Analyzing Side-Tracking of Developers using Object-Centric Process Mining

Saimir Bala[1[0000−0001−7179−1901]], Thanh Nguyen[1], and Jan Mendling[1,2[0000−0002−7260−524X]]

[1] Humboldt Universität zu Berlin, Germany
[2] Weizenbaum Institute, Berlin, Germany
firstname.lastname@hu-berlin.de

Abstract. Managers need to analyze the software development process to pro-actively make decisions that meet quality, budget and time objectives. To aid this analysis, a number of data-driven approaches exist, which can be used for specific purposes, such as computing target key performance indicators (KPIs). In particular, process analysis techniques, like process mining, can analyze data from event logs of information systems and deliver actionable insights on how the *process* is conducted. However, traditional process mining techniques make strong assumptions on the structure of event logs, requiring the existence of a *case* identifier, used to group the traces. As a result, the output of such techniques only provides a narrow view of the reality, leading the manager towards wrong intepretations in cases of *side-tracking*, when a developer is involved in different processes that interleave one another. To account for these cases, we investigate the use of object-centric process mining (OCPM) to analyze software repositories. Our results help to explain performance issues by revealing the contributing factors that hinder the progress of development tasks.

Keywords: process analysis · software repositories · object-centric process mining

1 Introduction

Monitoring the software development process is a complex task as it involves many actors who must coordinate their work effectively. Fortunately, there are vast amounts of data collected in the system logs of software repositories that keep track of the work done by developers. To analyze these data from a *process* perspective, process mining techniques have been used [8]. Recently, object-centric process mining (OCPM) [2] has emerged as a new paradigm that allows for multi-dimensional process analysis [22]. OCPM approaches are particularly affine to data from software. Contrary to traditional process mining approaches, they do no require a notion of *case* (which is not present in software development processes [9]), but rather focus on capturing all relevant entities and their relations.

While approaches from literature have tackled the problem of monitoring tasks in software development in a myriad of ways [4, 31, 39, 42], there is still a lack of process analysis techniques. One reason for this, is given by the multi-dimensionality of the software process that includes various perspectives such as time, resource, control-flow (hand-over of work), and so on. In this context, literature has analyzed these perspective separately [8]. However, this is not sufficient to understand inter-perspective behavior and may lead to wrong interpretation of the results by the manager [2].

In this paper, we extend previous work [36] to adapt multi-dimensional process analysis techniques from OCPM to analyze software data. We show how to process the input gathered from real-world GitHub repositories to construct event knowledge graphs. We use these graphs to both compute key performance indicators (KPIs) and to perform process analysis. Our approach is able to provide deeper insights in the software development process than traditional process mining. In particular, it is able to uncover inter-perspective behavior such as developer *side-tracking*. With this work, we increase the manager's ability to make informed decisions and show the applicability of OCPM to software data.

The remainder of the paper is structured as follows. Section 2 illustrates the problem at hand, reviews the related literature and derives the research questions. Section 3 describes the research methodology, the goals and details all the steps towards the creation of an artifact to achieve the defined goals. Section 4 provides the results of our artifact applied to a real-world repository from GitHub. Section 5 discusses the results against the research questions and Section 6 concludes the paper.

2 Background

This section provides the background of our research. It is divided in three subsections. Section 2.1 states the problem addressed. Section 2.2 presents existing contributions from literature that tackle the problem. On this basis, Section 2.3 derives our research questions.

2.1 Problem Statement

Software development is complex and includes many processes aimed at targetting specific aspects that contribute to the engineering of a working artifact. In reality, building a software product is rarely a sequential process. In fact, this is rather a creative endeavour that depends, among other things, on what is the current status of the development and what issues arise along the way.

This nature of software creates a problem for *managers*, who want to help the team and make the best decisions, but do not have precise information about the progress. Therefore, questions like *will we meet the deadline* or *is there more workforce required in a specific task* become hard to answer. Wrong evaluation of the current development status may, among others issues, lead to loss of revenue,

trust in the customer, low quality of the delivered product, increased maintenance cost, higher technical debt and other issues [11, 17, 50].

To aid the analysis of software projects, trace data generated from software development is used [8, 18]. Specifically, the software development methodology is analyzed in the details of its constituting elements [30, 47]. Especially, data analysis techniques, such as process mining [1, 2, 49] are used to provide results that are best understood by managers. In particular, process analysis helps to address key difficulties [49] by *i)* providing means to handle the complexity of information via filering techniques; *ii)* measuring key performance indicators (KPIs); and *iii)* providing a model that is easier to understand by the manager.

Figure 1 illustrates a common scenario in software development. To implement a specific feature, an actor (e.g., senior software developer) from the development team starts to *Implement functionality A*. While working on this task, the developer is called by a second team because complex issue has occured which needs their expertise. The actor, after finishing their current task in the *Feature development* process, starts to work on the activity *Enter Issue to Backlog*, belonging to the *Issue resolution* process. Working on the issue also requires reporting. The actor does this via completing activity *Report issue* which belongs to the *Issue reporting* process. Then, the senior developer helps the team by contributing to *Resolve current issue* and *Resolve related issues*. Finally, the senior developer can go back to work in *Feature development* by moving to activity *Implement functionality B*.

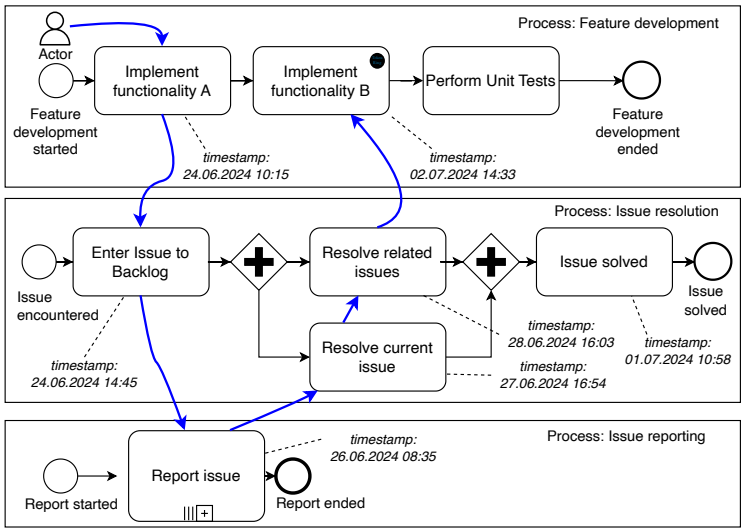


Fig. 1: Illustration of the problem of understanding the progress of a specific software development process. Only the process view is not sufficient to explain why Implement functionality B takes longer.

In this context, a manager needs to analyze the *Feature development* process to understand how long it normally takes to perform the activities that require implementation. Thus, the data about this process would be collected and analyzed. Specifically, the manager can see that *Implement functionality A* was finished at the time marked by timestamp=`24.06.2025 10:15` and *Implement functionality B* was finished at timestamp=`02.07.2024 14:33`. As traditional process mining tools force the choice of the process to analyze via the requirement of a *case*, the manager has only the information about these two timestamps and no information about any event that happened outside the *Feature development* process. Therefore, while in reality the actor performed a number of activites, the manager can only see that it took roughly one week to reach the status in which *Implement functionality B* is done. In this scenario the actor peformed useful activities (connected by blue sequence flows) to help the team, but this is not always the case. Performing different actities by developers is also referred to as *side–tracking*. Side–tracking by developers is not always a desired behavior. Furthermore, it makes it hard for managers to assess the current status and the effort that is put in the development, thus hindering their effectiveness.

2.2 Literature Review

Related to the problem illustrated above, there are three research stream upon which we can draw: *i)* Mining Software Repositories; *ii)* Information Systems Research on Open Source Development; and *iii)* Process Analysis and Object-Centric Process Mining.

Mining Software Repositories. In this research stream fall those works which use data mining techniques to compute quantitative analysis (e.g., KPIs) about the software. In this context, the focus is to learn how users relate to the artifacts in the repositories [37], or at analyzing the evolution of changes over time [55]. To achieve that, these techniques are based on identifying process events [54] from the software repository data and abstracting them onto higher-level activities [37]. The goal is to measure aspects of software such as the type of work (i.e., what kind of files are being worked on) [48], the type of resources [5] or measure KPIs [42]. All these works provide valuable insights into the software development efforts done in the project, but focus on low-level indicators or relations.

Furthermore, there are works that target more the managers, by applying process analysis. Pasuksmit et al. [39] evaluate the state of the art on log parsing to prepare input for process mining techniques. Milewicz et al. [34] look into who is driving the evolution of scientific software in collaborative reserach projects. Schipper et al. [43] propose the use process mining for bottlenecks in sprint planning. All these works only take into account traditional process mining, and are therefore not able to directly detect side-tracking.

Information Systems Research on Open Source Development. There is a substantial body of literature in information systems that uses software data, particularly from open source development, to theorize about the process.

Key theoretical works and empirical studies include the following. Carlo et al. [16] studied the impact of adoption timing on innovation outcomes during disruptive innovation cycles, demonstrating how early and late adoption of new technologies influence innovation performance and strategic decision-making. Crowston [19] applied coordination theory to analyze software change processes, identifying mechanisms for task assignment, resource sharing, and managing dependencies in large-scale software development. Adolph et al. [4] examined how individuals' perspectives about a software project converge towards a common understanding, highlighting the importance of reconciling diverse viewpoints for effective collaboration in open source projects. Yu and Petter [53] used shared mental models theory to explain how teams work together to complete common tasks, emphasizing the need for effective communication and shared understanding, especially in virtual and distributed open source teams. Lindberg et al. [31] investigated coordination mechanisms for resolving pull requests in open source projects, providing practical guidelines for integrating contributions from multiple developers.

Furthermore, Sedano et al. [44] studied waste in software development, developing a taxonomy that identifies nine types of waste, including overproduction, waiting, and defects. This research is valuable for enhancing efficiency in open source projects. Werner and Berry [51] highlighted the use of trace data in analyzing large development projects, showing how detailed trace data can provide insights, identify bottlenecks, and improve project management practices. These studies provide a foundation for designing and developing effective approaches to analyzing the software development process.

Process Analysis and Object-Centric Process Mining. Works in this area aim at understanding how events in software development unfold over time. For that they take into account various elements [49] of the software development. There are approaches to transform software development data in process-mining compatible event-logs [29, 41]. There are also approaches that enable process analytics of fine-grained events from evolving artifacts [13]. More complex approaches use repository data to analyse well-known processes. The work from [32] uses process mining [1] to analyze bug resolution processes, while [7, 10, 26] use version data to analyse commits, and gather insights respectively about the project timeline, hidden dependencies and *de-facto* teams.

Most of the techniques in the process mining stream require the input data to have well-defined attributes (i.e., an event log with defined case, activity, and timestamp). These works cannot be readily applied to data from software development [46]. As well, they only focus on discovering and analyzing predefined relations, by fixing the notion of case and following its traces in the data. However, the notions of case and activity of a process, especially in software data, are in practice loosely defined [9].

In recent years, process mining approaches are moving toward multi-dimensional analysis. Indeed, concepts like object-centric process mining (OCPM) [2] and standards like OCEL [23] are increasingly gaining interest. Thus, the tendency

is to use as much information as possible. One way to holistically capture the information contained in the event logs is through the use of so-called *event knowledge graphs* [28] and store them in graph databases [20].

2.3 Research Questions

Given the opportunities arising by the multi-dimensional process analysis offered by OCPM, we derive the following research question *How can we exploit process analysis to analyze software development traces in a repository?*. As OCPM is a rather new technique, it makes sense to first test its applicability on software data. Furthermore, OCPM should specifically address the scenario illustrated in the problem statement (Section 2.1). Therefore, we divide our main research question into the following two.
RQ1: *How can we exploit Object-Centric Process Mining (OCPM) to analyze software development traces in a repository?*
RQ2: *How can we exploit OCPM to discover developer side-tracking?*

In other words, **RQ1** wants to investigate the applicabilty of OCPM to software development data and its effectiveness to deliver useful input for the manager and **RQ2** wants to test whether OCPM can help identify developer side-tracking, which is not possible to detect with traditional process mining techniques.

3 Approach

In the following, we describe our methological approach. Section 3.1 describes design science research method (DSRM) [25]. Section 3.2 details the Design and Development of the artifact created through DSRM. Section 3.3 describes how the input is preprocessed to be consumed by the artifact. Section 3.4 describes how the information is linked together and event-knowledge graphs are built. Section 3.5 shows what types of analyses are delivered to the manager.

3.1 Research Method

Design Science Reserach Methodology (DSRM) [25] is a research paradigm that focuses on the creation and evaluation of *artifacts* designed to solve identified organizational problems. Originating from the fields of engineering and computer science, DSRM emphasizes the development of innovative solutions through iterative processes of design, implementation, and refinement [40, 52].

Key elements of DSRM include the following. *Problem Identification and Motivation*: Clearly defining the problem to be addressed, including its importance and relevance to stakeholders. This step ensures the research is grounded in real-world needs. *Objectives of a Solution*: Establishing criteria for what constitutes a successful solution. These objectives guide the development and assessment of the artifact. *Design and Development*: Creating the artifact, which can be a model, method, construct, or instantiation. This phase involves leveraging existing

theories and technologies to craft a novel solution. *Demonstration*: Showing how the artifact can solve the problem through experiments, case studies, or simulations. This step provides initial evidence of the artifact's utility. *Evaluation*: Systematically assessing the artifact's effectiveness and efficiency against the predefined objectives. Various methods, including analytical, experimental, and observational techniques, are employed to validate the artifact. *Communication*: Disseminating the results to both technical and managerial audiences. Effective communication ensures that the findings and artifacts can be utilized by others and contribute to the broader knowledge base.

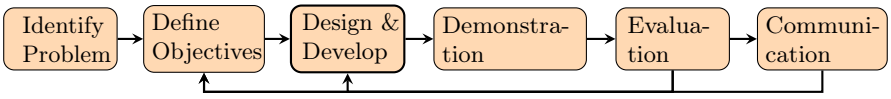


Fig. 2: Design Science Research Process (Peffers et al., 2007 [40])

Figure 2 summarizes the design science research process. DSRM is iterative, often cycling through these steps multiple times to refine and enhance the artifact. Its rigorous approach ensures that the solutions developed are both innovative and practical, addressing the specific needs of organizations while contributing to academic knowledge [25, 40]. This research uses design science to develop a software artifact. The contribution serves the purpose of applying a process mining solution to a new domain, also referred to as exaptation [24]. Next, we describe the details of the design and development phase.

3.2 Design and Development of the Artifact

To implement the Design and Development step of DSRM, we followed a four-steps approach. These steps consist in pre-processing the repository data to extract event, building a knowledge-graph from these events, analysing them via multiple queries, and presenting the results to the end user. Next, we present the goals of our approach following the Goal-Question-Metric (GQM) paradigm from [12]. Then, we describe each step of our approach.

We define the following three main goals and related questions.

GQ1. *Goal 1*: analyze the development of individual modules, especially those showing signs of delay, to identify potentially problematic areas. For instance, if `developerA` discontinues work on `moduleA`, understanding the reasons behind such disruptions is crucial to address underlying workflow or project management issues and provide timely solutions. *Question 1*: What are the reasons behind discontinued work?

GQ2. *Goal 2*: comprehend the impacts of shifting developer attention or side-tracking, thus addressing and mitigating impacts on overall productivity. *Question 2:* How to trace task transitions and discern whether such shifts arise from urgent issues or mismanaged priorities?

GQ3. *Goal 3*: inform long-term planning and resource allocation, especially for specialized knowledge areas. *Question 3*: How to investigate code complexity and recurrent issues in files that necessitate frequent developer attention?

The following five *metrics* are identified to gauge progress towards our goals-questions. As metrics can be used to address more than one question, we provide a their description separately and map them to the relative goal-question (GQ). *Module/Task Progress* [38] measures the completion rate of tasks or modules, focusing on indicators like "task completion time" (GQ1). *Developer Activity* [45] assesses a developer's activity on a project by evaluating the number and frequency of commits (GQ1). *Code Churn* [45] examines the amount of code rewritten or revised, indicating issues with code complexity or quality (GQ1, GQ3). *Issue Tracking* [33] metrics (e.g.,"Number of Open Issues") signal potential problems within the software (GQ1, GQ3). *Task Switching* [14] records instances of developers switching tasks, aiding in identifying its frequency and impact (GQ1, GQ2, GQ3).

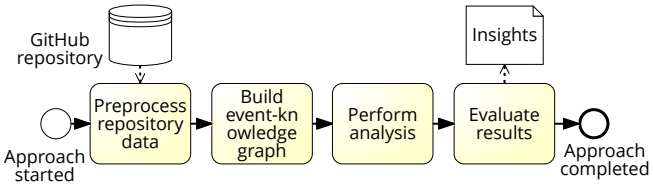


Fig. 3: Artifact for Analyzing Side-Tracking in Software Development Projects

An overview of the steps involved in the artifacts' design are illustrated in Figure 3. The approach takes as input a reference to a software repository and performs four steps. First, it pre-processes the data stored in the repository in order to extract various events. Second, these event data are employed to populate an event-knowledge graph. Third, various analyses are performed on the event-knowledge graph by means of queries. Fourth, the results are collected and presented to a user. In the following sections, we will detail the first three steps, while the evaluation follows in the next section.

3.3 Preprocessing Repository Data

We focused on feature selection from the GitHub API to shape our event knowledge graph, choosing parameters that reflect various dimensions of the development process. Thus, we fetch the datasets from the software repository (e.g., GitHub) and select the most relevant dimensions for our purpose. We include *commit data*, *issue event data*, *pull requests* and *branch data*.

When extracting commit data we include fields like SHA, committer, author's name, commit message, verification status, commit parents, merge status, URL, stats, files involved, author-login, repos, URL, organizations URL, and branch

name. These parameters are chosen to visualize the essence, trajectory, and workflow of the software development process based on file modifications over time after each commit.

When extracting issue event data, we include data about both open and closed issues from the chosen repository, and their correlated events. We fetch information about the issue, event type, commit-id, event-creator, state, and a timestamp indicating whether the issue was closed and when.

When extracting the data of pull requests we include the pull-request number, title, state (closed, open or all), user who made the PR, creation time of the PR, merge time (if merged), and the merge commit's SHA code. With these parameters, we can get a better understanding of what transpires when a developer decides to merge a pull request from one branch into another and how commits are being handled for both the merging branches. We can also compare the data of two states: what data are being generated when users open a pull request versus what data are being recorded after it gets closed.

Finally, when extracting the branch data, we include the branch name for the purpose of better navigation the data from commits and subsequently from pull requests. To fetch the data in a systematic manner, we wrote a Python script and used it together with the requests library. These data served as the foundation for our event knowledge graph and were critical in reflecting the intertwined relationships among different software development events and activities.

3.4 Building an Event Knowledge Graph

In the following, we describe the steps to convert the acquired data from the repository into the event knowledge graph. In order to do so, we rely on Neo4j[3] graph database and its Cypher query language.

Creating Nodes. The first phase consists of loading event data extracted from the repository during the pre-processing step and stored into CSV files. That is, for each of the four datasets *commit data*, *issue event data*, *pull requests* and *branch data*, we have a corresponding .csv file. With these we create the primary nodes of the knowledge graph. Each type of node represents a unique entity in the software development process.

We used a Cypher query[4] to create entities from the *commits* dataset (i.e., `commit.csv` file). From this dataset we can create the following nodes: `Commit`, `Author`, and `File`. Each `Commit` node is associated with an `Author` node representing the individual who made the commit and several `File` nodes representing the files that were altered in the commit. The `Commit` nodes include properties such as `commit_id`, `message`, `URL`, `stats`, `date`, and `merge`, which provide detailed information about each commit while `Author` nodes contain information

[3] https://neo4j.com

[4] All the queries can be found in our GitHub repository https://anonymous.4open.science/r/Multi-Dimensional-Process-Analysis-on-Software-Data-F33F

or each individual. Nodes for `File` help to show the state of each file after a modification has been done by a commit.

Queries for creating entities from the issue, events and pull requests nodes are similar. For commits we create `Author` nodes and for issue events we create `Users` nodes to distinguish the different type of GitHub's users. `Author` are the GitHub users or developers that are actively involve in the process of the development process. `Users` are the people from the open community that contributed to the issue through activities such as comment or reference.

Creating Relationships. Next we describe how we create the relationships.

Branches, Authors, Files and Commits. In this step, relationships `:COMMITTED` (between an `Author` and a `Commit`) and `:BELONGS_TO` (between a `Commit` and a `Branch`) are formed and each commit is linked to the files (`File`) it modifies through the `:MODIFIES` relationship. These are established by connecting the `Author` node who `:COMMITTED` to the `Commit` node and linking each `Commit` to the `Branch` node it belongs to. This will relatively show which commit belongs to which branch and eventually who worked on a specific branch or file.

Directly Follows Relation for Commits. This relationship, represented as `:DF`, connects two commit nodes that directly follow one another in time, regardless of the branch they belong to. This is similar to the commit history displayed on GitHub but not limited to a specific branch. The `:DF` relationship makes it possible to trace the chronological sequence of commits across all branches.

Directly Follows Relation of Commits-Modification. This relationship, symbolized as `:DF_M`, connects two commit nodes that directly follow each other only if they have modified the same file. Like `:DF`, this relationship also tracks the sequence of commits, but it narrows down the scope to those modifying the same file. This allows a detailed view of how individual files evolve over time. It also helps at detecting patterns to highlight developers who stop working on the file.

The relationships `:DF` and `:DF_M` enrich the structure of the event knowledge graph by adding a time dimension. Queries for creating `Issue`, `Event` and `PullRequest` relationships can be found in our GitHub repository.

3.5 Performing the Analysis

We leverage graph databases to extract the following Key Performance Indicators (KPIs) and process models.

Basic KPIs. We start with describing the general (i.e., not process oriented) software development KPIs our approach offers.

Code Churn Analysis. Code churn [35] is the measure of lines of code added and removed from a file over time. The code churn can be calculated using the formula Code Churn = Lines Added + Lines Deleted. We compute this for all the developers. This also allows to rank the developers by contribution (e.g., by sorting the respective churn value in decreasing order).

Ratio of Closed and Open Issues. This metric provides insight into the project's issue management efficiency and effectiveness [33]. Assesses the project's issue management efficiency and effectiveness and indicates well-managed projects and areas for improvement in the development process.

We compute this metric as Ratio (State) = $\frac{\text{Issues (State)}}{\text{Total Issues}}$, where the input parameter *State* can take the values *Open* or *Close* to indicate respectively opened or closed issues.

Cycle Time. This is a performance-related software development metric, representing the time taken to implement, test, and correct a piece of work from the moment work begins until it is ready for delivery [6]. It measures the time taken from beginning work to delivery, providing a more granular view of the development process.

The *cycle time* in software development can be calculated using the formula Cycle Time = Completion Date − Start Date. We do this for all the issues. We are also able to compute the cycle time of each user that has worked on a given issue. With this, we allow for identifying patterns or anomalies in cycle times associated with specific users, providing a more granular view of the development process.

KPIs for Process Analysis. Next, we provide some metrics for process analysis. We focus on, i) issue resolution time, ii) collaboration, iii) file/commit dependency, and iv) issue escalation.

Issue Resolution Time Analysis. With this metric we analyze how long it takes to resolve different types of issues. Specifically, we look into two aspects: the individual issues that take the longest to resolve, and the users who, on average, take longer to complete issues. Utilizing a Cypher query we can perform the analysis by extracting information of issues or users with the longest cycle time with simple queries. More specifically, this is computed as the cycle time of each user that is associated to an issue. That is, it computes the amount of time elapsed between the first and last events of that user on the issue.

Collaboration Analysis. Analyzing collaboration [15] between team members reveals patterns in how team members interact on issues and files. For example, we can identify which team members often work on the same issues or files.

We compute this as two KPIs regarding respectively the issues and the files that were collaboratively worked on. We consider all the *collaborative* events from two users `u1`, `u2` that were recorded within the same issue. We apply the same logic for what concerns the commonly modified files. We return the number of shared issues as the collaboration value. Same holds for the shared files.

File/Commit Dependency Analysis. This metric helps to identify relationships between different parts of the codebase [10]. For instance, it makes it possible to identify files that are frequently modified together, revealing areas of the codebase that are tightly coupled and may benefit from refactoring to improve modularity. We compute this KPI by considering the set of shared commits among the various files. Files that appear together in more commits have a higher dependency with one another.

Issue Escalation Analysis. Analyzing issue escalation [27] helps to identify patterns in issue evolution over time. We identify issues that undergo a larger number of events and consider them potentially problematic as they may require more management attention. To do so, we navigate the event knowledge graph and collect all the issues along with their related events, sorted in decreasing order.

Extracting a Process Model. To extract a process model we construct `:DF` (directly-follows) relationships between event nodes correlated to the same entity node. These relationships link together events of the same type that follows one another according to their timestamps. We repeat this for all the processes or dimension we want to investigate. Then, we classify the event nodes to event classes and retrieve a multi-entity directly-follows graphs (DFGs) through aggregation. Ultimately, through this approach we can obtain a process model that represents multi-entity DFGs.

We applied the techniques from [20, 22]. Hence, we could aggregate the graph nodes to class nodes. Then we constructed filtered directly-follows relationships and retrieved a *proclet* model [3] that provides one distinct behavioral model per entity. We aggregate the event class nodes for branch and commits nodes, after analyzing the resulting graph.

Next, we raise the level of abstraction. We adapt the query for DFG discovery to aggregate `:DF` relationships between classes. To obtain the *proclet* model, we proceed by adding synchronized edges between event classes of the same activity in different entity types.

Finally, we simplify the resulting proclet model by raising again the level of abstraction. We create a higher-level class node for branch (that could be considered as an *activity* in a process) corresponding to how the event nodes and class nodes were constructed.

This concludes the steps required to perform the analysis of the software development process. The evaluation of the results can then be carried out by a domain expert (e.g., a manager or a senior software developer) who can then use the extracted KPIs and models to gather insights into the status of the project.

4 Evaluation

To demonstrate the insights gathered by our approach, we tested it on the GitHub repository of Microsoft Visual Studio Code[5] (`vscode`).

[5] https://github.com/microsoft/vscode

4.1 KPIs and Process Analysis

We start with showing the applicability of our approach to generate custom KPIs. Given the underlying graph-structure, it is possible to define both custom and well-known software engineering KPIs. One commonly used KPI for process analysis is *cycle time*. Cycle time for an issue, means the time it take from its opening to its closing (resolved). Listing 1.1 shows Cypher query to retrive the issues that took the longest times to be closed. The result of this query can also be combined with another query that retrieves the people associated to the issue. Table 1 reports the values of the issue resolution time analysis KPI in the `vscode` repository. We can observe the user who spent more time on average on resolving issues. All the Cypher queries and analyses to achieve these results can be found in our GitHub repository.

Listing 1.1: Issue nodes with `opened_at` and `closed_at` timestamps

```
MATCH (i:Issue)
WHERE i.state = 'closed'
RETURN
  i.issue_id,
  i.closed_at - i.opened_at
    as resolution_time
ORDER BY resolution_time DESC
LIMIT 10
```

Table 1: Top 10 Users with the Longest Average Cycle Times [36]

User	Hours	Minutes
aeschli	23	44
bhavyaus	23	11
Danielmelody	23	11
weinand	23	11
dtroberts	18	56
christian-bromann	18	44
JacksonKearl	18	34
jzyrobert	18	34
joelday	18	34
AmitPr	18	34

4.2 Process Model

We derived a process model after filtering the entire database to focus on the progression of a specific file over time. For the purpose of demonstration we picked the file `quickInput.ts`. Using a Cypher query, we connect, label by category and display the events that happened on the file on the different dimensions. The resulting model can be seen in Figure 4.

In this model, we used branches to represent 6 process activities, focusing mainly on the progression of `File` nodes, the associated `Commits` and their `Authors`. Additionally, `Issue` nodes are also involved in the process where they got raised to signify that a file needed to be worked on and areas that need attention. Here we could also see the cycle time of each resolved issue as well as the users that were involved in helping to solve the issue.

In Figure 4 we can also observe that in Activity T3 and T4, there are some commits that belong to different branches, a deeper investigation revealed these

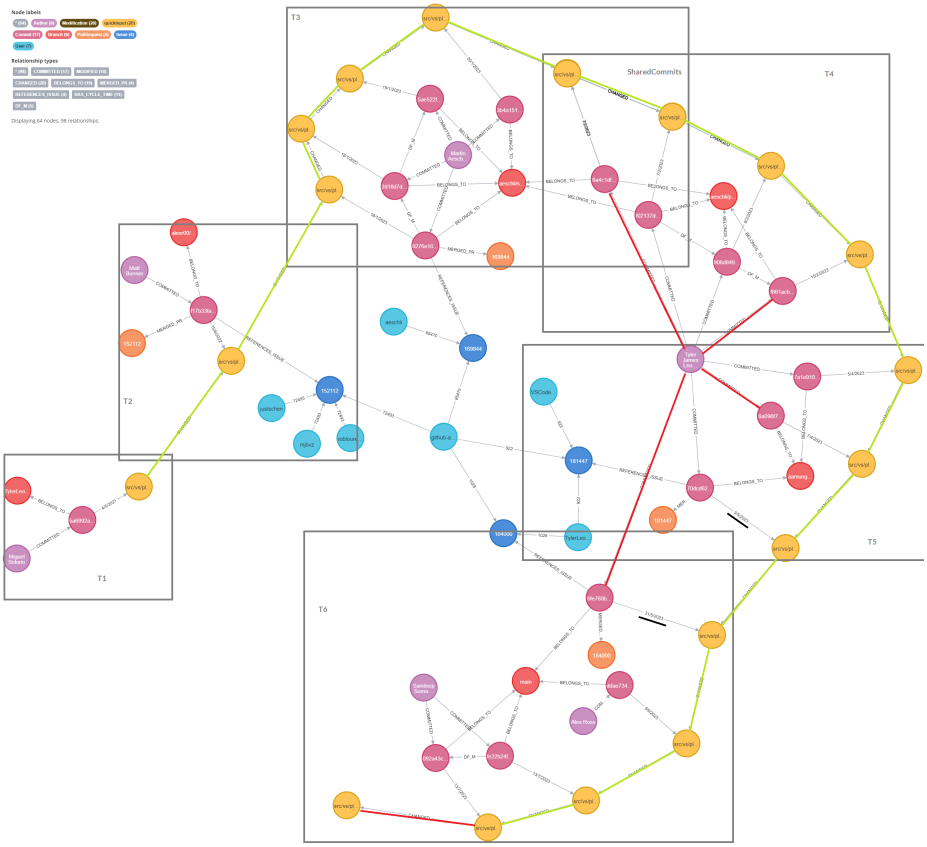


Fig. 4: The process of developing file `quickInput.ts` [36]

as revert commits from T4, occurring post-merge of the T3 working branch into the main branch, signaling an issue necessitating further work on the file. The author that was responsible for this action is highlighted in the Figure, and we can observe that he continued to work on the process in the next Activity T5 as well before pausing for a significant time period until there was an issue that require the process to be merged into the main branch. In Activity t6 we can also see that there were another few direct changes by 2 other authors before finishing the workflow for this file.

Further insights were sought on why the author from T4 and T5 paused before merging their work into the main branch. By filtering the workflow for this specific author during that timeframe, we can delve into the cause of this side-tracking and visualize their work progression during this period.

Figure 5 reveals the author worked on another branch during the hiatus. The graph depicts the states of the file in question (yellow nodes) and the other files the author worked on during that period (brown nodes). The multi-dimensional

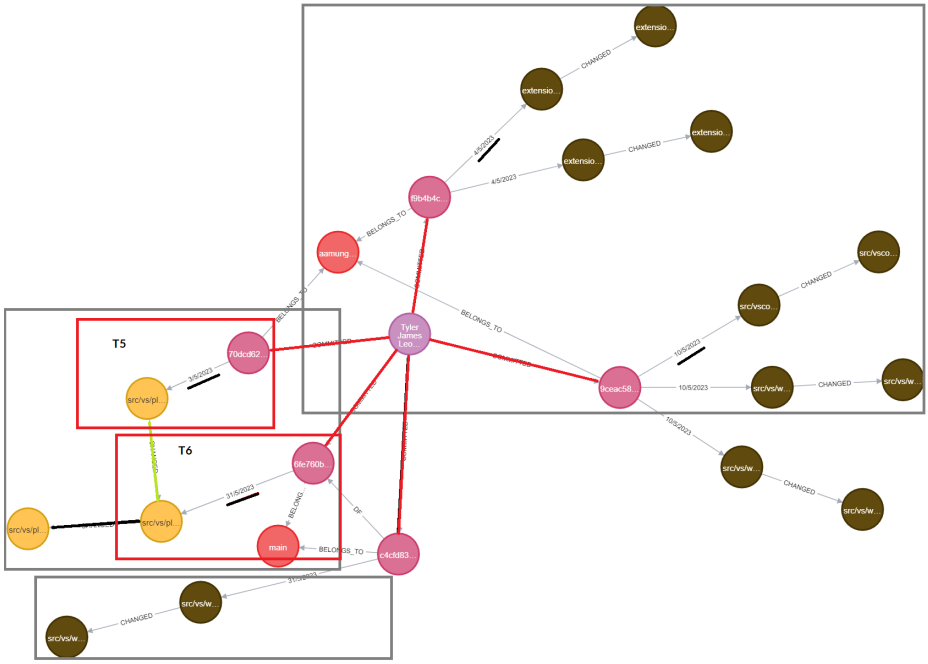


Fig. 5: Author side-tracking problem [36]

process analysis enabled us to identify the side-tracking issue and explain it by the help of another dimension with one simple query.

After utilizing the metric *code churn*, and filtering the time and the name of the author in question, we obtained a comprehensive list of what the author was working on or side-tracking during that time period in a relational form of data. This is shown is Table 2.

The relational database results can be exported in formats like `csv` or `json`. This allows for subsequent data loading into tools such as Python Library or NumPy, facilitating further analyses and visualizations by creating charts or statistics.

4.3 Proclet Model

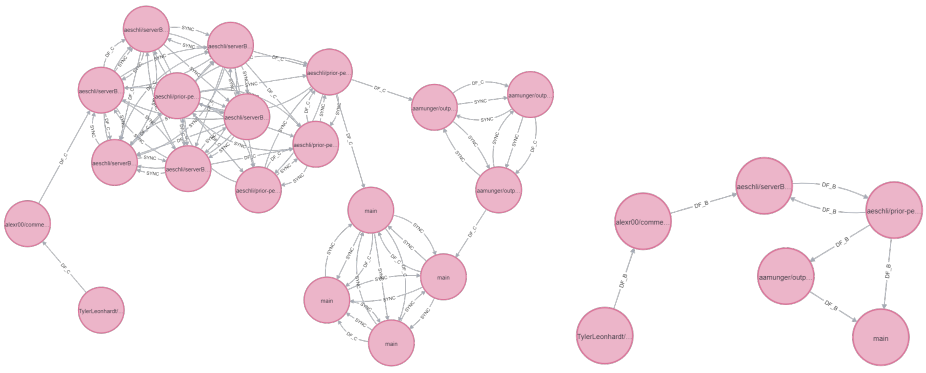
Next, we provide details on user behavior. When developers encounter an issue in their projects, their typical response is to create a separate branch to work on the file or module in question. From this perspective, a branch could be considered an *activity* of the process. To allow for better investigation of these extra activities, we resort to the notion of *proclets* [3].

Figure 6 shows the extraction of two proclet models (following the idea presented in [21]) related to the file `quickInput.ts` analyzed previously. We focused on two perspectives: i) the commits that were made to the file by actors

Table 2: Total churn of developer TJL. File path prefixes are left out [36]

File	Total Churn (lines)	Most Churning Dev
.../mainThreadAuthentication.ts	105	TJL
.../authenticationService.ts	105	TJL
.../vscode.d.ts	105	TJL
.../extHostAuthentication.ts	105	TJL
.../authentication.ts	105	TJL
.../vscode.proposed.getSessions.d.ts	105	TJL
.../extHost.protocol.ts	105	TJL
.../githubServer.ts	44	TJL
.../github.ts	44	TJL
.../quickInput.ts	10	TJL

(Figure 6a) and ii) the branches created before including changes to the main branch (Figure 6b). By observing Figure 6a we can notice that file `quickInput.ts` that after its creation, it is handed over to a next actor. Aftewards, it undergoes to a number of changes by the same actor as seen in the big connected component of the graph. This actor is also the one who then pushes the changes to the main branch. Subsequently, another actor works on subsequent changes and also pushes them to the main branch. Similarly, Figure 6b shows the change activites made to the, which do not appear in the main branch, but rather in a separate branch.



(a) Proclet model (commits) (b) Proclet model (branch)

Fig. 6: Proclet models of `quickInput.ts` based on different dimensions.

5 Discussion

The work presented in this paper is driven by the overall research question: *How can we exploit multi-dimensional process analysis to analyze software development traces in a repository?*. To answer this question we used multi-dimensional process mining.

With regard to **RQ1:** *How can we exploit Object-Centric Process Mining (OCPM) to analyze software development traces in a repository?*, we found that OCPM is applicable and useful to analyze software repositories. This is inline with previous work from [41] who applied traditional process mining. This work overcomes issues of applying traditional process mining such the project-orientation of development processes [7] and the non existence of clear case and activities [9]. With regard to **RQ2:** *How can we exploit OCPM to discover developer side-tracking?*, we found that OPCM can uncover these situations. This is due to the fact that all entities and interaction are linked together in the event-knowledge graph.

Key findings of this paper are that i) multi-dimensional process mining is suitable for analysing software repository data; and ii) it is possible to obtain a comprehensive view of the overall process that happens behind software development. The first point (i) is shown by the queries presented in the first part Section 3. Another evidence of the suitability of this approach for software development analysis is the fact that various KPIs can be reproduced by means of Cypher queries. The second point is evident by the analysis of Figure 4. In this picture, it is possible to observe not only the sequence of commit activities performed by one user to a file, but also other activities that influenced the user behavior. This, for instance, enables the explanation of why certain files took longer to be developed. In this case, we see that the reasons are that more developers were involved (purple nodes) and that some developers also worked on other files before continuing to contribute on the file `quickInput.ts`.

Compared to previous studies that focus on extracting a process workflow from their repositories, we argue that this study is a first attempt on gathering a multi-perspective view. Well-known approaches such as [29, 41] only tackle one dimension (i.e., they force the notion of a process case). Existing process mining approaches that output DFGs tend to make this assumption. Instead, our approach can show information beyond the discussed KPIs. For example, we can to observe in Figure 4 that certain users (like Tyler) contribute to more commits than others – suggesting that these maybe senior users – or that certain issues are more complex to solve because they are related to busy users or they simply require more work.

6 Conclusion

To overcome the problems of existing data-driven analysis techniques, we explore the use of Object-Centric Process Mining for software development data. More specifically, we designed and developed an open-source artifact that we applied to

real-world data from GitHub. We show OCPM is applicable to such data and can deliver valuable results for the managers. In particular, it can be used as a basis to compute commonly used KPIs and also to show process models that capture complex information. As an added value of OCPM over traditional process mining technique, we showed that it can well-capture inter-process behavior such as side-tracking. This helps to explain why determinate activites take longer.

Future work aims to delve deeper into possibilities for analyses that open up thanks to object-centricity. Therefore, we plan to assess the potentials of OCPM for software development data-analysis. Furthermore, we plan to exploitn the entities and relationships stored into the event-knowledge graph to support the development and testing of information systems theories (e.g., on coordination).

Acknowledgments. This research was supported by the Einstein Foundation Berlin under grant EPP-2019-524 and the Weizenbaum Institute under grant 16DII133

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. van der Aalst, W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In: SEFM. Lecture Notes in Computer Science, vol. 11724, pp. 3–25. Springer (2019)
3. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. Int. J. Cooperative Inf. Syst. **10**(4), 443–481 (2001)
4. Adolph, S., Kruchten, P., Hall, W.: Reconciling perspectives: A grounded theory of how people manage the process of software development. Journal of Systems and Software **85**(6), 1269–1286 (2012)
5. Agrawal, K., Aschauer, M., Thonhofer, T., Bala, S., Rogge-Solti, A., Tomsich, N.: Resource classification from version control system logs. In: EDOC Workshops. pp. 1–10. IEEE Computer Society (2016)
6. Agrawal, M., Chari, K.: Software effort, quality, and cycle time: A study of CMM level 5 projects. IEEE Trans. Software Eng. **33**(3), 145–156 (2007)
7. Bala, S., Cabanillas, C., Mendling, J., Rogge-Solti, A., Polleres, A.: Mining project-oriented business processes. In: BPM. LNCS, vol. 9253, pp. 425–440. Springer (2015)
8. Bala, S., Mendling, J.: Monitoring the software development process with process mining. In: BMSD. Lecture Notes in Business Information Processing, vol. 319, pp. 432–442. Springer (2018)
9. Bala, S., Mendling, J., Schimak, M., Queteschiner, P.: Case and activity identification for mining process models from middleware. In: PoEM. Lecture Notes in Business Information Processing, vol. 335, pp. 86–102. Springer (2018)
10. Bala, S., Revoredo, K., de A. R. Gonçalves, J.C., Baião, F.A., Mendling, J., Santoro, F.M.: Uncovering the hidden co-evolution in the work history of software projects. In: BPM. LNCS, vol. 10445, pp. 164–180. Springer (2017)
11. Balasubramanian, L., Mnkandla, E.: An evaluation to determine the extent and level of agile software development methodology adoption and implementation in the botswana software development industry. In: 2016 International Conference on

Advances in Computing and Communication Engineering (ICACCE). pp. 320–325. IEEE (2016)
12. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. Encyclopedia of software engineering pp. 528–532 (1994)
13. Beheshti, S., Benatallah, B., Motahari Nezhad, H.R.: Enabling the analysis of cross-cutting aspects in ad-hoc processes. In: CAiSE. LNCS, vol. 7908, pp. 51–67. Springer (2013)
14. Benbunan-Fich, R., Adler, R.F., Mavlanova, T.: Measuring multitasking behavior with activity-based metrics. ACM Trans. Comput. Hum. Interact. **18**(2), 7:1–7:22 (2011)
15. Biazzini, M., Baudry, B.: "may the fork be with you": novel metrics to analyze collaboration on github. In: WETSoM. pp. 37–43. ACM (2014)
16. Carlo, J.L., Gaskin, J.E., Lyytinen, K., Rose, G.M.: Early vs. late adoption of radical information technology innovations across software development organizations: an extension of the disruptive information technology innovation model. Inf. Syst. J. **24**(6), 537–569 (2014)
17. Chan, F.K.Y., Thong, J.Y.L.: Acceptance of agile methodologies: A critical review and conceptual framework. Decis. Support Syst. **46**(4), 803–814 (2009)
18. Choetkiertikul, M., Dam, H.K., Tran, T., Ghose, A., Grundy, J.: Predicting delivery capability in iterative software development. IEEE Trans. Software Eng. **44**(6), 551–573 (2018)
19. Crowston, K.: A coordination theory approach to organizational process design. Organization Science **8**(2), 157–175 (1997)
20. Esser, S., Fahland, D.: Multi-dimensional event data in graph databases. J. Data Semant. **10**(1-2), 109–141 (2021)
21. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Petri Nets. Lecture Notes in Computer Science, vol. 11522, pp. 3–24. Springer (2019)
22. Fahland, D.: Multi-dimensional process analysis. In: BPM. Lecture Notes in Computer Science, vol. 13420, pp. 27–33. Springer (2022)
23. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.P.: OCEL: A standard for object-centric event logs. In: ADBIS (Short Papers). Communications in Computer and Information Science, vol. 1450, pp. 169–175. Springer (2021)
24. Gregor, S., Hevner, A.R.: Positioning and presenting design science research for maximum impact. MIS Q. **37**(2), 337–355 (2013)
25. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS quarterly pp. 75–105 (2004)
26. Jooken, L., Creemers, M., Jans, M.: Extracting a collaboration model from VCS logs based on process mining techniques. In: Business Process Management Workshops. Lecture Notes in Business Information Processing, vol. 362, pp. 212–223. Springer (2019)
27. Keil, M.: Pulling the plug: Software project management and the problem of project escalation. MIS Q. **19**(4), 421–447 (1995)
28. Khayatbashi, S., Hartig, O., Jalali, A.: Transforming event knowledge graph to object-centric event logs: A comparative study for multi-dimensional process analysis. In: ER (2023)
29. Kindler, E., Rubin, V.A., Schäfer, W.: Activity mining for discovering software process models. In: Software Engineering. LNI, vol. P-79, pp. 175–180. GI (2006)
30. Kneuper, R.: CMMI: Improving Software and Systems Development Processes Using Capability Maturity Model Integration. Rocky Nook (2008)

31. Lindberg, A., Berente, N., Gaskin, J., Lyytinen, K.: Coordinating interdependencies in online communities: A study of an open source software project. Information Systems Research **27**(4), 751–772 (2016)
32. Marques, R., da Silva, M.M., Ferreira, D.R.: Assessing agile software development processes with process mining: A case study. In: CBI (1). pp. 109–118. IEEE Computer Society (2018)
33. Meneely, A., Corcoran, M., Williams, L.A.: Improving developer activity metrics with issue tracking annotations. In: WETSoM. pp. 75–80. ACM (2010)
34. Milewicz, R., Pinto, G., Rodeghero, P.: Characterizing the roles of contributors in open-source scientific software projects. In: MSR. pp. 421–432. IEEE / ACM (2019)
35. Munson, J.C., Elbaum, S.G.: Code churn: A measure for estimating the impact of code change. In: ICSM. p. 24. IEEE Computer Society (1998)
36. Nguyen, T., Bala, S., Mendling, J.: Multi-dimensional process analysis of software development projects. In: MODELSWARD. pp. 179–186. SCITEPRESS (2024)
37. Oliva, G.A., Santana, F.W., Gerosa, M.A., de Souza, C.R.B.: Towards a classification of logical dependencies origins: a case study. In: EVOL/IWPSE. pp. 31–40. ACM (2011)
38. Paasivaara, M., Lassenius, C.: Collaboration practices in global inter-organizational software development projects. Softw. Process. Improv. Pract. **8**(4), 183–199 (2003)
39. Pasuksmit, J., Jiang, F., Thornton, K., Friedman, A., Fuksmane, N., Kohout, I., Connor, J.: Improving agile planning for reliable software delivery. In: MSR. pp. 25–26. IEEE (2023)
40. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. Journal of management information systems **24**(3), 45–77 (2007)
41. Poncin, W., Serebrenik, A., van den Brand, M.: Process mining software repositories. In: CSMR. pp. 5–14. IEEE Computer Society (2011)
42. Rastogi, A., Gupta, A., Sureka, A.: Samiksha: mining issue tracking system for contribution and performance assessment. In: ISEC. pp. 13–22. ACM (2013)
43. Schipper, D., Aniche, M.F., van Deursen, A.: Tracing back log data to its log statement: from research to practice. In: MSR. pp. 545–549. IEEE / ACM (2019)
44. Sedano, T., Ralph, P., Péraire, C.: Software development waste. In: ICSE. pp. 130–140. IEEE / ACM (2017)
45. Shin, Y., Meneely, A., Williams, L.A., Osborne, J.A.: Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Trans. Software Eng. **37**(6), 772–787 (2011)
46. Tsoury, A., Soffer, P., Reinhartz-Berger, I.: A conceptual framework for supporting deep exploration of business process behavior. In: ER. LNCS, vol. 11157, pp. 58–71. Springer (2018)
47. Van Loon, H.: Process Assessment and ISO/IEC 15504: a reference book, vol. 775. Springer Science & Business Media (2004)
48. Vasilescu, B., Serebrenik, A., Goeminne, M., Mens, T.: On the variation and specialisation of workload - A case study of the gnome ecosystem community. Empir. Softw. Eng. **19**(4), 955–1008 (2014)
49. Vavpotic, D., Bala, S., Mendling, J., Hovelja, T.: Software process evaluation from user perceptions and log data. J. Softw. Evol. Process. **34**(4) (2022)
50. Vavpotic, D., Hovelja, T.: Improving the evaluation of software development methodology adoption and its impact on enterprise performance. Comput. Sci. Inf. Syst. **9**(1), 165–187 (2012)

51. Werner, C.M., Berry, D.M.: An empirical study of the software development process, including its requirements engineering, at very large organization: How to use data mining in such a study. In: APRES. Communications in Computer and Information Science, vol. 809, pp. 15–25. Springer (2017)
52. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer (2014)
53. Yu, X., Petter, S.: Understanding agile software development practices using shared mental models theory. Information and software technology **56**(8), 911–921 (2014)
54. Zimmermann, T., Weißgerber, P.: Preprocessing CVS data for fine-grained analysis. In: MSR. pp. 2–6 (2004)
55. Zimmermann, T., Weißgerber, P., Diehl, S., Zeller, A.: Mining version histories to guide software changes. IEEE Trans. Software Eng. **31**(6), 429–445 (2005)