2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science)

Susereum: Towards a Reward Structure for Sustainable Scientific Research Software

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

Wahab Hamou-Lhadj
Concordia University, Montreal
Quebec, Canada
wahab.hamou-lhadj@concordia.ca

Swapnil Chauhan
University of Texas, El Paso
Texas, U.S.A
sschauhan@miners.utep.edu

***Abstract*—Research software has opened up new pathways of discovery in many and diverse disciplines. This research software is developed under unique budgetary and schedule constraints. Its development is driven by knowledge discovery goals often without documented requirements. As a result, the software code quality is impacted which often hinders its sustainability beyond the immediate research goals. More importantly, the prevalent reward structures favor contributions in terms of research articles and systematically undervalues research codes contributions. As a result, researchers and funding agencies do not allocate appropriate efforts or resources to the development, sustenance, and dissemination of research codebases. This paper presents Susereum, a Blockchain based platform that aims at achieving two goals. First, restructuring prevalent incentives by awarding permanent immutable credit to research code authors similar to the credit awarded to the authors of scientific articles. Second, distributing sovereignty by empowering peers through a consensus process to define code sustainability and impact metrics.**

Keywords*—*Susereum, software sustainability, research codes, Blockchain, consensus algorithm, distributed sovereignty.

I. Introduction

Research software has, and continues to, open up fundamentally new pathways of discovery in broad disciplines of research. In the course of inquiry and discovery, many researchers, postdocs and graduate students have become software developers and practitioners. They develop research software under unique budgetary and schedule constraints. The workforce is highly transient and often practice software development without formal training or education. They adopt discovery-driven development processes to achieve their research goals and objectives.

This research software is unique, not only because of how its developed and the nature of its workforce. It is also unique because the prevalent academic reward structure systematically undermines its sustainability. Principle investigators document, preserve, disseminate their research findings in scientific articles. The impact and scientific value of those articles are relatively well-understood and are broadly accepted and recognized by the respective communities. Unfortunately, research software does not enjoy a comparable status, even in cases where the software may achieve broader impacts. This results in reward structures and incentives that systematically undermines the quality and sustainability of research software. Consequently, the significant intellectual investments by domain experts in the development of research software does not achieve its potential impacts and returns. The specialized domain knowledge embodied in the software codes and algorithms become inaccessible due to the growing arbitrary complexities. As a result, reproduction and extension of research findings become significantly hampered and the dissemination of knowledge is greatly restrained.

This is a multi-facet challenge that includes preparation and trainings in software development, effective collaboration between researchers and professional engineers, as well as fundamental challenges in the quantification of research software impact. Recognizing this challenge, many research funding agencies has adopted a number of initiatives, including: 1) requiring that software elements be made publicly accessible; 2) mandating explicit dissemination activities, and more recently, 3) requiring that budgets include support for professional software engineers to aid in the development tasks. These initiatives, while potentially effective, they do not address the fundamental deficiencies in academic ecosystems that have evolved over hundreds of years. These ecosystems systematically promote contributions of peer reviewed articles and overlook the emerging reality that many articles have become software-dependent. Unfortunately, it is very difficult to subject software codes to the same peer-review processes of scientific articles for many reasons.

This paper presents Susereum, a Blockchain based platform that rewards scientific research code developers with immutable credit for their research code contributions. The platform adopts a peer-driven consensus protocol that is similar to how academic articles are reviewed and evaluated. Susereum is a functioning open source platform that awards 'Suse' as credit for contributions.

II. Background

We discuss background related to the context in which scientific research software is developed. This is followed by a review of fundamental software sustainability quantification approaches.

A. Research Software

Research software codes embody knowledge and algorithms pertaining to specialized domains, processes, algorithms, and data representations. Many of these software codes do not adhere to software engineering best practices and continue to degrade in quality as they evolve. By the time some of these

978-1-7281-2276-2/19/$31.00 ©2019 IEEE
DOI 10.1109/SE4Science.2019.00015

software systems reach the broader communities, their code size and complexity limit their reusability; consequently, limiting the ability to reproduce and extend research results. Code complexities obscure knowledge, design decisions, goals, and data representations, further limiting the dissemination of the software and the knowledge embodied in its codes. These current practices are detrimental to the scientific software sustainability and is a major limiting factor for dissemination of knowledge. This challenge is echoed in the recent National Science Foundation (NSF) report on "Computational Science and Engineering Software Sustainability and Productivity (CSESSP) Challenges" [1].

B. *Software Sustainability Quantification*

Software sustainability is often assessed by means of quantifying adverse symptoms. One broadly recognized notion is 'Code Smells' [11][12]. 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 [13]. A similar code smell is 'Large Class' that refers to a single code unit that has become too big and complex to sustainably maintain. Such smells often have arbitrary thresholds above which the code is flagged with the smell. For example, a 'Large Class' smell is identified typically when the class lines of code exceed 750 lines [9]. Similarly, 'large Parameter List' smell refers to methods typically with more than five parameters. A related notion is the concept of 'Design Smell', which refers to structures in the design that indicate a violation of fundamental principles that may negatively impact design quality and the sustainability of the software system [14]. Some of the common Design Smells include 'Missing Abstraction' where elements of data or computations are not assigned to appropriate abstractions [15]. Another related and important notion is Technical Debt (TD) [16]. Technical Debt refers to the effort required to refactor the code or design to remove or minimize the undesired characteristics or smells.

Sustainability quantification methodologies that are based on measures of code and design smells suffer from key fundamental limitations. First, such methodologies are reactive in nature as they are dependent on the symptoms to manifest and become identifiable in the code with little upfront guidance to avoid introducing the smells from the first place. Second, such measures are based on arbitrary thresholds that largely ignores the broad and diverse development technologies, individual and team priorities, and the diverse and changing project contexts. A software designed for low-power IoT devise will inevitably have unique characteristics compared to software running on specialized high-performance computing platform. Third, current sustainability quantification methods ignore the fundamental incentive structures that drives decision making processes for individuals, teams, and organizations.

III. SUSEREUM PLATFORM

Susereum is a purely distributed Blockchain based platform. Susereum's goal is to restructure prevalent incentives to reward authors of scientific research codes. This is achieved by rewarding research code authors with immutable credit for their research code contributions. In this section, we present Susereum's operational steps. A fundamental element of Susereum is the consensus protocol that is at the core of the sustainability quantification. We discuss the consensus protocol in the following section.

A. *Susereum's Openrational Steps*

Susereum's steps initiates whenever there is a change in the research codebase (Fig. 1). Susereum measures the resulting codebase sustainability using metrics formulated as a result of the consensus protocol (discussed next). The result of this quantification forms a transaction, which contains information pertaining to the source and destination revisions, the author, and the associated quantification metrics. Each transaction translates to a credit which can be either positive or negative. Positive credit indicates an improvement in the codebase sustainability.

A fixed number of transactions are combined together into patches. Similarly, a fixed number of patches forms blocks in the chain. The chain is distributed to all peers in the network (i.e. developers in the open source project) along with the codebase. Peers can verify transactions in a fashion similar to how transactions are verified in Blockchains. These operational steps are depicted in Fig. 1.

B. *Proof of Work and Proof of Stake*

Proof of Work (PoW) and Proof of Stake (PoS) are fundamental protocols in Blockchains where they are used to formalize which peer in the network is awarded credit for creation of new Blocks. To incentivize participation, Blockchains often award credit to peers that are selected for the creation of new blocks. This is typically achieved through a protocol known as Proof of Work (PoW). Unfortunately, PoW requires extensive computations and consumes significant resources to demonstrate the evidence for having done the 'work' (in Bitcoin, this is the work required to solve a complex crypto problem). Proof of Stake (PoS) has emerged to achieve the same goal without requiring the significant work to be performed, especially that the work is typically unproductive and does not serve any meaningful purpose beyond the determination of the privileged peer. PoS selects peers deterministically based on their stake in the network.

Susereum uses both PoW and PoS as follows. PoS is used to deterministically select a peer based on their stake in the network (lines of code committed by the peer). Once the peer is selected, the peer must demonstrate PoW by performing the calculations required by the sustainability quantification module. This work is both productive and needed to calculate the baseline for future credit awards. In Susereum, PoW and PoS do not create credit (or Suse). Credit is only created when the code base sustainability is improved.

C. *Susreum Design and Strcuture*

Susereum runs in the cloud with zero installation footprint. Researchers can use Susereum without changing their development infrastructures or tools. Susereum currently provides seamless integration with GitHub repository with plans to extend integration with other public code repositories.

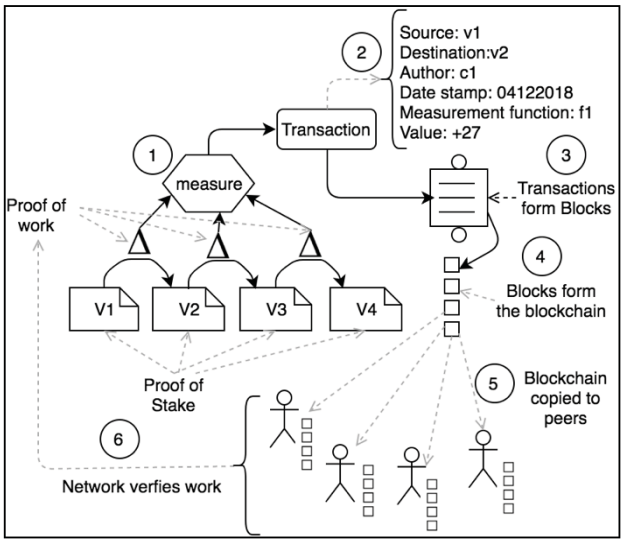

Fig. 1: Susereum Operational Steps [10]

Metrics Specification Language. Susereum uses a Domain Specific Language (DSL) to define sustainability metrics. The DSL defines quantification metrics and their scope. The scope can be a specific code unit (i.e. a method or a class), a module, or a universal scope. The language is defined using Tom's Obvious Minimal Language (TOML) [17]

Consensus Module. This module serves two key purposes. First, it empowers peers to continuously redefine appropriate sustainability metrics that are most suited to their project unique context. Second, it ensures that these metrics are defined through a process of consensus. The use of this module is optional for those teams who do not wish to modify the default sustainability metrics. It is possible for small research teams to use the default metrics without ever participating in this consensus process. Fig. 2 illustrates this consensus protocol.

The protocol is based on a continuous timeline covering the entire project lifecycle. Susereum's peers can propose new metrics or propose a modification to an existing metric. 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 for the metric inclusion. If the proposal is voted for inclusion, a single peer 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 constitutes the Proof of Work (PoW) in the blockchain as discussed earlier. The protocol allows for a second

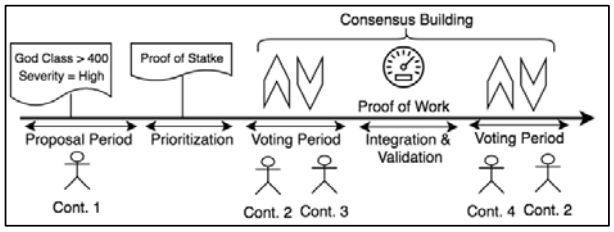

Fig. 2: Consensus Protocol [10]

voting period to so that peers can recast their vote based on the new knowledge about the impact of the new sustainability metric.

Code Analysis Integration Module. Susereum integrates with *SonarQube* and *SourceMeter* code analysis tools to perform code analysis. These two code analysis tools run on Susereum's server and the integration is seamless to the end user. This module also performs pre and post processing for the analysis results. This is because metrics provided by the analysis tools are often very different than the metrics defined through the consensus protocol.

Suse Quantification Module. This module takes the metrics values and coverts them into crypto units, Suse. The calculation is performed following a series of calculations as discussed below.

$$Suse_{\Delta code} = F(Sus)_{after} - F(Sus)_{before}/F(Sus)_{before} \quad (1)$$

That is, the amount of Suse awarded is measured by the change in the sustainability of the entire codebase as a result of the code change.

$$F(Sus)_{module} = rw - \frac{\sum_{i=1}^{n}(cm - Ac_{module})^2}{Ac^2} * rw \quad (2)$$

That is, the sustainability of a code module is the square difference between a code metric (cm) and the actual value of the code module under evaluation (Ac_{module}). Where n is the number of code metrics and *rw* is a reward constant set by default to 100. As such, sustainability is reduced as code modules deviate further from code metrics.

$$F(Sus)_{codebase} = \sum_{i=1}^{j}[F(Sus)_{module}] \quad (3)$$

That is, the sustainability of the entire codebase is measured by the summation of the sustainability of each code module.

D. Uniform Research Software Citation

Uniform citations is fundamental for appropriate attribution of credit and for the assessment of contributions impact. Unlike code, academic papers enjoy broadly adopted and uniform citations. Impact of papers is therefore quantifiable and is relatively well understood.

Uniform citation for research software maybe straightforward only for single investigators and small teams. Uniform citations become more challenging for large long-living projects particularly as the development teams change overtime. More importantly, individual code contributors to a research codebase is often untraceable and its quality and its future potential impact is largely unknown.

Susereum aims to address this challenge by providing a uniform immutable citation for the research codebase at any moment in time. Susereum orders all code authors by their level of stake (or Suse) in the codebase. For codebases with large and changing contributors, citations becomes more dynamic as it changes over time. While citations for academic papers is fixed over time, Susereum generates a unique time-stamped citation at any moment of time.

IV. Related Initiatives

The Networking and Information Technology Research and Development (NITRD) workshop report on Computational Science and Engineering Software Sustainability and Productivity Challenges has identified a number of key gaps related to the development of sustainable scientific software [2]. The report concludes that research software is developed in unique context and under unique constraints that standard software engineering best practices are not applicable without significant modifications. This has motivated the establishment of some key initiatives to address research software sustainability. One notable example is the "US Research Software Sustainability Institute" (URSSI) which is established with the goal of serving as a community hub and providing services to scientists that will help them create improved, more sustainable research software [3]. Similar initiatives have been established outside of the U.S. For example, Software Sustainability Institute has been recently established in the U.K aiming at gathering intelligence from researchers and offering multiple programs and fellowships for researchers [4].

There are ongoing efforts that explores applications of blockchain technologies to reform sustainability of the scientific research [5]. *Scienceroot* aims at addressing deficiencies in the status quo and the extended time durations required to attain funding, collaborate, and publishing research findings [6]. A related effort is *Pluto Network* [7] that aims at decentralizing communications between researchers and scientists. *ImpactStory* [8] is a nonprofit effort dedicated to making scholarly research more open, accessible, and reusable.

V. Conclusion

Modern research is driven by advanced and specialized software developed with the primary goal of inquiry and discovery. This software is developed in research labs under unique budgetary and schedule constraints frequently with insufficient consideration of its sustainability beyond the immediate research goals. As a result, the software dissemination and impact are hindered and the investments in its development do not reach its full potential and returns. Developing research and experimental software is a challenge. This software is developed following a discovery driven process and its developers often lack formal software engineering training. Moreover, the nature of the funding cycles of research for three to five years means that there is very little incentive to sustain the codebase beyond the immediate lifecycle of the research project. The potential impact of improving sustainability in the research domain is momentous. Research software codes embody knowledge and algorithms pertaining to specialized domains, processes, algorithms, and data representations.

This paper presents an approach and a supporting platform that addresses the fundamental deficiencies in the prevalent academic incentive structures. The approach aims at restructuring incentives by appropriately crediting research code authors based on the sustainability and impact of their contributions. Susereum, the blockchain platform, aims at 1) creating immutable credit to authors, and 2) empowering peers to define their own appropriate sustainability metrics. In effect, Susereum aims are establishing an ecosystem where research codes are peer-reviewed, their impact are quantified, and their authors are appropriately awarded the credit.

[1] Heroux, A. M., Allen, G. (2016, September). Computational Science and Engineering Software Sustainability and Productivity (CSESSP) Challenges Workshop Report. Arlington, VA: Networking and Information Technology Research and Development (NITRD) Program. Retrieved from NITRD Website: https://www.nitrd.gov/PUBS/CSESSPWorkshopReport.pdf

[2] Heroux, A. M., Allen, G. (2016, September). Computational Science and Engineering Software Sustainability and Productivity (CSESSP) Challenges Workshop Report. Arlington, VA: Networking and Information Technology Research and Development (NITRD) Program. Retrieved from NITRD Website: https://www.nitrd.gov/PUBS/CSESSPWorkshopReport.pdf

[3] US Research Software Sustainability Institute" (URSSI). Available: http://urssi.us/ Accessed: September 2018.

[4] Software Sustainability Institute. Available: https://www.software.ac.uk/programmes-and-events/fellowship-programme. Accessed: September 2018.

[5] Andy Extance , "Could Bitcoin technology help science?" Available: https://www.nature.com/articles/d41586-017-08589-4. Nature magazine article. Accessed: September 2018.

[6] ScienceRoot. Available: https://www.scienceroot.com/ Accessed: September 2018.

[7] Pluto Network. Available: https://pluto.network/ Accessed: September 2018.

[8] ImpactStory. Available www.impactstory.org Accessed September 2018.

[9] García-Munoz, Javier, Marisol García-Valls, and Julio Escribano-Barreno. "Improved metrics handling in SonarQube for software quality monitoring." In *Distributed Computing and Artificial Intelligence, 13th International Conference*, pp. 463-470. Springer, Cham, 2016.

[10] Badreddin, Omar. "Powering software sustainability with blockchain." In Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering, pp. 315-322. IBM Corp., 2018.

[11] 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 conference on, pp. 101-110. IEEE, 2014.

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

[13] Santos, Jose Amancio M., and Manoel G. de Mendon√ßa. "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.

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

[15] 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.

[16] 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.

[17] Preston-Werner, T. "TOML-Tom's Obvious, Minimal Language.". Available: https://github.com/toml-lang/toml; Accessed January 2019.

[18] Baddreddin, Omar, and Khandoker Rahad. "The impact of design and UML modeling on codebase quality and sustainability." In Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering, pp. 236-244. IBM Corp., 2018.