficient or the need may exist for a broad and diverse protocols and structures.

Finally, how to measure the impacts of the new incentives on peers' motivations and ultimately their impact on software sustainability? To what extent will the proposed platform affect individuals, teams, and organizational decisions.

9. Conclusion

Software sustainability is a systematic challenge, and not a manifestation of an individual or team fallings. This challenge will most likely become more prominent as software systems continue their exponential increase in complexity and diversity.

Current methodologies focus on the symptoms and largely fail to address the fundamental incentive structures that drives individual, team, and organization decision making processes. More importantly, current methodologies are insensitive to the unique nature of project contexts and priorities. They operate under the assumption that sustainability measures and assessments are universally applicable to the majority of software projects. The validity of this assumption is increasingly becoming weakened as platforms and middleware go through this period of flux. Moreover, prevailing methodologies fail to adapt as the codebase evolves over time. As a result, they often become irrelevant and are increasingly perceived as invalid.

To address these fundamental challenges, the paper proposes a novel sustainability methodology that is driven by project peers. The paper also proposes Susereum, a novel blockchain platform to materialize the proposed methodology. Susereum empowers project peers to dynamically define sustainability measures. Susereum aims at restructuring incentives by creating permanent public ledger of sustainable contributions. In effect, Susereum creates permanent credit for code contributors. The potential impacts of this methodology are transformative for open source and professional spheres, as well as scientific and research software codes.

The new methodology opens new fundamental research questions, including how to aggregate unit measures into wholistic ones that assess the entire code base sustainability, and to what extent will the peer-driven methodology result in valid trusted sustainability measures.

The proposed methodology highlights the concept of distributed sovereignty. In the future, this concept could extend to include other aspects of software engineering well beyond sustainability.

REFERENCES

[1] Palomba, Fabio, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrea De Lucia. "Do they really smell bad? a study on developers' perception of bad code smells." In Software maintenance and evolution (ICSME), 2014 IEEE international conf' on, pp. 101-110. IEEE, 2014.

[2] Nakagawa, Elisa Yumi, Rafael Capilla, Eoin Woods, and Philippe Kruchten. "Sustainability and Longevity of Systems and Architectures." Journal of Systems and Software, (2018).

[3] Santos, José Amancio M., and Manoel G. de Mendonça. "Exploring decision drivers on god class detection in three controlled experiments." In Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 1472-1479. ACM, 2015.

[4] Mantyla, Mika, Jari Vanhanen, and Casper Lassenius. "A taxonomy and an initial empirical study of bad smells in code." In *Software Maintenance, 2003. ICSM 2003. Procs. Int’l Conference on*, pp. 381-384. IEEE, 2003.

[5] Gupta, Aakanshi, Bharti Suri, and Sanjay Misra. "A Systematic Literature Review: Code Bad Smells in Java Source Code." In International Conference on Computational Science and Its Applications, pp. 665-682. Springer, Cham, 2017.

[6] Suryanarayana, Girish, Ganesh Samarthyam, and Tushar Sharma. Refactoring for software design smells: managing technical debt. Morgan Kaufmann, 2014.

[7] Chaudron, Michel RV, Brian Katumba, and Xuxin Ran. "Automated Prioritization of Metrics-Based Design Flaws in UML Class Diagrams." In Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on, pp. 369-376. IEEE, 2014.

[8] Li, Zengyang, Paris Avgeriou, and Peng Liang. "A systematic mapping study on technical debt and its management." Journal of Systems and Software 101 (2015): 193-220.

[9] Huang, Qiao, Emad Shihab, Xin Xia, David Lo, and Shanping Li. "Identifying self-admitted technical debt in open source projects using text mining." Empirical Software Engineering 23, no. 1 (2018): 418-451.

[10] Zheng, Zibin, Shaoan Xie, Hong-Ning Dai, and Huaimin Wang. "Blockchain challenges and opportunities: A survey." Work Pap.-2016.

[11] Cachin, Christian, Marko Vukolic Sorniotti, and Thomas Weigold. "Blockchain, cryptography, and consensus." (2016).

[12] Narayanan, Arvind, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press, 2016.

[13] Porru, Simone, Andrea Pinna, Michele Marchesi, and Roberto Tonelli. "Blockchain-oriented software engineering: challenges and new directions." In Proceedings of the 39th International Conference on Software Engineering Companion, pp. 169-171. IEEE Press, 2017.

[14] Destefanis, Giuseppe, Michele Marchesi, Marco Ortu, Roberto Tonelli, Andrea Bracciali, and Robert Hierons. "Smart contracts vulnerabilities: a call for blockchain software engineering?." In Blockchain Oriented SE (IWBOSE), 2018 International Workshop on, pp. 19-25. IEEE, 2018.

[15] Herbert, Jeff, and Alan Litchfield. "A novel method for decentralised peer-to-peer software license validation using cryptocurrency blockchain technology." In Proceedings of the 38th Australasian Computer Science Conference (ACSC 2015), vol. 27, p. 30. 2015.

[16] Knirsch, Fabian, Andreas Unterweger, and Dominik Engel. "Privacy-preserving blockchain-based electric vehicle charging with dynamic tariff decisions." Comp Sci’ -Research and Dev’ 33, no. 1-2 (2018): 71-79.

[17] Sankar, Lakshmi Siva, M. Sindhu, and M. Sethumadhavan. "Survey of consensus protocols on blockchain applications." In Advanced Comp’ and Comm’ Sys’ (ICACCS), 2017 4th In’l Conf on, pp. 1-5. IEEE, 2017.

[18] Groen, Eduard C., Joerg Doerr, and Sebastian Adam. "Towards crowd-based requirements engineering a research preview." In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pp. 247-253. Springer, Cham, 2015.

[19] Mao, Ke, Licia Capra, Mark Harman, and Yue Jia. "A survey of the use of crowdsourcing in software engineering." *Journal of Systems and Software* 126 (2017): 57-84.

[20] Harel, David, Idan Heimlich, Rami Marelly, and Assaf Marron. "Crowd-based programming for reactive systems." In *CrowdSourcing in Software Engineering (CSI-SE), 2017 IEEE/ACM 4th International Workshop on*, pp. 9-13. IEEE, 2017.

[21] Omar Badreddin. Susereum Public Web Page. Available: www.susereum.com. Accessed August 2018.

[22] Quantum GIS (QGIS) Development Team. "QGIS Geographic Information System. Open Source Geospatial Foundation Project." (2012). Available: https://github.com/qgis/QGIS

[23] How the Internet of Things Took Down the Internet. Accessed Dec. 2017. Available: https://www.technologyreview.com/s/602713/how-the-internet-of-things-took-down-the-internet/

About the Author

Dr. Omar Badreddin is a software design professional and researcher. He has authored many books and scientific articles on software design and sustainability. He has contributed to many prominent open and closed source software that has reached global userbase, including the Eclipse platform. He is an active open source contributor and an advocate for sustainable software engineering practices. Dr. Badreddin is the primary author of Susereum (www.Susereum.com), the blockchain platform that establishes permanent credit for code authors and contributors. Susereum’s vision is to fundamentally transform software sustainability by tackling the incentive structures of individuals, teams, and organizations.