Process Inspection Support: an Industrial Case Study

Christoph Mayr-Dorn
Institute for Software Systems Engineering
Johannes Kepler University, Linz
Linz, Austria
christoph.mayr-dorn@jku.at

Johann Tuder
formerly ACME
Linz, Austria
johann.tuder@gmail.com

Alexander Egyed
Institute for Software Systems Engineering
Johannes Kepler University, Linz
Linz, Austria
alexander.egyed@jku.at

***Abstract*—Organizational factors such as team structure, coordination among engineers, or processes have a significant impact on software quality and development progress. Projects often take much longer to complete than planned and miscommunications among engineers are common. Yet, the process for exploring the project-specific or organization-specific root causes why this happens is still poorly supported. Investigations are cumbersome and require significant effort. In the context of this industrial case study, our industry partner was interested in measuring and assessing how the organization structure and issue handling processes ultimately affected software quality and time. Reducing the effort of such investigations/retrospectives and speeding up fact finding is important as it allows for more frequent, informed engineering process improvements and feedback to managers, team leads, and engineers. This paper describes our approach of pairing process metrics with visual historical inspection of issues. Stakeholders such as managers, team leads, or quality assurance engineers inspect metrics (and deviations from expected values) for individual issues and utilize a historical visualization of the affected (and related) issues to obtain insights into the reason for the metric (deviation) and its root cause. We demonstrate the usefulness of our approach based on our *ProcessInspector* prototype providing access to data on four real industry projects and a qualitative evaluation with team leads and group leads from our industry partner.**

***Index Terms*—issue, organizational structure, software engineering process, metrics, prototype, JIRA, history visualization**

I. Introduction

Software engineering projects often last much longer than planned and miscommunication between engineers occurs on a regular basis. The process for exploring the exact project- or organization-specific root causes why this happens is still poorly supported. Investigations are cumbersome and require significant manual effort. Retrospective analyses, such as done at the end of sprints in agile development environments, are important to happen jointly as a group effort but benefit significantly from tools that assist in better understanding the cause of undesirable situations such as missed milestones. Reducing the effort of such investigations/retrospectives and speeding up fact finding is important as it allows for more frequent, informed engineering process improvements and feedback to managers, team leads, and engineers.

Research over the past decades has shown that organizational factors such as team structure, coordination among engineers, or processes have a significant impact on software quality and development progress [1]. Engineers working on strongly coupled artifacts tend to require frequent communication to coordinate their engineering efforts [2]. Hence, achieving socio-technical congruence (STC) is one aspect towards improving software development performance [3].

Therefore, in the context of this industrial case study, our industry partner was interested in measuring and assessing how its organizational structures and issue handling processes ultimately affect coordination among engineers and timely delivery in order to obtain insights in how and where to focus on improvements. Typically, software process metrics provide such insights and multiple research efforts aim to improve them [4]–[7]. Yet, determining which metrics are useful and accurately describe the ongoing development efforts is non-trivial as this differs among companies and often also among departments and groups within the same company.

To this end, we propose an approach of pairing process metrics with visual historical inspection of issues to overcome the limitations of metric inspection without context on the one hand and visualization without guidance on the other hand. Stakeholders such as managers, team leads, or quality assurance engineers inspect metrics (and deviations from expected values) for individual issues and utilize a historical visualization of the affected (and related) issues to obtain insights into the reason for the metric (deviation) and its root cause. We designed the accompanying prototypical tool (*Process Inspector*) light-weight and flexible to support easy integration and adaptation of metrics. We demonstrate the usefulness of our approach based on a prototype providing access to data on four real industry projects and a qualitative evaluation with team leads and group leads from our industry partner.

The contributions of this paper are:

- A flexible approach for combining process metrics and issue history visualization
- A prototypical tool implementation (*Process Inspector*)
- A data set describing the complete set of issues from four industry projects
- A qualitative evaluation of the approach

The remainder of this paper is structured as follows: Section II provides case study background information. We introduce

our approach in Section III and the corresponding prototype in Section IV. We qualitatively evaluate and discuss our approach in Section V. Section VI compares our approach to related work, before Section VII concludes this paper with an outlook on future work.

II. Case Study Background

A. Industry Context

ACME is in the business of hosting a recreational activities web platform. The company's identity and project names have to remain confidential due to the sensitive nature of the analyzed data. At the time of data extraction and paper evaluation, the company was structured into ten departments. Those departments consisted of 22 groups. The following groups are of primary interest for this paper: software developers-frontend, software developers-backend, database, graphics, quality assurance, and product & project management.[1] These groups are heavily involved in the software development process. The departments are physically distributed across two buildings. A project team usually consists of at least one member from each of the above listed groups of interest. When a project's software is released for public access the project team remains responsible for maintenance. This implies that the team works together during the product's complete lifecycle from initial project setup, to implementation, and ongoing support even though team members are situated in different offices. This cross-departmental/group organization style comes with a significant number of necessary meetings and informal communication channels within the team. From experience, this often resulted in miscommunication, unclear assignments, and in undocumented decisions in any of the used tools: the company uses JIRA, Confluence, Skype, rocketChat and Outlook to keep track of projects. JIRA is used as ticketing system and also provides further information such as comments and lists of changed source code files. The intended process to follow for software development is reflected in JIRA issues, issue relations, and issue properties such as state, milestones, releases, and assignee.

B. An Industry Challenge

As most software development companies, ACME is interested in improving its workflows and software development process. After each completed major project the current process is evaluated and adapted to fit new organizational circumstances such as new teams. To this end it needs to compare differences among teams at project and at issue level (e.g., issues bottlenecks at a specific group) to establish how differences in process and structure affect performance. ACME also needs to distinguish if problems result from e.g. workflow flaws, inefficient organizational structure, or insufficiently accurate artifacts such as unclear requirements. Given ACME's organizational structure, a particularly interesting question was whether vertically organized teams (team members from different groups become co-located) perform better than horizontally organized teams (team members from different groups remain with their groups). This evaluation process is cumbersome currently as the reasons for deviations and comparisons across teams and projects are not easily obtainable from ACME's tool landscape.

[1]We identify a member of such groups as an *engineer* and use the term *developer* when explicitly referring to a member from the frontend or backend group.

C. Process Improvement Method

One typical approach to process improvement is through a Goal-Question-Metric (GQM) driven method [8]. In the context of our industry partner, the purpose of such a study is to evaluate the impact of the current team structure and issue ticket usage on efficiency and coordination effort from the point of view of team-leads (manage projects) and group-leads (manage an expertise-centric set of engineers such as testers) in the context of lightly distributed teams.

According to the taxonomy by Smite et al. [9], ACME's teams can be classified as Location: Onshore, Legal Entity: Insourcing, Geographic Distance: Close, Temporal Distance: Similar. These teams can nevertheless be considered distributed as already a separation by floors or buildings can significantly reduce informal contacts and thus influence coordination [10].

ACME identified four key questions that they need to answer on their path to process improvement:[2]

1) Is engineering effort accurately estimated?
2) How much coordination is happening?
3) How efficient are coordination actions?
4) How efficient is project planning?

Here questions and metrics (see Section III-A) are iteratively refined upon feedback from our industry partner.

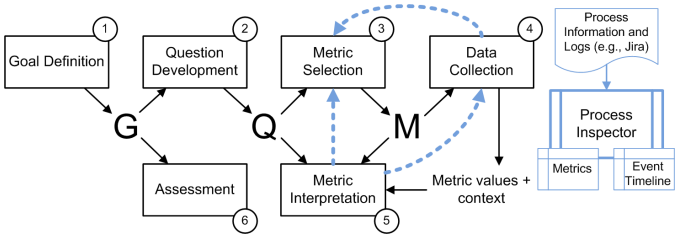


Fig. 1. Approach to assisting process inspection in the scope of a GQM method: full/black arrows show input/output between GQM steps, dashed/blue arrows depict supported feedback between steps applying the *Process Inspector* prototype.

III. Approach

Our approach (see Figure 1) supports the stakeholders during metric interpretation by pairing metric calculation (and presentation) with issue event timelines as metric context. As a side effect, our approach assists the stakeholders to determine which metrics are useful and applicable in measuring process improvement (in our case study with a focus on the subgoal of measuring team coordination).

[2]Note, that these are not the research questions to be answered in this paper.

The GQM method (as applied by ACME) typically consists of following six steps:

1) Define improvement goals.
2) Develop questions that allow the goals to be quantified.
3) Determine which metrics answer these questions.
4) Implement data collection mechanisms.
5) Collect and interpret the metrics for immediate feedback/improvement.
6) Assess how the gathered metrics support reaching the goal and derive recommendations.

Deriving accurate and meaningful metrics is difficult as their measurement context changes over time (e.g., teams are restructured, processes change) or they become applied in situations they are not sufficiently suitable for (low priority, short term, technology exploration project). As a consequence, metrics need careful interpretation in their measurement context.

Our approach addresses this concern by supporting a stakeholder to easily move between metric calculation, respectively inspection, (here on the level of issues) and the context that gave rise to that particular metric instance (here the history of the issue and its related issues).

In our particular case, this allows a stakeholder to become aware (at a coarse-grained level) of projects that seem to be poorly performing, and (at a fine-grained level) of issues that need attention. The stakeholder then inspects the visual history of these issues that give raise to a critical metric value to understand if there is truly a problem at hand, whether the particular situation constitutes an exception to the rule, or whether the metric is not suitable (anymore) in the larger context of this project. For example, metrics that measure how long an issue is in state *Open* and *In Progress* before it is *Closed* become unreliable when engineers forget to transition an issue upon starting their work and only briefly set the issue to *In Progress* shortly before completing it. This type of deviation from expected behavior results from the flexibility engineers require. Too rigid, explicit process control greatly limits engineers' freedom, respectively forces engineers to work outside the process to handle unforeseen situations and optimizations not foreseen by the process. Diebold and Scherr [11] show that in industrial practice the majority of processes, therefore, focus on description rather than using formal notations or models. Visual inspection provide, on the one hand, rapid feedback on the brittleness of metrics and, on the other hand, point out potential for tool improvement to assist engineers in following the intended process.

Likewise, the timeline visualization allows stakeholders to browse issue progress and, upon finding suspicious looking event sequences, the stakeholder can easily cross-check with metrics how that particular issue measures against other (similar) issues.

In short, our approach, respectively prototype, directly addresses step 4 and 5 of the GQM method, and indirectly supports the re-evaluation of metric selection in step 3.

Ultimately, the research question addressed in this paper (and answered in the evaluation section) is: Is the combination of the provided process metrics and issue history visualization effective for obtaining insights into coordination problems? Note that its not a goal of this paper to investigate whether the proposed metrics, our approach, or the GQM method indeed lead to process improvement.

A. Metrics

We selected the following set of metrics and refined them together with engineers at ACME to ensure they are relevant in ACME's development context. The metrics come at various granularity levels: per-issue, per-occurrence (multiple times per issue possible), informational/aggregating (information at issue level, project-level metric value).

1) *Issue resolved/closed date <= issue due date:* The resolved date is set e.g., if an engineer finishes a bug-fix implementation. The closed date is set when e.g., a tester finished testing of a bug-fix. In the ideal case every issue should be closed before its due-date. For this metric the relevant fields of an issue are the *closed-, resolved-* and *due-date*. A negative difference value describes an issue that was closed/resolved before its due-date, a positive value that it missed it's due-date deadline, respectively. Large deviations in either direction are indicators of estimation errors. Positive deviations indicate that engineers required less time, negative deviations indicate unforeseen complexity in implementation or coordination. (per-issue metric: duration in days)
2) *Due dates aren't changed:* Due dates are set at the beginning of a project. Project managers and engineers then arrange the workload and determine when the various components of a project should be completed. Due date changes indicate that project planning was not accurate (e.g., lower/higher effort/coordination estimation) or an assigned engineer (no longer) needed to work on a higher prioritized task. Specifically, this metric counts the number of *due-date* changes per issue (not considering the initial setting of the due-date, which is also a change event) and derives an project overall ratio. (per-occurrence metric: duration in days for from/to due date change; project-level metric: ratio of "issues with changes" compared to "issues without changes")
3) *Fix version wasn't changed:* A fix version identifies the release or milestone this issue's result should be available in. Fix version changes reflect issues being moved between milestones during planning or implementation (or even when already completed). Many such changes may indicate challenges in project planning and also the release workflow. This metric is calculated similar to M2 but based on changes to the *fix version* property. (per-occurrence metric: changes: count of from/to fix version changes; project-level metric: ratio of "issues with changes" compared to "issues without changes")
4) *Duration in approve state:* Some issues have to be approved by team leads before work can start. Usually such an approval process does not take very long and this metrics informs the responsible decision maker when

delays happen repeatedly. Approval delays block the actual engineers from working on the issue. This metric is calculated only for issues that have had a status changed to "Approval necessary". The temporal distance to the next status entry is then the metric value. (per-issue metric: duration in days)

5) *Re-open distance to due date:* Issues that are re-opened multiple times might indicate an engineer-to-issue assignment mismatch or unclear/incomplete requirements. Whenever an issues state is changed to "Open", "Open Again" or "Reopened" the change date's difference to the due date is calculated. A positive value describes re-opening after the due date, a negative value a re-opened before the due date. (per-issue metric: how often changed; per-occurrence metric: how long before due date in days)
6) *Assignee Changes:* Assignee changes are part of the workflow. For example, a project manager creates an issue and assigns it to an engineer. After the implementation the issue is assigned to a tester. A high number of changes potentially indicates communication problems between departments or that an issue was not implemented correctly. Especially assignments within a department, e.g., from front-end developer to another front-end developer, should not happen. Therefore the metric is split into the number of intra-department changes and the number of cross-department changes. (per-issue metric: number of changes)
7) *Use of comments:* ACME expects complex issues to require additional refinement and clarification via comments, especially when an issue involves engineers from multiple departments. This metric subset also provides an indication whether discussions, decisions, and assumptions are documented within Jira or using other tools. (per-issue metric: comment count, commenter count, count per commenter, min/max/avg comment length).
8) *Issue re-assignment without documented communication:* Changing the assignee typically requires no documentation when the engineers follow the intended workflow, e.g., a developer finishes the implementation and a tester has to start. However if an issue re-assignment happens multiple times and deviates from the usual workflow then this should be documented in the comments. The metric uses the changelog entries of type "assignee". The metric counts for each issue, how many pairs of such changes exist without a comment in between. For each of these change pairs, the metric also collects the duration between assignment changes in days. (per-issue metric: number of assignments, total duration in days between two assignments without comment)
9) *Issue re-assignments without status changes:* Usually an issue is re-assigned to another engineer if the issue's status changes, e.g. the developer finishes implementation and sets the status to "Resolved", assigns it to a tester who changes the status to "In Testing". If issues are re-assigned without a status change something could be wrong either in the workflow or how engineers handle issues. This metric uses the same *assignee* changelog entries as the previous metric M8 but considers *status* changelog entries instead of comments. (per-issue metric: number of reassignment, total duration between two assignments without status change)
10) *Duration between re-assignment and subsequent status change:* Issue re-assignment indicate that another engineer can start their work, indicating this start by updating the issue's state (see previous metric). This metric measures for each re-assignment the duration until the subsequent status change. A long timespan could indicate that the assigned engineer is overloaded and that the issue should have been better worked on by someone else, respectively that project planning overlooked/created a bottleneck. (per-issue metric: duration in days)
11) *Issue resolved date compared to code freeze date:* The metric shows the difference between the resolve date of an issue and the code freeze date. Usually issues should be resolved before code freeze because during code freeze no new features are implemented for the current milestone and the time to release is reserved for bug fixing and testing. Issues resolved after code freeze indicate work overload and wrong project planning or normal bug fixing. A positive value indicates that an issue was resolved before the code freeze, a negative value the opposite. (per-issue metric: difference in days to code freeze date)
12) *Issue of future milestone started in an earlier milestone:* The metric calculates the time span between the first progress of an issue to the start date of its milestone's predecessor. This metric highlights which and how many issues of a project were started earlier than planned. This indicates that some engineers do not have enough issues assigned or that project planning was not accurate enough: towards the end of the milestone, engineers were already doing work for another milestone and they could have moved more features into the current one. A positive time span shows that an issue was started before the official milestone start, a negative number shows the opposite. (per-issue metric: difference in days to milestone)

All these metrics rely on a small set of issue change event (i.e., timeline events). Displaying them in their temporal order as they have occurred (sometimes in isolation, somethings almost simultaneously) allows to better place the metric values in the engineering process context.

B. Issue Timeline Events

The selected issue timeline events are driven by the metrics that make use of these events. As such, these events are exemplary and we make no claim for completeness (additional/different metrics may introduce other events). These

events, however, are central to coordinating engineering efforts. In general, we distinguish between two event categories: events that are placed at the time they occurred and events that represent predefined dates such as fix-versions and due dates. The currently considered event list comprises:

1) created: marks the date the issue was added to the issue tracker (here Jira).
2) status: marks changes to the status, e.g., from "Open" to "In Progress".
3) assignee: marks handing over the ticket from one responsible team member to another.
4) fix version set: marks changes to the release version in which this issue should be included.
5) resolved: marks when the issue was set to resolved.
6) due date: marks when the issue is planned to be completed.
7) due date changes: marks updates to the due date.
8) comment: marks when a engineer commented on the issue.
9) last updated: marks the last change to any of the issue properties.

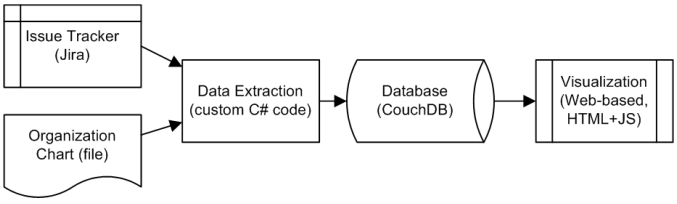


Fig. 2. Prototype Architecture.

IV. Prototype Tool Support

In this section we present the current prototypical tool support for calculating metrics and inspecting issue histories. We deliberately aimed for a maximally light-weight prototype to allow for rapid prototyping and iterative refinement.

The current prototype (see Figure 2) consists of three main components: the *Jira Extractor*, a *Database* for caching processed issues, and a *Web Frontend* for calculating and displaying metrics and issue history.

The *Jira Extractor* is responsible for accessing a Jira server's REST API for retrieving all issues of a given project as json documents (in our industry partner's case identified by a root issue with all related issues obtained via an issue's "Part" relations). It then reads an engineer's department from a configuration file, anonymizes the engineer id, prepends the department id to the anonymized engineer id, and replaces this throughout the issue json document before storing each issue json document in the CouchDB.

CouchDB is a schemaless JSON database merely used to store the processed issues and making them available to the Web Frontend.

The *Web Frontend* is a locally hosted HTML page making heavy use of javascript libraries for calculating and displaying metric result tables, timeline visualization, and connecting (directly) to the CouchDB. The set of metrics can be easily adapted as each metric is implemented as a separate class and located in its respective, separate javascript file (e.g., metric 1 in M1.js). Each metric class simply needs to provide a *calculate*, *getDataTable*, and *getSummaryDataTable* method for triggering metrics calculation across all issues of the project, returning the overall metric result table with values for each issue (for per-issue metrics) or occurrence (for per-occurrence metrics), and a summary table displaying, for example, averages, maximum, and minimum values (customizable per metric), respectively. A standard workstation is sufficiently fast to carry out issue retrieval (from the CouchDB), metric computation, and visualization on demand without the need to include server side data (pre)-processing facilities. The Web Frontend consists of the following UI elements (see also Figure 3):

- *Project Selector* (A, top left corner) is a dropdown input that allows the user to pick a project. Each project is stored in a separate "database" in CouchDB.
- *Timeline Scope* (B, left) enables the user to select start and end dates for the timeline. The timelines for all visible issues shrinks or expands to the desired values. A press on the "Reset to default Dates" button adjust all timelines to the default values.
- *Metric Selector* (C, left) allows a user to switch between indiviual metrics (one at a time). Upon selecting an option input, the metric algorithm runs on the fly and outputs a result table above the timelines (shown in Figure 5).
- *Issue Selector* (D, left) supports the filtering for issues in a project. The checkbox beside the issue key includes, respectively removes, a timeline for this issue.
- *Timeline Area* (E, middle) contains for each selected issue a separate timeline. A single timeline shows all events that occurred during the issues lifetime. By hovering over a single event additional information is displayed (see Figure 4).
- *Event Selector* (F, right) lists all the symbols, i.e., event types, that may occur on a timeline. Un/selecting a symbol hides/includes all corresponding event instances from all timelines.

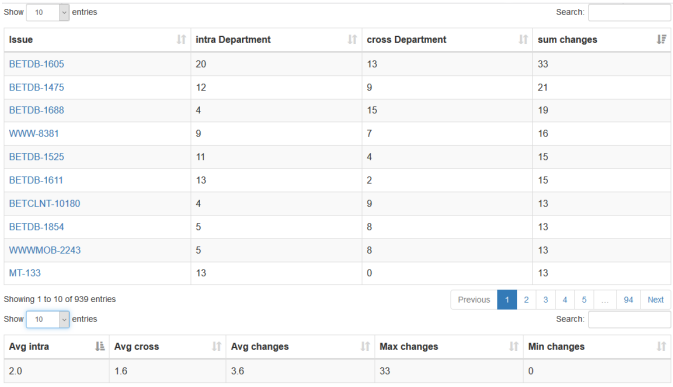
V. Qualitative Evaluation

A. Study Design

We conducted semi-structured interviews with four stakeholders (i.e., actual end-users of the prototype): team-leads and group-leads. Group-leads are responsible for the engineers in their group and assign work packages to them. Whereas team-leads are responsible for the group-leads and assign them projects and want frequent updates about project progress and workload. Each (separate) interview consisted of an introduction of the prototype including explanation of the metrics, used data, and user interface features. Subsequently, each participant was asked to assess three issues with respect to identifying coordination problems (see V-D). Thereafter, participants were asked to choose the three most insightful

Fig. 3. Prototype screenshot.

Fig. 4. Example icon hover-over information for *status*, *assignee* changes as well as *comments*.



Issue	intra Department	cross Department	sum changes
BETDB-1605	20	13	33
BETDB-1475	12	9	21
BETDB-1688	4	15	19
WWW-8381	9	7	16
BETDB-1525	11	4	15
BETDB-1611	13	2	15
BETCLNT-10180	4	9	13
BETDB-1854	5	8	13
WWWMOB-2243	5	8	13
MT-133	13	0	13

Avg intra	Avg cross	Avg changes	Max changes	Min changes
2.0	1.6	3.6	33	0

Fig. 5. Prototype screenshot for metric M6.

metrics (see V-E), score the user interfaces on a Likert scale from 1 to 5 starts (5 stars being best) along Jacob Nielsen's heuristics [12] (see V-F), before providing free-form feedback on positive and negative impression as well as ideas for additional prototype features.

B. Data Set

The participants had access to data from four projects via the prototype. These four real ACME projects are: P1, a low-priority Android app development project; P2 and P3, two business-critical Android app development projects; and project P4 integrating two types of recreational activities that involved experts beyond front-end, business logic, design, and database engineering (e.g., marketing and legal departments). The four projects P1–P4 contain a total of 1017, 2676, 1052, and 939 issues, respectively. During data extraction, the department prefixes in Table I were added to the anonymized engineer ids.

TABLE I
DEPARTMENT PREFIXES

Department	Prefixed short name
Development Web	devW
Development Client	devC
Database	db
Graphics	gfx
Product & Project Management	ppm
Quality Assurance	qa
Marketing	mkt
IT Administration	adm
Controlling	ctrl
CIO	cio

C. Participant Demographics

The voluntary participant list consisted of two team-leads and two group-leads. The job title for both team-leads is "Teamlead WebDev". Group-lead job titles were "Grouplead Architecture & Performance" and "Grouplead App" thus spanning a range of the industry partner's engineering departments. The group-leads are in their position since 0.5 and 1 year, the team-leads 1 and 10 years with software engineering

experience for 12 and 5 years, and for 6 and 17 years, respectively.

D. Task: Issue Assessment

Each participant received the same three issues selected from across three projects. They were asked to assess these issues only using timelines and metrics (from the whole respective project) with no additional information from Jira available (see Figure 6 for the timelines of the three issues: BETDB-1475, BAH-71, WWW-7370). The issues BETDB-1475 and BAH-71 were chosen, because they show problems in organization and workflow. The issue WWW-7370 represents a normal issue without anomalies. Since this task was part of an interview, we report the answers for the following two questions together:

- Can you spot any problem that occurred during the issues' lifetimes?
- Can you identify high coordination efforts?

BETDB-1475 (Figure 6 top)
BETDB is an issue primarily involving the database team. Every participant noted that status changes appear in quick succession from which they concluded (also from experience): the workflow for handling database issues is badly designed. When an SQL query needs to be fixed, nearly the whole workflow has to be executed again, except for an initial approval step. One problematic aspect, as mentioned by one of the participants, is that it is not clearly defined what should be done in the approval process. Furthermore approval does not seem to work if issues are re-opened that often. Participants remarked that the ill-design workflow is mitigated currently by the database team reacting very quickly on changes and immediately tackling them so that long delays are prevented.

Another hypothesis postulated by the participants is that requirements were not written well enough. This hypothesis is based on the fact that preparation (at the beginning of the lifetime) lasted one month and this is not the usual case. This is a problem because other teams have to wait. Also the issue was often re-assigned within the database team and it was also once re-assigned to a developer.

BAH-71 (Figure 6 bottom)
For this issue all participants identified problems in how milestone and status fields are used and the overall lifetime of project issues. All participants stated that the lifetime is very long and it has to be asked if it was planned for that long at the beginning. Furthermore they mentioned that the issue was moved a lot to other milestones without any work in between and this indicates that issue planning is not efficient. Also status for issues are not handled correctly. Some are superfluous, e.g. "QA Test" which indicates that the whole project is in testing. However sub-issues are already tested before and the status does not make sense at project level. This problem is supported by the event on March 31st, where status changed from "In Progress" to "Internal Review" to "External Review" and then to "Softwaredesign". A participant mentioned that it is not possible to review specifications of a whole project within a single day. Furthermore a project of this size should not stay one month in the design-stage, this is usually done faster. At the other end it seems that the whole project was tested in only three days which also cannot have occurred in reality. It was mentioned that the end of the timeline (beginning in May) represents the usual workflow.

WWW-7370 (Figure 6 middle)
All participants declared that this is the usual workflow of a developer issue. This is the ideal case and it was probably a simple bug fix.

Participants identified the three causes for these specific issues mainly through timeline investigations. When the participants wanted to analyze the whole project they used the metrics. When a result of a metric was displayed they used timelines to analyzed the outliers (e.g. issues with minimum or maximum metric values).

TABLE II
MOST VALUABLE METRICS FOR PARTICIPANTS

Metric	GL1	GL2	TL1	TL2
M1	x	x	x	x
M3	x	x	x	
M5	x			
M6		x	x	
M9				x
M10				x

E. Top Rated Metrics

The next question asked the participants to select the three most valuable metrics for their work (see Table II GL = group-lead, TL = team-lead). We further report summarized participant statements regarding the applicability of a metric together with the historical timeline visualization.

As shown in Table II, every participant listed Metric 1 (*Issue resolved/closed date <= issue due date*) as one of the most valuable metrics. For the participants the metric is suitable for comparing projects and the timeline visualization can assist in answering the following questions:

- How long is an issue at a specific department?
- How was the project planned?

Metric 3 (*Fix version wasn't changed*) was mentioned by 3 participants. It may indicate work overload and the visualization of the affected issue (and other contemporary issues) may provide insights to:

- Why were issues moved?
- How many issues are moved to the backlog or come from the backlog?
- Was an issue planned for an unrealistic time-frame?

Metric 5 (*Re-open distance to due date*) assists in finding out why issues were re-opened and the visualization supports detecting if there is a pattern for the re-open changes.

Metric 6 (*Assignee Changes*) supports investigations if requirements were not formulated clearly or if perhaps task responsibilities were vague.

Metric 9 (*Issue re-assignments without status changes*) highlights problems with long-running issues and visualization provides clues whether the workflow should be revised.

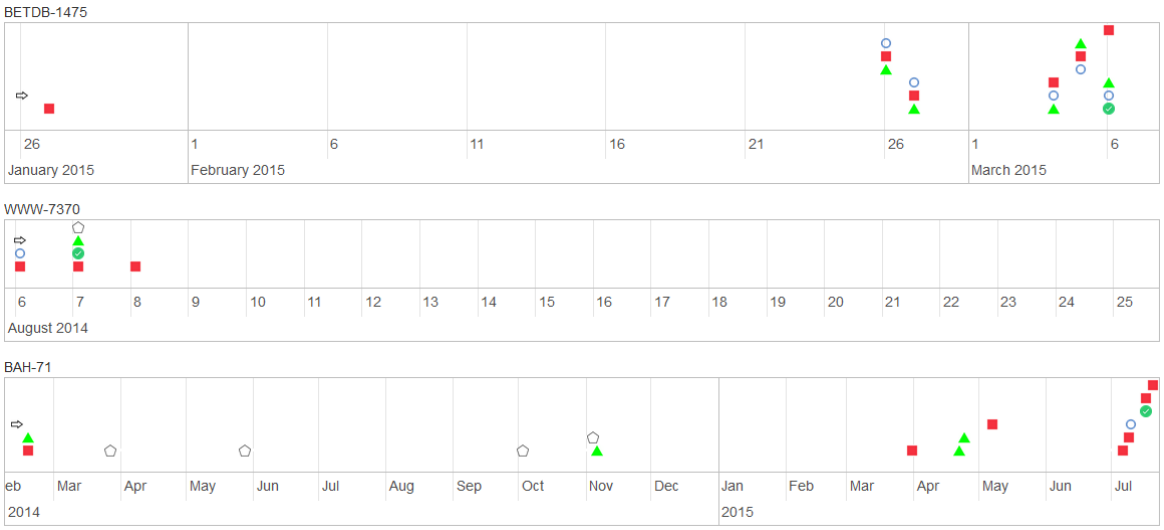


Fig. 6. Issues under evaluation by study participants.

TABLE III
AVERAGE TOOL SCORES.

Measure	Average
Responsiveness (performance with a large amounts of data)	4.25
Forgiveness (allows exploration)	4.00
Intuitivity (easy to navigate)	4.00
Icons & Symbols understandable	4.00
Information overload (sufficient/too much information)	4.50
Low learning curve (easy to understand)	4.00
Match to the real world (terminology)	3.25
Flexibility/Efficiency of use (expert vs novice)	4.00

Metric 10 (*Duration between re-assignment and subsequent status change*) helps a participant to identify poorly planned projects.

F. Usability Scores

The next part of the questionnaire asked the participants to rate the tool on a Likert scale of 1 (not good) to 5 (very good). Table III reports the chosen eight aspects (based on Nielsen's heuristics [12]) and the average participant rating. Given the prototypical nature of our tool, the participants rated it very well across the eight aspects, with exception to "Match to real world (terminology)". Here participants remarked that some metric names could be improved to more accurately convey their semantics.

G. Discussion

The answers to the two questions within scope of the assessment task (see Subsection V-D) revealed that metrics allow for quickly analysing a project and finding anomalies. When investigating an issue in more detail, the participants always used the timeline visualization. The feedback on the tools usability (see Subsection V-F) lets us assume that the *Project Inspector* prototype was sufficiently mature to assist the participants in their task. Most explicit negative criticism by participants concerned minor usability aspects such as tool tips, descriptions, font size.

In the free-form feedback part, participants explicitly highlighted the usefulness of combining metrics and timelines, quickly finding process flaws in short time, comparing projects, and the prototype's simple design. Participants even suggested as future work to intensify this aspect by highlighting the events on the timeline that are relevant to the chosen metric.

Key Observation 1: switching back and forth between metrics and timeline is essential to quickly, easily obtaining insights into situations that would benefit from process improvement.

Participants also noted that for properly interpreting metrics and timelines, knowledge about how Jira is used by the teams is necessary: for understanding based on what data/events a metric is derived, but also to understand when engineers deviate from expected process behavior. During metric development our industry co-author but also the case study participants noticed that state transitions occurred often within unreasonable long or short time, implying that engineers were not accurately following the prescribed process. As potential future work, participants suggested to include additional coordination-centric meta information such as displaying the department of the engineer that made a status change.

Key Observation 2: an iterative cycle of metric selection, data collection, and metric interpretation is vital for ensuring that the metrics really measure something meaningful and for identifying additional useful metrics. Being able to flexibly integrate new or updated metrics is thus a key requirement, as enabled by our prototype.

The scoring of metrics (see Subsection V-E) indicated that not all metrics were considered immediately useful during the assessment task. The most useful metrics either utilized changes to re-assignment of issue responsibility, state changes, or date changes. Participants mentioned in the free-form part

of the questionaire the lower-than-initially-expected usefulness of comment-centric metrics. They noted that the semantics of comments would need to be considered as well (most likely also other informal communication channels such as skype) and still potentially miss relevant face-to-face communication during the occasional physical meetings.

Key Observation 3: avoid, if possible, message content-based metrics (e.g., based on analysing comments, emails, etc.) that would require extensive coverage of multiple channels.

H. Threats to Validity

Internal Validity. We address researcher bias by analyzing data from an actual company rather than conducting controlled experiments. The approach works on arbitrary issue events and was not specifically tailored to Jira issues or the used dataset.

External Validity. Rather than claiming for wide generalizability of our results, we argue in line with Briand et al. [13] that context-driven research will yield more realistic results. In this paper, we thus evaluated the usefulness of the process metrics and issue timeline visualization with multiple engineers working at our industry partner. We analyzed data from this single company only as such data from real-world, industrial environments is extremely hard to get. Companies are very reluctant to provide insights into their working processes at that level of detail. As different engineers and roles (e.g., database expert, designer, team lead) have different concerns, they might evaluate the metrics' usefulness differently. The positive feedback from multiple engineers familiar with the development processes under study, however, at least shows that the approach is applicable and indeed useful in the observed context.

We make no claim that our approach will yield equally useful results when applied to data from a Jira server used for open-source development. Often these servers, such as hosted by the Apache Software Foundation[3] provide only the default issue types and issue states and thus require engineers to limit their coordination to processes based on this limited set of states, or apply other non-structured mechanisms such as comments, mailing-lists, and tacit knowledge to manage processes. Even less structured are issues in projects hosted on GitHub. GitHub issues[4] are either "Open" or "Closed", any intermediary state needs managing via arbitrary labels (i.e., tagging) with no support for defining or restricting valid transitions. This restrict the ability to determine fine-granular metrics. The visualization prototype, however, is flexible to incorporate custom build metrics as long as they are provided with the JSON data items in the CouchDB.

Engineers assigned to departments and roles are a second, context-specific characteristic of the analyzed data set. Jira itself is unaware of roles (i.e., who should be changing a issue's state) and hence such information cannot easily be extracted via its REST API. As many of the metrics derived from the data set include roles, we cannot infer how useful our approach will be for environments where no role information is available or where roles and departments are not clearly assignable. This will often be the case in open-source projects because they rarely exhibit a clear department and/or role assignment structure. Hence a comparison of the four projects, respectively the metric usefulness, to open source projects would make little sense.

[3]https://issues.apache.org/jira

[4]https://help.github.com/articles/about-issues/

VI. Related Work

Issue trackers have become an important tools for teams to coordinate their work. Managing the increased number of issues, however, has become a challenge [14] that multiple researcher aim to address.

Luijten et al. [5] introduced a tool to generate three different views that enable assessment of the issue handling process: a high-level (Issue Churn View), a quantitative (Issue Risk Profiles) and a detailed life-cycle (Issue Lifecycle View) view. Knab et al. [15] visualize the duration of a process step (submitted, in_analysis, in_resolution, in_evaluation) with a pie chart and provide a state transition view for problem reports.

Sarma et al. [4] proposed Tesseract, a socio-technical dependency browser that enables exploration of relationships between artifacts, developers, bugs, and communications, for example highlighting developers that are modifying interdependent code but are not communicating with each other.

Dal Sassc and Lanza [6] implemented in*Bug, a web-based software visual analytics platform. Extracting data from bug tracking systems, different panels describe highlevel information such as duration (as a horizontal stacked bar chart) and status of bugs as well as fine grained views describing changes to a bug report's properties.

Similar, D'Ambros et al. [16] focus on becoming aware of critical issues. Their "Bug Watch" visualization helps to understand the various phases that it traversed. They note that the criticality of a bug is not only dependent on its severity and priority but also on its life cycle. Frequently opened bugs indicate deeper problems.

Halversion et al. [17] describe problematic patterns of change management for example recurrent loops (e.g. repeatedly resolving and reopening or reassigning) or unattended issues (when an issue remains too long without resolution).

Tüzün et al. [18] describe their progress towards a unified project monitoring solution based on the Essence language and kernel. Also Brandt et al. [19] build on the Essence framework for project state visualization but focus on a Kanban style visualization rather than metrics and issue history.

Poncin et al. [20], [21] introduced the Framework for Analyzing Software Repositories (FRASR) for combining data from source code repositories, email lists, and bug trackers. They subsequently utilize the ProM process mining framework for obtaining insights such as classifying developers in open source software projects to roles such as project leader, core member, peripheral developer, bug fixer, or reader. They also analyzed the typical transitions between bug report states on Bugzilla.

Gupta et al. [22] conducted process mining across an issue-tracking system, a code review system, and a version control system. They map events from these systems into a single process (based on states) and determine transition occurrences. Based on this annotated transition diagram, they analyze the bug-fixing process from reporting to resolution to discover bottlenecks, deviations from the intended process, joint activities, and work handover.

In our previous work, we investigated an alternative approach to defining explicit metrics [23]. We applied constraints mining to issue histories from multiple projects to derive meaningful metrics for describing the software development process. Such an approach is suitable to identify additional metrics for integration into our prototype.

VII. Conclusions and Future Work

In this paper we presented an industrial case study and approach for supporting metric-driven process improvement. Specifically, we focused on coordination-centric metrics and consequently targeted process-related data and events in our evaluation prototype *Process Inspector*. Qualitative analysis with team-leads and groups-leads from our industry partner demonstrated that the combination of metric data with issue timeline visualization is a powerful approach to obtain process insights and quickly identify flaws in the process, inefficient coordination in issues, and comparing coordination aspects across projects.

As part of future work, we plan to improve the prototype along the received feedback, but more importantly, evaluate the use of the prototype across a project's lifetime. This includes also evaluating to what extent these metrics are able to detect concrete differences when project teams are structured vertically as opposed to horizontally.

Acknowledgment

This work was supported in part by the Austrian Science Fund (FWF): P29415-NBL funded by the Government of Upper Austria; and the FFG, Contract No. 854184. Pro^{2}Future is funded within the Austrian COMET Program—Competence Centers for Excellent Technologies — under the auspices of the Austrian Federal Ministry of Transport, Innovation and Technology, the Austrian Federal Ministry for Digital and Economic Affairs and of the Provinces of Upper Austria and Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

References

[1] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality," in *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 2008, pp. 521–530.

[2] M. E. Sosa, S. D. Eppinger, M. Pich, D. G. McKendrick, and S. K. Stout, "Factors that influence technical communication in distributed product development: an empirical study in the telecommunications industry," *IEEE transactions on engineering management*, vol. 49, no. 1, pp. 45–58, 2002.

[3] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864–878, 2009.

[4] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 23–33.

[5] B. Luijten, J. Visser, and A. Zaidman, "Assessment of issue handling efficiency," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 94–97.

[6] T. Dal Sassc and M. Lanza, "A closer look at bugs," in *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on*. IEEE, 2013, pp. 1–4.

[7] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: A project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 446–465, 2005.

[8] V. R. B. G. Caldiera and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, pp. 528–532, 1994.

[9] D. Šmite, C. Wohlin, Z. Galviņa, and R. Prikladnicki, "An empirically based terminology and taxonomy for global software engineering," *Empirical Software Engineering*, vol. 19, no. 1, pp. 105–153, 2014.

[10] T. J. Allen *et al.*, "Managing the flow of technology: Technology transfer and the dissemination of technological information within the r&d organization," *MIT Press Books*, vol. 1, 1984.

[11] P. Diebold and S. A. Scherr, "Software process models vs descriptions: What do practitioners use and need?" *Journal of Software: Evolution and Process*, vol. 29, no. 11, pp. e1879:1–e1879:13, 2017.

[12] J. Nielsen, *Usability engineering*. Elsevier, 1994.

[13] L. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh, "The Case for Context-Driven Software Engineering Research: Generalizability Is Overrated," *IEEE Software*, vol. 34, no. 5, pp. 72–75, 2017.

[14] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*. ACM, 2005, pp. 35–39.

[15] P. Knab, B. Fluri, H. C. Gall, and M. Pinzger, "Interactive views for analyzing problem reports," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 527–530.

[16] M. D'Ambros, M. Lanza, and M. Pinzger, "" a bug's life" visualizing a bug database," in *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*. IEEE, 2007, pp. 113–120.

[17] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg, "Designing task visualizations to support the coordination of work in software development," in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. ACM, 2006, pp. 39–48.

[18] E. Tüzün, Üsfekes, Y. Macit, and G. Giray, "Towards unified software project monitoring for organizations using hybrid processes and tools," in *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, May 2019, pp. 115–119.

[19] S. Brandt, M. Striewe, F. Beck, and M. Goedicke, "A dashboard for visualizing software engineering processes based on essence," in *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, Sep. 2017, pp. 134–138.

[20] W. Poncin, A. Serebrenik, and M. van den Brand, "Process mining software repositories," in *Proc. of the 15th European Conference on Software Maintenance and Reengineering*. IEEE CS, 2011, pp. 5–14.

[21] ——, "Mining student capstone projects with FRASR and ProM," in *Proc. of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. ACM, 2011, pp. 87–96.

[22] M. Gupta, A. Sureka, and S. Padmanabhuni, "Process mining multiple repositories for software defect resolution from control and organizational perspective," in *Proc. of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 122–131.

[23] T. Krismayer, C. Mayr-Dorn, J. Tuder, R. Rabiser, and P. Grünbacher, "Using constraint mining to analyze software development processes," in *Proceedings of the International Conference on Software and System Processes, ICSSP 2019, Montreal, QC, Canada, May 25-26, 2019*, S. M. S. Jr., O. Armbrust, and R. Hebig, Eds. IEEE / ACM, 2019, pp. 94–103. [Online]. Available: https://doi.org/10.1109/ICSSP.2019.00021