Understanding What Software Engineers Are Working on

The Work-Item Prediction Challenge

Ralf Lämmel, Alvin Kerber, and Liane Praza*
Facebook Inc.

ABSTRACT

Understanding what a software engineer (a developer, an incident responder, a production engineer, etc.) is working on is a challenging problem – especially when considering the more complex software engineering workflows in software-intensive organizations: i) engineers rely on a multitude (perhaps hundreds) of loosely integrated tools; ii) engineers engage in concurrent and relatively long running workflows; ii) infrastructure (such as logging) is not fully aware of work items; iv) engineering processes (e.g., for incident response) are not explicitly modeled. In this paper, we explain the corresponding '*work-item prediction challenge*' on the grounds of representative scenarios, report on related efforts at Facebook, discuss some lessons learned, and review related work to call to arms to leverage, advance, and combine techniques from program comprehension, mining software repositories, process mining, and machine learning.

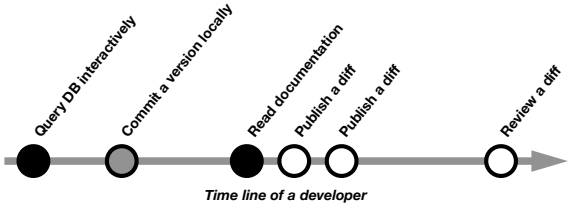
KEYWORDS

developer workflow, loose tool integration, concurrent workflow, process mining, machine learning, code similarity, word correlation

1 INTRODUCTION

A '*process-unaware (information) system*' [9] does not expose processes in a direct manner at an architectural and user level. In this paper, we are concerned with a very similar problem in the context of the ecosystems and processes that engineers use to develop, to deploy, and to maintain software systems: the challenge of predicting what work item a software engineer is working on:

The work-item prediction challenge



The events on the timeline concern different 'diffs' (i.e., system changes all the way from committing a change locally to landing the change in production) as work items. White events are trivially associated with diffs. Gray events require dedicated data integration for association. Black events are hard to associate; advanced heuristics and machine learning may be of use. That is, which of the three diffs should be associated with the DB query and the documentation access?

Figure 1: Dark Matter in Engineering Workflows.

Abbreviation: WIP challenge.[1] We speak of a challenge here because of these **defining characteristics**: i) engineers rely on a multitude (perhaps hundreds) of loosely integrated tools; ii) engineers engage in concurrent and relatively long running workflows; ii) infrastructure (such as logging) is not fully aware of work items; iv) engineering processes (e.g., for incident response) are not explicitly modeled. In combination, these characteristics give rise to what we call '*dark matter*'; see Figure 1 for an illustration.

Being able to predict the work item along the timeline of each developer has profound applications, for example, in the context of incident response in engineering (with relevance for reliability, integrity, privacy, and security) or the aggregation of key performance indicators for engineering processes.

Call to Arms. While there exists significant related work on capturing and analyzing workflows of software engineers (e.g., in terms of the use of VCSs or IDEs [10, 15, 18]), this paper calls to arms on research addressing the WIP challenge in terms of the defining characteristics to enable work-item prediction for software engineering workflows in software-intensive organizations. Future work is needed to leverage, advance, and combine techniques from program comprehension, mining software repositories, process mining, and machine learning.

Roadmap of the Paper. We explain the WIP challenge in more detail on the grounds of representative scenarios (Section 2), report on related efforts at Facebook (Section 3), discuss some lessons learned (Section 4), and review related work (Section 5).

*This paper appears in Proceedings of 28th International Conference on Program Comprehension, ICPC 2020. The subject of the paper is covered by the first author's keynote at the same conference.

[1] WIP tends to serve also as an acronym for 'work in progress', which is very fitting for our purposes because predicting what work item is being worked on essentially boils down to tracking all work in progress, at all times, as we will discuss in more detail.

2 REPRESENTATIVE SCENARIOS

At Facebook, we have encountered the need for and engaged in efforts towards addressing the WIP challenge on several occasions. We now describe two different scenarios in which work-item prediction is required. We inject some data points or illustrations to underline the defining characteristics of the WIP challenge. For non-disclosure reasons, we cannot report more specifics of any actual WIP scenarios at Facebook, but we consider the selected scenarios as representative of what is needed in software-intensive organizations.

2.1 The 'Incident Response' Scenario

Summary. Engineers need to respond to an incident (an alert) such as suboptimal performance of an important system component. To this end, engineers would like to reuse workflow steps by experienced engineers who investigated and mitigated similar incidents in the past.

Details.

- For some types of incidents, there may exist team-specific or more generic documentation with workflows for incident response, but it may be outdated or too unspecific. In fact, there are so many different kinds of incidents and the response workflows change over time. Thus, some form of 'automatic documentation' is needed.
- If we were to extract workflow steps from past incident responses, we need to identify all events associated with a given incident ID. Available logging does not suffice for such association in practice, due to a multitude of loosely integrated tools. Simple heuristics are insufficient because engineers engage in concurrent and relatively long running workflows.
- We may attempt reverse engineering and data mining to recover the incident IDs from logs. This would be a continuous and possibly prohibitively expensive effort, given the complexity and the evolution of the tool suite to be considered.
- We may instead attempt re-engineering to improve, for example, logging, thereby improving tool integration. This would risk reliability of the infrastructure / the ecosystem. This would also cause disruption, as the integration would affect engineering processes, which would also be the case, if we were to start from the premise of explicit modeling of engineering processes.

Illustration – Multitude of Tools. Figure 2 illustrates tool usage at Facebook. The chart shows the number of tools used per employee per day. The employees with more tools used per day are mostly developers and other engineers. We note that the counting scheme for tools is idiosyncratic; we often treat tool suites (sometimes of significant size) as single tools because it is easier for us to count that way; we also filter tools in some ways.

2.2 The 'Aggregate Performance' Scenario

Summary. Organizations aggregate performance data at different levels by means of Key Performance Indicators (KPIs) to inform decision making, for example, regarding the effectiveness of engineering practices; see [11] for some broader context on productivity metrics. Let us consider two KPIs related to 'diffs' (i.e., system changes all the way from committing a change locally to landing the change in production) as work items: i) 'time spent on reviewing a diff'; ii)'time spent on a diff in total'. (In practice, these KPIs would be set up in a nuanced manner, for example, by grouping by product area.)

Details.

- Consider the KPI 'time spent on reviewing a diff' first. This KPI is still implementable, if we assume that we can leverage logging for individual review comments, review decisions, and engagement with the reviewing UI (such as clicking, typing, and scrolling).
- Now consider the more general KPI 'time spent on a diff in total'. Logging may not cover important slices of work on a diff during 'idea formation' (e.g., reading documentation) or extra investigations independent of the diff's code (e.g., interactive database queries).
- Concurrency of working on diffs is entirely to be expected, especially in keeping several published diffs active for reviews and revisions until eventually shipping them independently or together. These workflow are also potentially long-running; think of diff authors and reviewers residing in different time zones.

Illustration – Concurrent and Long-Running Workflows. Figure 3 illustrates diff-related workflows of a developer over a few days. It is clear that the developer works on several diffs concurrently, as indicated by the overlapping distances between 'diff created' and 'diff shipped' / 'diff abandoned' events. In some cases, key events of several diffs occur together on the timeline, but it is reasonable to assume that some 'context switching' must have occurred prior to these 'checkpoints'. Note also the 'work-in-progress' events which appear on the timeline at a point when the work item of interest, i.e., the diff, did not even exist yet.

3 A SYSTEM FOR DIFF PREDICTION

We will now sketch a system for work-item prediction which we developed and deployed at Facebook. The system is focused on diffs as work items, but it incorporates other types of work items, as a matter of design – notably tasks (i.e., work items for project management and planning at Facebook). The system leverages heuristics and machine learning.

While the system can be used, for example, to address the 'Aggregate Performance' scenario of Section 2, it was initially developed and deployed to address a specific security-related use case, which is not discussed here for non-disclosure reasons.

3.1 The Prediction Architecture

The system essentially relies on a logging foundation that integrates data for employee activity, a notion of time windows for work-item prediction, a notion of candidate work-item IDs from which to pick, and, ultimately, a ranking process for candidate work items per employes.

3.1.1 Logging Foundation. Diff prediction relies on data from a number of logs; the most important ones are these:

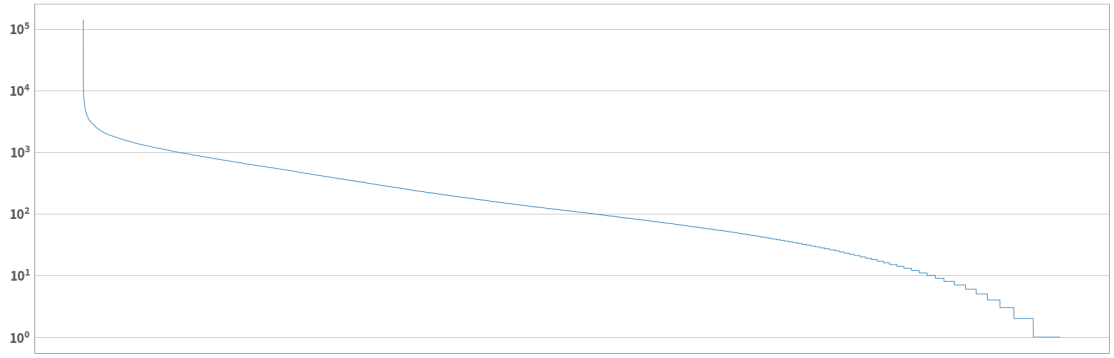


Figure 2: Number of (selected) tools used per employee on a given day for many of Facebook's employees.

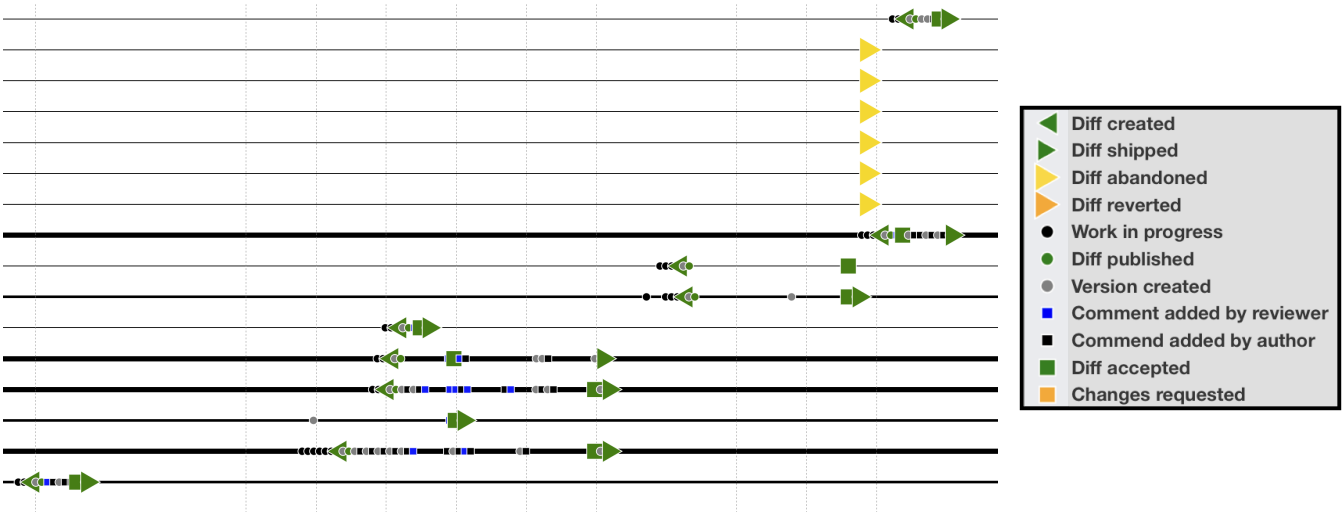


Figure 3: Concurrent workflow by a developer on several diffs (y-axis) over a few days (x-axis).

- Interactive, web-based tools – some of these tools cover development directly (e.g., the Phabricator UI, see below), but there are hundreds of tools, which are logged generically.
- Version control (Mercurial, Git, etc.) – Checkouts are performed; commits are created, amended, and rebased. These actions are logged from the command line and the IDEs.
- Code reviewing and continuous integration (Phabricator UI and CLI et al.) – Commits are published for review, tested in a sandbox, commented on, revised, accepted or rejected, landed in production or abandoned, etc.
- Task management – Tasks are created, assigned, commented on, associated with diffs, closed, etc.
- Development tools – Build project; run test; debug code; query a database; etc.

3.1.2 Time Windows Into Dark Matter. We aim at probabilities for a certain employee to work on a certain work item (diff) at a certain time. We use time windows of 10 minutes as the granularity on the time axis for aggregating signal and learning correlations. For each employee (engineer), we use basic logging data to determine the windows during which the employee was active and for which prediction is thus relevant; see Listing 1.

Listing 1: Data Model for *Active Time Windows*

```
CREATE TABLE active_time_windows (
    employee BIGINT COMMENT 'Employee ID',
    first_time BIGINT COMMENT 'Window first time',
    last_time BIGINT COMMENT 'Window last time'
)
PARTITIONED BY (ds STRING) -- YYYY-MM-DD - for the day of the data
```

Throughout this section, we use such relational table schemas to hint at the data model used by the system for diff prediction, which is implemented in Facebook's data warehouse while relying on Hive, Spark, Presto, and scheduled pipeline and ML runs. All tables are partitioned by day (see 'ds'), i.e., work-item prediction is is generally approached on a per-day basis.

The time windows associate with corresponding event sequences as of the logging foundation. (Think of join conditions based on time boundaries.) We also refer to the time-windowed event sequences as 'dark-matter sequences'; see the introduction for our use of the term 'dark matter'.

3.1.3 Candidate Work Items. Prediction cannot make up work items (or IDs thereof) by itself. Instead, prediction is given access to a set of candidate work items that can be at all expected to be worked on during a given day by a given person.

To this end, we use high-confidence signal such as version control- and reviewing-based interaction of employees with diffs such as the events illustrated in Figure 3. We propagate backward b days and forward f days. (Let's assume here $f = b = 2$.) Thus, we end up with a set of candidate diffs as modelled in Listing 2.

Listing 2: Data Model for *Candidate Diffs per Employee*

```
CREATE TABLE candidate_diffs (
    employee BIGINT COMMENT 'Employee ID',
    diff_number BIGINT COMMENT 'Candidate diff'
)
PARTITIONED BY (ds STRING)
```

3.1.4 Prediction by Ranking. On a given day, for each employee, for each active time window, and for each candidate work item, we need to determine the probability of the employee working during the time window on the work item. Many of these probabilities should be expected to be zero. All these probabilities, when combined, define directly a ranking of work items per time window of the employee. The actual prediction is determined by a combination of heuristics, applied to the dark-matter sequences, subject to heuristic averaging. Some of the heuristics use machine learning.

When outputting a prediction, we also record the contribution of each heuristic, thereby contributing to an explainable model for work-item prediction [20]. The output format for predictions is modelled in Listing 3.

Listing 3: Data Model for *Diff Predictions*

```
CREATE TABLE diff_predictions (
    employee BIGINT COMMENT 'Employee ID',
    first_time BIGINT COMMENT 'Sequence first time',
    last_time BIGINT COMMENT 'Sequence last time',
    diff_number BIGINT COMMENT 'Candidate diff',
    prediction DOUBLE COMMENT 'Probability of employee working on diff',
    contributions MAP<STRING, DOUBLE> COMMENT 'Contributions of heuristics'
)
PARTITIONED BY (ds STRING)
```

The individual heuristics compute either a Boolean value or a probabilistic value in the range $[0, 1]$. Thus, there is a table of per-heuristic output, subject to additional partitioning as modelled in Listing 4.

The extra partitioning simplifies the process of computing the individual heuristics in the data warehouse in a distributed manner. (We could also use one table for each heuristic instead.)

3.2 Selected Heuristics

Let us sketch some of the heuristics used by the system for diff prediction.

3.2.1 Heuristic 'Diff Analysis' (DA). We extract 'strong confidence' events for authors and reviewers interacting with diffs. All those events are labeled with '1.0'. There are 'obvious' events when diff authors submit diffs or new versions thereof for reviewing or reviewers submit reviews or parts thereof. We also incorporate so-called

Listing 4: Data Model for *Diff Heuristics*

```
CREATE TABLE diff_heuristics (
    employee BIGINT COMMENT 'Sequence employee ID',
    first_time BIGINT COMMENT 'Sequence first time',
    last_time BIGINT COMMENT 'Sequence last time',
    diff_number BIGINT COMMENT 'Candidate diff',
    label DOUBLE COMMENT 'Positive (1.0) / negative (0.0) label'
)
PARTITIONED BY (
    ds STRING,
    heuristic STRING -- Name of heuristic
)
```

work-in-progress events which are about an employee's interaction with the repository and local commits — also before an actual Phabricator diff is created. Consider the following sample workflow of an employee:

(1) Update to master on the developer's machine.
(2) Start editing and commit locally.
(3) Continue editing and amend locally.
(4) Split the amended commit.
(5) Submit a stack of two diffs for review.

At point (1), a checkout identifier (also referred to as 'work-in-progress' identifier) is created. Each of the operations (2)–(4) creates more checkout identifiers and also commit hashes as the result of the mutations. Once operation (5) creates two diffs, actual diff IDs become available. We can now travel into the past and connect timestamps (1)–(4), in this case, with both diffs. This heuristic leverages an integrated event log for version control and code review, as modelled in Listing 5.

Listing 5: Data Model for *Integrated Event Log for Diffs*

```
CREATE TABLE diff_event_log (
    id BIGINT COMMENT 'Diff event ID',
    time_started BIGINT COMMENT 'Time the event (action) started',
    time_ended BIGINT COMMENT 'Time the event (action) ended',
    actor BIGINT COMMENT 'ID of employee acting on the diff',
    event_type STRING COMMENT 'Type of diff event',
    diff_number BIGINT COMMENT 'Diff number',
    version_number BIGINT COMMENT 'Version number of diff',
    owner BIGINT COMMENT 'ID of employee owning the diff',
    data STRING COMMENT 'Extra metadata in JSON'
)
PARTITIONED BY (ds STRING)
```

3.2.2 Heuristic 'Task Events' (TE). Tasks support project management and planning at Facebook. There are tasks for features to be developed, incidents to be investigated, bugs to be fixed, etc. Diffs are typically also associated with tasks eventually. Once this association is revealed, we count task interaction events retroactively towards associated diffs.

3.2.3 Heuristic 'Diff URIs' (DU). The web-based, internal tools used at Facebook may track work items, to some extent, through URI parameters. For instance, diff IDs are generally represented in this format "`D(\\d+)`" (using regular expression syntax here) and this representation is also used within URIs. There is a few specific tools, for which we are readily aware of their usage of diff IDs in the URIs and specific positions or parameters thereof. The present

heuristic extracts diff-ID occurrences more generically. Without a heuristic like this, we would need to continuously invest into extraction functionality for a multitude of evolving and new tools. The heuristic is prone to false positives, but aims at minimizing those by incorporating knowledge of the valid length of contemporary diff IDs. (For instance, the string 'D42' would be considered too short to count as a diff ID.)

3.2.4 Heuristic 'Diff Comparison' (DC). Based on backup system-like support for tracking local repository changes, we track 'file changes', ahead of actual commits so that they can be correlated with diffs, eventually.

We tokenize (featurize) diffs and changes:

- filename extensions;
- filename words (directory names, basenames, parts thereof);
- symbols 'used' (e.g., program identifiers referenced).

For instance, filename extensions help to already group diffs largely by language (technology); filename words help with aligning changes based on the affected regions in the file tree. We leverage similarity metrics (cosine et al.) and clustering for comparison, in fact, similarity analysis.

3.2.5 Heuristic 'Word Indexing' (IX). The 'Diff URIs' (DU) heuristic suffices when diff IDs are explicitly mentioned in URIs. We devise the 'Word Indexing' (IX) heuristic to better address the defining characteristics of the WIP challenge to involve a multitude of loosely integrated tools along concurrent, long-running workflows.

Consider the following (obfuscated) URI, which we keep trivial for ease of explanation, but it should be noted that URIs for internal tool usage at Facebook can be rather complex, as significant session/context data is captured by the URIs:

```
https://internal.acme.com/tools/tool-42/resources/123/?task=T4711
```

The heuristic relies on tokenization of URIs. That is, the following words would be extracted for the example: tool, 42, resources, 123, task, T4711. The heuristic performs word indexing and, in fact, word-correlation learning such that, for example, we may infer that 'T4711' associates generally (probabilistically) with a certain diff ID, if 'T4711' co-occurs with the diff ID in enough time windows elsewhere. This is explained in Section 3.3 in more detail.

3.2.6 Heuristic Averaging. The heuristics are combined to compute a final prediction by weighted averaging. We use larger weights for higher confidence signal (such as the DA heuristic) and lower weights for lower confidence signal, to account for noise (false positives), subject to the overall assumption that low weights are still sufficient to create a rank for work items and a combination of low weights may also increase the rank.

Figure 4 shows two basic metrics for the selected heuristics, as they affect the final predictions – this is a sample for a recent day and some group of employees. The *nonzero* column counts how many time windows are labeled by the heuristic. The *significant* column counts how many times a significance threshold is passed, subject to internal validation for reducing false positives. We mention in passing that we use yet other metrics, for example, a quantification of whether a work item is included exclusively due to a specific heuristic or a distribution of the distance between

	nonzero	significant
da	130090	130090
dc	80001	76419
du	394751	383693
ix	3015564	734323
te	82013	73376

Figure 4: A Sample of Two Basic Heuristics Metrics.

high-confidence signal (DA and TE) and low-confidence signal (DC and IX) per pairs of work item and employee.

3.3 Word-Correlation Learning

Let us discuss the IX heuristic for word-correlation learning, as introduced in Section 3.2.5, in more detail, as this component addresses the defining characteristics of the WIP challenge in a relatively advanced manner.

3.3.1 Tokenization. The tokens (words) extracted from URIs due to tool usage are the starting point for word indexing; see Listing 6.

Listing 6: Data Model for *Words in URIs due to Tool Usage*

```
CREATE TABLE uri_words (
    employee BIGINT COMMENT 'Employee ID for tool usage',
    time BIGINT COMMENT 'Time the tool was used',
    uri_words ARRAY(STRING) COMMENT 'Words in a URI for tool use'
)
PARTITIONED BY (ds STRING)
```

3.3.2 Word Frequency. Many of the URI words (tokens) extracted from the URIs for tool usage are 'noise' in the sense that they occur all too often and we need to filter the set of words to ever be considered for prediction. To this end, we aggregate, over time, the frequency of words; see Listing 7.

Listing 7: Data Model for *Word Frequency*

```
CREATE TABLE word_frequency (
    uri_word STRING COMMENT 'Word in a URI for tool use',
    word_days BIGINT COMMENT 'Number of employee days the word appears in',
    all_days BIGINT COMMENT 'Total number of available employee days',
    inverse_frequency DOUBLE COMMENT 'IDF computed from above numbers'
)
PARTITIONED BY (ds STRING)
```

In particular, we compute an inverse document (-like) frequency with all documents corresponding to the days seen by our analysis versus those days where the word occurs. We omit here details how exactly we apply filters, but we use this frequency table in an obvious sense to focus on words of interest and to maintain scalability of the prediction process. Here we note that several 100K new words show up every day in our dataset.

3.3.3 Co-Occurrence with Diff IDs. We determine co-occurrences of diff IDs for candidate diffs with other words within URIs. We build a corresponding index of such word overlaps per day from a number of past days and we track the employee for whose dark-matter sequence the co-occurrence occurred; see Listing 8.

Listing 8: Data Model for *Overlaps of diff IDs and Other Words*

```
CREATE TABLE diff_id_overlaps (
    employee BIGINT COMMENT 'Employee ID',
    first_time BIGINT COMMENT 'Sequence first time',
    last_time BIGINT COMMENT 'Sequence last time',
    diff_number BIGINT COMMENT 'Candidate diff',
    word STRING COMMENT 'Word that co−occurs',
    index_employee BIGINT COMMENT 'Employ with overlap in index',
    index_time BIGINT COMMENT 'Overlap time in index',
    index_ds STRING COMMENT 'Overlap day in index'
)
PARTITIONED BY (ds STRING)
```

3.3.4 Word-Indexing Features. At this point, we can extract features from word overlaps; see Listing 9 for some examples of count-based features.

Listing 9: Data Model for *Word Index Features*

```
CREATE TABLE word_index_features (
    employee BIGINT COMMENT 'Employee ID',
    first_time BIGINT COMMENT 'Sequence first time',
    last_time BIGINT COMMENT 'Sequence last time',
    diff_number BIGINT COMMENT 'Candidate diff',
    ft_ct_total_overlaps INT COMMENT
        'Total number of overlaps of diff ID with other words.'
    ft_ct_overlaps_more_distinct INT COMMENT
        'Number of distinct triples (word, employee, ds) from among
        all the overlaps. In particular, this does not count repeated visits
        to the same word within the sequence.',
    ft_ct_overlapping_words INT COMMENT
        'Number of distinct words from among all the overlaps. This counts
        how many related words there are, without looking at how closely
        related each individual word is.',
    ft_ct_overlapping_employee_days INT COMMENT
        'Number of distinct employee/ds pairs from among all the overlaps.
        This counts how many times the sequence and candidate can be related,
        without looking at how many words they are related by each time.',
    ft_min_overlap_employee_day_freq INT COMMENT
        'Smallest employee−day frequency of an overlapping sequence word.
        That is, from all the overlaps, find the sequence word that is rarest
        in general (by employee−day frequency, i.e., by counting the number of
        employee/ds pairs in which it appears in the past).',
    ...
)
PARTITIONED BY (ds STRING)
```

For each feature, one may have an intuition as to why the feature could be possibly helpful for work-item prediction. For instance, *ft_min_overlap_employee_day_freq* is included because overlapping words that are rarer in the 'entire' past of an employee may be more meaningful.

3.3.5 Prediction with Decision Trees. We leverage a decision tree (DT) algorithm, in fact, GBDT with permutation-based feature importance. We rely on a weak supervision approach as follows. For

	Feature	Standard Deviation	Mean	Coverage	Importance
4	ft_ct_overlapping_employee_days	11.4963	8.6574	100.00%	33.086%
2	ft_ct_overlaps_more_distinct	50.9225	24.1994	100.00%	15.398%
1	ft_ct_total_overlaps	172.4915	74.2752	100.00%	12.265%
8		4.0626	11.5704	100.00%	7.728%
10		4.5395	13.0049	100.00%	7.278%
7		29.0013	32.3214	100.00%	6.712%
9		4.3625	12.5191	100.00%	6.571%
5	ft_min_overlap_employee_day_freq	11.3123	6.6926	100.00%	4.931%
6		16.2191	14.0425	100.00%	3.121%
3	ft_ct_overlapping_words	16.8993	6.0462	100.00%	2.911%

Figure 5: Feature Importance of the GBDT Model.

positive labeling, we use high-confidence events. For *negative labeling*, essentially, we select dark-matter sequences without high-confidence events, but instead with high confidence of no diff-related work based on an observable, prolonged hiatus regarding diffs in terms of absence of high-confidence events, despite though being active overall.

In Figure 5, we illustrate feature importance. For instance, the importance of *ft_ct_overlapping_employee_days* is evident, while our hopes for *ft_min_overlap_employee_day_freq* are shattered. (Disclaimer: feature importance is shown here at some point during development, feature engineering, and feature selection.)

4 LESSONS LEARNED

Based on our work on work-item prediction at Facebook (diffs and other types of work items), we share some lessons learned. For non-disclosure reasons, we cannot report empirical results on work-item prediction. Instead, we focus here on more general findings.

4.1 The Dominance of Event Log Integration

Events of interest are scattered over diverse available logs. Thus, filtering, abstraction, cleaning, alignment, and quality checking efforts are crucial. In our experience, this work combined with recovering the semantics of the given data dominates efforts on work-item prediction. Some of these integration efforts can be technically involved and require significant computational and storage investments. For instance, diff prediction required an extra component to perform chasing from diffs to local commits based on version control data — this turned out to be a non-trivial graph-based computation.

4.2 The Effort of Managing Prediction DAGs

We implement prediction architectures as directed acyclic graphs (DAG) with tables as nodes and scheduled pipeline and ML runs in the data warehouse as edges. For instance, the DAG for diff prediction includes many dozens of tables and scheduled runs. (Arguably, splitting up data into tables and computations into scheduled runs is subject to some design choices. Also, some of the tables are reused or reusable.) We suffice with offline, day-by-day prediction and thus, all computation is scheduled for daily execution.

The challenge is that a prediction architecture has significant internal and upstream dependencies; we are affected by delays, failures, or changes in the data semantics. It requires continuous effort and rigorous engineering (e.g., monitoring, data-quality checking, responding to alerts) to keep the overall DAG up and running.

4.3 Limits of Human-Based Validation

A naive approach to human-based validation would require subjects to accept or reject triples of the form time window, employee, and work item, thereby providing labeled data for machine learning and enabling measurements of model performance (e.g., precision and recall) in the most direct manner. The costs of such an approach are arguably prohibitive in an industrial setting. Also, it is relatively hard to sufficiently help the subjects to make an informed decision. There are more indirect means of human-based validation, which also can be used for weak supervision. i) Gamification (For instance, 'Hello X, were you working on diff D123, when you were running the database query Q456 at 2:42pm yesterday? Please, answer with Yes or No.') ii) Efficiency (Can we measure that, for example, the incident response time is lower, on average, when responders use predicted work items as opposed to using more basic means of identifying work items based on, for example, simple queries for high-confidence events?) iii) Relevance (Can we measure how often an incident responder relies on predicted work items?)

4.4 The Need for Internal Validation

Especially because human-based validation is so limited, we need to use internal validation. To this end, some quantitative means are used: i) Predictions associate a significant number of time windows with specific work items. ii) A significant number of the time windows in i) do not contain high-confidence events, but only low-confidence events for the some work item. iii) With increasing life time of a work item (say, the span between first and last high-confidence event), the probability increases that there are interruptions in time windows being associated with the work item. These forms help in establishing basic properties of prediction. That is, prediction finds work items, the findings are not just based on proximity, and association of time windows with work items is on and off over time.

4.5 The Value of Similarity Measures

When we started our efforts on work-item prediction, we were enthusiastic about clustering work items (diffs) to correlate events (such as file changes or database queries, debugging sessions, or code searches) with eventual diffs. Perhaps obviously, clustering turned out to be the wrong tool for the problem because any discoverable clustering semantics within developer 'utterances' turns out to be too coarse-grained to be useful in distinguishing, for example, diffs of the same developer, especially when clustering is performed for all developers at once. Also, the size of many utterances is just too small to be useful for clustering; consider, for example, short debugging sessions or trivial database queries. It turned out that relatively simple similarity measures (e.g., cosine) are more effective for correlating 'utterances' with diffs.

4.6 Systematic Dark Matter Elimination

In our work, we included additional heuristics based on an ad-hoc process. In the case of diff prediction, we started from obvious, high-confidence events (such as 'publish a diff'); we advanced towards inclusion of correlated events (such as 'commit a version locally'), subject to some data integration; eventually, we leveraged some data mining and machine learning components. While the inclusion of any heuristic was driven by domain knowledge, the process is prone to premature optimization and gaps or delays regarding (internal) validation. We are now looking for a more data science-driven approach, where each decision to include an additional or to advance an existing heuristic can be supported by an appropriate analysis.

5 RELATED WORK

Let us review related to work so that we connect the WIP challenge and the sketched efforts at Facebook to prior art. Related work is from the areas of program comprehension, mining software repositories, process mining, and machine learning. We group related work entries by some common themes.

5.1 Developer Workflow Mining

Summary. Work in this group tends to involve explicit models of developer workflow; making such an assumption would make the WIP challenge only harder. None of the scenarios that we have in mind require an explicit model. Further, some of the work in this group touches upon the issue of using multiple tools, but the more general characteristics of a multitude of loosely integrated tools along concurrent, long-running workflows are not addressed. However, work in this group submits important techniques for event modeling and data mining, overall.

Visualizing IDE sessions [14]. The approach is inspiring in terms of setting up some basic concepts such as IDE sessions (to provide scope), events and classifying them more abstractly (e.g., inspection, editing, and navigation), and navigation paths to express that different entities are manipulated. The frequency of events and navigation paths are visualized by size/width. Colors are used to encode classification properties.

Process mining for mining software repositories [18]. In one of the case studies, the paper uses the ProM tool for process mining to extract the bug life cycle in a software project. Much of the work is concerned with preprocessing data sources for use with the ProM tool for process mining.

Identification of usage smells in IDEs [5]. This work helps assessing the usability of an IDE. Event sequences are analyzed through stages of pattern mining (mostly counting common sequences with some degree of variation), pattern filtering (e.g., to remove too short sequences), and pattern clustering (to compress the large number of patterns to be amenable to visual inspection).

Workflow mining from IDE usage [10]. The Disco tool is used to extract the actual workflows in terms of graphs with events and frequency-annotated transitions. Events are classified in some interesting ways (e.g., discrete versus continuous events). Workflows are determined in an experiment-based manner such that developer subjects solve certain tasks with the given IDE.

Prediction or recommendation of developer behavior in the IDE [4]. Temporal Latent Dirichlet Allocation (Temporal LDA) for topic modeling is applied to IDE interaction data. In this manner, high-level task behavior such as structured navigation is discovered based on interpreting sequences of lower level interaction events and commands.

Developer behavior across GitHub and StackOverflow [23]. The accounts are associated and cross-site developer behaviors are analyzed through T-graph analysis, topic-based clustering (based on LDA) and cross-site tagging.

Discovering software processes from OSSD Web repositories [12]. A general methodology is presented, combining techniques for text analysis, link analysis, and patterns of repository usage and update. This work presents a process entity taxonomy and relies on the construction of a social graph to better capture who is working with whom how. Probabilistic relational modeling (PRM) is used to model what the developers are doing and how they are doing it.

Correlation of code changes with reviewer input [2]. This correlation is measured while assuming a certain process for code review with involvement of a task management system. The work does not use any advanced information retrieval techniques, but it relies instead on carefully collected datasets and subject-based ground truth. A significant percentage of code changes is classified as not responding to reviewer comments.

5.2 Case ID Recovery in Process Mining

Summary. Work in this group applies to *process-unaware systems* [9] in so far that case IDs (say, work-item IDs – for our purposes) are not assumed. However, other characteristics of the WIP challenge (the multitude of evolving tools, the significance of concurrent workflows, the limitations of logging) are not taken into account.

Key alignment by composite key conditions [16]. The work aims at integrating separate logs using different kinds of keys. Alignment of the keys relies on composite key conditions of some specific format, which assumes key attributes, equality, conjunction, and disjunction, so that entries from different logs are assigned to the same workflow instance, when the conditions hold. The conditions do not need to be designed manually, but they can be explored semi-automatically by a heuristic, thereby providing a form of case ID learning.

Case ID assignment based time proximity [8] et al. The probability of certain events to follow certain other events can be inferred from an (unlabeled) log. Once such a probability matrix has been estimated, assignment of case IDs is essentially an optimization problem. This approach is of limited use in a setting, like the one of the WIP challenge, with concurrent and long-running workflows and an event log that necessarily contains much 'noise' in terms of irrelevant events.

Case IDs in a mobile app context [6]. The service log, which logs requests of the app, is translated to an event log while filling in missing case IDs on the grounds of several heuristics: some form of time proximity, spatial continuity, grouping of requests by source, and an overall time bound (due to app specifics). The work caters for multiple versions of the underlying process, thereby addressing system evolution to some extent.

5.3 Machine Learning in Process Mining

Summary. Work in this group discusses machine learning in the process mining context. The work does not directly address the defining characteristics of the WIP challenge, but the work inspires potential advances of any WIP solution to better address event abstractions, ML concept drifts, and representations for embeddings, or to enable additional applications such as anomaly detection.

Event abstraction [21]. Supervised learning is leveraged for event abstraction based on annotations with high-level interpretations of low-level events while relying on extensions of the XES event-stream format and using conditional random fields for machine learning. (Without event abstraction, raw events may be too fine-grained for process discovery to return useful results.)

Discovering and understanding potential concept drifts [3]. (This term is used in machine learning to refer to situations when the relation between the input data and the target variable, which the model is trying to predict, changes over time in unforeseen ways, thereby letting accuracy of the predictions degrade over time.) This work provides a generic framework and specific techniques for discovering and understanding potential concept drifts in process mining. In particular, the effects of 'noise' (random replacements or insertions of incorrect symbols or missing symbols) and imbalance (largely different priorities of certain branches) are addressed.

Representation learning [13]. The work is based on the observation that, we quote, "real-life event logs present a large number of cases, potentially representing a highly varied set of distinct event sequences, and usually also containing information on resources and a diverse set of other event- or case-related attributes". As a consequence, featurized event data suffers from a dimensionality problem, which the paper addresses by representation learning architectures for activities, traces, logs, and process models.

Anomaly detection [17]. Neural networks are applied to address a general objective in process mining: verification of the absence of certain undesirable properties or the compliance with certain desirable properties [1, 7, 22].

6 CONCLUDING REMARKS

In this paper, we have established the work-item prediction challenge, as it applies to software engineering workflows in software-intensive organizations.

The defining characteristics of the challenge are these: i) engineers rely on a multitude (perhaps hundreds) of loosely integrated tools; ii) engineers engage in concurrent and relatively long running workflows; ii) infrastructure (such as logging) is not fully aware of work items; iv) engineering processes (e.g., for incident response) are not explicitly modeled.

In practice, an ensemble of heuristic- and ML-based components is needed to predict work items on the timelines of employees. We have described related efforts at Facebook, where diffs (system changes), as a type of work item, are predicted. Such prediction is readily useful in the context of incident response in engineering (with relevance for reliability, integrity, privacy, and security) or the aggregation of key performance indicators for engineering processes.

We have provided an extended related work discussion which connects the work-item prediction challenge to research in the areas of program comprehension, mining software repositories, process mining, and machine learning. This discussion documents the need for and the potential of leveraging, advancing, and combining existing techniques to tackle the work-item prediction challenge more efficiently in practice.

REFERENCES

[1] Rafael Accorsi, Thomas Stocker, and Günter Müller. 2013. On the exploitation of process mining for security audits: the process discovery case. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13*. ACM, 1462–1468.

[2] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Jürgens. 2014. Modern code reviews in open-source projects: which problems do they fix?. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings*. ACM, 202–211.

[3] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indre Zliobaite, and Mykola Pechenizkiy. 2014. Dealing With Concept Drifts in Process Mining. *IEEE Trans. Neural Netw. Learning Syst.* 25, 1 (2014), 154–171.

[4] Kostadin Damevski, Hui Chen, David C. Shepherd, Nicholas A. Kraft, and Lori L. Pollock. 2018. Predicting Future Developer Behavior in the IDE Using Topic Models. *IEEE Trans. Software Eng.* 44, 11 (2018), 1100–1111.

[5] Kostadin Damevski, David C. Shepherd, Johannes Schneider, and Lori L. Pollock. 2017. Mining Sequences of Developer Interactions in Visual Studio for Usage Smells. *IEEE Trans. Software Eng.* 43, 4 (2017), 359–371.

[6] J. Du, H. Cai, L. Jiang, and C. Huang. 2017. Methods of Introducing Continuous Process Mining to Service Management for Mobile APPs. In *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*. IEEE, 134–141.

[7] Stefano Ferilli and Floriana Esposito. 2013. A Logic Framework for Incremental Learning of Process Models. *Fundam. Inform.* 128, 4 (2013), 413–443.

[8] Diogo R. Ferreira and Daniel Gillblad. 2009. Discovering Process Models from Unlabelled Event Logs. In *Business Process Management, 7th International Conference, BPM 2009. Proceedings (LNCS)*, Vol. 5701. Springer, 143–158.

[9] Sukriti Goel, Jyoti M. Bhat, and Barbara Weber. 2013. End-to-End Process Extraction in Process Unaware Systems. In *Business Process Management Workshops - BPM 2012 International Workshops. Revised Papers (Lecture Notes in Business Information Processing)*, Vol. 132. Springer, 162–173.

[10] Constantina Ioannou, Andrea Burattin, and Barbara Weber. 2018. Mining Developers' Workflows from IDE Usage. In *Advanced Information Systems Engineering Workshops - CAiSE 2018 International Workshops, Proceedings (Lecture Notes in Business Information Processing)*, Vol. 316. Springer, 167–179.

[11] Ciera Jaspan and Caitlin Sadowski. 2019. No Single Metric Captures Productivity. In *Rethinking Productivity in Software Engineering*. Apress / open Springer, 13–20.

[12] Chris Jensen and Walt Scacchi. 2004. Data Mining for Software Process Discovery in Open Source Software Development Communities. In *Proceedings of the 1st International Workshop on Mining Software Repositories, MSR@ICSE 2004*. 96–100.

[13] Pieter De Koninck, Seppe vanden Broucke, and Jochen De Weerdt. 2018. act2vec, trace2vec, log2vec, and model2vec: Representation Learning for Business Processes. In *Business Process Management - 16th International Conference, BPM 2018. Proceedings (LNCS)*, Vol. 11080. Springer, 305–321.

[14] Roberto Minelli and Michele Lanza. 2013. Visualizing the workflow of developers. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 1–4.

[15] Audris Mockus. 2003. Analogy Based Prediction of Work Item Flow in Software Projects: a Case Study. In *2003 International Symposium on Empirical Software Engineering (ISESE 2003)*. IEEE, 110–119.

[16] Hamid R. Motahari Nezhad, Boualem Benatallah, Régis Saint-Paul, Fabio Casati, and Periklis Andritsos. 2008. Process spaceship: discovering and exploring process views from event logs in data spaces. *PVLDB* 1, 2 (2008), 1412–1415.

[17] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. 2018. BINet: Multivariate Business Process Anomaly Detection Using Deep Learning. In *Business Process Management - 16th International Conference, BPM 2018. Proceedings (LNCS)*, Vol. 11080. Springer, 271–287.

[18] Wouter Poncin, Alexander Serebrenik, and Mark van den Brand. 2011. Process Mining Software Repositories. In *15th European Conference on Software Maintenance and Reengineering, CSMR 2011*. IEEE, 5–14.

[19] Liane Praza. 2019. The untapped potential of analyzing complete developer workflows. In *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019*. IEEE / ACM, 178.

[20] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1 (05 2019), 206–215. https://doi.org/10.1038/s42256-019-0048-x

[21] Niek Tax, Natalia Sidorova, Reinder Haakma, and Wil M. P. van der Aalst. 2016. Event Abstraction for Process Mining using Supervised Learning Techniques. *CoRR* abs/1606.07283 (2016), 10. arXiv:1606.07283

[22] Wil M. P. van der Aalst, H. T. de Beer, and Boudewijn F. van Dongen. 2005. Process Mining and Verification of Properties: An Approach Based on Temporal Logic. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Proceedings, Part I (LNCS)*, Vol. 3760. Springer, 130–147.

[23] Yunxiang Xiong, Zhangyuan Meng, Beijun Shen, and Wei Yin. 2017. Developer Identity Linkage and Behavior Mining Across GitHub and StackOverflow. *International Journal of Software Engineering and Knowledge Engineering* 27, 9-10 (2017), 1409–1426.