Process mining software repositories: do developers work as expected?
Poncin, W.; Serebrenik, A.; van den Brand, M.G.J.

Published in:
ERCIM News

Published: 01/01/2012

Document Version
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. Jan. 2019
Maintainer scripts are executed during upgrades and they are fully-fledged programs usually written in POSIX shell language. Moreover, they are run with system administrator rights and may perform arbitrary changes to the whole system. EVOSS defines a domain-specific language (DSL) to specify script behaviour: this is a way to limit the expressive power of the language used in maintainer scripts, without reducing the functionality of the scripts themselves. The DSL includes a set of high-level clauses with a well-defined transformational semantics expressed in terms of system state modifications: each system state is given as a model and script behaviour is represented by corresponding model transformations.

The idea of EVOSS is to simulate system upgrades through its upgrade simulator component (see Figure 1), which allows system users to identify upgrade failures before performing the upgrade on the real system. The simulator takes into account both fine-grained static aspects (e.g., configuration incoherencies) and dynamic aspects (e.g., the execution of maintainer scripts). The simulator is based on model-driven techniques and makes use of a model-based description of the system to be upgraded. In order to build the system’s configuration and package models, EVOSS makes use of model injectors that extract models from existing artifacts. The outcome of system injection is a model that represents, in a homogeneous form, different aspects of a running system, such as installed packages, users and groups, mime type handlers, alternatives, implicit dependencies, etc. The outcome of package injection contains modelling elements encoding both the package and its scripts (as DSL statements).

The fault detector is then used to check system configurations for incoherencies. The coherence of a configuration model is evaluated by means of queries which are embodied in the fault detector. Obviously, it is not possible to define once-and-for-all a complete catalogue of faults because they are based on experience and acquired know-how. The fault detector has thus been designed to be open and extensible so that new queries can be added whenever new classes of faults are identified.

In summary, EVOSS represents an advancement, with respect to the state of the art of package managers, in the following aspects: (i) it provides a homogeneous representation of the whole system’s configuration in terms of models, including relevant system elements that are currently not explicitly represented, (ii) it supports the upgrade simulation with the aim of discovering failures before they can affect the real system, (iii) it proposes a fault detector module able to discover problems on the configuration reached by the simulation.

Links:
http://www.mancoosi.org
http://evoss.di.univaq.it/

Please contact:
Davide Di Ruscio, Patrizio Pelliccione or Alfonso Pierantonio
University of Aquila, Italy
E-mail: davide.diruscio@univaq.it, patrizio.pelliccione@univaq.it, alfonso.pierantonio@univaq.it

Process Mining Software Repositories: Do Developers Work as Expected?

by Alexander Serebrenik, Wouter Poncin and Mark van den Brand

Modern software development commonly makes use of a multitude of software repositories. How can these help us to understand the ongoing development process? Researchers of Eindhoven University of Technology design new methods revealing how software has been developed.

Modern software development commonly makes use of various configuration management systems, issue tracking systems, automated quality control and testing systems. Analysis of data in the repositories can give valuable insights into the development process of legacy systems. For example, it can answer the following software engineering questions: is the project development documentation kept up-to-date with the implementation? How fast are bugs resolved and feature requests implemented? Which (groups of) developers are responsible for the introduction of bugs or overly complex code and code that does not meet company-specific and/or industry guidelines and standards such as MISRA? When and where have particular standard violations been introduced and (how) are...
they related to later discovered bugs? If there is no relationship, is it worth narrowing the scope of routine quality checks to the actually important ones? What is the share of software artifacts covered by tests and how does this share change in time (eg from version to version) and project space (from subsystem to subsystem or one developer group to another)?

In the ongoing project at Eindhoven University of Technology, we design a generic framework that allows the user to choose repositories, as well as analysis techniques depending on the software engineering question at hand. This is in sharp contrast with many existing repository mining approaches that consider a single specific question (eg which developers are responsible for introduction of bugs) or one specific kind of repository or group of repositories (usually a version control system or a version control system together with the bug tracker).

To create the generic framework we need to separate the preprocessing step, consisting of choosing and combining different repositories, from the analysis and visualization step. The first step takes care of specific challenges pertaining to the presence of multiple repositories, such as matching identities of developers and artifacts mentioned across different repositories, as well as synchronization of the time stamps. This step is carried out by a tool called FRASR. FRASR extracts from various data sources such as version control systems, bug trackers, mail archives, news groups, Twitter messages or issue reports, and combines this information in one log file. Given the log file, the second step implements a broad spectrum of analysis and visualization techniques supporting the user in answering the software engineering question. To implement the second step we make use of the existing work on process mining and ProM, a generic open-source framework for implementing process mining tools. Process mining aims at discovering, analyzing and visualizing business process models given a log of an information system. We stress that decoupling log preprocessing (FRASR) from the actual mining (ProM) makes a broad spectrum of analysis and visualization techniques readily available, and creates a highly flexible platform for application of repository mining.

We have successfully applied process mining software repositories to study a number of open-source software systems and student projects. We have considered such aspects of the development process as roles of different developers and the way bug reports are handled. In student projects we have focused on investigating whether the students adhered to the prescribed software development process. Using FRASR+ProM combination we have produced the following visualizations. For one of the projects the visualization clearly shows two triangles corresponding to the prototype implementation and the final implementation. For another project the visualization shows one triangle starting from the beginning of the project and continuing throughout, ie the prototype was reused as part of the final implementation delivered to the customer.

Process mining of software repositories provides a solid basis for prediction-based analysis. While the preceding questions aimed at providing insights into the way software and related documents have evolved so far, prediction-based analysis focuses on providing insights into the way software and related documents can or will evolve in the future. We consider developing appropriate prediction techniques as an important direction of the future work.

Process mining of software repositories is a novel promising approach that allows developers, designers and managers to rapidly gain insights in the way the development process is progressing, obstacles it is facing and challenges it has to address.

**Link:** http://www.frasr.org/

**Please contact:**
Alexander Serebrenik, Eindhoven University of Technology, The Netherlands
Tel: +31 402473595
E-mail: a.serebrenik@tue.nl

**Figure 1:** Visualizations of FRASR+ProM: the horizontal axes represents time, lines correspond to files, colored spots indicate events pertaining to the files such as creation (blue), modification (white) and deletion (red). Big red triangles have been added for readability purpose. The figure on the left shows only one triangle starting at the beginning of the project, ie, despite the prescribed development guideline, the prototype has been reused in the final implementation. The figure on the right shows two triangles: the triangle on the top corresponds to the prototype, the triangle on the bottom corresponds to the final implementation, ie, software development adhered to the prescribed guideline.