Increasing Software Quality using the Provenance of Software Development Processes

Andreas Schreiber <andreas.schreiber@dlr.de>
German Aerospace Center (DLR)
Berlin / Braunschweig / Cologne
Outline

• Introduction

• Provenance

• Software Development Processes

• Queries
Introduction

Problem

• Today’s software development processes are complex
• Massive interaction between developers and tools as well as between tools (manually or automatically)
• Tracing and understanding the process is hard
• Software isn’t reused because of lack of trust and quality

Solution

• Recording of process information during runtime
• Analysis of recorded information for insight and confidence

Standardized (W3C) solution: Provenance
Provenance Definition

Provenance is defined as a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing.

(W3C Provenance Working Group, http://www.w3.org/2011/prov)
Provenance
Research Area Since 2002

- Simmhan, Yogesh L., Beth Plale, and Dennis Gannon: A survey of data provenance in e-science.
Provenance Application Areas

General Areas

• Information systems: Origin of data, who was responsible for its creation
• Science applications: How the results were obtained
• Publications: Origins and references of published results

Applications involve

• Engineering
• Finance
• Security
• Climatology & earth sciences
• Medicine, pharmacy & biomedicine
• Software Development

http://www.w3.org/2011/prov/wiki/ISWCProvTutorial
Provenance Goal

Express special “meta” information on the data

- Who played what role in creating the data
- View of the full revision chain of the data
- In case of integrated data, which part comes from which original data and under what process

http://www.w3.org/2011/prov/wiki/ISWCProvTutorial
Realizing Provenance

Provenance requires a complete model

• Describing the various constituents (actors, revisions, etc.)

• Balance between
  • simple ("scruffy") provenance: easily usable and editable
  • complex ("complete") provenance: allows for a detailed reporting of origins, versions, etc.

http://www.w3.org/2011/prov/wiki/ISWCPprovTutorial
W3C Provenance Data Model (PROV-DM)

Concepts

**Nodes**
- Entity
- Activity
- Agent

**Edges**
- association
- responsibility

Diagram:
- Entity
  - wasDerivedFrom
  - used
  - wasGeneratedBy
  - wasStartedBy
  - wasEndedBy
  - wasAssociatedWith
- Activity
  - wasDerivedFrom
  - used
  - wasGeneratedBy
  - wasStartedBy
  - wasEndedBy
  - wasAssociatedWith
- Agent
  - actedOnBehalfOf
Baking a Cake

- 100 g Butter
- 2 Eggs
- 100 g Sugar
- 100 g Flour

baking wasGeneratedBy Cake
Provenance Life Cycle

- Application
- Data (Result)
- Recording of process Information
- Provenance database
- Administration of Provenance database
- Query for Provenance of data
Software Development Processes
Typical DLR Software Development Process
Process Steps

**Issue Tracking** (Requirements, Bugs)

**Development** (Planning, Design, Coding, Testing)

**Continuous Integration**

**Documentation** (Developer, User)

**Release**
Provenance Model

Activities
• Issue Tracking
• Development
• Continuous Integration
• Documentation
• Release

Entities and Agents
• User
• Issue
• Revision
• Release
<table>
<thead>
<tr>
<th>Questions and Problems</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Error detection</strong></td>
<td><em>Which change set resulted in more failing unit tests?</em></td>
</tr>
<tr>
<td><strong>Quality assurance</strong></td>
<td><em>How many releases have been produced this year?</em></td>
</tr>
<tr>
<td><strong>Process validation</strong></td>
<td><em>From which revision was release X built?</em></td>
</tr>
<tr>
<td><strong>Monitoring</strong></td>
<td><em>How much time has been spent implementing issue X?</em></td>
</tr>
<tr>
<td><strong>Statistical analysis</strong></td>
<td><em>How many developers contributed to issue X?</em></td>
</tr>
<tr>
<td><strong>Developer rating</strong></td>
<td><em>Which developer is most active in contributing documentation?</em></td>
</tr>
<tr>
<td><strong>Information</strong></td>
<td><em>Which features are part of release X?</em></td>
</tr>
<tr>
<td>Tool Type</td>
<td>Simple</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>Single</td>
<td>What is the current overall code coverage?</td>
</tr>
<tr>
<td>Multi</td>
<td>Developer: How many issues were implemented by developer X for release Y?</td>
</tr>
<tr>
<td>Multi</td>
<td>Errors: Which requirement causes the most build failures?</td>
</tr>
</tbody>
</table>
Implementation
Collecting Data

MoinMoin Wiki (Documentation)
- Page
  - Change
  - Revision
  - Comment
  - User
  - Date

Eclipse (IDE)

Mantis (Issue Tracking)
- Issue
  - Change
  - User
  - Date

Subversion (Version Control) + Repoguard (Hook)

Checkstyle (Coding Style)
- Changeset
  - Result
  - Log
  - Comment
  - Issue ID
  - Files
  - Diff
  - User
  - Date

Hudson (Continuous Integration)

JUnit (Unit Tests)
- Unit Test
  - Result
  - Log
  - Revision
  - Date

Cobertura (Code Coverage)
- Coverage
  - Class
  - Coverage
  - Revision
  - Date

Release Script (Deployment)
- Release
  - Revision
  - File
  - Version
  - Date

Maven (Build Automation)
- Build
  - Result
  - Log
  - Revision
  - Date

Legend
- User Related Tool
- Independent Tool
- Artifact
  - Attributes
- User Interaction
- Trigger Interaction
- Tool Interaction

Continuous Integration
Implementation
Graph Database and Query Language

Graph Database *Neo4j*

- High-performance NoSQL graph database

Query Language *Gremlin*

- Graph-based programming language for property graphs
Queries
How many commits did developer X contribute to release Y?
How many commits did developer X contribute to release Y?

$$release := g: key(_g, 'string', string($release))$$

$$commits := release/outE/inV/inE/outV[@type='commit']$$

$$relevant := commits[outE/inV[@type='user' and @name=string($developer)]]$$

$$count := count(relevant)$$
Which requirement causes the most build failures?

```xml
$ids := g:dedup(g:key($g, 'type', 'issue')/@identifier)
$results := g:map()
foreach $id in $ids
    $issues := g:key($g, 'identifier', string($id))
    $revision := $issues/inE/outV[@type='commit']
        /inE/outV[@type='revision']
    $build := $revision/inE/outV[@type='build']
        /inE/outV[@exit_code>0]
    g:assign($results, $id, count($build))
end

$most := g:keys(g:sort($results, 'value', true()))[1]
```
Open Research Topics

• Hiding the complexity of queries
• Visualization of query results
• Standardized semantics/ontology for software development processes
Questions?

Summary

- Recording Provenance during run-time
- Deep insight into software dev. processes
- Higher trust in software quality
- Allows reuse with more confidence
- Current research field!

Andreas Schreiber
Twitter: @onyame
http://www.dlr.de/sc