Discovering Activities in Software Development Processes

Saimir Bala, Paul Kneringer and Jan Mendling

Vienna University of Economics and Business (WU), Welthandelsplatz 1, 1020 Vienna (AT)

Abstract

Software development processes are complex to monitor as they involve the coordination of many resources working with different tools. This makes it hard to apply mining techniques for monitoring the process. A key challenge for using traces of tools such as version control systems (VCS) is to find meaningful abstractions in order to identify the work that was actually done. In this paper, we use data from VCS to analyze the actual progress of software-development processes. We develop a technique that is able to mine the activity types of which the development processes consists. We implement our technique as a prototype in Java and evaluate its outputs in terms of effectiveness. In this way, we are able to graphically uncover new behavioural patterns in real-world data from existing open-source GitHub repositories.

Keywords

activity discovery, fine-grained event data, mining software

1. Introduction

Software development processes involve the coordination of multiple resources working on different parts of the software at the same time. As these resources focus on specific parts of the overall development process, it is difficult to obtain transparency of the current status of the project. At the same time, monitoring such process is required in order to avoid risks of running out of time, budget or not meeting established quality objectives.

These type of processes have been referred to as *project-oriented* business processes or simply as *projects*. Monitoring a project is complex because there is hardly any central control of the work progress. One type of system that is extensively used in software development are version control systems (VCSs). They are used to keep track of the changes of the different files that constitute a project. Trace data generated by VCS is a starting point for mining these type of processes. However, only a few approaches [1, 2] focus on analyzing the status of software development from these fine-grained VCS traces. Thus, there is a need to fill in this gap with further techniques that allow to identify additional aspects of the development process, such as the actual activities done by developers.

PoEM'20 Forum: 13th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling, Forum, November 25–27, 2020, Riga, Latvia
EMAIL: saimir.bala@wu.ac.at (S. Bala); paul.kneringer@gmail.com (P. Kneringer); jan.mendling@wu.ac.at (J. Mendling)
URL: https://github.com/s41m1r (S. Bala); https://github.com/PaulKner (P. Kneringer)
ORCID: 0000-0001-7179-1901 (S. Bala); 0000-0002-7260-524X (J. Mendling)

In this paper, we provide a technique for capturing the progress of a project in such a way that it becomes clear what work activity is being done over time. We define fundamental concepts for representing these processes, upon which we develop a novel analysis technique. Our prototypical implementation of this technique is able to represent the status of the development process as well as the activity that is being done.

The rest of the paper is organized as follows. Section 2 details the problem, positions our contribution against existing literature, and defines the requirements for the design of the artifact that solves the stated problem. Section 3 defines preliminary concepts and presents the approach to mine the activities. Section 4 describes the implementation of the artifact and shows its application to real-world projects. Section 5 concludes the paper.

2. Background

Problem description. The problem discussed in this paper is the monitoring of software engineering projects. These projects present the following characteristics. First, they are hardly repetitive. That is, while best practices learned from one project endeavour can be reused, it is never the case that the project is rerun exactly in the same way. Second, they are worked on collaboratively by many participants who regularly document their progress in a semi-structured way. Third, the work is performed under clear constraints in terms of time, budget and quality. Fourth, the workflow does not follow an imperative process model and is not managed by a process engine. Fifth, despite the aforementioned limitations, project managers require transparency of the process in terms of being able to distinguish what kind of work activities where done when and by which resource. In this paper, we consider the terms “type-of-work” and “activity” as synonyms.

Project managers need tools that help them understand where the project is currently standing and how they have performed in retrospect. Being able to see what work was done and when opens up possibilities to understand inefficiencies about how the process was conducted. Moreover, managers need to access information in a timely manner. Therefore, a representation of progress over time is important. Theoretical models of the development process, such as the Rational Unified Process (RUP), are useful for planning software projects but they fall short when it comes to monitoring.

Fortunately, software development projects store rich trace data. Participants (e.g., resources, users) use many tools for different purposes. A common tools is used in any professional software engineering project are VCSs. These systems allow users to collaboratively work on the same project. They manage the different versions of files created by users at any point in time. As well, they keep track of all the changes done by resources at all times. These kind of systems represent a starting point for analyzing projects. As software projects may contain hundreds of thousands files, it becomes prohibitive to manually check what work was done in the project. This calls for automatic analyses and reporting.

Typical *activities* (i.e., type of work) traced by VCSs fall under the categories *Code*, *Documentation*, *Test*, etc. Major VCSs like Git and Subversion, do not provide direct support for understanding the activities executed by developers. However, these tools provide rich event logs which record all the changes made to any file. Listing 1 presents an excerpt of trace data from a publicly available GitHub repository. Specifically, the trace data shows information about two commits, which are activities performed by developers to save their work progress. These commits have a unique identifier and provide information about *i)* the author (i.e., the human resource who issued the commit); *ii)* the date (i.e., a timestamp recording the instant when the commit was made); *iii)* a natural-language textual description filled in by the author; and *iv)* a list of files that were either modified (M), added (A) or deleted (D).

Listing 1: Excerpt from VCS log data from git

```
commit b0346a47df142394da820e1e5d0f7e31b41a70d3
Author:  s41m1r
Date:    Wed Feb 4  13:05:14  2015  +0100

    Deleted  TODOs

D  MiningSVN/TODOs
M  MiningSVN/src/reader/GITLogReader.java

commit 30c5e536e88501295aa3f226645953c69e8f3947
Author:  s41m1r
Date:    Wed Feb 4  15:31:05  2015  +0100

    Works with GIT (hopefully)

M  MiningSVN/src/reader/GITLogReader.java
A  MiningSVN/src/test/TestReadGIT.java
```

As real-life event logs may contain a large amount of commits, it is imperative to use automatic tools to discover the activities. While it is possible to take into account the commit messages and apply Natural Language Processing (NLP) techniques to classify the various changes [3], Listing 1 suggests that there are no guarantees that the textual descriptions are informative about the activity. For this reason, this work focuses on the type of file that was modified rather than relying on the commit comments.

Therefore, the problem is how to exploit low-level trace data for extracting project knowledge that is informative to the manager. We translate this problem into the following requirements.
RQ1. (Processing of VCS event logs). The prototype must extract valuable information from VCS data.
RQ2. (Identification of the activities). The prototype shall classify what activity is done and when.
RQ3. (Computation of KPIs). The prototype shall provide Key Performance Indicators (KPIs) that are understandable by project managers.
RQ4. (Visualization of project status). The prototype shall provide a high level overview of the project.

Related work. Literature related to the aforementioned requirements can be classified into two main groups: *(i)* software engineering; and *(ii)* business process management.

Contributions in *(i)* focus on event data generated by systems like VCS, issue tracking, bug tracking, mail archives, etc. They mainly aim at either finding correlations between activities performed by resources and the artifacts in the repositories [4] or at analyzing the evolution of changes over time [5]. These works typically provide powerful techniques that help with processing events [6] from software data and further abstracting them into coarse-grained activities [4, 7, 8] and understanding type of work (i.e., activites) and KPIs [9, 10]. While these works are fundamental in dealing with software repositories, they are typically process unaware. Therefore, do not provide a process representation.

Contributions in *(ii)* fall under the *process mining* umbrella. Typical approaches focus on transforming software development data in process-mining compatible event-logs [11, 12], by making assumptions on what to consider as a case identifier. Other works focus on enabling process analytics on top of fine-grained events from evolving artifacts [13]. Finally, there are process-aware works that deal with software repositories. In particular, the work from [14] analyses bug resolution processes and the work from [2] uses VCS data to analyse teams. Most of the techniques in the process mining area have specific requirements about their input (i.e., an event log with defined case, activity, and timestamp attributes). These works cannot be readily applied to data from software development [15]. In this paper, we focus on automatic identification of the activities based on file types as described in [9]. Moreover, this work is process aware and presents the data from a perspective which is more targetted towards project managers.

3. Discovering software development process activities

We developed a technique that takes as input an event log from VCS and extracts visual insights as well as KPIs about the development process. In the following, we describe the steps of our technique.

Preprocess VCS log file. The input of this phase is a log in the *unified diff format*, which is offered by major VCSs such as Git and Subversion. The information retrieved by the VCS is configurable by the user. More specifically, it is possible to obtain the basic information shown in Listing 1 along with details on the differences among versions of the same file (i.e., which and how many lines changed from version 1 to version 2 of file X). In order to extract such information, the raw event log is parsed. We used the parser from [16]. This parser generates events which are then stored into a Database Management System (DBMS) for further processing.

From a log event, we extract the following entities: *i)* Project; *ii)* User; *iii)* Commit; *iv)* File; and *v)* Edit. *Project* represents the software project at hand. By including this entity in the data model, we can gather data over multiple projects and store them in the same database. *User* is the user who performs change operations on the files. *Commit* represents the status of the repository at a given point in time. Commits must contain a revision number, a timestamp and the user who issued them. *File* is a single file of the repository identified by its full path. *Edit* captures the change as numbers of lines added to or removed from a file. This step fulfills **RQ1**.

Classify activities. Having the data stored in a DBMS, enables us to run several analyzes already at this level by simply issuing SQL queries. For example, we can obtain all changes that happened to single files during their lifetime. For the scope of this work, we collect all the file paths, all the changes that happened to files, the amount of change in terms of lines of code (LOC), the type of change (e.g., addition, modification, deletion), the commit identifier, and the user who did the change. Next, we automatically categorize the type of change. For this, we apply regular expressions on path attribute in accordance to the classes provided in [9]. Examples of classes are *Testing*, *Coding*, *User interface*, etc. There are in total fourteen classes. When a type of change does not belong to any class, it is put under the category *Unknown*.

Table 1
Set of activities that are discovered and main regular expressions

Activity	Abbreviation	Regular Expression
Unknown	unknown	.*
Documentation	doc	.*\/doc(-?)book(s?)\/.* .*\/info .*\.txt((\.bak)?) .*\.man .*\.tex
Image	img	.*\.jpeg .*\.bmp .*\.chm .*\.vdx .*\.gif
Localization	loc	.*\/locale(s?)\/.* .*\.po(~?) .*\.charset(~?)
User interface	ui	.*\.ui .*\.gladep(\\d?)((\.bak)?)(~?) .*\.theme
Multimedia	media	.*\.mp3 .*\.mp4 .*\/media(s?)\/.* .*\.ogg
Code	code	.*\.jar(~?) .*\/src\/.* .*\.r((\.swp)?)(~?)) .*\.py((\.swp)?)(~?) .*\.php((\.swp)?)(\\d?)(~?)
Meta	meta	.*\.svn(.*) .*\.git(.*) .*\.cvs(.*)
Configuration	config	.*\.conf .*\.cfg .*\.project .*\.ini .*\.prefs
Build	build	.*\.cmake .*\/install-sh .*\/build\/.* .*makefile.*
Development documentation	devdoc	.*readme.* .*\/changelog.* .*\/devel(-?)doc(s?)\/.*
Database	db	.*\.sql .*\.sqlite .*\.mdb .*\.db
Test	test	.*\.test(s?)\/.* .*\/.*test\..* .*/test.*\..*
Library	lib	.*\/library\/.* .*\/libraries\/.*

We have adapted the regular expressions to our case and enriched the list of rules from literature. Table 1 shows the activity types and the main regular expressions we use to classify files onto specific types of work. For the sake of space, the majority of the regular expressions is left out. The reader can access the full list of regular expressions on our GitHub repository whose link is provided in the following section.

For the categorization we consider both the extension of the file and its path. For example, a file with the path `/test/file.java` is labelled as *Testing* rather than *Coding*. To achieve this, we sort the matching rules in order of specificity. At a higher level, a commit involves multiple files. In order to fit the commit into a specific class, we rely on majority voting as follows. We iterate over the list of changes affected by the commit and sort the number of changes by their activity and the amount of change. We select the activity that is associated to the highest number of changes. With this step we fulfill **RQ2**.

Compute KPIs. Next, we compute KPIs with the help of the DBMS. This allows for a customized set of KPIs to be implemented. In the scope of this paper, we reproduced some of the main KPIs form literature [9]. We divide them into basic (absolute and relative) and specialization metrics. Basic metrics focus on descriptive statistics such as frequency counts of how many times each user works on a file. Specialization metrics focus on the measuring imbalance of work towards a specific file or author. Imbalance is captured by the Gini inequality index [17].

Table 2
KPIs computation details

KPI	Description	Calculation
PW	(absolute) project workload	$\sum_{t \in T, u \in U} UTW(t, u)$
TW (t)	workload of a specific activity	$\sum_{u \in U} UTW(t, u)$
NAP	number of authors in the project	$\sum_{u_j \in U} j$
NTP	number of activities in the project	$\sum_{t_j \in T} j$
PIS	specialization of user involvement the activities of the project	$Gini_{t_k \in T}(\sum_{u \in U} UTI(u, t_k))$
RPIS	specialization of relative user involvement over the activities of the project	$Gini_{t_k \in T}(\frac{\sum_{u \in U} UTI(u, t_k)}{NAP})$
RPWS	specialization of relative workload across all activities in the project	$Gini_{t_k \in T}(\frac{\sum_{u \in U} UTW(u, t_k)}{TW})$

We implemented the following basic project metrics: project workload (PW), type of change workload (TW), number of authors in project (NAP), number of types of work in project (NTP). Furthermore, we also implemented the following specialization metrics: specialization of author in each activity type (PIS), specialization of relative author in each activity type (RPIS), and specialization of relative project workload (RPWS).

Let U be the set of all unique users in the project, T the set of all activities in the project. Files that were edited in the context of a commit can be associated to a user $u \in U$ and an activity $t \in T$ computed as described above. We adapted the definition of KPIs from literature to support the extraction of information about the activity types as follows. As a first step, we redefine the two basic KPIs that are involved in the calculation of all other KPIs. User–Activity–Workload (UTW) is the number of files relative to activity t, a user u edits over the entire history of the activity. User–Activity–Involvement (UTI) is 1 if a user u has been involved in (i.e., has edited at least once) a file with activity w. It is 0 otherwise. Finally, using these definitions, we can present in Table 2 how the rest of the KPIs is computed. With this step we fulfill **RQ3**.

Visualize results. The last step of our technique deals with the presentation of the results. As our prototype needs to be informative to project managers, we chose to graphically display the results of the previous two steps on a friendly user interface. The user interface takes as input the results of the classification of the activities as well as the set of the computed KPIs. The main goal of the user interface is to show two fundamental aspects of the project at hand. First, it visualizes the evolution of each activity aggregated by customized periods of time (e.g., weeks, months). By doing so, our prototype helps at vizualing the general behaviour as RUP phases. This enables the project manager to compare the ideal project evolution to the actual one. Second, we display the various KPIs in a dashboard (e.g., barchart). Further information, such as the commit identifiers, file names, amount of changes, and users who worked on the files are also made available. This allows the project manager to zoom into specific parts of the project for more detailed analyses. With this step we fulfill **RQ4**.

4. Preliminary results and discussion

Our prototype is named ActiVCS and is available as open source on GitHub (https://github.com/PaulKner/ActiVCS).

Visualization of projects. Figure 1 shows one output of our prototype. This visualization plots the evolution of the automatically extracted activities *code* and *test*. This visualization is part of a UI that shows several other properties. The domain expert can execute a number of further actions. These actions are the following: *i)* load an event log; *ii)* save an event log for reuse and avoid parsing it anew; *iii)* get help information; *iv)* change granularity of the timeline by choosing to display data daily, weekly or monthly; *v)* change data series on X-axis between commit-level and file-level; and *vi)* four buttons to change size and type of the plots. The plots shown in the frame illustrate the evolution of the identified types of work over time. The user can choose between having this information on commit- or file-level.

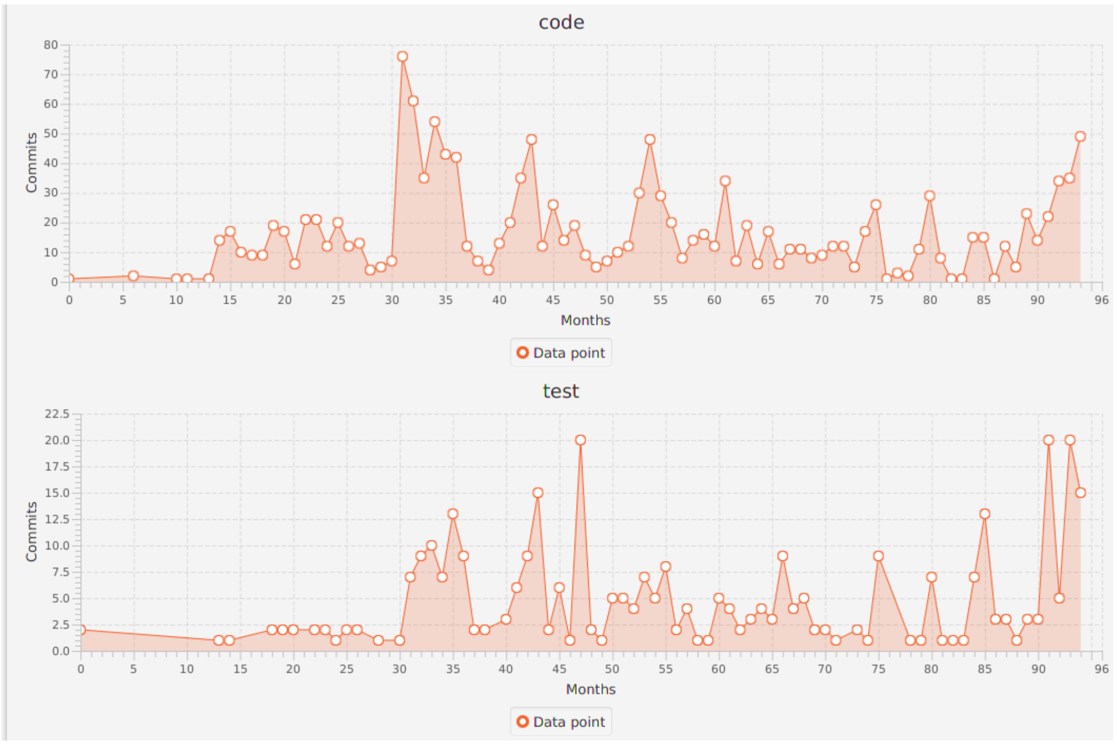

Figure 1: Zoom on the Code and Test types of work from the *ok* project

It is possible to interact with each point on the plot shown in the main frame. Depending whether the X-axis is showing the information at commit-level or at file-level, a pop-up menu that summarizes the information related to that point of the plot is shown. KPIs about the overall project are also available. Especially, our tool offers a separate tab in which it is possible to visualize bar-charts based on the values of the KPIs (e.g., what are the actual activities being done and by how many authors, the size of the project, the GINI index, etc). Further information offered by the tool consists of tabular information about the measures presented in Section 3.

With this information at hand and their knowledge of the domain, the project manager can investigate, for example whether the team is *de facto* following a specific software methodology.

Analyses of real-life open-source projects. We use ActiVCS to analyze real-life open source projects from GitHub. We chose thirteen projects of different size, age, user counts, and programming languages. Selection criteria included having, *i)* a representative variety in terms of programming language, *ii)* variety in the size, *iii)* at least having two similar projects, *iv)* highly active versus inactive, and *v)* fast versus slow growing projects.

This resulted in the following projects. *MPAndroidChart (MP* is a visualization library for Android platforms. *Torque2D (Torque)* is a software engine for the development of video games. *Openage (openage)* is an open source clone of the Age of Empires II engine. *Incubator-dubbo (inc)* is a RPC (remote procedure call) framework for Java. *jekyll* is a blog-aware written Ruby that generates websites from user content. *scrapy* is a Python based framework to extract data from 24 websites *brew* is a software that automatically installs missing packages for MacOS and Linux operating systems. *Algorithms – Java (Java)* is a collection of different Java algorithms. *Flask (flask)* is a lightweight Web Server Gateway Interface (WSGI) web application framework created in Python. *Tablesaw* is a framework for the transformation and visualization of data, implemented in Java. *Okhttp (ok)* is an HTTP client for Java and Android. *Retrotfit (retro)* is another HTTP client for Java and Android. *editor.js (editor)* is a JavaScript based editor software for the creation of documents and the transformation into JSON format.

Table 3
KPIs giving insights on real-life projects.

Name	COM	PW	NAP	NTP	PWS	PIS	C	D	T	U
MP	2012	8552	86	8	**0.8**	0.5	7352	53	24	351
Torque	970	7978	46	9	0.75	0.34	5410	134	241	1561
openage	3136	10529	168	9	0.74	0.5	7697	596	366	1126
inc	3308	31667	240	8	0.69	0.58	16116	6	8232	164
jekyll	10388	14369	1028	9	0.69	0.6	4085	1384	1492	7001
scrapy	7045	15014	409	9	0.68	0.57	7995	612	2784	525
brew	18410	35299	815	6	0.65	0.46	23087	1276	7348	3161
Java	755	902	175	5	0.65	0.58	636	5	7	203
flask	3505	5679	623	10	0.65	0.65	1950	206	1027	481
tablesaw	1893	23700	39	7	0.63	0.39	8779	11149	2188	652
ok	3826	11567	245	7	0.63	0.51	5837	225	3348	260
retro	1721	5168	173	7	0.6	0.43	2467	69	1039	103
editor	494	2337	27	7	0.56	0.24	927	314	0	814

Table 3 shows the KPIs resulting from these projects. Columns contain the following information: number of commits (COM), project workload (PW), type of change workload (TW), number of authors in project (NAP), number of types of work in project (NTP), specialization of project workload (PWS), specialization of author in each activity type (PIS), code (C) , documentation

(D) , testing (T) and unknown (U). Entries have been sorted by the specialization of project workload (PWS). This means that, for instance, resources of project *MP* are highly occupied. Therefore, load balancing should be considered if the managers want to improve the capacity of the team to handle new tasks in the near future.

Finally, the types of work code (C), documentation (D), testing (T) and unknown (U) that are displayed in the table make up the most frequently identified types that were detected with ActiVCS. As expected from software development processes, the main workload in most analyzed projects were coding activities. One exception to this observation is the jekyll project. ActiVCS has detected 7001 file changes with an unknown type within the Ruby software application. Projects like these would provide a solid basis for the identification of additional file types. The total amount of workload that was captured across all projects is 172761. The number of file changes which could not be classified by ActiVCS and were marked as unknown is 16402. This represents 9,49% of the total workload. Excluding the jekyll project from the analysis would reduce this margin to 5,94%. Many of the identified projects neglect the creation of documentation. All of the analyzed projects with a workload value for coding activities of 8000 and more also have a noticeable amount of registered testing activities. This could refer to the fact that large software development processes have a need for automated test activities which must be frequently updated.

Discussion. ActiVCS can be used as a tool for checking conformance to existing development methodologies. Figure 1 shows a screenshot of the types of work Code and Test, taken from the *ok* project, described previously. A project manager can now check that work of the type Code and Test was consistently done throughout the lifetime of the project. Moreover, it is possible to observe that Code and Test were active together most of the time, with Code starting earlier. A typical development methodology that presents such pattern is *agile*.

This might confirm that the *de facto* software development method corresponds with what the enterprise has decided. Alternatively, the enterprise might be following a Waterfall development model. In that case, this pattern may point at a lack of control on the project. The managers can use their domain knowledge along with the provided *factual* information for better decision making.

5. Conclusion

This paper provides an artifact for analyzing event logs from VCS. Our tool is able to visualize the type of work that was executed in a software development process. Starting from fine-grained changes in the different versions of files, it allows to understand what activity was done and when. It also provides important KPIs, such as the effort distribution. These features are important for managers to understand whether the project is deviating from target goals, and in case take corrective actions.

In the future, we will integrate the ActiVCS tool with the Gantt chart miner from [1]. This would offer to the project manager complementary views on the current status of the project. We plan to conduct user studies with managers in order to receive more feedback from domain experts. Finally, we have already conducted a study on selecting KPIs for software development and we plan to incorporate them in the ActiVCS tool.

References

[1] S. Bala, C. Cabanillas, J. Mendling, A. Rogge-Solti, A. Polleres, Mining project-oriented business processes, in: BPM, volume 9253 of *LNCS*, Springer, 2015, pp. 425–440.

[2] L. Jooken, M. Creemers, M. Jans, Extracting a collaboration model from VCS logs based on process mining techniques, in: Business Process Management Workshops, volume 362 of *LNBIP*, Springer, 2019, pp. 212–223.

[3] K. Agrawal, M. Aschauer, T. Thonhofer, S. Bala, A. Rogge-Solti, N. Tomsich, Resource classification from version control system logs, in: EDOC Workshops, IEEE Computer Society, 2016, pp. 1–10.

[4] G. A. Oliva, F. W. Santana, M. A. Gerosa, C. R. B. de Souza, Towards a classification of logical dependencies origins: a case study, in: EVOL/IWPSE, ACM, 2011, pp. 31–40.

[5] T. Zimmermann, P. Weißgerber, S. Diehl, A. Zeller, Mining version histories to guide software changes, IEEE Trans. Software Eng. 31 (2005) 429–445.

[6] T. Zimmermann, P. Weißgerber, Preprocessing CVS data for fine-grained analysis, in: MSR, 2004, pp. 2–6.

[7] A. Zaidman, B. V. Rompaey, S. Demeyer, A. van Deursen, Mining software repositories to study co-evolution of production & test code, in: ICST, IEEE Computer Society, 2008, pp. 220–229.

[8] A. Rodríguez, F. Tanaka, Y. Kamei, Empirical study on the relationship between developer's working habits and efficiency, in: MSR, ACM, 2018, pp. 74–77.

[9] B. Vasilescu, A. Serebrenik, M. Goeminne, T. Mens, On the variation and specialisation of workload - A case study of the gnome ecosystem community, Empirical Software Engineering 19 (2014) 955–1008.

[10] A. Joonbakhsh, A. Sami, Mining and extraction of personal software process measures through IDE interaction logs, in: MSR, ACM, 2018, pp. 78–81.

[11] E. Kindler, V. A. Rubin, W. Schäfer, Activity mining for discovering software process models, in: Software Engineering, volume P-79 of *LNI*, GI, 2006, pp. 175–180.

[12] W. Poncin, A. Serebrenik, M. van den Brand, Process mining software repositories, in: CSMR, IEEE Computer Society, 2011, pp. 5–14.

[13] S. Beheshti, B. Benatallah, H. R. Motahari Nezhad, Enabling the analysis of cross-cutting aspects in ad-hoc processes, in: CAiSE, volume 7908 of *LNCS*, Springer, 2013, pp. 51–67.

[14] R. Marques, M. M. da Silva, D. R. Ferreira, Assessing agile software development processes with process mining: A case study, in: CBI (1), IEEE Computer Society, 2018, pp. 109–118.

[15] A. Tsoury, P. Soffer, I. Reinhartz-Berger, A conceptual framework for supporting deep exploration of business process behavior, in: ER, volume 11157 of *LNCS*, Springer, 2018, pp. 58–71.

[16] S. Bala, K. Revoredo, J. C. de A. R. Gonçalves, F. A. Baião, J. Mendling, F. M. Santoro, Uncovering the hidden co-evolution in the work history of software projects, in: BPM, volume 10445 of *LNCS*, Springer, 2017, pp. 164–180.

[17] C. Gini, Measurement of inequality of incomes, The Economic Journal 31 (1921) 124–126.