Modelling Knowledge about Software Processes using Provenance Graphs and its Application to Git-based Version Control Systems

Andreas Schreiber  
German Aerospace Center (DLR)  
Cologne, Germany  
andreas.schreiber@dlr.de

Claas de Boer  
German Aerospace Center (DLR)  
Berlin, Germany  
claas.deboer@dlr.de

ABSTRACT
Using the W3C PROV data model, we present a general provenance model for software development processes and—as an example—specialized models for git services, for which we generate provenance graphs. Provenance graphs are knowledge graphs, since they have defined semantics, and can be analyzed with graph algorithms or semantic reasoning to get insights into processes.

KEYWORDS
provenance, software development process, knowledge graphs, git, version control systems

ACM Reference Format:

1 INTRODUCTION
Today, many research fields—including computer science—use Provenance [5] to verify data products and to analyse processes that led to them. Provenance can be used to form assessments about quality, reliability or trustworthiness of a piece of data. The knowledge of provenance includes aspects such as sources and processing steps as well as dependencies and contextual information.

Provenance can be expressed in many formats. We focus on the standard W3C PROV [2], which defines the provenance data model PROV-DM [6] to support the interoperable interchange of provenance in heterogeneous environments such as the web. The core structure of PROV-DM relies on the definition of the model class elements entities (Entity), activities (Activity), and agents (Agent) that are involved in producing a piece of data or artifact and on definitions of relations to relate these class elements, such as wasGeneratedBy, wasAssociatedWith, wasAttributedTo, and used. Each of the class elements and relations can have additional attributes.

2 PROVENANCE OF SOFTWARE ARTIFACTS
Due to the complexity of today’s software many development process models evolved, together with many tools. A typical tool suite consists of an integrated development environment (IDE), a version control system, an issue tracker, a continuous integration framework, and a documentation management system.

Based on previous work [10], where we developed a provenance model for software development processes using the Open Provenance Model (OPM) notation, we develop an extensible PROV model that currently covers issue tracking (requirements, bugs), development (planning, design, coding, testing), continuous integration, documentation (developer, user), and release (Figure 1).

The general model can be extended with further activities such as editing or deployment and further actors such as software bots or software analytics tools. If used for concrete processes, each of the PROV class elements must be defined with specialized class elements. For example, the generic actor role “User” has to be specialized to roles such as “Author” or “Test Manager”.

To get knowledge from provenance graphs, one has to query the provenance database. Examples queries include questions related to quality assurance (e.g., “How many releases have been produced this
As a practical example for retrospective provenance, we consider (excerpt; for clarity, some relation types and most attributes (Figure 2).

\[
X \text{ built?}
\]


\( \text{database (or the API’s (GitHub, GitLab, and the PROV entities for the modified file file_v-1 it contains the specialized PROV activity commit), or developer performance (e.g., “Which developer is most active in contributing documentation?”).} \)

To practically extract the PROV graph from repository’s, the re-

...extracting knowledge otherwise is possible by using Cypher queries (Listing 1) or graph algorithms in Neo4j. For knowledge, we consider the semantics and an ontology for git [4].

Listing 1: Cypher query for the number of files that were edited by an agent.

MATCH
(user:Agent) - [:wasAttributedTo] - (:fileVersion:Entity), (:fileVersion:Entity) - [:specializationOf] - (:file:Entity)
WHERE
fileVersion.prov:type = "file_version" AND
file.prov:type = "file"
RETURN
user.name, COUNT(DISTINCT file) AS file_count
ORDER BY file_count DESC

4 CONCLUSIONS AND FUTURE WORK

We presented a draft PROV model for software development processes and an example for retrospective provenance. The resulting provenance graphs are knowledge graphs given their specified semantics for nodes and edges.

Future work includes to add more sources to the provenance graph. For example, from design documents (UML2PROV [9]) or IDE’s. The provenance graph can also be extended with provenance for (running) algorithms, data, or machine learning processes [3].

REFERENCES


