Provenance of Software Development Processes

Heinrich Wendel, Markus Kunde, and Andreas Schreiber

Simulation and Software Technology
German Aerospace Center (DLR)
51147 Cologne, Germany
{Heinrich.Wendel,Markus.Kunde,Andreas.Schreiber}@dlr.de
http://www.dlr.de/sc

Abstract. "Why does the build fail currently?" - This and similar questions arise on a daily basis in software development processes (SDP). There is no easy way to answer these questions, the required information is stored throughout different tools, the version control and continuous integration systems in this example. The tools mainly live in isolated worlds and no direct connection between their data exists. This paper proposes a solution to such problems, based on provenance technologies. After outlining the complexity of a SDP, the questions arising on a daily basis are categorized. Finally an approach to make the SDP provenance-aware is proposed based on PRiME, the Open Provenance Model and a SOA architecture using Neo4j to store the data, Gremlin to query it and REST webservices as connection to the tools.

1 Introduction

Research in provenance focuses on a variety of topics, ranging from suitable models to useable libraries and informative visualizations. Those technologies have been tested on real world use cases, mainly scientific workflows from areas like engineering, medicine and bioinformatics. Moreau provides a very detailed overview over the research performend in this area [1]. This paper focuses on a new field of application, namely software development processes (SDP).

Some effort has been invested to record the execution of programs, e.g., by recording all Java method calls [2]. Traceability deals with the links between requirements, design artifacts, tests and code in both directions [3]. Application Lifecycle Management Systems (ALM) provide integrated tool suites to manage artifacts and their relationships in an SDP [4]. Recording the interaction of tools has been handled in the Taverna project, related to scientific workflows [5]. Automatic reasoning on collected information, stored using semantic web technologies, can be done using the Proof Markup Language [6].

In contrast to those approaches the paper focuses on recording the interactions between developers and a distributed tool suite in an SDP and the resulting artifacts. Stored in a graph databases provenance questions can be executed using a graph query language.

Chapter 2 describes a typical SDP and the tools used in it, showing the need of the proposed solution. Chapter 3 gives a categorization of questions

D.L. McGuinness, J.R. Michaelis, and L. Moreau (Eds.): IPAW 2010, LNCS 6378, pp. 59–63, 2010.

occuring during the process. Those questions have to be answerable through the information collected by the provenance system outlined in chapter 4. Finally a summary and evaluation of the approach is given in chapter 5.

2 Software Development Processes

Due to the complexity of today's software a large number of development process models evolved. Although those processes not necessarily force the usage of certain tools, the development can be simplified and sped up by their usage. A typical tool suite at DLR consists of an integrated development environment, a version control system, an issue tracker, a continuous integration framework and a documentation management system.

A lot of interaction occurs between developers, the tools they use during the development process and automatically between different tools. Examples of those interactions are: i) discussion about a feature request; ii) entering or changing requirements in an issue tracking system; iii) automatic code style checks during a check-in.

Information about those processes is, if available at all, distributed over the different tools used. Version control systems feature a history of all files and their editors, issue tracking systems a list of all comments for an issue. Still, the missing link between these different tools makes it either impossible to draw conclusions from this information or is at least very time-consuming, especially when the immense amount of available data is considered.

3 Questions

A lot of questions arise during such complex processes. Based on an internal survey at DLR they have been categorized into one or more of the following groups:

Error Detection: During day to day development it often happens that builds or unit tests suddenly fail. In such cases it is important to identify the source of the error. In many cases this might be the responsible developer, who should be contacted first, because he has the most knowledge to fix the problem. Sometimes it might also be the failure of a tool, e.g., ocurring after an upgrade.

Quality Assurance: Customers are always interested in a product with maximal quality, therefore quality assurance has always been a very important topic, not only in the domain of software engineering. The number of unit tests or code coverage percentages give important hints on where the quality of the product is very good or still deficient.

Process Validation: Following a defined process is another way of performing quality assurance. By following norms like ISO 9001 the quality of the final product is not assured, but the quality of the process that led to the product. This is especially important in the area of medical software, where a process validation is required.

Monitoring: Often problems do not become visible until a closer look at the project. Automatic monitoring and notifications can help to identify those problems, e.g., to see if an issue takes longer to implement than expected.

Statistical Analysis: Statistics help to interpret and draw conclusions out of collected data. They can be used by managers, e.g., to decide if the project is in time, needs more resources or if developers can be put into another project. Developers are interested in statistics to see how they perform or compare to others.

Process Optimization: Another use of monitoring, error detection and statistics is the optimization of the process itself. E.g., a lot of commits related to one issue might show that the issues should be split more fine-grained next time. A build tool that fails in a lot of cases because of segmentation faults might be replaced by a new one.

Developer Rating: There is no widely accepted method to rate the productivity of developers; and it might not be a popular topic. Still the collected data can give some hints in order to decide which developer to assign to a specific problem. It might show that some developers are better in writing unit tests and some in documentation and helps to improve the process.

Informational: Sometimes data has to be collected for informational purposes, e.g., when creating a release announcement it shall contain a list of all bugs fixed.

4 A Provenance-Aware Software Development Process

In order to answer such questions the SDP has to be made provenance-aware using PRiME [7]. Originally PRiME was created for applications, not for processes or the later developed Open Provenance Model (OPM), therefore a few adaptions are needed to apply it to SDPs. First, the breackdown into individual application components, has to be changed to a breakdown into individual subprocesses. Second, instead of using interaction graphs, OPM graphs are used to picture the interactions between the actors. Using this methodology an OPM meta-model for the individual SDP can be created.

Based on the Neo4j graph database, which has successfully been used to store provenance information [8], a service oriented architecture to record information using this meta model can be implemented. Served by a web server, individual interfaces are exposed via REST to allow the insertion of new data. The services are secured using HTTP basic authentication. Afterwards each tool has to be extended to call the appropiate REST interface when new actions are performed. Usually the core of the tool must not be changed, because they provide some kind of hook mechanism which allow the execution of actions on specific events.

Finally a second interface is provided, allowing to query the recorded data using the graph programming language Gremlin [9]. The questions, previously

stated in a human-readable format and analyzed by its starting item and scope using PRiME, can be translated into Gremlin queries. The queries can be executed using a provenance console served by a webserver.

5 Evaluation and Conclusions

The proposed approach has been implemented and evaluated using the SDP of the distributed simulation framework Remote Computing Environment [10] at DLR. The adapted metholodogy and selected technolgies could be successfully used and offers the possibility to answer questions from all categorizes summarized in chapter 3. The integration into the distributed tool suite was possible and showed a reasonable performance.

Some minor issues in the detailed modelling process regarding index structures and best practices for using Neo4j remain. Furthermore it is questionable if OPM is really needed for modelling or any arbitrary graph would suffice. OPM produces some overhead and could still be used as data exchange format.

Although it was possible to answer all given questions using the Turing-complete language Gremlin it is not intuitive to use. More work can be spend on visual query technologies. The Eclipse plug-in Neoclipse already offers graph navigation mechanisms. Combined with a meta-model definition, currently under development for Neo4j, a way to graphically specify queries could be developed.

Medical software must be developed following certain process models, the provenance model could be used to verify the compliance to the process. Even if the process itself is valid the tools might fail and prevent the process from working, which could also be detected. Moreover it might be possible to extend the approach to development processes in general, not focusing on software, but, e.g. system design or other engineering domains.

References

1. Moreau, L.: The foundations for provenance on the web. Technical report, University of Southampton (2009)
2. Miles, S.: Automatically adapting source code to document provenance. In: Proceedings of the 3rd International Provenance and Annotation Workshop, Springer, Heidelberg (2010)
3. Kannenberg, A., Saiedian, D.H.: Why software requirements traceability remains a challenge. CrossTalk The Journal of Defense Software Engineering (2009)
4. Schwaber, C.: The changing face of application life-cycle management. Technical report, Forrester (2006)
5. Zhao, J., Goble, C., Stevens, R., Turi, D.: Mining taverna's semantic web of provenance. Concurrency and Computation: Practice and Experience 20(5), 463–472 (2008)
6. da Silva, P.P., McGuinness, D.L., Fikes, R.: A proof markup language for semantic web services. Inf. Syst. 31(4), 381–395 (2006)

7. Munroe, S., Miles, S., Moreau, L., Vázquez-Salceda, J.: Prime: a software engineering methodology for developing provenance-aware applications. In: SEM 2006: Proceedings of the 6th International Workshop on Software Engineering and Middleware, pp. 39–46. ACM, New York (2010)
8. Tylissanakis, G., Cotronis, Y.: Data provenance and reproducibility in grid based scientific workflows. In: GPC 2009: Proceedings of the 2009 Workshops at the Grid and Pervasive Computing Conference, pp. 42–49. IEEE Computer Society, Los Alamitos (2009)
9. Gremlin - a graph-based programming language, `http://wiki.github.com/tinkerpop/gremlin/`
10. Remote component environment, `http://www.rcenvironment.de/`