ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/338367995

# BLINKER: A Blockchain-Enabled Framework for Software Provenance

**Conference Paper** · December 2019

DOI: 10.1109/APSEC48747.2019.00010

CITATIONS
12

READS
48

**4 authors:**

Jagadeesh Chandra Bose R.P.
Eindhoven University of Technology
**53** PUBLICATIONS **2,747** CITATIONS
SEE PROFILE

Kanchanjot Kaur
Indraprastha Institute of Information Technology
**3** PUBLICATIONS **19** CITATIONS
SEE PROFILE

Vikrant Kaulgud
Accenture Labs Bangalore
**63** PUBLICATIONS **414** CITATIONS
SEE PROFILE

Sanjay Podder
Accenture
**41** PUBLICATIONS **279** CITATIONS
SEE PROFILE

**Some of the authors of this publication are also working on these related projects:**



Test Automation from the Visual Requirements View project



Software enginnering View project

All content following this page was uploaded by Kanchanjot Kaur on 09 September 2021.

The user has requested enhancement of the downloaded file.

# BLINKER: A Blockchain-enabled Framework for Software Provenance

R.P. Jagadeesh Chandra Bose, Kanchanjot Kaur Phokela, Vikrant Kaulgud and Sanjay Podder
Accenture Labs, Bangalore, India
Email: {jagadeesh.c.bose, kanchanjot.k.phokela,vikrant.kaulgud, sanjay.podder}@accenture.com

***Abstract*—There has been a considerable shift in the way how software is built and delivered today. Most deployed software systems in modern times are created by (autonomous) distributed teams in heterogenous environments making use of many artifacts, such as externally developed libraries, drawn from a variety of disparate sources. Stakeholders such as developers, managers, and clients across the software delivery value chain are interested in gaining insights such as *how* and *why* an artifact came to where it is, *what* other artifacts are related to it, and *who* else is using this. Software provenance encompasses the origins of artifacts, their evolution, and usage and is critical for comprehending, managing, decision-making, and analyzing software quality, processes, people, issues etc. In this paper, we propose an extensible framework based on standard provenance model specifications and blockchain technology for capturing, storing, exploring, and analyzing software provenance data. Our framework (i) enhances trustworthiness of provenance data (ii) uncovers non-trivial insights through inferences and reasoning, and (iii) enables interactive visualization of provenance insights. We demonstrate the utility of the proposed framework using open source project data.**

## I. Introduction

As enterprises are struggling to deliver software efficiently at scale, software development practices have witnessed significant changes that have greatly improved velocity, agility, and innovation [1]. Primary among them is the notion of software delivery to be viewed as a *supply chain* that relies on open source or third party component *building blocks*. Modern software development is characterized by a highly distributed environment comprising of (autonomous) heterogeneous teams consuming extraordinary amounts of open source components drawn from a variety of disparate sources. Such a phenomenon is mandated both for technical and economic reasons (e.g., lack of IT workforce, cheap labor etc.). This paradigm shift in software development brings along with it several challenges ranging from *communication* of information/knowledge to *coordination* of teams, activities and artifacts so they contribute to the overall objectives, and *control* of teams adhering to goals and policies and artifacts adhering to quality, visibility, and management [2].

As the number of open source components balloons, even knowing what they all are let alone effectively managing them is a challenge. Reports indicate that at least $75\%$ of organizations rely on open source as the foundation of their applications with as much as $90\%$ of their applications assembled from open source components [3]. In such times, knowing the source of artifacts and the transformation of artifacts to final software product is critical for analyzing the quality of software and its development/delivery processes. Different stakeholders of the software supply chain such as developers, QA team members, managers, and clients are interested in understanding *why* and *how* a variety of kinds of software entities/artifacts have evolved to be as they are e.g., *how* and *why* a piece of code, or a test suite came to *where* it is. TABLE I lists some example questions that are of interest to different stakeholders of a software supply chain.

TABLE I: Illustrative set of questions by different stakeholders.

| Stakeholder | Provenance Questions |
|---|---|
| Developer | • What is the history of this artifact? How did it come to be what and where it is currently?<br>• What other entities is an artifact related to, and how? |
| QA team | • What and who caused this build to break?<br>• What were the most recent changes to the test that caused build failure?<br>• What has changed between two builds and by whom? |
| Manager | • Which developers have been working on an artifact that resulted in way too many problems?<br>• Were the bug triages handled properly?<br>• Do all critical systems have the most recently released, appropriate software patches? |
| Intellectual Property team | • What was the design history of an artifact that is under open source licensing violation? |
| Client | • Are the licences of various used components compatible with the released system?<br>• Which specific variants of FOSS components were used in a given product?<br>• Does the bill of materials clearly reflect the FOSS components? |

There are a multitude of tools that are being used to automate development activities or aid the human roles in performing their activities faster and better. Most of these tools record event data and software development companies have started to adopt data-driven practices in parts of their business over time [4]–[6]. Although some meaningful insights can be obtained through this, they suffer from the following shortcomings:

- it is not uncommon for cross-organizational and distributed teams to use diverse set of tools for a particular purpose
- lack of standardization on the semantics and syntax (what to log and how to log) poses challenges in integration of disparate data sources and performing analysis
- since most tools/data reside within the boundaries of an organization, the trustworthiness of data is not reliable (e.g., data can be fudged or fabricated) in global software development

Software provenance encompasses the origins of artifacts, their evolution, and usage and is critical for comprehending, managing, decision-making, and analyzing software quality, processes, people, issues etc. Provenance deals with systematically capturing meta-data describing the relationships among all the elements such as source code, components, processing steps, contextual information and dependencies. For example, provenance should capture the version of the compiler, the compilation options used, the exact set of libraries used for linking etc., when a binary was created.

Software provenance can be used for many purposes, such as understanding how an artifact was collected, determining ownership and rights over an artifact, making judgements about information to determine whether to trust an external library, verifying whether the process and steps used to obtain an artifact are compliant with given requirements, and reproducing how something like a build failure was generated.

To cope with global software development challenges, there is a growing need for a *reliable, trustworthy*, and *shared* provenance knowledge base, keeping track of all software components at various granularities ranging from project releases down to individual source files. Such a base can be referenced and augmented with other software-related facts, such as license information, and used by various stakeholders, software build tools and processes.

In this paper, we propose a *blockchain-enabled framework for software provenance*. We use an extended version of the PROV family of specifications [7] to capture provenance related data. The base specifications are extended to suit the meta-data requirements of software provenance. Our framework:

- enables an access controlled way to capture/query provenance related data
- ensures the authenticity of provenance data through verification/voting mechanisms by authorized personnel
- ensures integrity and immutability of provenance data
- uncovers implicit knowledge and infers new knowledge using ontologies, inference rules, and inference engines/reasoners
- enables an interactive hierarchical visualization of provenance data with seamless zoom-in/out

The remainder of the paper is organized as follows: Related work is presented in Section II. Section III presents some background on blockchain, smart contracts and the PROV standard specification for capturing provenance. Section IV presents our blockchain-enabled framework for software provenance while Section V discusses the realization of the framework with preliminary results. Finally Section VI concludes with some future directions.

## II. Related Work

Initial attempts at provenance of software products was confined to specific aspects of software such as provenance of code [8] and configuration management [9]. One of the earlier attempts at provenance for software processes in an integrated/holistic manner was reported in [10], where the aim was to follow and reproduce the entire process of developing a software product and be able to answer direct questions like 'to which requirement does this commit belong?', 'Who is responsible for a failing unit-test?' etc. [10] adopts the PrIMe methodology [11] and was based on the Open Provenance Model (OPM) specification [12]. The resulting model is converted into a graph and stored in a graph database Neo4j[1]. Provenance questions are converted into queries using the graph programming language, Gremlin[2] to obtain insights. Our work in contrast is based on the more robust PROV family of specifications proposed as a gigantic leap over OPM. The capabilities of PROV, e.g., use of ontologies allow a wider range of provenance queries to be addressed. Furthermore, our provenance solution framework is built on the blockchain technology to ensure trust, integrity of data etc.

[13] reports several facets of software provenance. [14] addresses the question of provenance of software entities, i.e., where did this entity come? within a large corpus of possibilities through an *anchored signature matching* approach.

The first work of using ontologies based on the PROV family of specifications for software process provenance was done in [15]. A software process specific extension called PROV-Process was defined, which was later extended as PROV-SWProcess in [16], [17]. Our work adopts PROV-SWProcess for provenance data specification but builds the framework for capturing and addressing provenance, and providing mechanism to enable trust using blockchain in a decentralized application Furthermore, our solution enables a hierarchical visualization of provenance graphs and query results.

The use of blockchains as a platform for capturing provenance of data was reported in [18], [19]. Neither of these works address software provenance and the capturing of data according to standard specifications like PROV-*. [19] focusses primarily on enabling trust of content recorded on the blockchain through voting mechanisms. We adopt [19] as one of the enablers of trust mechanisms in our framework but also provide recommendations on other enablers.

## III. Background

In this section, we present a brief overview on the concepts used in this paper, viz., blockchain, smart contracts, and PROV family of specifications to capture/query provenance related data.

### A. Blockchains and Smart Contracts

Blockchains provide a secure, decentralized mechanism for sharing transactions and/or data across a large network of untrusted participants without relying on a central trusted authority to record and validate data or transactions. Blockchain technology relies on a *network of participants*, *consensus protocols* such as proof-of-work and proof-of-stake, *cryptographic hashes* and *digital signatures*. The blockchain data structure is a cryptographically linked chain of blocks of data.

Blockchains provide the following key properties:

- *Non-repudiation:* non-repudiation means that one cannot reject (cannot repudiate) the validity of a certain action

[1]https://neo4j.com

[2]https://tinkerpop.apache.org/gremlin.html

or claim. Since the data on the blockchain is digitally signed by the submitter, the submitter cannot dispute the data.

- *Immutability (tamper-evidence):* immutability pertains to the inability of a block to be deleted or modified once it is in the blockchain. Immutability assures trust in that the contents on the blockchain cannot be tampered with. Most blockchains' immutability in its true sense refers to them being *tamper evident* but not *tamper proof*.
- *Availability:* since blockchains are a fully decentralized peer-to-peer system, there is no single point of failure. Data is replicated across the nodes in the network and consensus is maintained.
- *Transparency:* transactions are recorded on a distributed ledger publicly. This brings in transparency of activities to all participants.

When compared to centralized storage, blockchains provide more trust and robustness due to its immutability.

There are three types of blockchains (a) public (b) private or permissioned and (c) consortium blokchains. Bitcoin [3] and Ethereum [4] are popular public blockchain platforms while Hyperledger [5] is a private blockchain and Quorum [6], R3 Corda [7] etc., are examples of consortium blockchains.

The power of blockchains is compounded by its ability to embed and execute logic. While Ethereum refers this as *smart contracts*, Multichain calls them *smart filters* and Hyperledger Fabric uses the nomenclature *chain code*. Smart contracts are programs that can be deployed on the blockchain and run across the blockchain network. It can express triggers, conditions, and complex/sophisticated logic such as an entire business process. Computational results upon execution of smart contracts are verified by the participants of the network and recorded on the blockchain.

Smart contracts has two types of data storage: *state storage* which stores data of the variables in the smart contract and the *event logs*. Events are notification mechanisms in the smart contract that allow it to trigger some external functionalities. Event logs maintain an immutable record of the sequence of events that are emitted by a smart contract.

Although Blockchain technology originated as an immutable ledger of transactions for cryptocurrencies, with the power of smart contracts, it has evolved into a programmable interactive environment for building reliable decentralized applications.

### B. PROV Family of Specifications

PROV is a family of specifications to express, model, access, exchange, serialize, and reason over provenance records [7]. Provenance records contain descriptions of the entities, activities, and agents involved in producing and influencing a given object. The core concepts of provenance model is specified in PROV-DM [20] and is centered around (i) entity (ii) activity and (iii) agent. Entity refers to a physical, digital, conceptual, or other kind of thing. Activity refers to actions using or creating entities while Agents refer to something/someone responsible for execution of an activity or the existence of an entity, or for another agent's activity. Fig. 1 depicts the core concepts of PROV. Fig. 2 depicts an example instantiating the same. The example says that the entity `Z.jar` *was generated by* the compilation activity *using* entities `X.java` and `Y.java` by agent Adam. The entity `Z.jar` is said to be *attributed to* agent Adam and *derived from* `X.java` and `Y.java`. The activity compile is said to be *associated with* agent Adam.

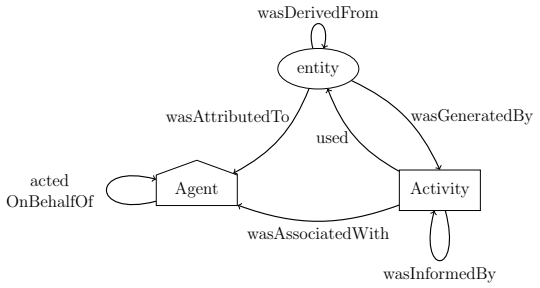



Fig. 1: Core concepts of PROV as defined in [7].

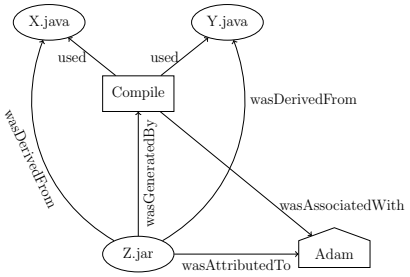



Fig. 2: Example instantiation of the PROV model.

We highlight some of the key requirements around which the PROV data model was designed. The reader is referred to [21] for a comprehensive overview on the requirements and [22] for an overview of the PROV data model.

- PROV has to support the *responsibility view*, *data flow view*, and *process flow view*. Responsibility view pertains to assigning responsibility for a given result or for what had happened in a system. The data flow view is concerned with the flow and transformation of information in the digital or the transormation of things in the physical and/or imaginary world. The process flow view is concerned with the activities that occurred, the entities they used, how they started and ended, as well as their start and end times.
- PROV is aimed to describe past executions, as opposed to specify potential future executions. Hence the relations are expressed in past tense, e.g., wasGeneratedBy, wasDerivedFrom etc.
- PROV is to model the attributes of entities, activities, agents, and most relations.
- PROV is to model the versioning of elements

[3]https://bitcoin.org
[4]https://www.ethereum.org
[5]https://www.hyperledger.org
[6]https://www.jpmorgan.com/global/Quorum
[7]https://www.r3.com/corda-platform

Some of the other set of specifications in the PROV family that are of interest in this paper include an OWL ontology (PROV-O), XML serialization (PROV-XML), a set of constraints and inference rules (PROV-CONSTRAINTS), PROV-AQ for accessing and querying and PROV-SWProcess for software processes.

The main extensions of PROV-SWProcess lies in the (a) renaming of entities to artifacts and defining categories of artifacts and relationships between them and (b) finer-granular classification of agents to *stakeholders* (for human agents) and *resources* (such as software and hardware) and (c) definition of procedures to capture the methods, techniques and document templates adopted by the software process activities. Fig. 3 depicts the core concepts and the model for PROV-SWProcess[8]. In addition, PROV-SWProcess defines a set of *eight* inference rules that uncover implicit information. The inferred results can act as new provenance statements.

We have adopted the PROV-SWProcess specification in this paper.

## IV. Framework

Fig. 4 depicts our framework for software provenance. Our framework comprises of

- *Data Services:* There are a multitude of tools that are being used to automate software development activities or aid the human roles in performing their activities efficiently. For example, IDEs like eclipse and visual studio, build tools such as maven, test tool such as JUnit, version control systems such as Git and subversion, quality assessment tools such as PMD etc. Several of these tools emit data forming a *data exhaust* that can aid in provenance. For example, state of the art version control systems record commit messages, precious information that detail specific changes that have been made to a given software at a given moment. Provenance data capturing should not interfere in the software process execution and be non-intrusive. *Data services* layer comprises of data ingestion tools/plugins for these disparate sources and the transformation of this data into forms compliant to PROV-specifications, that resides in off-chain databases. Furthermore, the data services also contain 'oracles' that faciliate communication between smart contracts and the external world entities and services.
- *Trust Services:* Insights gained from provenance data are worthy only if the captured data is truthful. Users may try to corrupt the provenance chain by logging incorrect details to the chain. Such user activity may be intentional or unintentional and the reasons for this can be manifold. For example, a team lead may log that code review was successfully done without actually doing so due to deadline, incentive and penalty pressures. Provenance details recorded on the blockchain should be verifiable by authorized personal without compromising on the privacy and violating the ownership of the data. The system should encourage truthful behavior by penalizing the users who submit wrong data.
  Voting is one of the prominent mechanisms used for validating the content/transactions on the blockchain [19]. The basic principle here is that all or selected participants of the network are required to vote approving the validity of the transaction or submitted data. The choice of who can vote for what can be defined through *voting policies*. The accumulated votes are recorded on the blockchain. Voting strategies such as $m$ out of $k$ or *majority voting* are typically adopted. Users submitting transactions/data that are rejected by voting can be penalized.
  Social certification based enablement of trust is loosely based on *reputation systems*. Unlike voting that seeks proactive votes, social certification relies on participants' voluntary rating of content on the blockchain. For example, *replication*, *reuse*, and *reproduction* are some of the benefits of capturing provenance data. If users of the network are able to benefit from the specified provenance, they can provide their ratings, which act as reputation. The higher the reputation, the more trustworthy is the data. Note that social certification mechanisms should have proper counter measures to detect and mitigate *collusion*.
- *Smart Contracts:* Smart contracts form the core services of the blockchain framework. Various sets of smart contracts are to be created to cater to different functions such as registering (storing) provenance trails, enabling trust by voting, and compliance checking. There are challenges in building decentralized blockchain applications for data intensive applications as storage of data on some blockchain platforms like Ethereum cost cryptocurrency. In such cases, the smart contracts can choose to store critical provenance data along with cryptographic hashes of the actual data on the blockchain (e.g., as event logs on Ethereum) and decide to store the full data on off-chain databases. Cryptographic immutable distributed databases such as IPFS[9] can be used for this purpose.
- *Utility Services:* These pertain to services that facilitate comprehension from provenance data by discovering non-trivial insights. These comprise of:
  - *Provenance Query Services:* Provenance related queries are typically grouped into three broad categories:
    - Agent-centered provenance: this deals with queries related to people, roles, organizations etc.
    - Artifact-centered provenance: this pertains to questions over tracing the origins of artifact.
    - Process-centered provenance: this relates to the actions and steps taken to generate the information or artifact in question.
  - *Inference Services:* As discussed in Section I, stakeholders would be interested in different aspects of provenance at various levels of abstraction. Primitive relations as captured using provenance specifications

[8]The latest version of the PROV-SWProcess specification can be accessed at http://www.gabriellacastro.com.br/provswprocess/v3.html

[9]https://ipfs.io/

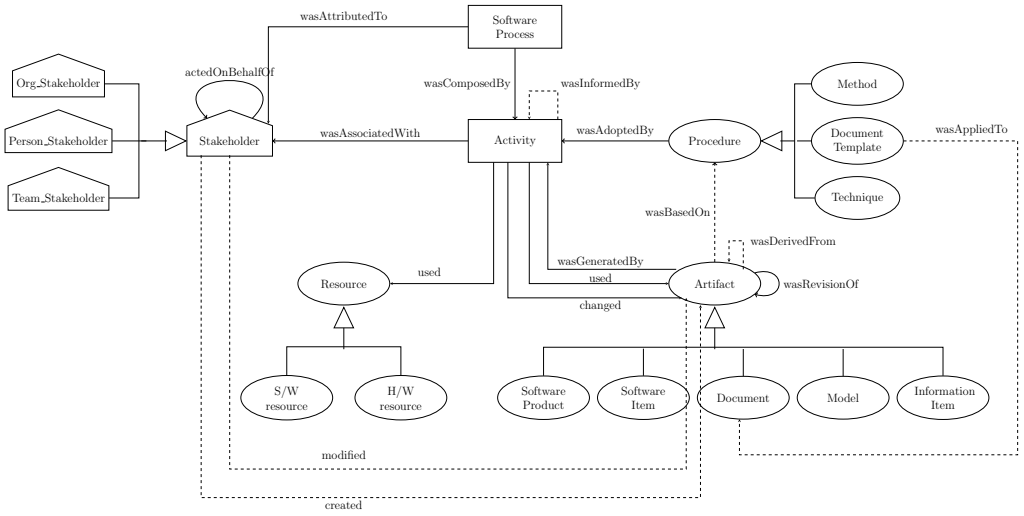



Fig. 3: PROV-SWProcess conceptual model (taken from [17]). Dashed edges depict relations that are inferred.

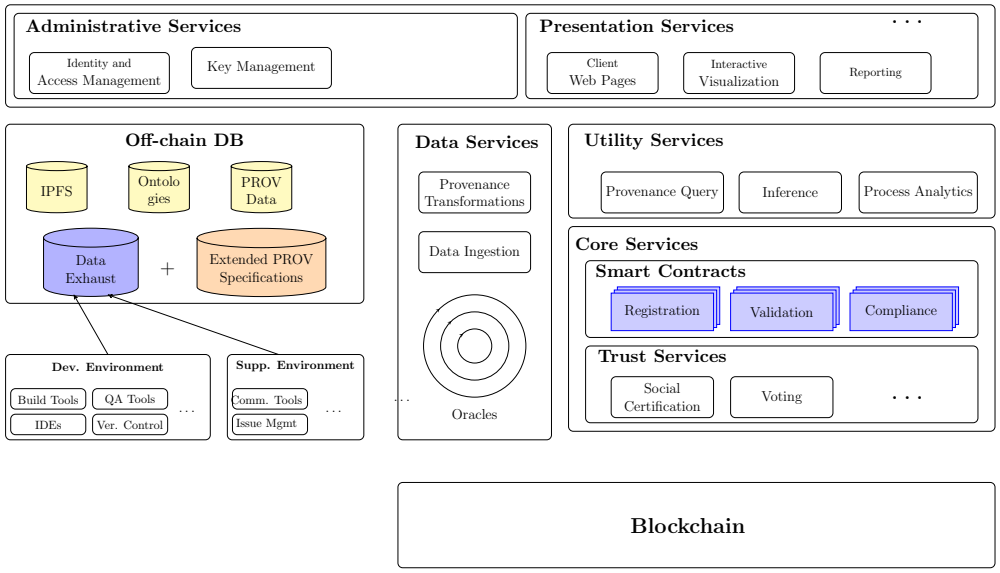



Fig. 4: Blockchain-enabled framework for provenance.

can be combined to infer new relations manifested *implicitly*. The framework allows the specification of inference rules and uses a inference engine to uncover insights. Similarly, domain ontologies that capture relationships between concepts, abstractions of low-level elements, etc., For example, although provenance data is recorded at a source code level, with proper ontology relating source code to components, one can answer queries at component level.

– *Process Analytics:* Since provenance specifications facilitate the capturing of data pertaining to activities, data, resource, and time, provenance data stores act as *data gold mines* for process analytics. For example, one can use process mining techniques [23] to discover process models, check for conformance to expected behavior, uncover bottlenecks etc. Such insights can help in process optimization and reengineering.

- *Administrative Services:* Any form of data provenance management system should ensure that the data is protected against unauthorized access and secured against privacy violations. Blockchain paticipants access the network via their addresses and sign transactions using their public-private key pairs. It also facilitate as an *authentication provider*, manages the access control polices (e.g., role-based accesses) and identity information of users.
- *Presentation Services:* These are concerned with the front-end interfaces of the provenance solution such as the client web pages for querying/accessing provenance information, the visualization aspects of provenance data/results, and reporting. Efficient and user-friendly mechanisms are needed for easy comprehension

of provenance data [24]. One of the powerful means of comprehending provenance data is through graph visualization [25]. However, most graphical visualizations of provenance data depict a flat structure, which become *spaghetti-like* even for a few dozens of nodes. *An interactive hierarchical visualization with seamless zoom-in/out enriched with additional metrics will have a profound impact on provenance comprehension*, e.g., one can use the size, color, thickness properties of nodes/edges to convey rich insights. Domain ontologies are used for abstractions over low-level elements For example, one can have a seamless drill-down of changes in releases from product level to component level to individual source code files.

## V. Experimental Results and Discussion

In this section, we discuss about the realization of the framework and some preliminary experimental results. The framework discussed in the previous section has been implemented over the TestRPC Ethereum platform[10]. For inferencing and querying, we have used protégé[11], and for visualizing provenance graphs, we have adapted the SPPV tool [25]. We extended SPPV to enable the notion of abstractions, and seamless drill-down on abstract entities, and annotations of edges.

*DataSet:* Code reviews are an integral part of software delivery and several studies have reported that peer review fosters quality assurance [26]–[29]. Many large-scale OSS projects have adopted peer reviews and web-based code review tools such as Gerrit[12] have become popular, e.g., Google uses Gerrit for Android Open Source Project. Gerrit automatically records and tracks all merges into the source code, including details related to the code patch and the peer review process activities.

Fig. 5 depicts a typical review process of a change request. The code review process starts with a commit of a patch-set, i.e., the code change submitted for review. Reviewers, which are selected as per project management policies, perform the review on the patch-set. The reviewed patch-set needs to be verified and approved before merging it into the code repository. Verification and/or Approval is performed by experienced or core members. Automated bots can also do verification in the build and test stage of the patch-set Approvers decide on the acceptance of the patch-set for source code merge. Note that reviewers cannot verify or approve reviews. Review process participant are only authorised to re-assess patch-set quality and give feedback as comments.


Fig. 5: Review process of a change request.

[10]https://nethereum.readthedocs.io/en/latest/ethereum-and-clients/test-rpc/
[11]https://protege.stanford.edu/
[12]https://www.gerritcodereview.com

We considered a collection of modern code review data reported in [30][13]. The data showcases mined data from both an integrated peer review system and source code repositories. The core of the dataset comes from five projects, viz., *OpenStack, Qt, AOSP, Eclipse,* and *LibreOffice*, that use git as their source code repository, and are also integrated with the Gerrit code review system. We considered the Eclipse dataset within this repository for our analysis, which we present below.

Let us consider the change with id 14250. This change pertains to the project `papyrus/org.eclipse.papyrus` and corresponds to the bug 'Bug 482483 - [Component diagram] The behavior of the component semantic should be the same'. Fig. 6 depicts the change review process. This change resulted in the creation of four patch sets. The verification (build) of the first patch set resulted in failure upon which this patch $\text{set}_1$ was **rebased** to patch $\text{set}_2$. This phenomenon is observed for all the patch sets and finally the change was *abandoned* as illustrated in Fig. 7. Note that objects in the provenance graph can carry properties (attributes). For example, as highlighted in the figure, the Verify Patch Set 1 activity has startedTimeAt, endedTimeAt and status attribute

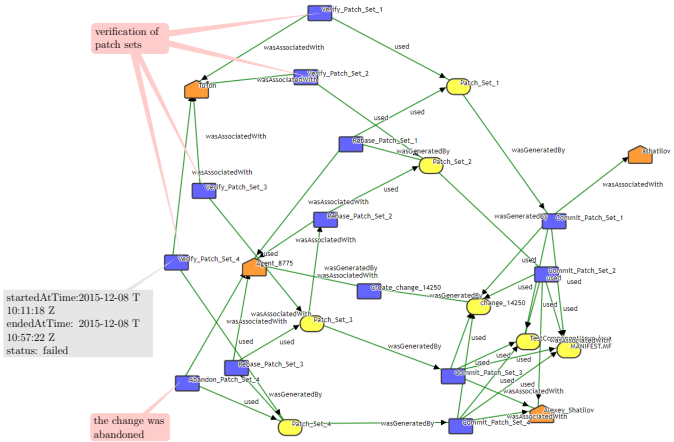



Fig. 6: Provenance graph of the change review process for the change with id 14250.

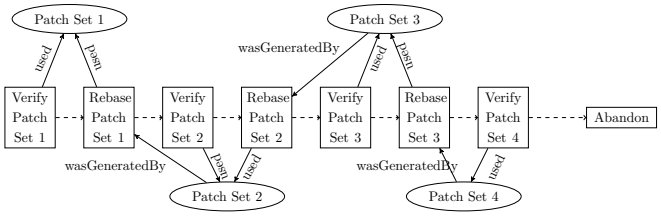



Fig. 7: Phenomenon of build failures for change with id 14250.

*Inferencing, Abstractions, and Querying:* One can derive new information from existing provenance data by defining inference rules. For example, we can define the `filesChangedBy` relation based on existing associations between entities, agents, and activities as below:

[13]http://kin-y.github.io/miningReviewRepo

$\text{prov} : \text{Activity}(?a) \ \wedge \text{prov} : \text{Entity}(?e) \ \wedge$
$\text{prov} : \text{used}(?a, ?e) \wedge \text{prov} : \text{Agent}(?ag) \ \wedge$
$\text{prov} : \text{wasAssociatedWith}(?a, ?ag) \implies$
$\text{prov} : \text{filesChangedBy}(?e, ?ag)$

This states that for all activities 'a' and file entities 'e' , if activity 'a' used entity 'e' and if the activity 'a' wasAssociatedWith agent 'ag', then we can infer that the file entity 'e' was changed by agent 'ag'.

Provenance data can become huge if we intend to capture fine-grained provenance data. Provenance graphs visualized over such data is highly spaghetti-like and incomprehensible. Fig. 8 depicts the graph captured using the provenance data of agent Mattias Sohn for the change with id 12572. We can see that the graph is less comprehensible. One of the reasons for this spaghettiness is the high number of files that Mattias worked on during the change. For better comprehension, domain knowledge about the entities can be used to abstract things out, e.g, files can be abstracted to their corresponding components.

Since we do not have direct domain knowledge about the datasets used in the experiments, we use the package folder structure to illustrate the ideal. For instance, we can define ontology based on the package structure of code as shown in Fig. 9. This ontology depicts the package structure at certain level (from the root), comprises of other (sub-)packages and can have files.

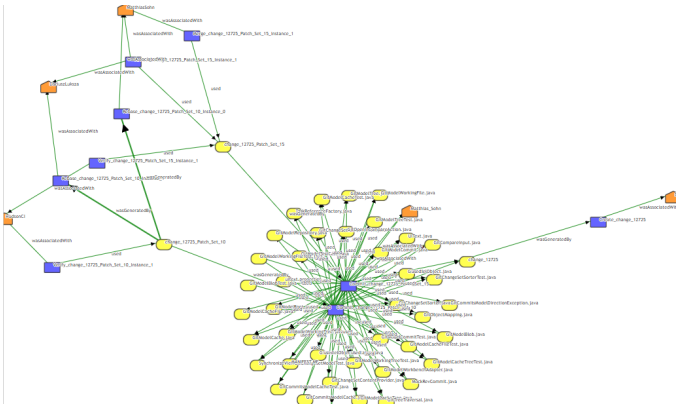


Fig. 8: Provenance graph of agent Mattias Sohn for change id 12752.

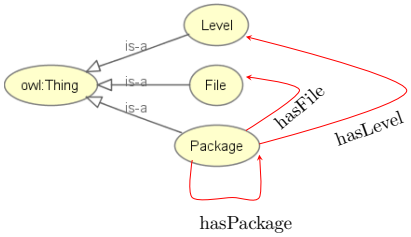



Fig. 9: Ontology for the package structure.

If we are interested in the question 'Which components (packages at level 'k') have agent Mattias Sohn contributed to in a change request'?, We would be able to answer them using the package structure ontology and the following query:

```
Package and hasLevel value k and hasFile
some (prov:Entity and prov:filesChangedBy
value prov:Matthias_Sohn and
prov:influenced some prov:Activity and
prov:used value prov:change_12725
```

As an illustration, Mattias Sohn has contributed to 40 files for change id 12752. A stakeholder like a manager who might be interested in course granular information such as, the components that Mattias contributed to. For example, if we choose a package hierarchy at level 5, we get two components viz., `egit` and `jgit`; for a package hierarchy at level 6, we get `core`, `internal`, `revwalk`, and `ui` as the components that Mattias contributed to. We can use these high-level abstractions to visually inspect provenance graphs. Fig. 10 depicts the graphs at abstraction levels of 5, 6, and 7. We can see that Fig. 10 is a lot more comprehensible than that of Fig. 8. The abstract nodes are depicted in 'brown' color in the graph. Abstract nodes can be seamlessly zoomed-in, upon which the detailed graph pertaining to that abstraction will be shown.

*Enabling Trust using Blockchain:* In a real deployment of our framework, provenance data is expected to be captured in near realtime as activities are executed. Since we have used a historical dataset in the illustration above, we couldn't demonstrate the utility of the trust services.

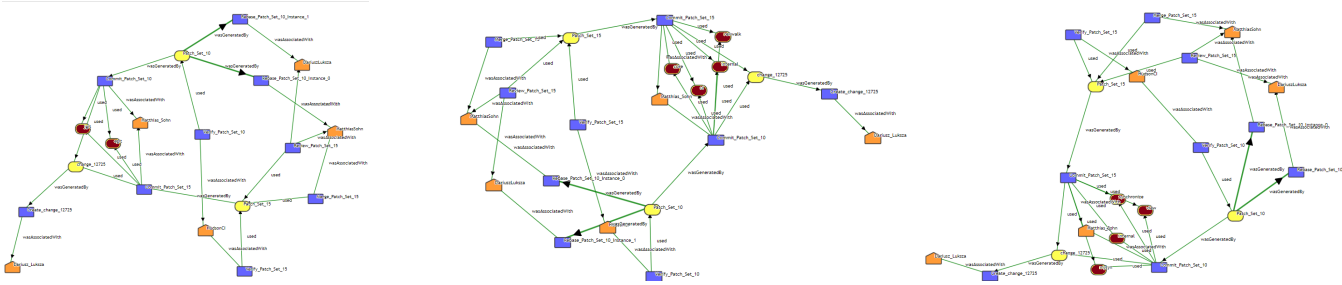
For the code review process described earlier, as and when the quality assessment/review of the patch-set happens, the feedback is provided by the reviewer, which is logged on the blockchain as provenance data. Reviewers are selected based on their seniority and proximity with the module to which the patch-set belongs, generally team leads of that module. Approvers and/or verifiers evaluate the feedback of the patch-set and give their consent or rejection, which is logged on the blockchain as a vote. Selection of the approvers/verifiers is generally as per the *voting policies*.

Now, using the *voting* smart contacts of trust services, which are implemented based on the voting strategies, cumulative vote scores are calculated. For example, majority voting is implemented as a voting strategy in this case. Based on this vote score, the patchset is either merged in the source code repository or rejected. The framework supports pluggable voting strategies, as described by organization guidelines.

The voting mechanism using the blockchain ensures trust, immutability and transparency in software delivery process, as described for the review process.

## VI. Conclusions and Future Work

In this paper, we proposed a blockchain-based conceptual framework for software provenance based on PROV family of specifications customized for software processes. Our framework can be used to represent, query, reason-about, and analyze provenance meta-data for gaining insights on software artifacts, processes, and people. We demonstrated the utility of the framework on a historical code review process. The proposed framework should be understood as a foundation for blockchain-enabled provenance management systems that can be built upon, e.g., the framework is designed to be extensible



(a) level 5 (b) level 6 (c) level 7

Fig. 10: Provenance graph with different levels of abstraction of agent Mattias Sohn's contribution to change id 12752.

with new elements such as new architectures of trust. As future work, we would like to perform case studies on live projects and critically assess the ability and limitations of our solution. For example, we would like to study the scalability of the solution for high volume provenance data requirements. We would like to study questions such as: (a) can we design a methodology that assesses what goes on-chain and what goes off it? (b) how do we seamlessly integrate the two? and (c) how do we keep track of availability and usage of assets off-chain?

## REFERENCES


[1] Sonatype, "State of the software supply chain," 2015, last accessed: 1 feb 2019. [Online]. Available: https://cdn2.hubspot.net/hubfs/1958393/White_Papers/2015_State_of_the_Software_Supply_Chain_Report-.pdf?t=1466775053631

[2] S. Sahay, B. Nicholson, and S. Krishna, *Global IT outsourcing: software development across borders*. Cambridge University Press, 2003.

[3] Sonatype, "Open source development and application security survey," 2014, last accessed: 1 feb 2019. [Online]. Available: https://securosis.com/assets/library/reports/Securosis_OpenSourceSurvey_Analysis.pdf

[4] V. S. Sharma and V. Kaulgud, "Pivot: Project insights and visualization toolkit," in *Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics*. IEEE Press, 2012, pp. 63–69.

[5] V. Kaulgud and V. S. Sharma, "Software development analytics: Experiences and the way forward," in *Automated Software Engineering Workshop (ASEW), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 10–13.

[6] T. Menzies and T. Zimmermann, "Software analytics: so what?" *IEEE Software*, no. 4, pp. 31–37, 2013.

[7] P. Missier, K. Belhajjame, and J. Cheney, "The w3c prov family of specifications for modelling provenance metadata," in *Proceedings of the 16th International Conference on Extending Database Technology*. ACM, 2013, pp. 773–776.

[8] Y. B. Dang, P. Cheng, L. Luo, and A. Cho, "A code provenance management tool for ip-aware software development," in *Companion of the 30th international conference on Software engineering*. ACM, 2008, pp. 975–976.

[9] P. Xu and A. Sengupta, "Provenance in software engineering-a configuration management view," *AMCIS 2005 Proceedings*, p. 515, 2005.

[10] H. Wendel, "Using provenance to trace software development processes," *Masterarbeit, Institut für Informatik III, Universität Bonn*, 2010.

[11] S. Miles, P. Groth, S. Munroe, and L. Moreau, "Prime: A methodology for developing provenance-aware applications," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 3, p. 8, 2011.

[12] L. Moreau, J. Freire, J. Futrelle, R. E. McGrath, J. Myers, and P. Paulson, "The open provenance model: An overview," in *International Provenance and Annotation Workshop*. Springer, 2008, pp. 323–326.

[13] M. W. Godfrey, "Understanding software artifact provenance," *Science of Computer Programming*, vol. 97, pp. 86–90, 2015.

[14] J. Davies, D. M. German, M. W. Godfrey, and A. Hindle, "Software bertillonage: Determining the provenance of software development artifacts," *Empirical Software Engineering*, vol. 18, no. 6, pp. 1195–1237, 2013.

[15] H. L. Dalpra, G. C. B. Costa, T. F. Sirqueira, R. M. Braga, F. Campos, C. M. L. Werner, and J. M. N. David, "Using ontology and data provenance to improve software processes." in *ONTOBRAS*, 2015.

[16] G. C. B. Costa, H. L. Dalpra, E. N. Teixeira, C. M. Werner, R. M. Braga, and M. A. Miguel, "Software processes analysis with provenance," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2018, pp. 106–122.

[17] G. Castro, "Supporting software processes analysis and decision-making using provenance data," Ph.D. dissertation, Federal University of Rio de Janerio, 2018.

[18] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 2017, pp. 468–477.

[19] A. Ramachandran and M. Kantarcioglu, "Smartprovenance: A distributed, blockchain based dataprovenance system," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. ACM, 2018, pp. 35–42.

[20] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker *et al.*, "Prov-dm: The prov data model," *W3C Recommendation*, 2013. [Online]. Available: \url{http://www.w3.org/TR/2013/REC-prov-dm-20130430/}

[21] L. Moreau, P. Groth, J. Cheney, T. Lebo, and S. Miles, "The rationale of prov," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 35, pp. 235–257, 2015.

[22] Y. Gil, S. Miles, K. Belhajjame, H. Deus, D. Garijo, G. Klyne, P. Missier, S. Soiland-Reyes, and S. Zednik, "Prov model primer," *W3C Working Group Note*, vol. 30, 2013.

[23] W. M. Van der Aalst, *Process Mining: Data Science in Action*. Springer, 2016.

[24] W. Oliveira, L. M. Ambrósio, R. Braga, V. Ströele, J. M. David, and F. Campos, "A framework for provenance analysis and visualization," *Procedia Computer Science*, vol. 108, pp. 1592–1601, 2017.

[25] G. C. Costa, M. Schots, W. E. Oliveira, H. LO, C. M. Dalpra, R. Braga, J. M. N. David, A. Marcos, V. S. Miguel, and F. Campos, "Sppv: Visualizing software process provenance data," *Sociedade Brasileira de Computação–SBC*, p. 49, 2016.

[26] A. F. Ackerman, L. S. Buchwald, and F. H. Lewski, "Software inspections: an effective verification process," *IEEE software*, vol. 6, no. 3, pp. 31–36, 1989.

[27] P. Anderson, T. Reps, T. Teitelbaum, and M. Zarins, "Tool support for fine-grained software inspection," *IEEE software*, vol. 20, no. 4, pp. 42–50, 2003.

[28] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: a case study at google," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. ACM, 2018, pp. 181–190.

[29] K. Hamasaki, R. G. Kula, N. Yoshida, A. Cruz, K. Fujiwara, and H. Iida, "Who does what during a code review? datasets of oss peer review repositories," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 49–52.

[30] X. Yang, R. G. Kula, N. Yoshida, and H. Iida, "Mining the modern code review repositories: A dataset of people, process and product," in *Proceedings of the 13th International Conference on Mining Software Repositories*, 2016, pp. 460–463.


View publication stats