Beyond the Four Key Metrics: Quantifying DevOps Practice

Sridhar Venkatesh

***Abstract*—This project aims to extend the standardized methods of managing software delivery, to ensure developers are provided with meaningful in-depth feedback on their practices. The problem with the standardized metrics is that they're too broad and don't consider the wide-ranging statistics that GitLab repositories will have to offer. Some repository statistics such as milestones, pipelines, epics to name a few, are presently unexamined by management features such as GitLab's 'Value Stream Analytics'. This project aims to provide richer statistics that should be analyzed when the software delivery process is being managed, which will also enhance and give guidance to how features such as 'Value Stream Analytics' can be extended. This project follows the structure of 'Process Mining' streamlining the process of collecting source workflows, comparing them against the model workflows, to perform discovery methods to understand how to improve the model workflows. This allows the chance of introducing important statistics which are pivotal for measuring DevOps performance - through taking source workflows and judging their performance against the existing DevOps metrics, along with additional analytics that should be considered when measuring productivity in a DevOps environment.**

I. Introduction

THE aim of this project is to refine the way we analyse and manage processes that are adopted for software delivery, to deliver enhanced feedback on demand. Feedback and metrics of process management performance are highly sought after in many DevOps environments. Through reading the metrics and feedback available to those operating in a DevOps environment, invites the opportunity for process improvement and learning. Depending on the richness of the analytics and metrics, this can determine the speed of process refinement and enhancement on the workers end. Reinforcing the need for valuable and definitive metrics that provide insight of developers' productivity, workflow management, feature planning etc. Currently DevOps teams utilise four standardised metrics as a basis to measure their workflow adherence and productivity. These metrics include 'Lead Time', 'Cycle Time', 'Deployment Frequency' and 'Change Failure Rate'.

On the surface, these metrics do a solid job outlining speed and stability of the work developers are doing [7], however it doesn't capture the full image/performance and could lead to potential false positives/negatives regarding process performance. For example 'Change Failure Rate' is a metric that determines the ratio of code deployments to failure in the production cycle. If a company is deciding to add minimal or low-complexity level features to it's code base, the valuation for 'Change Failure Rate' would be largely positive as there's minimal risk with some of the changes decreasing the likelihood for failure. An event such as this would ensure an inaccurate way to determine stability in a DevOps environment making changes - and could reduce the efficiency of translating the results of performance into actual process improvement.

The aim of this project is to find additional metrics beyond the four key metrics to ensure richer statistics/metrics/analytics are presented to developers, to ensure they are presented with a definitive measurement of their workflow management. Providing metrics that will apply to projects being conducted outside of those that will support business operations. This can be achieved through the technique of 'Process Mining' [10]. Which includes validating/comparing source workflows to a model workflow, and invites the opportunity to introduce new metrics off the basis on the gaps in the source workflows.

The source workflows in question will originate from GitLab repositories used by ENGR301 students. The reason for the use of GitLab repositories specifically is that it contains planning tools such as issues, milestones, and also contains support for automated processes such as CI/CD pipelines - this is quite convenient as this now ensures all the tools and services for the repositories are all in one place. The contributions a project member can make to a GitLab repository such as creating an issue, making a merge request or triggering a pipeline - are available for analysis through GitLab's full feature API [5] and python library [6]. The choice of ENGR301 repositories specifically and only ENGR301 repositories, is due to the fact all students are practicing the same workflow and will be working under the same project requirements which ensures all source workflows are practiced under the same context - evading any problems with bias or misinformation in the analysis.

There are many implications surrounding sustainability that can be supported by the outputs of this project. The ability to find new metrics and statistics improves the feedback which in turn gauges faster process improvement. Enhancing the insights based on the speed and stability of large-scale integrations and planning - ensures the optimization of resources. If there are more metrics putting more emphasis on the planning stage of a project than some metrics such as 'Lead time' or 'Cycle time' for example - this allows DevOps teams to plan understand the amount of usage for their resources better. Stronger feedback on the planning phase of a project ensures resources are adequately dedicated to meeting project requirements at hand, as opposed to wasting resources on unnecessary features that don't align with project specifications. An example that would be avoided with stronger feedback on

This project was supervised by James Quilty

the planning phase - would be an avoidable upgrade to server capacity to service unnecessary features that don't correlate to project requirements, optimizing energy consumption and costs [9]. Richer statistics that will lead to an enhanced level of feedback ensures both faster iterations and reduced waste. Producing a faster transition into process improvement throughout the project will ensure a better understanding of the cause of for example a deployment failure/failure in production and hence ensure for more responsible consumption of computational resources, electricity and time. These sustainability implications align with the 12th goal which depicts 'Responsible consumption and production' in the list of goals of sustainable development provisioned by the United Nations [14].

II. Related Work

This project adopts the technique of process mining to validate/compare git workflows to a model workflow. There are no current examples of Process Mining tools specifically for git Repositories. However there are examples of Process Mining which help establish common practices for dealing with each step. All of the discussion surrounding Process Mining universally agree on the phases that should be undertaken when performing process mining workflows.

A. Event Logs

The first phase which is discussed amongst the readings is the Event Logs stage. The reading provided here [10], provides an example of event logs tracking different cases of a web application. This provides a software specific example of how event logs could be adapted for a software solution, comparative to the process mining handbook [15], which sites a general business model example for the production of pizzas for a pizzeria. This simplified example helps to bridge the gap for people new to process mining and provides an example of how to structure a simplified event log to provide a structure latter phases.

The web application example aligns more with the needs of process mining git repositories. Each iteration of the model workflow in the provided example starts with an expected triggering point - in this case a 'request from a client' is the event which starts a case. This example alone groups the events into it's own cases which can be directly applied to that of a git workflow. Each iteration of the gitlab workflow [4] starts with a specific triggering point (specifically creation of an issue), which allows for it to be grouped into it's respective cases making it easier to group it's connected events (such as an attached merge request or tagged milestone).

		Attributes					
Case id	Event id	Activity		Lifecycle	Timestamp	Resource	...
1	1.1	*a*	receive request	start	30-10-2017 11:02:45.000	main-thread	...
	1.2	*a*	receive request	complete	30-10-2017 11:02:45.500	main-thread	...
	1.3	*g*	parse request head	start	30-10-2017 11:02:45.650	worker-1	...
	1.4	*d*	check authorization	start	30-10-2017 11:02:45.651	worker-3	...
	1.5	*d*	check authorization	complete	30-10-2017 11:02:45.710	worker-3	...
	1.6	*e*	reject request	start	30-10-2017 11:02:45.820	main-thread	...
	1.7	*e*	reject request	complete	30-10-2017 11:02:45.870	main-thread	...
2	2.1	*a*	receive request	start	30-10-2017 11:03:12.150	main-thread	...
	2.2	*a*	receive request	complete	30-10-2017 11:03:12.450	main-thread	...
	2.3	*g*	parse request head	start	30-10-2017 11:03:12.670	worker-2	...
	2.4	*b*	read data part	start	30-10-2017 11:03:12.670	worker-1	...
	2.5	*g*	parse request head	complete	30-10-2017 11:03:13.110	worker-2	...
	2.6	*b*	read data part	complete	30-10-2017 11:03:13.160	worker-1	...
	2.7	*f*	prepare query	start	30-10-2017 11:03:13.320	worker-1	...
	2.8	*f*	prepare query	complete	30-10-2017 11:03:13.400	worker-1	...
	2.9	*h*	process request	start	30-10-2017 11:03:13.670	main-thread	...
	2.10	*h*	process request	complete	30-10-2017 11:03:14.220	main-thread	...
3	3.1	*a*	receive request	start	31-10-2017 09:43:16.030	main-thread	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Fig. 1. Scalable software application of event logs.

This differs from the example provided from the process mining handbook [15], which collects all the events and sorts them by timestamp, without grouping the events by case to note the other events in that specific iteration. This example would not be suitable for the application of process mining git repositories - as it is important to be able to trace the events in the same iteration. Grouping them into individual cases allows for easier aggregating of the workflows to produce a cumulative overview of the logs in general for performance/conformance analysis.

B. Conformance Checks

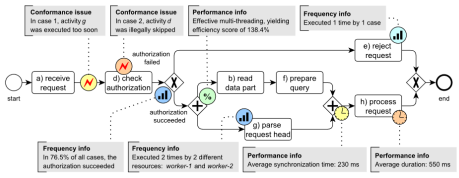

Fig. 2. Example of process model used to support conformance analysis.

The need for producing event logs is to ensure the source workflows are structured in a way to assess their conformance to the model workflow. Both examples provided from the respective readings require the need for both an event log and a process model to act as inputs for the conformance analysis phase. The model is designed to aggregate the typical behavior from the event logs. While the structure of the event logs (grouping the connected events into cases for better traceability) makes it easier for validating the source workflows. The major disadvantage with process modelling is that it purely depends on the stability of the event logs, if there isn't a largely consistent scheme which is prevalent through the event logs it would be the upmost difficult to cumulate it into a visualised model.

There also needs to be careful consideration on the outputs/goals of the project - which is to find metrics beyond the four main metrics [7]. Conformance analysis on the source workflows needs to not only spot gaps in them in comparison to the model workflow, but to also assess it's performance against the four main metrics. What both the handbook and software-oriented applications don't discuss about this phase is how quickly the processes are being performed - which can be assessed through the four main metrics previously discussed.

C. Process Discovery

The final stage orients around the results from the Conformance Checks/Performance Analysis to take in what processes dominated and the general gaps in the source workflows. This is what would lead into finding the additional metrics that should be added to assess the source workflows against. This differs from the definitions provided from both the handbook and the scaled software example [15] [10]. In the examples provided from the readings, the source workflows are assessed in the conformance checks phase to spot potential gaps - which is then sent into the discovery phase to determine how the model/reference workflow should be changed to adapt to the source workflows. This is adaptable for a business operations setting where it's important to understand customer behavior and adjust your services to suit them, however for an internal project setting this is not applicable.

The model workflow which is an adaptation of the GitLab flow [4], will not be adjusted to conform to the source workflows. Instead the source workflows performance according to the four key metrics is assessed to judge the relevance of the existing metrics, and to also understand what processes dominated. This invites the opportunity to introduce new metrics based on the processes that did dominate. This is due to the needs of the project not aligning with the typical needs/applications of process mining - the examples provided in both readings explain the need to support the source workflows and form internal processes that cater to customer needs. This project focuses more on the areas in the workflows that were common and creating new metrics to assess them, ensuring DevOps metrics are provided and applicable for a internal project setting.

III. Design of Solution

The solution will automate the event logs and conformance checking phases of process mining. It is planned to utilised GitLab's logging of contribution events to support both the logging and analysis. This is expected to be a CLI tool which will both generate and collect the contribution events from a given repository and specific author, and will generate an html based report similar to the scheme of another tool named 'GitInspector' [2].

A. Event Logs Design

The structure for the event logs will follow the same format as the scalable software example [10], which will group the events into it's relevant cases (in this case an issue would represent a case). Previously, in the preliminary report the event logs were expected to be grouped based on repository. In the example provided below - this follows more of the examples provided within the process mining handbook [15], however it does create some of the traceability issues previously cited, it doesn't align with the aim of tracking all the events in that specific iteration hence it makes it difficult to assess workflow conformance.

Project ID	event-type	timestamp
2002	created-issue	2023-07-08
19670	closed-issue	2023-04-06

Hence the change to structure the cases of event logs based on issues as an event such as creating a merge request/tagging to a milestone can be traced to that specific gitlab issue.

Case ID:	Event-type:	Timestamp:
1	create-issue	2022-10-14T14:19:26.536+13:00
	closed-issue	2022-10-14T14:44:45.377+13:00
2	create-issue	2022-09-30T12:57:09.212+13:00
	closed-issue	2022-10-07T14:04:46.303+13:00
3	create-issue	2022-09-30T12:56:34.032+13:00
	closed-issue	2022-10-14T13:57:41.356+13:00
4	create-issue	2022-09-30T12:55:59.402+13:00
	closed-issue	2022-10-17T13:25:22.904+13:00

Fig. 3. Soft example of issue event logs.

The example provided in figure 3, shows a preliminary example of how the event logs should be structured based on issue - with the unique identifier of the CaseID being generated off the Issue IID. This change to use the Issue ID (provided by GitLab) as a unique identifier for each case would mean that the event logs generated will be from each repository, meaning that an event log will have to be generated for each repository that is a subject for the research. Along with tracking the timestamps of the created issue - event logs will be structured as a .csv file to support the visualisation aspect of the conformance analysis phase. This will mean a .csv file (event log) would have to be generated for each repository that has been provisioned for the research. Causing unnecessary bloating for the file system at hand.

Additionally we would also have to consider the ethical considerations attached with gathering repository event logs. These source repositories owned by the students of the ENGR301 Project Management class would need to provide full permission to utilise the event logs. On top of this these repositories also are owned by groups of people - meaning that we would require permission from all the members of a specific repository to collect all the events from that given project. This would guarantee that filtering based off the specific project member is also required to address these specific ethical issues as we wouldn't be allowed to gather events from students who don't provide permission.

It was also previously mentioned that the event logs would utilise dependencies such as GitLab's REST API resource [5], which contains several endpoints necessary for tracking workflow conformance, along with python-gitlab [6] which is a python module capable of gathering issues, merge requests and pipelines from an inputted repository. Both of these will serve as dependencies for generating the event logs.

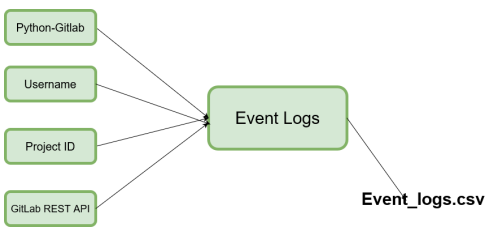

Fig. 4. Event logs and it's dependant modules and inputs

From the figure provided above - this shows the expected inputs and outputs of the events logs portion of the CLI tool. Expecting both a username and the repository ID to be provisioned as support. The output generated from the event logs is a .csv file which is expected to act as input for the conformance analysis/performance calculations.

Another consideration is the adequate structure of the sequence of events for creating the event logs. To collect all the possible issues, merge requests and pipelines associated with the inputted user, this would require a few API calls along with appending such results to the csv, potentially meaning this phase could be a bottleneck for the rest of the tool. Another problem would be the traceability of the events - it is completely possible that the user in question could be an author of a merge request which is attached to an issue they didn't create. Meaning we would have to filter based off the merge requests they authored, to find the connected issue and additional events etc.

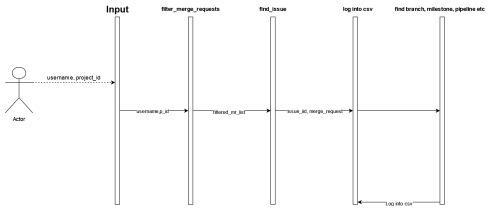

Fig. 5. UML Sequence Diagram showing the order of procedures in event logs

From the figure above, this shows the sequence of processing events within event logs, with the additional events using the same scheme for finding connected issues to merge requests - this ensures for each entry in the .csv file will have an issue grouped with it's associated events and the timestamps that those events occurred. This is also structured to ensure functions or procedures aren't repetitively called during the event logs to optimize performance and not put too much load or dependency on the GitLab server which would have to respond to all the API calls etc. The timestamp format used by GitLab is 'iso8601', naturally this timestamp format would be unrecognizable for many graphing tools. However the tools to convert that into a datetime readable format will not be performed before logging the details into the csv, this is done to restrict the amount of dependent libraries for this stage as we only want to have dependencies on the gitlab-api and the python library.

B. Conformance Analysis

The next phase in question is the conformance analysis done on the source workflows. At this point of completion in the event logs - it is expected that both the event logs are structured necessarily to calculate conformance according to the four key metrics. The outputs of this phase is an html report similar to the scheme of 'GitInspector' [2].

Fig. 6. Example run of GitInspector on own repository

The figure above shows an example of GitInspector's ability to collect the contribution events within the given repository and display such information for all members. Tracking lines of code that have been contributed into the current revision, weekly contributions etc. These statistics displayed in GitInspector won't specifically be queried or displayed on the conformance check page - however its established a good scheme to display the statistics - which will be adopted for displaying conformance to the four standardised metrics.

Considering the inputs for the conformance check phase is a csv - we can utilise libraries such as pandas [12], to structure the data into tables to ensure easy conversion into some visualisation of a graph. Another consideration is how we would allow for both calculations based off the event logs as well as producing graphs and finding a natural way to export them into html for the report. Tools such as Bokeh [1], mpld3 [11] and plotly [13] - were all experimented with to decide which library was suitable for exporting the graphs/figures into the html report. Each module going off the basic documentation and showcase would be suitable for exporting the graphs into html - however it is desired that a solid user experience can be felt when using the reports. The reports are supposed to be sent out to students with the graphs and calculations being the focal point of the report.

When trialling out Bokeh it was clear to see it had a wide variety of graph styles/chart styles meaning it was fit for purpose, along with having a good level of interactivity such as hovering over points on a line graph and showing the value in a popup.

Plotly can perform this too, as well as provide a wide ranging amount of graphs that would help for displaying workflow conformance such as Sankey Diagrams, Value Stream Maps etc. Plotly's high level API makes it easy to customise the diagrams and charts. Plotly has a more gentle learning curve than Bokeh - whilst Bokeh has the capability of providing abstract, interactive and complex data visualisations it requires abstract ways to structure your dataset to be able to make these complex visualisations. Along with this it takes less lines of code to produce the visualisations in 'Plotly' comparative to creating the same graphs in Bokeh and Mpld3. This design decision can have sustainability implications - particularly

surrounding 'Green Computing', reducing the lines of code that needs to be parsed by a processor leading to greater memory and energy usage overall. While this may be a small change to make a partial impact, it still contributes to the scheme of writing more 'green software' [8].

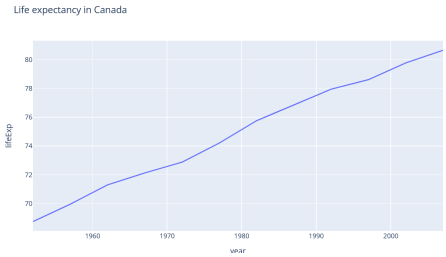

Fig. 7. Plotly Example of producing a line graph

Taking into account the customization capabilities and how easy it is to adjust to, plotly suits the needs of the project in comparison to the other modules such as 'Bokeh' and 'mpld3'. Also taking into account time constraints 'plotly' is the most suitable and is the most capable of matching the needs of the project.

After determining the packages to support both the calculation and visualisation of the event logs, we also need to determine the sequence of calculations that will happen in the conformance analysis phase.

'Lead time' - for an issue calculates the time taken from the time an issue is scoped to a milestone, to the time that the issue is closed. The overall Lead Time for the repository takes into account the median value for all these time differences for each issue. 'Cycle time' - takes into account the time taken from a merge request opening to the issue that it was attached to closing, overall cycle time is calculated in a similar fashion to 'Lead time'. Both deployment frequency and change failure rate take into account the amount of deployments over a specific time frame, where change failure rate considers the ratio of failed deployments in comparison to deployment attempts. These calculations can be calculated by using different fields each line of the csv file. Each of the formulas required for each metrics take into account 1 or 2 fields each - meaning that there doesn't need to be any specific order established to make a difference in speed or performance.

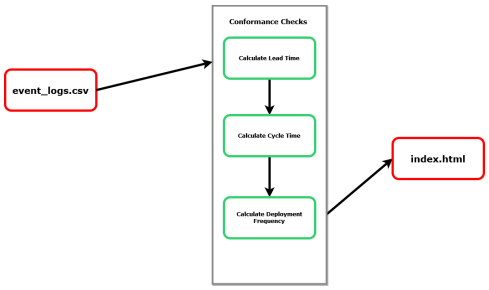

Fig. 8. Structure of Conformance Analysis phase.

In figure 8 the block diagram shows the order of events that should be processed in the conformance analysis phase. Each portion such as 'Calculating Lead Time' is responsible for calculating lead time for each issue that's relevant, as well as calculating overall lead time and producing the respective graphs - and output them into the .html file.

C. Overall Design

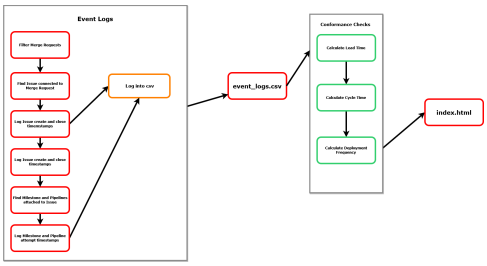

Fig. 9. Diagram showing format of modules and tasks of CLI tool.

This is the final format of all the modules and tasks performed in each phase of process mining. The event logs will take a Project ID and member username to perform each step, mainly for polling events from the GitLab API and python-library, it's only purpose is to log and format the .csv in a structure where it is easy to perform the conformance analysis calculations in the final stage.

IV. Artefact Implementation

Before commencing on the general implementation of the CLI process mining tool - the first step was to collect and gather permission from students to use their repository contribution event logs for testing and to support results for the research. Considering the amount of time this can take - we had to adjust for testing the scripts on my previous GitLab projects. This can be quite problematic for collecting event logs as most of the projects (that were conducted from 1st year/2nd year), don't work accordingly to the GitLab flow [4]. This means when generating the event logs there will be minimal logs (issues, and general events tagged to issues) to visualise and produce - making it difficult to fully test and verify the robustness of the solution before collecting and assessing the logs of the ENGR301 repositories.

To address this issue for testing the solution, an attempt to automate the creation of a test repository was done with the GitLab supported python library 'python-gitlab' [6]. This library has the helpful tools to automate the creation issues/milestones/register merge requests from python scripts - however this does come with some limitations, such as not having the ability to create a merge request and branch simultaneously with the reference of an issue. This makes it difficult for logging and testing the tools ability to spot linked merge requests - we want to emulate an environment and repository that aligns closely with the repository events performed by the students. A solution/compromise to this issue was to use the repository hosting the artefact itself as a basis

for testing and collecting the logs. That way it properly simulates planning, development, and review stages in a typical DevOps environment practicing the GitLab workflow [4].

Work commenced by prioritising the collection of event logs and using the GitLab repository itself as a test base. Specifically using 'python-gitlab' and the GitLab API interchangeably to search for events and statistics to log. Whilst orienting myself with the capabilities of the API - it was also important to note the push rules/configurations of the repositories itself, some repositories and the 301 projects in particular can be set up with the scheme of squash commits/ or deleting branches on merge. These events are hard to track as they're deleted from the projects internal storage and tracking events such as commits or pushing branches aren't accessible from a simple endpoint such as '/branches' or '/commits' [3]. A solution to this is the GitLab API contains an endpoint such as '/events' which contains all the contribution events under one endpoint - it is preferable to limit use of this endpoint as it can have a negative impact on performance having to filter through 1000 different events searching for a 'branch creation' event or searching for the existence of a specific commit under a merge request event - however this is the only endpoint which makes these commit logs available for analysis.

Another consideration which was previously mentioned was the instance of a student starts a merge request on an issue they didn't create - specifically in an instance that the student being analysed follows in this example in the sense they're starting the merge request. GitLab stores the information based on author meaning that the log itself is traceable to the student to authored the event - implying an ethical issue with collecting logs of students who didn't consent to the research [5]. This was fixed by filtering based on merge requests authored by the inputted username (a student who volunteered for the study).

The .csv which is acting as output for the event logs portion was structured to dedicate each line for an issue, and each field dedicated to the events connected to the issue.

Issue ID	ms tagged at	mr created at	mr merged at
67	2023-07-08	2023-04-06	2023-07-08
12	2023-04-06	2023-07-08	2023-08-09

This table shows how the event logs were formatted and collected, where the fields 'ms tagged at' means for the timestamp which the issue was tagged to a milestone, and 'mr created at' and 'mr merged at' means the timestamp for opening and closing merge requests respectively. We've also decided to separate pipeline event logs into it's own file as it'll be more suitable for the calculations of metrics such as 'Deployment Frequency' and 'Change Failure Rate' if we collected all the pipelines triggered, and there's no need to trace the logic of pipelines to issues.

Using this change - we could use the information from the 'issues-event-logs' file to calculate metrics such as 'Cycle time' and 'Lead time', which require evidence of adhering to the workflow (through the existence of certain events).

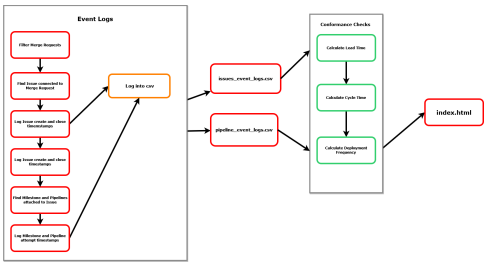

Fig. 10. Structure of final design

This image here shows the revised format of the event logs producing two csv files for calculation and visualisation in the conformance check phase, including another event logs portion for pipelines specifically to help calculate 'Deployment Frequency' and 'Change Failure Rate'.

After establishing such changes and completing the logic for collecting issue traceable logs and pipeline event logs - then it was time to perform the necessary calculations and visualise the logs in conformance analysis. Each metric is calculated as follows:

$$LT = i_c - ms_o$$

ms_o = Milestone tagged at
i_c = Issue closed at
LT = Lead time

$$CT = mr_m - mr_o$$

mr_o = merge request opened
mr_m = merge request closed
CT = Cycle time

*Note Deployment Frequency and Change Failure Rate has been adjusted to pipeline attempts and pipeline failures (which differs from normal convention), to match the events of the source workflows.

$$DF_m = np/31$$

np = Number of pipeline runs
DF_m = Deployments/Pipeline frequency by month

$$CFR_m = np/np_f * 100$$

np_f = Number of pipeline failures
CFR_m = Change Failure Rate by month

After establishing the calculations using both pipeline event logs and issue event logs, graphs were created to visualise the outputs.

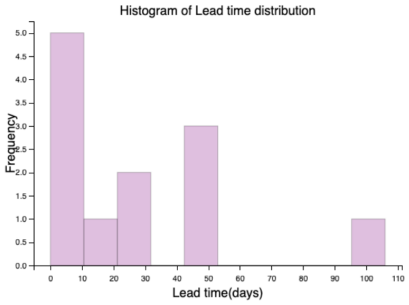

Fig. 11. Histogram of Lead time

This histogram of the lead time distribution is done from the event logs of my own repository. This is regarded as a positive case where majority of the issues I have development work for, match the scheme of scoping an issue into a milestone, and connecting the merge request of work to the issue so it's traceable through the algorithm.

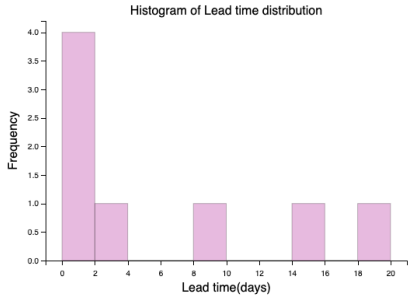

Fig. 12. Histogram of Lead time in students repositories

After trialling out the scripts for calculating and visualising Lead time, on the selected ENGR301 Repositories who provided consent for analysis - we get the output shown above. This already provides great insights on the processes that dominated - as through the calculations and visualisations we're also filtering out issues that don't have a milestone attached to the issue. Only 8 out 40 issues collected we're able to provide valuations for all of milestone tag, merge request open, merge request merge and issue closing timestamps. The main bottleneck here is the lack of milestone tags for issues - which means the calculation will never be triggered.

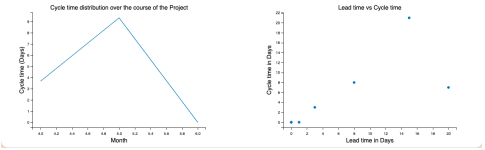

Fig. 13. Average Cycle time over the months of the project.

For the Cycle time calculations on the selected repositories we found cycle time would improve within the final phase of the project - and the development cycle (merge request lifespan) taking up majority of the lead times for each issue. Even some instances when the merge request starts before the issue is scoped into a milestone (due to Cycle time being greater than Lead time) - which is another point to raise for process discovery. We also need to consider how long merge requests last due to the lack of frequent approvals/reviews done by other team members, this can unfairly inflate the development cycle time and read out negatively on a members productivity.

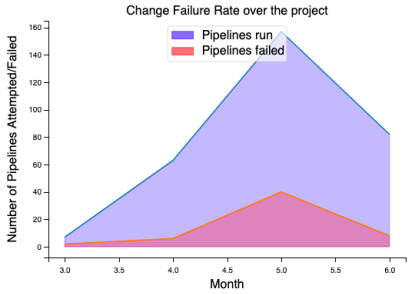

Fig. 14. Average Deployment Frequency and Change Failure Rate over the project.

After this we have the analysis based on pipeline attempts and failures - this can indicate that throughout the project - students were gaining enough in-depth feedback to familiarise themselves with the project environment to reduce pipeline failures. Another indications can be the low severity of the changes people are making gradually reducing throughout the back-end of the project to reduce the risk of a pipeline failure.

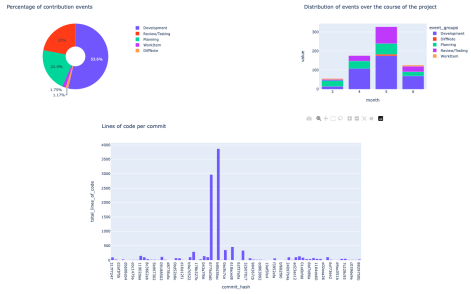

Fig. 15. Additional events and reviews to be considered

Beyond the four key metrics, analysis is also performed on the additional activities a member can perform outside of Developer work - this function mainly uses the '/events' GitLab API endpoint to collect all possible events committed

at the repository. Such as giving credit for reviewing merge requests/work done on testing and integration and general discussion. Another important aspect to track is the amount of atomic commits - 'atomic commits' are a great way of preventing unneccessary bloat of memory usage and resource consumption from using a repository. From the graph provided above we can see the presence of binaries being committed ranging from about 3500-4000 lines of code in one commit - which gives an idea for another metric we could introduce to measure stability and consistency of commits.

V. Evaluation

After taking into account the findings from assessing the source repositories against the four key metrics - we're able to introduce new metrics that would suit the nature of the source workflows. The first metric proposed is:

$$IRR = \frac{Ni_c}{Ti_o} * 100$$

Ni_c = Number of issues closed within a given time frame
Ti_o = Number of issues opened at the start of a given time frame
IRR = Issue Resolution Rate

Issue Resolution Rate caters to the huge gap in the workflow where a minimal percentage of issues have been scoped into a milestone. Basing it off a specific time frame, for example a month, this tracks how fast issues are resolved - and also prompts the need for using milestones as this would provide a great way to track productivity and performance within specific iterations.

Another metric introduced to cater to the nature of the workflows is:

$$M_dt = M(mr_r - mr_o)$$

M_dt = Median development time
mr_r = Time when Merge request is marked as ready
mr_o = Time when Merge request is opened

Median development time takes into account the issue of the development cycles being unfairly bloated due to the time considered to review/approve their body of work. There is the ability to mark when a merge request is ready for review and indicate that to other team members. Tracking the time taken from merge request opening, to the last instance of marking the merge request as 'ready' - to get closer to reproducing the general time taken in a development cycle. The overall development cycle time is determined by the median of all the time differences, to reduce the impact of potential outliers.

Based on the ability to find new metrics to align with the source workflows found in internal projects - this meets the requirement of finding metrics from process mining the source repositories.

The last metric introduced revolves around detecting the presence of binary files being committed to the repository to prevent avoidable resource consumption [8].

$$BFCR = \frac{No_cb}{No_c} * 100$$

$BFCR$ = Binary File Commit Rate
No_{cb} = Number of commits that exceed the limit (¿1000 lines of code)
No_c = Number of commits in total

Binary File Commit Rate acts more of a detection of adherence to the workflow compared to the other metrics proposed. Being able to track and identify instances of binaries being needlessly committed to the repository can help DevOps engineers to place practices to prevent instances like these occurring down the track of the project and ensure engineers are following the 'green coding scheme' to also take responsibility of the sustainability implications of the project.

Performance was compared through the runs on my own repository with a minimal amount of events to the ENGR301 repositories. The main bottleneck spotted in performance came through the collection of issue oriented event logs and the calculation of 'Lead time' and 'Cycle time' from it.

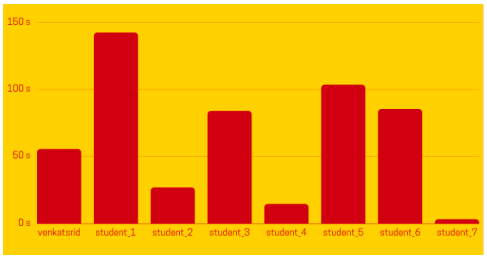

Fig. 16. Performance Test comparing the time taken to collect issues event logs of own repository compared to source repositories.

From the figure provided above majority of the runs take under 2 minutes to generate the events logs as well as calculate 'Lead time' and 'Cycle time'. This huge gap in performance sits well outside the window of an efficient tool that is effective for process mining - and the event logs should be serviced under a minute. The reason for this huge difference in performance for collecting the logs from my own repositories in comparison to the student repositories - is due to the amount of filtering for events that need to occur. On my own repository where I am the sole contributor, almost no time will be dedicated for scanning and ensuring that event belongs to that user - whereas for the ENGR301 source repositories, filtering and verification has gone in to ensure that event belongs to the user in question, this is to ensure compliance with previous ethical issues such as collecting logs from non consenting users.

VI. Future Work and Conclusions

There are many aspects to build off from the findings of this project. Firstly - through the ability of a tool named 'Value Stream Analytics' - which is built in to all GitLab repositories to assess the overall performance against the four key metrics [7]. The next point would be to push some of the metrics proposed to be actioned and reviewed within 'Value Stream

Analytics' to determine it's relativity to real-world projects and is serviceable for GitLab to provide calculations for.

Another is to add more visualisations that tell more about the processes being conducted as opposed to measuring conformance against the four key metrics - whilst measuring against the four main metrics allowed for identifying some key gaps in the workflow, it could be even more identifiable if the tool were to provide visualisations of the logs collected to tell more about the specific workflows conducted. Using graphs such as a Sankey Diagram, or a 'Value Stream Map' to aggregate the workflows.

REFERENCES

[1] BOKEH. Bokeh documentation/showcase. https://bokeh.org/. Accessed: 2023-10-13.

[2] EJWA. Git inspector repository. https://github.com/ejwa/gitinspector. Accessed: 2023-10-12.

[3] GITLAB. Branches api endpoint. https://docs.gitlab.com/ee/api/branches.html. Accessed: 2023-10-15.

[4] GITLAB. Gitlab flow. https://about.gitlab.com/topics/version-control/what-is-gitlab-flow/. Accessed: 2023-10-11.

[5] GITLAB. Gitlab rest api. https://docs.gitlab.com/ee/api/api_resources.html. Accessed: 2023-10-10.

[6] GITLAB. 'python-gitlab' documentation. https://python-gitlab.readthedocs.io/en/stable/. Accessed: 2023-05-27.

[7] GREGORY, P. Agile processes in software engineering and extreme programming. https://directory.doabooks.org/handle/20.500.12854/70808.

[8] IBM. Why green coding is a powerful catalyst for sustainability initiatives. Accessed: 2023-10-14.

[9] IMPACT, E. Resource optimization. https://www.engieimpact.com/expertise/resource-optimization. Accessed: 2023-10-10.

[10] LEEMANS, M. Hierarchical process mining for scalable software analysis. https://pure.tue.nl/ws/portalfiles/portal/109318783/Leemans_thesis_normal.pdf. Accessed: 2023-05-23.

[11] MPLD3. Mpld3 showcase and documentation. https://mpld3.github.io/. Accessed: 2023-10-13.

[12] PANDAS. 'pandas' table formatting. https://pandas.pydata.org/docs/user_guide/style.html. Accessed: 2023-05-29.

[13] PLOTLY. Plotly showcase. https://plotly.com/python/. Accessed: 2023-10-13.

[14] UNITEDNATIONS. United nations 17 goals of sustainable development. https://sdgs.un.org/goals. Accessed: 2023-06-02.

[15] VAN DER AALST, W. M. P. Process mining handbook. https://link.springer.com/book/10.1007/978-3-031-08848-3. Accessed: 2023-10-11.