Faculty of Science, Technology and Communication

Measurements and Visualisation based on data generated by deployment pipeline executions

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Information and Computer Sciences

Author:
Harisha Prakash

Supervisor:
Dr. Alfredo Capozucca

Reviewer:
Prof. Domenico Bianculli

August 2022

Declaration

I hereby declare that the contents of this dissertation are entirely original except where specific references to the work of others are made. They have not been submitted in whole or in part for consideration for any other degree or qualification at this or any other university. Except as specified in the text and Acknowledgements, this dissertation is entirely my work and contains nothing that results from collaborative work with others.

Harisha Prakash
August 2022

Acknowledgement

Firstly, I would like to express my sincere gratitude to Dr. Alfredo Capozucca for supervising this master's thesis. I have gained valuable transferable skills from working on this project under his supervision. His organized way of doing research helped me achieve more in a short time. Moreover, his immense knowledge in the field, motivation, joy, and tolerance helped in all times of thesis and thesis writing. His method of systematic research taught me an organized way to approach a challenging problem.

I thank the reviewing committee: Alfredo Capozucca and Domenico Bianculli, for reviewing the thesis, their insightful comments, and challenging questions.

Last but not least, I would like to thank my wife Asha Krishnamurthy and my parents for being supportive during the whole master's study.

Contents

Abstract

Continuous delivery is a well-established method known for enabling software development teams to enhance their performance on software delivery. Applying continuous delivery implies (technically speaking) to implement a deployment pipeline. It is of utmost importance for development teams practising this method not only to implement the deployment pipeline, but also to perform measurements to assess their progress towards high performance. There exists a plethora of metrics in the domain of continuous delivery that is both contradictory and confusing.

The aim of this thesis is threefold:

1. To find the the state-of-the-art metrics related to continuous delivery and evaluate which ones can be measured using data produced by the execution of the deployment pipeline
2. Operationalization of some of these metrics
3. To create dashboards for the visualization of metrics

In order to achieve this goal, a systematic literature review and content analysis are performed to collect the state-of-the-art metrics related to continuous delivery from the scientific community. The metrics are operationalised by performing an experiment. A rapid review and content analysis are performed to collect the types of charts that can be used for visualization of the measured metrics. To determine the most effective chart for visualizing the metrics by creating dashboards, a web-based survey is carried out.

List of Figures

List of Tables

Chapter 1

Introduction

We live in a time where practically all businesses manage their operations using software. With its constantly changing customer needs, tightening time-to-market deadlines, and unpredictable market conditions, software development has become a demanding area of business [12]. Release of new software versions frequently and early has become a practical option for an increasing number of practitioners with the transition to the continuous deployment pipelines[39, 30]. Teams must gather, examine, and monitor a variety of metrics in order to deliver products of higher quality faster in order to live up to the continuous delivery promise. These metrics related to continuous delivery give teams the crucial information they need to see and manage their deployment pipeline. However, there is evidence that knowing what are the metrics to be measured in the area of continuous delivery it is still an open question for practitioners[36].

This thesis aims to collect the metrics related to continuous delivery published in the scientific community, measure those metrics using data produced by the execution of a deployment pipeline using a software product, and visualise these metrics using open-source visualisation tools. The scope, problem statement, and the research questions addressed by this thesis are explained in detail in the following sections in this chapter.

1.1 Scope

This thesis mainly focuses on to measure the state-of-the-art metrics related to continuous delivery, which are based on data generated by the execution of the deployment pipeline. Furthermore, instrument and visualise the found metrics using open-source visualisation tools. The detailed information of these metrics are provided in chapter 3.6 and instrumentation and visualisation of the metrics measured are given in chapter 4 and 5. A web-based java project using Spring Boot framework with Apache Maven as automated build tool is used as the product. A deployment pipeline is defined and implemented to automate the build, deploy, test, and release processes of the product. A detailed description of the product and deployment pipeline is given in section 4.3 and 4.4 respectively.

1.2 Problem statement

For software to survive in a fast-paced environment, it is critical for the software development team to collect, analyze, and monitor a variety of metrics in order to deliver higher-quality products faster and meet customer needs. There have been very few studies conducted on metrics to be measured in the area of continuous delivery. However, there is no much studies available that explains on which metrics can be measured using data produced by the execution of the deployment pipeline and also the instrumentation of these metrics are not discussed much in the scientific world.

In this thesis, a systematic literature review is made to determine the state-of-the-art metrics that can be measured using the data produced by the execution of the deployment pipeline. It also consists of the answers to the following questions:

1. How to make it operational of the found metrics into a technical environment that implements a deployment pipeline.?

2. How to visualise the measured metrics?

1.3 Research questions

This thesis tries to investigate the problems defined in section 1.2 by performing a systematic literature review. Following are the research questions on which the SLR is based-

RQ1 *What are the state-of-the-art metrics related to DevOps, which are based on data generated by the execution of the deployment pipeline?*

RQ2 *How are these metrics (operationally speaking) measured?*

RQ3 *What do these metrics allow us to conclude? In other words, what are the properties that such metrics allow us to measure?*

RQ4 *What are the best suitable charts to visualise each metric?*

1.4 Research methodology

In this section, we will discuss how this research is structured

1. SLR to find out metrics
   (a) To find the state-of-the-art metrics related to a continuous delivery, an SLR is conducted, and only empirical papers were considered for this study. The detailed selection procedure for the research papers is explained in the section 3.3
   (b) To evaluate which metrics can be measured using data produced by the execution of the deployment pipeline.
2. Instrumentation for data collection
   (a) An experiment is conducted to measure a subset of metrics from the list of metrics obtained from the previous step.
   (b) A general information about the software product used in this experiment is explained in detail.
   (c) The infrastructure and the instrumentation required to collect the data for each metrics are explained in detail.

3. Instrumentation for data visualisation
   (a) A rapid review(RR) is conducted to determine the state-of-the-art charts used for data visualisation.
   (b) A survey research is carried out to find the best suitable chart type to visualise the each of the metric measured.
   (c) The implementation of dashboards to visualise the metrics which are measured are explained in detail.

1.5 Contributions

The contribution made in this thesis is five-fold:

1. First, a SLR performed to find the state-of-the-art metrics related to a continuous delivery.
2. Second, an experiment is conducted to apply the results obtained through the SLR to measure the metrics using a software product.
3. Third, a RR and survey is conducted to find the best chart types to visualise each of the measured metric.
4. Fourth, a dashboard is created to visualise each of the metric.
5. Fifth, a replication package is created which can be used by practitioners as is or with minor modifications to measure the metrics.

Chapter 2

Background

In this chapter, we will discuss the basic terminologies and concepts used in this thesis.

2.1 Continuous Integration

In CI, team members regularly integrate their work and use automation to build, test, and validate software [45]. It facilitates the rapid identification and elimination of defects while enhancing software quality.

2.2 Continuous delivery

Continuous delivery is an Agile software production method in which operation teams iteratively deliver a valuable software artifact in a short cycle time to ensure software is always in a release state while developers make thousands of changes on a daily basis. [10]. According to Humble ([23], page 12) continuous delivery, the means for software development teams, is a software delivery process meant to deliver high-quality, valuable software in an efficient, fast, and reliable manner. The core component in continuous delivery is the Deployment pipeline or Continuous delivery pipeline [46, 10]. The detailed information on

deployment pipeline is given in section 2.4.

2.3 Continuous deployment

Continuous deployment is an extension of the continuous delivery process, where the application or product is automatically deployed to the production environment ([23], page 266) or deployment of new features directly to the end users several times a day [14].

2.4 Deployment pipeline

A deployment pipeline is an automated implementation of an application's build, deploy, test and release process ([23], page 3). It is also referred to as a build pipeline, a deployment production line, or a continuous integration pipeline ([23], page 110) or continuous delivery pipeline [46, 10]. An example of a deployment pipeline is given in figure 2.1.

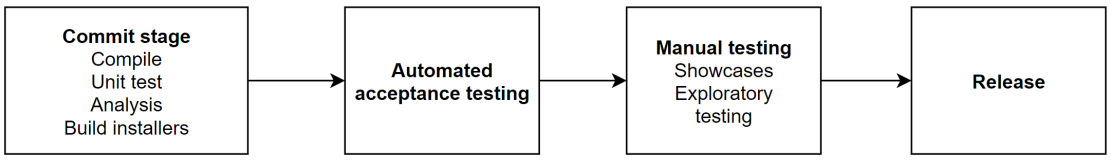


Figure 2.1: The deployment pipeline [23]

According to Humble [23], in general, pipeline consists of three stages which are *Commit*, *Test* and *Release* stage and each stage contains a set of activities. Following types of activities are assumed as those covered by a deployment pipeline.

- *Commit:* any activity meant to assert the product under development works at the technical level: i.e. it compiles, passes suite of automated tests, and code analysis,
- *Automated test:* any activity meant to assert the product under development works at the functional and nonfunctional level: i.e. it meets the needs of end-users,

- *Manual test:* any activity with the same purpose as in "Automated test", but whose execution is not (or cannot) be automatically executed, and

- *Release:* any activity meant to transfer the latest available version of the product under development to its production environment such that it becomes available to its end-users.

These type of activities lead to the definition of stages (i.e. set of activities) which are grouped based on their purpose. In this study, it is assumed that a deployment pipeline contains (at least) three ordered stages. The stages are:

1. *Commit stage:* to enclose "Commit" activities,

2. *Test stage:* to enclose "Automated and Manual test" activities

3. *Release stage:* to enclose "Release" activities.

2.5 Metric

Metric is defined as a tuple of the formula($\mathcal{F}$) and its unit($\mathcal{U}$) $\langle \mathcal{F}, \mathcal{U} \rangle$. Moreover, it is a quantifiable and measurable entity. E.g. NLOC(number of lines of code) is a metric which is quantified using a formula and could be measured using a code analysis tool.

2.5.1 Measurement

Measurement is the real-time data collected for a metric. The data are collected and stored in the database along with the timestamp at which the metric is measured. These time-series measurements help us to make decisions that are strongly backed by the evidence. E.g., Suppose we NLOC is increasing by very large number. In that case, we could look at the complexity of the software product which allows to predict required efforts to maintain the product.

2.5.2 Operationalisation

The term operationalisation is used to describe the act of translating a construct into its manifestation[48]. In this study, term operationalisation is used to show how some of the metrics which are found from the SLR conducted can be measured using the deployment pipeline.

Chapter 3

Systematic literature review

In this chapter, to synthesize the evidence for metrics related to continuous delivery, a systematic literature review as described by Cartaxo et al. is done [4].

3.1 Methodology

The methodology chosen to find answers to the research questions was a particular kind of systematic literature review (SLR) known as *rapid review* (RR) [4]. The reasons to perform a RR (rather a SLR) were two: (1) the targeted state-or-the-art had to be found in a timely manner to allow the retrieved knowledge to be quickly transferred to practice, and (2) practitioners were required to actively participate in the execution of the RR. It is worth mentioning that, when possible, the RR was enlarged to adhere to a classic SLR. Details about how characteristics of the RR that have been expanded towards a SLR are provided in Chatper 6.

3.2 Search strategy

To carry out the RR, ACM digital library with ACM Guide to Computing Literature database was chosen as the scientific repository. This

database was queried to get all the relevant papers that discuss about the research questions (RQ) mentioned in Section 1. To fetch relevant papers, some search query trials were executed using keywords aligned with the RQs. Running these trials was the first step taken to identify studies and prevalent synonymous words related to the research questions. The goal of these search query trials us can be summed up as follows:

1. To find the papers related to “Continuous” delivery and “Metrics”
2. To find keywords related to “Continuous delivery” and “Metrics”

The remaining of this section describes the steps that were performed to obtain the final corpus of the RR. This corpus was then used as baseline to identify and extract relevant data. The results of the identification, extraction and classification of the information allowed to perform the synthesis of the findings reported in this study. These activities, along with their sequential order are depicted in Figure 3.1.

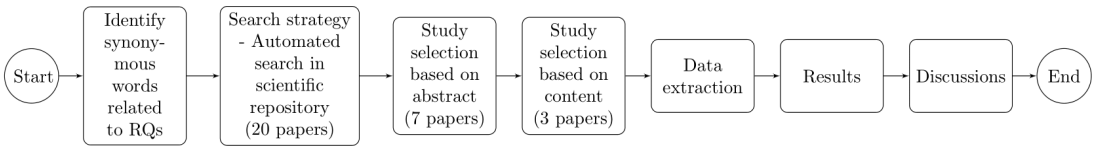


Figure 3.1: Flow chart - Methodology

Following filters are applied sequentially to obtain the final corpus for the SLR-

F1 Firstly, selected the papers related to the topic ”Continuous Delivery”. The prevalent synonymous (or closely connected) words related to “Continuous Delivery” used in the scientific literature are identified as “DevOps”, “Continuous Deployment”, and “Deployment Pipeline”. So, the filtered papers should have either DevOps, Continuous Delivery, Continuous Deployment, or Deployment Pipeline in their title or abstract.

F2 Secondly, out of all those papers selected in the F1, shortlisted the papers related to topic ”metric”. The prevalent related words for metric are assessment, measure or monitor. So, shortlisted papers

should contain either metric, assessment, measure or monitor in the title or abstract.

F3 Thirdly, select papers only between the duration of 2017 and 2021. The reason for selecting the papers only published in the last 5 years is to include only the recent studies. From figure 3.2, it can be observed that around 70% of the papers are from the last 5 years. Thus, this represents a good trade-off between coverage and review effort.

F4 Fourthly, select papers only related to the research article. The reason for selecting only research articles is to retrieve the latest strong validated [1] outcomes from the research community. Thus, these research papers provide a strong foundation for the content analysis.

The final search query with all of the above filter is-

"query": Title:(("DevOps" OR "Continuous Delivery" OR "Continuous Deployment" OR "Deployment Pipeline") AND ("Metric" OR "assessment" OR "Measure" OR "Monitor")) OR Abstract:(("DevOps" OR "Continuous Delivery" OR "Continuous Deployment" OR "Deployment Pipeline") AND ("Metric" OR "assessment" OR "Measure" OR "Monitor")) "filter": Article Type: Research Article,Publication Date: (01/01/2017 TO 12/31/2021)

3.3 Study selection

Candidate papers obtained after having applied the filter mentioned in step 3.2 needed to be reviewed to determine their suitability for the goal of the study (i.e. to convey relevant information that may allow one or more research questions to be answered).

The review of these candidate papers was performed by two independent reviewers. Their reviews were discussed until a final agreement was reached: i.e. the candidate paper was either excluded or kept into the final corpus.

[1] By the community itself through peer-review.

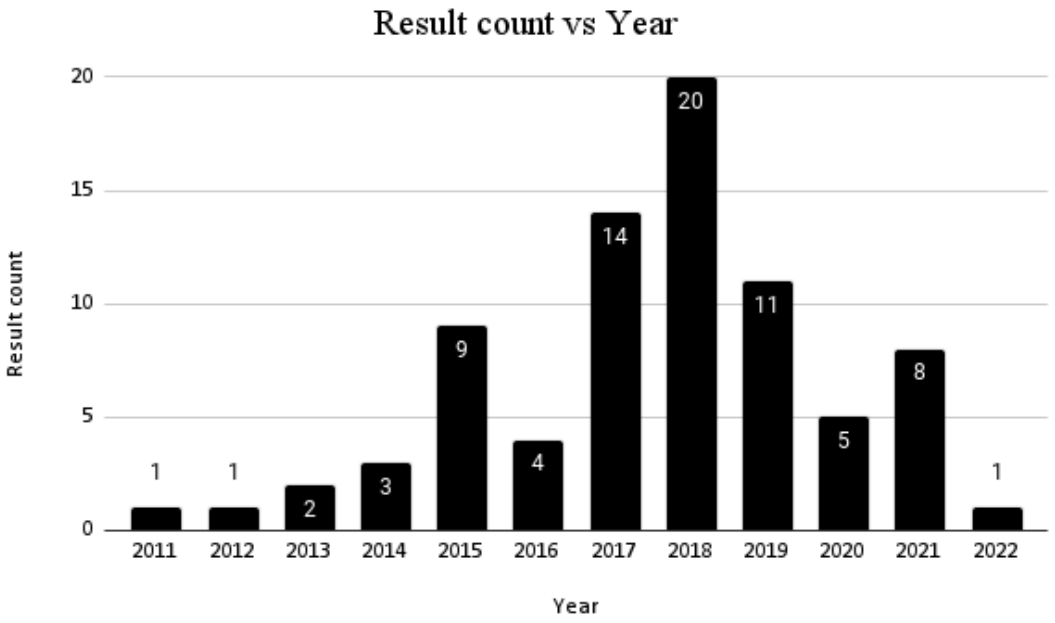


Figure 3.2: Paper distribution by year

The content analysis of each candidate paper was divided in two sub-steps. This was done not only to ease and systematise the review of the candidate papers, but also to speed up the entire process. These two sub-steps are described next-

3.3.1 Abstract

The first sub-step was meant to review only the abstract of each paper to check the written language and the type of study. Based on the goals of this step, the inclusion (hereafter labelled IC) and exclusion criteria (hereafter labelled EC) were defined as follows-

IC1 Papers written in English are included.

EC1 Papers related to systematic literature reviews are excluded

EC2 Papers related to extended abstract are excluded.

3.3.2 Content

The second sub-step required a complete review of each retained candidate paper after having applied the firs sub-step. The goal of this complete review was to determine whether the candidate paper's results were based on empirical evidence and connected with the goal of this study. The goal of this sub-step led to extend the IC and EC criteria to add the following conditions-

IC2 Papers related to empirical study are included. Empirical research papers are the studies for which the claimed results are supported by empirical evidence collected through studies on the field, or experiments performed in constrained environments. The main rationale behind selecting this type of studies was to look for insights about instrumentation of the metrics: i.e their operationalisation.

IC3 Papers deals with any one of the RQs are included. That means the paper includes information about one or more metrics used in the context of Continuous Delivery.

Criterion IC3 requires some clarifications about what were the assumptions to be considered by the reviewers when assessing it. A metric was considered as 'used" in a Continuous Delivery context if the candidate paper provided explicit information about (i) the activity that produces the required data to measure such metric, or (ii) the data required to compute the metric (which made possible to deduce the activity).

Therefore, to evaluate the IC3 criterion, it was required to know what are the activities required to be performed when practising Continuous Delivery. These activities correspond to any of those meant to accelerate the delivery of a product under development while ensuring its quality. In this study, it is assumed that these properties (speed and quality) are met by implementing a deployment pipeline. This assumption is supported by results reported in [21, 17].

In this manner, any activity that fits into those expected to be supported by a deployment pipeline it is considered as a valid activity

for the purposes of assessing IC3. To ease, clarify and systematise even more the decision of IC3, the types of activities that are assumed as those covered by a deployment pipeline is mentioned section 2.4.

The clarifications about the assumptions made about what it is a deployment pipeline in section 2.4, and what are the activities meant to be covered by its implementation helped reviewers when assessing the suitability of a candidate paper regarding IC3.

3.4 Data extraction

The objective of this step is to extract, from the retained papers, the data needed to address the research questions. To enhance the consistency of the data extraction among the reviewers a data extraction form was defined. The attributes of the form are listed next, along with their respective explanations and possible values:

- Metric ID: unique identifier of the retrieved metric. The identifier is made of the letter 'M" followed by an (incremental) integer value,
- Definition: a summary of the metric definition. The complete definition of the metric is provided by means of a narrative description. The definition of a metric should come along with a precise, and clear mathematical (symbolic) expression to assure consistent application and interpretation [19],
- Formula: mathematical (symbolic) expression used to calculate the value of the metric. The attribute holds the mathematical expression, if one is provided. Otherwise, it holds the label '×'
- Is the metric reliable or not?: it determines whether the primary study provides insights about consistency or repeatability of the measures. In other words, a measure is considered reliable if it gives the same result over and over again, assuming that what is measured is not changing [47]. This attribute holds the label YES to indicate that the primary study provides evidence about the reliability of the metric, otherwise the label is ×,

- Is metric operational or not?: it determines whether the primary study provides insights about how the metric can be implemented to collect the actual measurements (ideally as a series of operations or procedures. [47]). This attribute holds the label YES to indicate that the primary study provides the operations or procedures (along with evidence of its use). Otherwise the label is $\times$,
- What does the metric measure?: it is a summary of the property the metric is meant to assess,
- What is the required data to measure the metric?: it is a description of the data required to compute the metric,
- What is the activity that generates/collects the data?: it corresponds to an activity in the software development process that adheres to Continuous Delivery as explained in the previous section. The value of the attribute is the name of the activity.
- In which stage of the pipeline the metric can be placed?: it corresponds to one of the deployment pipeline stages as defined in the previous section. If the activity does not belong to any of these stages, then the attribute holds the label $\times$.

The shortlisted candidate papers were then carefully reviewed to extract the data meant to help answering the research questions. It is worth mentioning that the data extraction step was interleaved with the study selection. Both steps required several executions until the final conclusion was made: i.e. a paper that was initially excluded by a reviewer during the study selection step, was latter on included when doing the data extraction (or vice versa). That behaviour was mainly to the help of data extraction form that did help also as mechanism to drive the study selection.

3.5 Results

This section presents the results of having executed the methodology's steps as presented in Section 3.1.

Only empirical papers were selected for the content analysis as the empirical study relies on the collected data and is not based on any assumptions or theories. The method of data collection may involve qualitative or quantitative. So, depending on the empirical research provides a strong foundation for the content analysis and reduces the variability or bias resulting from this content analysis.

3.5.1 Result - Search strategy

The automated search in the ACM database corresponds to the first step. In this step, the filters F1, F2, F3, and F4 are applied in an accumulative manner. Table 3.1 shows the results of having applied these filters. Post applying these filters, 20 papers were selected for the second step: i.e. *Study selection.*

Step	Filters	Selected papers
1	**F1**	884
1	**F1+F2**	80
1	**F1+F2+F3**	58
1	**F1+F2+F3+F4**	20
2	**F1+F2+F3+F4+IC1**	20
2	**F1+F2+F3+F4+IC1+EC1**	19
2	**F1+F2+F3+F4+IC1+EC1+EC2**	17
2	**F1+F2+F3+F4+IC1+EC1+EC2+IC2**	13
2	**F1+F2+F3+F4+IC1+EC1+EC2+IC2+IC3**	3

Table 3.1: Research papers selected for each filter, inclusion and exclusion criteria

3.5.2 Result - Content analysis

The content analysis is performed in two sub-steps:

- Firstly, content was analysed based on the abstract of the paper. Papers were included and excluded based on the inclusion (IC1) and exclusion criteria (EC1, EC2) as mentioned in Section 3.4. Post applying inclusion and exclusion criteria, 13 papers were retained for the next sub-step.

- Secondly, content was analysed based on the entire content of the paper. Papers were included based on the inclusion criteria (IC1, IC2) as mentioned in Section 3.4. Post applying these criteria, 3 papers [2, 6, 33] were retained for the next step: i.e. *Data Extraction.*

Table 3.1 also shows the results of having applied the exclusion and inclusion criteria that led to the final corpus made of 3 papers.

3.5.3 Result - Data extraction

This section describes the information that has been extracted using the form and guidelines mentioned in Section 3.4. The list of retrieved metrics is shown in Table 3.2, whereas Table 3.3 and Table 3.5 provide the extracted data according to the defined form.

A narrative description of each metric is provided next. These descriptions represent a summary of the agreed understanding of the reviewers about the definition of the metric.

M1 - Percentage of effective time in the duration of all team members

$$PROG = max\{\frac{nT - \sum_{i=1}^{n}(SLACK_i + DELAY_i)}{nT}, 0\}$$

It measures project progress of each team. Given a team of n team members and project duration T, $PROG$ measures the percentage of effective time in the duration of all team members. Ineffective time is counted based on two parts,

Over-slack time ($SLACK$) is, the time of the period without any task assignments that has a length longer than expected Delay time ($DELAY$) is, the extra time of the tasks whose duration is beyond the scheduled time.

$SLACK_i$ is the over-slack time for a team member i and it is calculated as follows.

Metric ID	Metric Name	Reference
M1	Percentage of effective time in the duration of all team members	[2]
M2	Number of commits by individual	[2]
M3	Number of commits by team	[2]
M4	Merge your own code(MYOC)	[2]
M5	Early branching(EB)	[2]
M6	Merge-Often(MO)	[2]
M7	nloc	[6]
M8	added	[6]
M9	changed	[6]
M10	deleted	[6]
M11	change rate	[6]
M12	token count	[6]
M13	parameter count	[6]
M14	cyclomatic complexity	[6]
M15	effective cyclomatic complexity	[6]
M16	defect modifications	[6]
M17	defect density	[6]
M18	Production deployment	[33]
M19	Source code commits per day	[33]
M20	Number of defects carried over the next iteration	[6]
M21	List of open defects	[6]
M22	Software fix quality	[6]
M23	Functional coverage	[6]
M24	Number of planned/not planned/cancelled test cases	[6]
M25	Number of failed test cases	[6]
M26	Number of passed acceptance tests	[6]
M27	Mean time to failure (MTTF)	[6]
M28	Mean time between failure (MTBF)	[6]

Table 3.2: Retrieved metrics

$$SLACK_i = \sum_{j=1}^{m_i} Free_{i,j} \times f_{i,j}$$

where $\{Free_{i,j} | j = 1, ..., m_i\}$ is the set of time intervals that a team member i has no task assignments; $f_{i,j}$ is the tolerable factor of the slack time interval $Free_{i,j}$, defined as follows,

$$f_{i,j} = \begin{cases} 0 \ Free_{i,j} \leq T\_SLACK \\ 1 \ Free_{i,j} > T\_SLACK \end{cases}$$

Where T_SLACK is the threshold of the tolerable slack time.

Metric ID	Definition	Formula	Unit
M1	Given a team of n team members and project duration T, $PROG$ measures the percentage of effective time in the duration of all team members.	$PROG = max\{\frac{nT - \sum_{i=1}^{n}(SLACK_i + DELAY_i)}{nT}, 0\}$	%
M2	×	$COMMIT_i = \sum_{j=1}^{c_i} mod_{i,j} \times msg_{i,j} \times freq_{i,j}$	×
M3	×	$COMMIT\_TEAM = \frac{0.5 - \frac{1}{nC}\sqrt{\frac{\sum_{i=1}^{n}(COMMIT_i - \overline{C})^2}{n-1}}}{0.5}$	×
M4	×	$MYLOCI_i = \frac{M_i}{TM_i}$	×
M5	×	$EB = \frac{\overline{BR} \quad \overline{W}}{S}$	%
M6	×	$MO = \frac{\overline{BR} \quad \overline{MO}}{S}$	×
M7	Number of lines of code (nloc). This is calculated as the number of new-line characters per file (and function).	×	×
M8	nloc that have been added since previous version	×	×
M9	nloc that have been changed (not added or deleted) since previous version.	×	×
M10	The number of lines (nloc) that have been deleted from the previous version.	×	×
M11	The sum of added, changed, deleted since previous version.	×	×
M12	The number of white space delimited words in each file-function	×	×
M13	The number of parameters a function takes.	×	×
M14	Measures the number of linearly independent paths through a program's source code.	×	×
M15	Calculated by the sum of all the functions with cyclomatic complexity greater than 15 divided by the number of functions.	×	×
M16	The number of identified modifications for each defect per function.	×	×
M17	Number of defects per lines of code	×	×
M18	How often is code pushed to production for end user feedback	×	×

Table 3.3: Data extraction - Part a

Metric ID	Proven Reliability	Operational	What does it measure?
M1	×	×	Project progress of each team
M2	×	×	Commit Quality
M3	×	×	Commit Quality
M4	×	×	Activeness of developer in collaborative development
M5	×	×	Correlations among the commits in a branch
M6	×	×	How often branches in the iteration needs to be merged to the master branch
M7	×	×	×
M8	×	×	×
M9	×	×	×
M10	×	×	×
M11	×	×	×
M12	×	×	×
M13	×	×	×
M14	×	×	×
M15	×	×	×
M16	×	×	×
M17	×	×	×
M18	×	×	Efficiency of continuous delivery

Table 3.4: Data extraction - Part b

Metric ID	What is the required data to measure it?	Activity	Deployment Pipeline Stage
M1	Data from ticket management system	Task planning	×
M2	Commit details from version control repository	Compile	Commit
M3	Commit details from version control repository	Compile	Commit
M4	Merge details from the version control repository	Compile	Commit
M5	Branch details from the version control repository	Compile	Commit
M6	Merge details from the version control repository	Compile	Commit
M7	Source code from version control repository	Code analysis	Commit
M8	Source code from version control repository	Code analysis	Commit
M9	Source code from version control repository	Code analysis	Commit
M10	Source code from version control repository	Code analysis	Commit
M11	Source code from version control repository	Code analysis	Commit
M12	Source code from version control repository	Code analysis	Commit
M13	Source code from version control repository	Code analysis	Commit
M14	Source code from version control repository	Code analysis	Commit
M15	Source code from version control repository	Code analysis	Commit
M16	×	×	×
M17	×	×	×
M18	×	Release	Release

Table 3.5: Data extraction - Part b

Suppose$\{TASK_{i,j}|k = 1, ..., s_i \}$ is the tasks assigned to a team memeber i in the project duration, $TASK\_C_{i,k}$ is the sheduled time of a task $TASK\_S_{i,k}$, the total delay time for i, $DELAY_i$, is calculated as follows.

$$DELAY_i = \sum_{k=1}^{s_i} max\{TASK\_C_{i,k} - TASK\_S_{i,k}, 0\}$$

M2 - Number of commits by individual

Suppose $\{cm_{i,j}|j = 1, ...c_i\}$ is the set of commits by a team member i, each commit, $cm_{i,j}$, is evaluated by the three factors:

- $mod_{i,j}$ is the reasonable modification size of the each commit. A threshold is defined to avoid over-commit. A commit is reasonable only if its size is within the defined threshold.
- $msg_{i,j}$ is the reasonable message length of a commit. A commit message is reasonable only if message length is higher than the defined threshold.
- $freq_{i,j}$, is the reasonable frequency of commits. The intervals between two consecutive commits are restricted by upper-bound and lower-bound threshold.

The metric is calculated as follows.

$$COMMIT_i = \sum_{j=1}^{c_i} mod_{i,j} \times msg_{i,j} \times freq_{i,j} \tag{3.1}$$

where,

$$mod_{i,j} = \begin{cases} 1 & MLN_{i,j} \leq T\_MOD \\ 0 & MLN_{i,j} > T\_MOD \end{cases}$$

$MLN_{i,j}$, is the size of the j^{th} commit of a team member i in term of number of added/deleted lines of code, and T_MOD is the threshold of reasonable size of a commit.

$$mod_{i,j} = \begin{cases} 0 & MsgLN_{i,j} < T\_MSG \\ 1 & MsgLN_{i,j} \geq T\_MSG \end{cases}$$

Where, $MsgLN_{i,j}$ is the length of the message of the j^{th} commit of team member i, and T_MSG is the threshold of reasonable length of a commit message.

$$freq_{i,j} = \begin{cases} 0 & INTV_{i,j} < L\_FQ \\ 1 & L\_FQ \leq INTV_{i,j} < U\_FQ \\ \frac{E\_FQ - INTV_{i,j}}{E\_FQ - U\_FQ} & U\_FQ \leq INTV_{i,j} < E\_FQ \\ 0 & INTV_{i,j} \geq E\_FQ \end{cases}$$

Where, $INTV_{i,j}$ is the time interval of the j^{th} commit of the team member i, L_FQ and U_FQ are the lower and upper bound of acceptable time respectively, and E_FQ is the extended interval with progressively decreased one.

M3 - Number of commits by team

The metric is calculated as follows.

$$COMMIT\_TEAM = \frac{0.5 - \frac{1}{nC}\sqrt{\frac{\sum_{i=1}^{n}(COMMIT_i - \overline{C})^2}{n-1}}}{0.5}$$

where

$$\overline{C} = \frac{1}{n}\sum_{i=1}^{n} COMMIT_i$$

M4 - Merge your own code (MYOC)

One can only merge his/her own code with another branch to guarantee the familiarity with what do be done. This metric is calculated as follows.

$$MYLOCI_i = \frac{M_i}{TM_i}$$

Where, TM_i is the number of merges of his/her branches of a team member i in a team and M_i is the number of merges by a team member on his/her branches.

M5 - Early branching (EB)

It evaluates the correlations among the commits in a branch. This metric is calculated as follows.

$$EB = \frac{BR\_W}{BR}$$

where, BR is the number of all branches in a given sprint period and BR_W is the number of well-formed branches which conform to the early branching guideline.

M6 - Merge often (MO)

Given a sprint iteration, all the branches in the iteration need to be merged to the master branch. Given a project with S sprint iterations, suppose the number of sprints that follows the merge often (MO) guideline is S_MO, the conformation to the guideline is defined as follows:

$$MO = \frac{S\_MO}{S}$$

M7 - Nloc

This is calculated as the number of new-line characters per file (and function). It calculates the total number of lines in a file including the empty lines and commented code.

M8 - Nloc added

It is the nloc that have been added since previous version. It calculates the total number of lines that are newly added and which are not present in the previous version of the file.

M9 - Nloc changed

It is the nloc that have been changed (not added or deleted) since previous version.

M10 - Nloc deleted

It is the nloc that have been deleted from the previous version.

Metric ID	Metric Name	Reference	Activity	Deployment Pipeline Stage
M19	Source code commits per day	[33]	Compile	Commit
M20	Number of defects carried over the next iteration	[6]	×	×
M21	List of open defects	[6]	×	×
M22	Software fix quality	[6]	×	×
M23	Functional coverage	[6]	×	×
M24	Number of planned/not planned/cancelled test cases	[6]	×	×
M25	Number of failed test cases	[6]	Testing	Test
M26	Number of passed acceptance tests	[6]	UA testing	Test
M27	Mean time to failure (MTTF)	[6]	Product Monitoring	×
M28	Mean time between failure (MTBF)	[6]	Product Monitoring	×

Table 3.6: Under-defined metrics and extracted data.

M11 - Change rate

The sum of added, changed, deleted since previous version. It is the ratio of sum of nloc that are added, changed, deleted since the previous version over the nloc of previous version.

M12 - Token count

The number of white space delimited words in each file-function. It calculates the number of the strings which are composed of only of spaces, tabs or line breaks.

M13 - Parameter count

The number of parameters a function takes. It calculates the total number of arguments a function takes.

M14 - Cyclomatic complexity

It measures the number of linearly independent paths through a program's source code.

M15 - Effective cyclomatic complexity

It is calculated by the sum of all the functions with cyclomatic complexity greater than 15 divided by the number of functions.

M16 - Defect modifications

The number of identified modifications for each defect per function.

M17 - Defect density

Number of defects per lines of code. It calculates the number of defects found in the software product per size of the code.

M18 - Production deployment

How often is code pushed to production for end user feedback.

M19 to M28 - Under-defined metrics

There are 10 metrics for which the the primary research articles do not contain any information related to the elements that conform the extraction form. This is the reason why these metrics do not appear in Table 3.3 nor Table 3.5, but a specific table (see Table 3.6) was made to show the values obtained as result of the discussion among the reviewers about the the possibility to measured these metrics during the execution of the deployment pipeline.

Metric ID	Metric Name
M2	Number of commits by individual
M3	Number of commits by team
M4	Merge your own code(MYOC)
M5	Early branching(EB)
M6	Merge-Often(MO)
M7	Nloc
M8	Nloc added
M9	Nloc changed
M10	Nloc deleted
M11	Change rate
M12	Token count
M13	Parameter count
M14	Cyclomatic complexity
M15	Effective cyclomatic complexity
M18	Production deployment
M19	Source code commits per day
M25	Number of failed test cases
M26	Number of passed acceptance tests

Table 3.7: Metrics that can measured using data generated by deployment pipeline's activities.

3.6 Discussion

This section analyses the achieved results regarding the research questions and, to what extent, such results allow these questions to be answered.

The RR allowed to discover 3 peer-reviewed primary studies to answer RQ1. A total of 28 metrics related to continuous delivery were identified from the primary studies that made into the final corpus. The analysis of the extracted data allowed to identify 18 metrics related to continuous delivery: i.e. these metrics

can be measured on data generated by the execution of activities enclosed in a deployment pipeline. These metrics are listed in Table 3.7.

The paper [2] provides mathematical expression to calculate the metrics. However, it fails to provide a precise definition of metrics and how the metrics are measured operationally. The paper [6] provides definition for few of the metrics. However, it does not provide a mathematical expression for any of the metrics and has no information regarding the operationalisation of the metrics. The paper [33] also does not provide any formula nor operationalisation of metrics.

None of the primary studies that made it into the final corpus has given details on how the metrics can be operationally measured. In other words, primary studies do not provide any information on the steps taken to measure the metrics. Hence, no results were found to answer the RQ2.

This negative result led this thesis to study on how each of these metrics could be implemented using the components on which the implementation of a deployment pipeline may rely on: e.g. version control repository, build manager, package manager, and continuous integration server. The results of this thesis provides the evidence to answer the RQ2 which is given in chapter 4.

This SLR also allowed to address the RQ2. The state-of-the-art metrics found in this SLR can be categorised according to the asset each of them allow to assess. These assets are: (i) developers, and (ii) the product under development. A synthesis about what are the properties the found metrics allow to observe grouped by asset is presented next.

Developers

There are metrics that allow to assess the **performance** of an individual developer. The performance is understood as the time the developer requires to complete certain activity (i.e. code requirements). A way to assess his/her performance is by looking at the number of commits he/she does over certain period of time. Metric **M2** allows to observe such property over a particular developer, whereas metric **M3** does it for a group of developers.

The performance of an individual developer may also be observed by means of the metric **M4** as it measures the activeness of a developer based on the number of merges he/she does for the own produced code. A high number of merges may be see as positive as it shows evidence of integrating the changes on the main branch from where other developers obtain the latest version of the base code.

The performance of a development team can be observed by means of metrics **M5** and **M6**. For both metrics, closer to '1' is the measured value, the better. This represents a higher correlation, which can be understood as good

management of the branches (i.e. capacity to create multiple branches to speed up parallel development, along with their merge into the main branch).

The assessment of the process that leads to release the product can be made through the metrics **M18** and **M19**. The frequency in releasing the product to its production environment (metric M8) is a way to assess the level of adherence to Continuous Delivery by the developers in charge of the product. Thus, this metric can be used as proxy to assess the **software delivery performance** of the developers responsible of a product. It must be noticed that metric M18 is one of the Accelerate's metrics [16] proposed as proxy to assess the same property: i.e. software delivery performance.

The same property can be assessed by metric M19, as the number of (correct) commits made by developers on the same product are strongly connected with the release frequency. A high commit frequency is a necessary condition (but not sufficient) to reach a high release frequency. Nevertheless, it is worth noting that a development team may have a high commit frequency, but still release the product to a lower rate. In that case, it is said that the team practices "Continuous Deployment" (rather "Delivery").

The product under development

Metrics based on source code are used to assess the **complexity** of the software product. Knowing the complexity also allows to predict required efforts to **maintain** the product. Metrics of this kind have been widely explored and analysed by the research community [1, 18, 31, 40]. Metrics **M7 to M15** fall into this category.

For teams practising continuous delivery, the means to ensure **quality** is through testing. Recall that quality is defined as the level of adherence of the delivered product to the end-users' expectations. In this regard, the metrics **M25** and **M26** allow to assess the quality of the product. Whereas a high value of **M25** may indicate a low quality of the work committed by developers, this has to be double-check in the case that test are automatically executed (as an automated test failing to pass may be due to its lack of maintenance). Yet another valid observation is related to automated acceptance tests (metric **M26**): higher the number of passed acceptance tests shows the product is meeting the users requirement. If the count is lower, it shows that the testing done prior to user acceptance testing is not handling enough test cases.

Chapter 4

Instrumentation - Data collection

This chapter describes the hardware and software used for running the experiment. There is a detailed explanation of the product used and the pipeline created to instrument the metrics.

4.1 Hardware

The hardware used for the experiment is listed in the table 4.1

Name	**Specification**
Computer	Mac mini(2018)
Processor	3.2 GHz Intel Core i7
RAM Memory	64GB 2667 MHz DDR4

Table 4.1: Hardware technical specifications

4.2 Software

The list of software used for experiment is explain in this section.

4.2.1 Virtual box

VirtualBox[1] is a powerful x86 and AMD64/Intel64 virtualisation solution. It is a open-source software. Additional operating systems, known as Guest OS's, can be installed on VirtualBox and run in a virtual environment. We have used virtualbox to create multiple virtual machines which are used in the experiment. The virtualbox is installed on the local machine(Mac mini) and instructions for the installation is available on the URL `https://www.virtualbox.org/wiki/Downloads`.

4.2.2 Vagrant

Vagrant[2] is used to automate the creation of virtual machines used in the experimental setup. Vagrant is a tool that allows you to create and manage virtual machine environments all in a single process. Vagrant reduces setup time for development environments and improves production parity with an intuitive approach and an emphasis on automation. It provides easy-to-configure, reproducible, and portable work environments built on industry-standard technology and driven by a single, standardized workflow to help individuals and teams maximize productivity and flexibility.

Instrumentation required for Vagrant

Vagrant is installed on the local machine. The instructions for installation of Vagrant is available in the URL `https://www.vagrantup.com/downloads`. The important step in creating virtual machines in Vagrant is to create a vagrant file. Vagrant file should consists of below mentioned list of things.

- config.vm.box - This determines which machine will be brought up against.
- config.vm.hostname - This is where the machine's hostname should be stored.
- config.vm.network - This one configures the machine's networks.
- config.vm.synced_folder - Configures synced folders on the machine so that folders on the host machine can be synced to and from the guest machine.
- config.vm.provider - Provider-specific configuration is used to modify provider-specific settings.
- memory - It configures the memory of the virtual box
- cpu - It configures the CPU of the virtual box

[1]https://www.virtualbox.org/
[2]https://www.vagrantup.com/

- config.vm.provision - Configures provisioners on the machine so that software can be installed and configured automatically when the machine is created.

Once the vagrant file is ready, the virtual machines can be created by running the vagrant commands on the command-line interface. Few of the important vagrant commands[3] used in the experiment is listed below.

- vagrant up [name|id] - This command is used for createing and configureing guest machines based on the configuration present in Vagrantfile.
- vagrant ssh [name|id] - This will create a SSH(Secure Socket Shell) connection to running Vagrant machine and provides access to a shell.
- vagrant reload [name|id] - Running a halt followed by an up. This command is usually required for changes made in the Vagrantfile to take effect. Following any changes to the Vagrantfile, a reload should be performed.

The values mentioned within the square brackets i.e., [name|id] are optional. *name* is name of machine defined in Vagrantfile and *id* is the machine id, which could be found with command *vagrant global-status*. Using *id* allows to call command *vagrant up id* from any directory.

4.2.3 Ansible

Ansible is a Red Hat-sponsored open source community project, it provides a simplest way to automate the IT(Information Technology). It is the most popular DevOps tool for controlling, orchestrating, and automating IT infrastructure. Ansible is mainly used in this experiment to automate the installation and configuration of software inside the virtual machines. To install Ansible, requires a nearly any UNIX-like machine with Python 3.8 or newer installed. Steps to install ansible is available at `https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html`.

Ansible operates by establishing connections with nodes and distributing tiny programs known as ansible modules. Then, by default, it runs these modules over SSH, and after they're finished, it removes them. The main components in ansible scripts are modules, playbooks and roles [32].

- Playbook, which is a YAML file that contains the order of the commands, is made up of the modules. A sample ansible playbook is given below.

[3]https://www.vagrantup.com/docs/cli

```
- hosts: all
  remote_user: vagrant
  become: yes
  become_method: sudo

  vars:
    vHome: /home/vagrant
    script_path: /vagrant_scripts
    metrics_python_file: get_metrics.py

  tasks:
    - include_role:
        name: mysql
```

- Roles are complex structures that include tasks, modules, handlers and files. Role is made up of a directory with sub-directories that each contain a main.yml file that details the order in which operations should be carried out. A sample ansible role is given below to install MySQL database.

```
../playbook/roles/mysql/tasks/main.yml

- name: Installing Mysql  and dependencies
  package:
  name: "{{item}}"
    state: present
    update_cache: yes
  loop:
    - mysql-server
    - mysql-client
    - python3-mysqldb
    - libmysqlclient-dev
  become: yes
```

- Modules are short programs designed to handle basic system operations. They can be executed as a standalone command or as a component of playbooks, which are more intricate scripts. Below is an example of ansible module that reboots a machine and waits for it to shutdown, restart, and respond to commands.

```
    - name: Unconditionally reboot the machine with all defaults
      reboot:
        reboot_timeout: 600
```

4.2.4 Gitlab

Gitlab[4] is an open source web-based version control system which is based on git [41]. It provides a git repository with version control and collaboration option which helps development team collaborate and maximize productivity, sparking faster delivery and increased visibility. It also provides features such as CI/CD pipelines. The reason for selecting Gitlab is, it is open source and a DevOps platform which provides source code management, continuous integration and deployment.

Instrumentation required for GitLab

The installation and configuration of the GitLab is done automatically using ansible playbook. After successful installation of GitLab, it can be accessed with the URL http://<hostname>/gitlab. The *hostname* used in this experiment is 192.168.56.15. After successfully installing GitLab, a new project is created inside GitLab, and product used for the experiment is placed in this project. The most important in setting up CI/CD pipeline is creating GitLab Runner.

GitLab Runner[5] is an application that works with GitLab CI/CD to run jobs in a pipeline. In this experiment, runners are installed inside the virtual machines of each environment except the developer environment. After the installation of runners, the next step is to setting up communication between GitLab instance and the machine where GitLab Runner is installed by registering the runner in Gitlab instance. While registering the runner, a executor should be selected based on the requirement. An executor is the one which determines the environment in which each jobs run.

An access token is created in GitLab. This token is used to retrieve the metric data from the GitLab while CI/CD pipeline is running. Metric values are retrieved using a python script which does a API GET call to GitLab using the access token. The API for getting the individual *commits* details from a project is given below. The response of the API call will be in a json format.

```
http://192.168.56.15/gitlab/api/v4/projects/2/repository/commits/master
```

Where, *http://192.168.56.15/gitlab/* is the URL of the GitLab, *2* is the project ID, *repository/commits* is repository of commits and *master* is the name of the branch.

[4]https://about.gitlab.com/
[5]https://docs.gitlab.com/runner/

4.2.5 Sonarqube

To measure the source code metrics, we use the one of the most popular free, open-source code analysis tools called SonarQube[6]. SonarQube can be downloaded and run on a local server or it can be accessed as a service from the sonarcloud.io platform. SonarQube provides reports on metrics such as code coverage, unit tests, code complexity, and number of lines of code for around 29 programming languages (for example, Java, C#, JavaScript, Python, etc.,). The integration of SonarQube with GitLab Self-Managed and GitLab.com allows for the maintenance of code quality and security in GitLab projects. Analysis of design, architecture, and object-oriented metrics for Java programs are supported by Sonar, which interfaces seamlessly with build automation tools like Maven, Gradle and Ant[26].

Instrumentation required for Sonarqube

The installation of the Sonarqube is done automatically using ansible playbook. The sonar server starts in port 9000; It could be accessed on the local machine thorugh the URL `http://127.0.0.1:9000/`. We can also access the Sonarqube outside the local machine by using the host name of the machine. In this experiment it is accessed using the URL `http://192.168.56.15:9000/`. The project used in this study is based on Maven, the source code analysis of the project is done by executing the following command.

```
- mvn -f welcome-webapplication/pom.xml --batch-mode verify
--fail-never sonar:sonar -Dsonar.host.url=$SONAR_URL
-Dsonar.login=$SONAR_USER -Dsonar.password=$SONAR_PASSWORD
```

where, SONAR_URL is the URL of the Sonarqube, SONAR_USER is the username of the Sonarqube and SONAR_PASSWORD is the password of the Sonarqube user. These values are stored as variable in the Gitlab.

Once the analysis is completed the source code metrics are stored in the Sonarqube and these metrics can be accessed through the API using authentication token. To get the metric's data from Sonarqube, we can use the measurement fetching REST GET API with URL `http://192.168.56.15:9000/api/measures/component?component=com.harish:welcome-webapplication&metricKeys=ncloc`. The component name in this API call is the project key i.e., *com.harish:welcome-webapplication* and *metricKeys*[7] is the standard name of the metric which needs to retrieved from the sonarqube. The metricsKey parameter in the measurement fetching API is the value of the "key" field in the metric listing API JSON response. The list of standard metric keys used in this experiment is listed in the table 4.2.

[6]https://www.sonarqube.org/

[7]https://docs.sonarqube.org/latest/user-guide/metric-definitions/

Python code is written to automate the metric collection for the source code metrics via API, and these values are documented in section 4.5.

MetricKeys	Name	Description
ncloc	Lines of code	Number of physical lines containing at least one character that is not a whitespace, a tabulation, or part of a comment
complexity	Complexity	It is the Cyclomatic Complexity, which is determined by the number of paths through the code.
tests	Unit tests	Number of unit tests
test_failures	Unit test failures	The number of unit tests that failed due to an unexpected exception.

Table 4.2: Sonarqube metricKeys

4.2.6 MySQL

MySQL[8] is a widely used relational database management system (RDBMS). It is free and open-source and is ideal for both small and large applications. MySQL is used in this study to store all the metrics data collected from Gitlab and Sonarqube applications.

Instrumentation required for MySQL

The installation of the MySQL is done automatically using ansible playbook. The MySQL server starts in port 3306 by default. MySQL is connected using a Python code as given below. It requires a package called mysql.connector to work with Python code.

```
mysql_connnection = mysql.connector.connect(
                      host='localhost',
                      database='devops',
                      user='devops',
                      password='devops@2022')
```

A table with the name *metrics* is created with all the required columns with appropriate column type to store all the metric values in one place. The table also contains additional columns such as commit_id, commiter_name, and committed_date which is, ID or URL-encoded path of the project owned by the

[8]https://www.mysql.com/

authenticated user, name of the committer and date-time of the commit took place respectively. The structure of the database table is given as below.

```
CREATE TABLE  IF NOT EXISTS metrics(
                    commit_id varchar(100),
                    commiter_name varchar(100),
                    committed_date datetime,
                    nloc integer(100),
                    nloc_added integer(100),
                    nloc_deleted integer(100),
                    change_rate DOUBLE,
                    cyclomatic_complexity integer(100),
                    test_case integer(100),
                    failed_test_case integer(100),
                    acceptance_test_case integer(100),
                    failed_acceptance_test_case integer(100),
                    production_deployment datetime
                );
```

A python code is written to automate the insertion of collected metric data into the MySQL database table which is explained in section 4.5.

4.2.7 Grafana

We use Grafana as a tool for visualising the metrics. Grafana[9] is an open-source data analysis visualisation tool that allows visualisation, querying, alerting, and exploration of metrics. Grafana is one of the efficient tools which visualise the live-streamed data directly into the dashboard with accurate results.

The reason for selecting the Grafana as the visualisation tool, it is free, open-source, web-based and also it offers wide variety of data sources such as time series databases, logging and document databases, distributed tracing, SQL, Cloud and enterprise plugins[9]. One of the advantage of using Grafana is that the dashboards can shared with anyone. Dashboards can be exported in a JSON file and this file can be imported for re-using the dashboards. The comparison of visualisation tools is given in 5.3. There are multiple tools are available for visualisation but we choose Grafana as it is very simple to use, provides variety of charts and also has capability of extracting data from multiple sources.

[9]https://grafana.com/

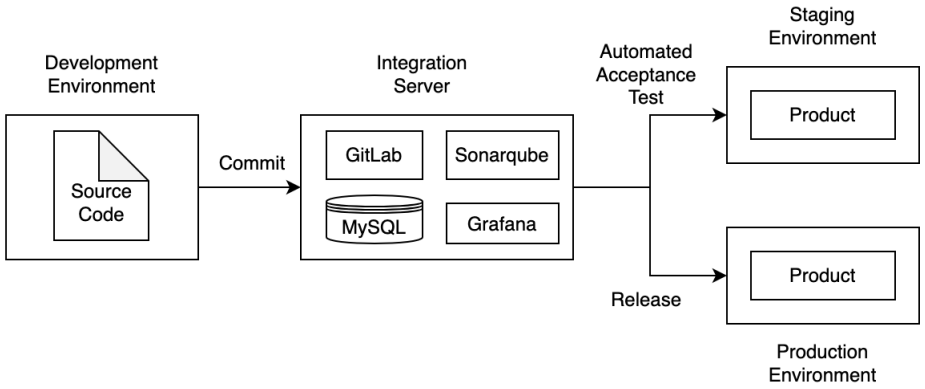


Figure 4.1: Environments

Instrumentation required for Grafana

The installation of the Grafana is done automatically using ansible playbook. The Grafana server starts in port 3000 by default. The main thing to setup in Grafana data sources from where the data is extracted. In this experiment, MySQL is used as the data source. The basic details required to connect MySQL database is given below.

```
Host: localhost:3306
Database: devops
User: devops
Password: Devops@2022
```

Once the data source is setup, next step is to create dashboards. It is done by creating dashboard and adding a panel to display the data. In panel, need to select the data source from where the data to be pulled and select the type of chart to display the data.

4.2.8 Environments

To run the experiments as part of the study, four different environments are created as shown in the figure 4.1. All the environments are created as virtual environments using the Vagrant file.

Development environment

The development environment is created with a “ubuntu” virtual machine(VM) box with virtual box provider as “virtualbox” with a 1024MB of memory and 4 CPU’s. This environment is mainly used for developing the source code of

the product. The details on the product used in this study is explained in the section4.3. Developers will make their changes to source code in the development environment and check in into version control system (VCS). In this study, Gitlab is used as the version control system.

Integration server

The integration server is created with a "ubuntu" virtual machine box with virtual box provider as "virtualbox" with 8192MB of memory and 4 CPU's. This environment is created using a virtual machine. The environment is used as a continuous integration server where Gitlab is used as VCS and CI, whereas docker is used to handle the integration environments. The compile of the code, running a set of commit tests, creating binaries for use by later stage takes place in this server. Sonarqube, MySQL and Grafana applications are also installed in this server.

Two GitLab runners are installed in integration server and registered with GitLab. One runner with *docker* as the executor with the *alpine:latest* as the docker image. When used with GitLab CI, docker executor[10] establishes a connection to docker engine and performs each build in a distinct container using the predefined image that is configured in. gitlab-ci.yml file. This runner is responsible for building and testing the product and also export metrics related to the product into Sonarqube.

Shell works as the executor for the second runner. This runner is used to executing a python script that will gather metrics data from SonarQube and GitLab applications and insert into a MySQL database table.

The interaction of the applications inside the integration server is depicted in the figure 4.2. GitLab is a CI/CD server which triggers the pipeline when a commit is done by developers. Some metrics are generated by the pipeline and few of the metrics are pused into Sonarqube service. A python code is developed to gather the metrics data from GitLab and Sonarqube applications via API call over HTTP protocol by using the tokens of GitLaba and Sonarqube. All the gathered metrics data are finally inserted into MySQL database table. Grafana connects to the MySQL database, queries the table for metrics data, and displays the information via dashboards.

Staging environment

The staging environment is created with a "ubuntu" virtual machine box with virtual box provider as "virtualbox" with 1024MB of memory and 4 CPU's. This environment is similar to the production environment and is primarily used for running automated acceptance tests.

[10]https://docs.gitlab.com/runner/install/docker.html

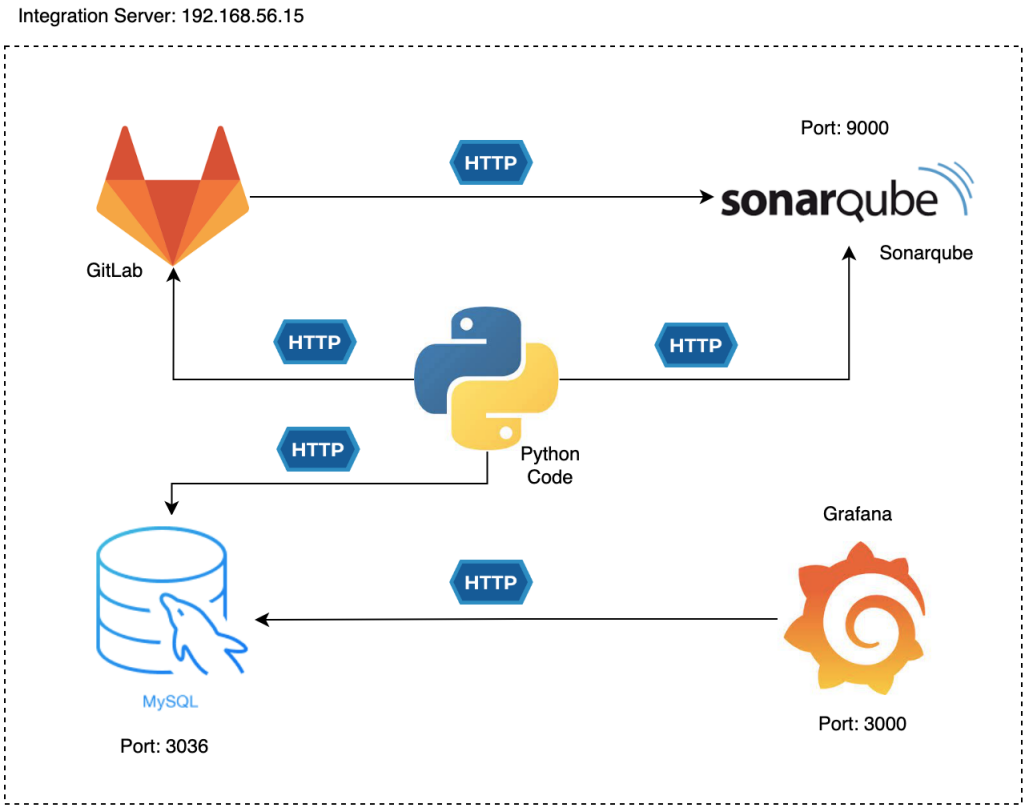


Figure 4.2: Interaction between applications in Integration server

In staging environment also two runners are installed and registered with GitLab. One runner with *docker* as the executor with the *alpine:latest* as the docker image. This runner is used for running automated acceptance tests and also used for running commands to export the metric data related to acceptance tests into Sonarqube application.

Second runner is created with a *shell* as the executor. This runner is used for running a python script that will gather acceptance test cases metrics data from SonarQube application and inserts into a MySQL database table.

Production environment

The production environment is created with a "ubuntu" virtual machine box with virtual box provider as "virtualbox" with 1024MB of memory and 4 CPU's. The final product is released into this environment. A single runner with *shell* executor is installed on this environment. The runner is used for copying the product from the integration server to the production environment.

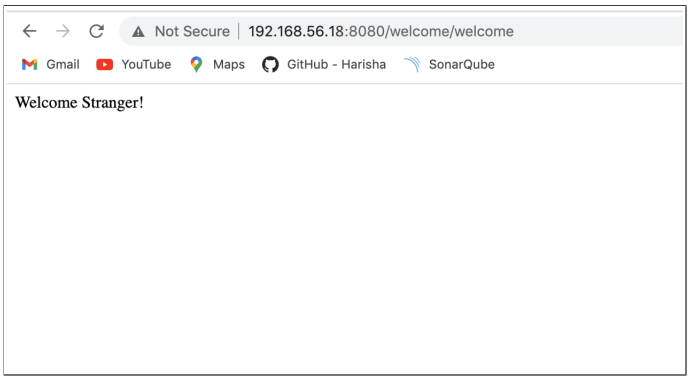


Figure 4.3: The result post running the product

4.3 The Product

In this section the product used in the experiment is discussed in detail. The product used in the experiment is a simple web-based java project using Spring Boot[11] framework. Apache Maven[12] is used as the automated build tool.

Especially for Java projects, Apache Maven is a well-liked build automation tool. Two facets of software development are covered by Maven. A software's dependencies are described in addition to how it is built. When the application is launched, Spring Boot will automatically identify the project's Spring MVC controller and launch an embedded Apache Tomcat instance. Apache Tomcat uses port 8080 by default.

When the product is run, a web page with the text "Welcome Stranger!" is opened in a browser. This web page should be accessible via the URL `http://<localhost>/welcome/welcome` to developer. When the product is deployed in the production environment, the web page should be accessible at URL `http://192.168.56.18/welcome/welcome` as shown in figure 4.3.

The test cases are written for the verification and validation of the product: namely, Unit, Integration and Acceptance test cases. Whereas unit and integration test cases are executed for the verification of the product i.e. A test of a product to prove that it meets all its specified requirements. The unit test cases are executed by the developer at their respective development machine. In deployment pipeline unit and integration are executed in the *Commit* stage. Acceptance test cases are used for the validation of the product from the end user i.e. A test that ensures that an end product stakeholder's true needs and expectations are met. Acceptance test cases are executed in the *Test* stage of the deployment pipeline. The detailed information on the test cases are given below.

[11]https://spring.io/projects/spring-boot
[12]https://maven.apache.org/

4.3.1 Unit test cases

These are first level of test cases which are usually performed by the developers on their local machines[13]. However, these test cases executed in the continuous delivery at commit stage of the deployment pipeline. Unit tests are developed to test the functionality of minor components of the program in isolation (say, a method, or a function, or the interactions between a small group of them)[22]. The unit test cases used in the product are written using JUnit[13], which is a unit testing framework for Java programming language.

4.3.2 Integration test cases

Integration test cases are used for testing each independent component of the application to make sure it functions properly with the services it depends on[23]. The integration test cases are written using Mockito[14], which is an open source testing framework[34].

4.3.3 Acceptance test cases

The purpose of an acceptance test case is to ensure that the acceptance criteria of a story or requirement have been met[23]. We have created the automated acceptance test cases using the TestNG[15] framework which is test automation framework for Java.

The product can be run using simple maven commands as given below.

- *mvn clean install* - It takes the Java source code, compiles it, tests it and converts it into an executable WAR(web application archive) file.
- *mvn clean compile tomcat7:run* - This command compiles web application and starts Tomcat container with application deployed under root of the server. After running the command, our product will be available on the URL `http://localhost:8080/welcome/welcome`

4.4 GitLab Pipeline

Pipelines are the highest level of integration, delivery, and deployment. GitLab pipelines are made up of the following components:

[13]https://junit.org/junit4/
[14]https://site.mockito.org/
[15]https://testng.org/doc/index.html

- Jobs define what one does. Jobs that compile or test code, for example. Runners executes the job. Multiple jobs in the same stage are executed in parallel if there are enough concurrent runners. Note that *activity* is referred to as a *job* in GitLab.
- Stages define when to run the jobs. For example, stages that run tests after stages that compile the code.

If any of the jobs fails in a stage, the pipeline will be stopped and next stages are not executed. The pipeline will run and complete successfully only if all the jobs in each stage is completed successfully. Though the pipelines can created in many different ways[16], we have created a simplest pipeline in GitLab i.e., Basic pipeline which executes everything in a stage concurrently, followed by one stage after the other.

In GitLab, to configure the CI/CD pipeline, a *.gitlab-ci.yml*[17] should be created which is of YAML(Yet Another Markup Language)[18] file type. This file contains details regarding the order in which the jobs should execute by the runners and the decisions that the runner should make when certain conditions are met.

4.4.1 Pipeline instance

Figure 4.4 shows the deployment pipeline used in this study. When developers commit changes to the version control system, the process begins. In response to the commit, the continuous integration management system starts a new instance of the pipeline. The first (commit) stage of the pipeline compiles the code, runs unit and integration tests, performs code analysis, and creates binaries. Executable code binaries is created and saved them in an artifact repository if the unit and integration tests are all successful and the code is valid. The second stage is test stage which executes the automated acceptance tests on the binaries which was created during the commit stage. If all the acceptance tests are passed, next stage is triggered i.e. release stage. In release stage, the binaries are transferred into production environment. The detailed information regarding each stage is explained in the below.

In our experiment a *.gitlab-ci.yml* file is created to configure the CI/CD pipeline which consists of 10 stages as given below. Each stage consist of one or more jobs in it. In GitLab by default the pipeline will be triggered when a commit occurs.

```
stages:
    - sonar_build
```

[16]https://docs.gitlab.com/ee/ci/pipelines/
[17]https://docs.gitlab.com/ee/ci/quick_start/
[18]https://en.wikipedia.org/wiki/YAML

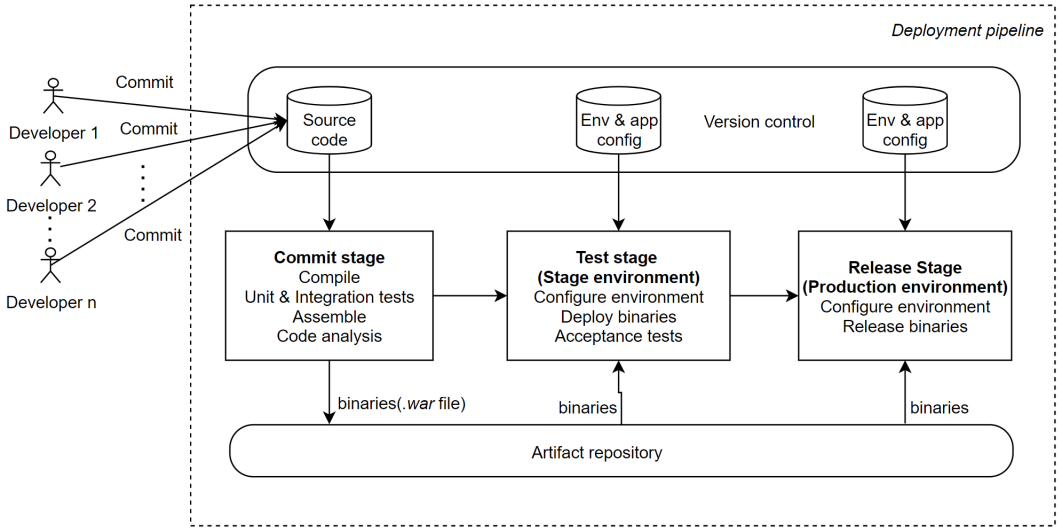


Figure 4.4: Deployment pipeline

```
- metrics_build
- build
- upload
- deploy
- sonar_test
- metrics_test
- test
- release
- metrics_release
```

Note that in this pipeline instance,

- stages sonar_build, metrics_build, build and deploy are part of the commit stage,
- sonar_test, metrics_test, test are part of test stage and
- release and metrics_release are part of release stage of the deployment pipeline.

The details of each stage in the pipeline instance is given below.

sonar_build

This stage includes a job that runs in a Docker executor on the integration server. This stage basically performs the code analysis, unit and integration tests activity. This job contains a script that uses the maven command to push the source code and test case metrics into the Sonarqube application.

```
- mvn -f welcome-webapplication/pom.xml --batch-mode verify
--fail-never sonar:sonar -Dsonar.host.url=$SONAR_URL
-Dsonar.login=$SONAR_USER -Dsonar.password=$SONAR_PASSWORD
```

metrics_build

It contains a job which runs on a shell executor inside the integration server. A python script is executed to extract the metrics data from the GitLab and Sonarqube application using API call. The command used for executing the python script is given below.

```
python /vagrant_scripts/get_metrics.py "build"
```

build

In this stage, the source code from the repository is integrated and product is built. This stage basically performs the *compile* activity. The code is compiled and checks for any errors in the source code. Also the unit test cases and integration test cases are executed at this stage. If any source code errors exists or any of the unit or integration test cases fails, then the product will not be built. Once the build stage is completed successfully, a *.war* file gets created in the path *welcome-webapplication/target/welcome-webapplication-0.0.1-SNAPSHOT.war*. The command used for building the package is given below.

```
- mvn -f welcome-webapplication/pom.xml $MAVEN_CLI_OPTS clean
install
```

upload

This stage is used for adding the *.war* file created in *build* stage into job artifact. This artifact is used in the later stages of the deployment pipeline.

deploy

In this stage, *.war* file which is generated in the build stage is deployed into the stage environment and tomcat server is started to product run the product. This is done to execute the automated acceptance test cases in the staging environment.

sonar_test

This stage is responsible for performing test activity by executing the automated acceptance test cases in the staging environment The data such as number of passed/failed acceptance tests cases are captured by the Sonarqube applcation. A script with maven command is used for pushing the data into the Sonarqube application as given below.

```
- mvn -f welcome-webapplication.testing.testng/pom.xml
--batch-mode verify --fail-never
-Denv.BASEURL=$STAGE_BASE_URL test $SONAR_ARGS
```

metrics_test

It contains a job which runs on a shell executor inside the integration server. A python script is executed to extract the metrics data related to acceptance tests cases from the Sonarqube application using API call. The command used for executing the python script is given below.

```
script: python /vagrant_scripts/get_metrics.py "test"
```

test

This stage is responsible for performing test activity by executing the automated acceptance test cases to assert the product under development works at the functional level i.e. it meets the end user requirement. If any of the acceptance test is failed, pipeline will be stopped. Otherwise, pipeline will execute the next stage i.e. release stage. The command used for running the acceptance test cases is given below.

```
- mvn -f welcome-webapplication.testing.testng/pom.xml
$MAVEN_CLI_OPTS -Denv.BASEURL=$STAGE_BASE_URL test
```

release

This staged is used for transferring the latest available version of the product under development to production environment.

metrics_release

In this stage, a python script is executed using shell executor inside the integration server. This script inserts metric data related to production deployment into MySQL database table. The shell command used for executing the python script is given below.

```
script: python /vagrant_scripts/get_metrics.py "production"
```

4.5 Data collection

In this section, data collected for each of the metrics is explained. Only eleven metrics are measured out of 18 metrics listed in the table 3.7. These eleven metrics are M2, M3, M7, M8, M10, M11, M14, M18, M19, M25 and M26. This is due to time constraint and few of the metrics can not be calculated with the current experiment set up. The data for each of the metrics are collected only during the execution of the pipeline.

4.5.1 M2 - Number of commits by individual

This metric is measured in the compile activity under the commit stage(sonar_build) of the deployment pipeline. To measure this metric, commit details such as id(commit id) and committer_name are extracted from the version control repository (GitLab) in the metrics_build stage of the deployment pipeline instance.

A python script is used to fetch these data from GitLab using REST GET API call through the use of GitLab authentication token. The URL used for API call is given below.

```
http://192.168.56.15:9000/api/v4/projects/2/repository/commits/master
```

A sample JSON result of the API call is given below. The result contains details regarding the latest commit which occurred in the version control system . The commit details is fetched using python script and finally the values of the key *id*, *committer_name*, *committed_date* is inserted in to the database table in the columns *commit_id*, *committer_name*, *committed_date* respectively.


```
{
"id": "7c57c688e796f57d6dbebbd3206e418dbd4c92d9",
"short_id": "7c57c688",
```

```
    "created_at": "2022-08-12T14:31:00.000+00:00",
    "parent_ids": [
        "4ef1a5997e86ea5ff652511e11085140b46d3236"
    ],
    "title": "Update welcomeapplication.java",
    "message": "Update welcomeapplication.java",
    "author_name": "devops",
    "author_email": "pharisha889@gmail.com",
    "authored_date": "2022-07-12T14:31:00.000+00:00",
    "committer_name": "devops",
    "committer_email": "pharisha889@gmail.com",
    "committed_date": "2022-07-12T14:31:00.000+00:00",
    "web_url": "http://192.168.56.15/gitlab/devops/
    welcomewebapplication/-/commit/
    7c57c688e796f57d6dbebbd3206e418dbd4c92d9",
    "stats": {
        "additions": 2,
        "deletions": 1,
        "total": 3
    },
    "status": "running",
    "project_id": 2,
    "last_pipeline": {
        "id": 48,
        "project_id": 2,
        "sha": "7c57c688e796f57d6dbebbd3206e418dbd4c92d9",
        "ref": "master",
        "status": "running",
        "created_at": "2022-07-12T14:31:01.444Z",
        "updated_at": "2022-07-12T14:31:04.187Z",
        "web_url": "http://192.168.56.15/gitlab/devops/
        welcomewebapplication/-/pipelines/48"
    }
}
```


4.5.2 M3 - Number of commits by team / M19 - Source code commits per day

To measured this metric, the data extracted for the metric M2 as mentioned in section 4.5.1 is only used. This is also calculated as M2, the only change is that the number of commits is calculated for all members in the development team by aggregating based on the date value of *committed_ date*.

4.5.3 M7 - NLOC

This metric is measured in the code analysis activity under the commit stage (sonar_build) of the deployment pipeline. The metric value is present in the Sonarqube application and it is extracted using API call through the Sonarqube authentication token. The URL used for API call is given below .

```
http://192.168.56.15:9000/api/measures/component?
component=com.harish:welcome-webapplication&metricKeys=ncloc
```

A JSON response is returned upon the API call as given below, where key "value" holds the value of number of lines of code i.e. 110. This value is inserted into the column *nloc* of the database table. It is important to note that it does not calculate the empty and commented lines which are present in the source code.

```
{
"component": {
    "key": "com.harish:welcome-webapplication",
    "name": "welcome-webapplication",
    "description": "pom provides dependency",
    "qualifier": "TRK",
    "measures": [
        {
            "metric": "ncloc",
            "value": "110"
        }
    ]
}
}
```

4.5.4 M8 - NLOC added

The data for this metric is collected similar to metric M2. However, this time only value of the key *additions* from the JSON file is extracted using the python script and inserted into *nloc_added* column of the database table.

4.5.5 M10 - NLOC deleted

The data for this metric is collected similar to metric M2. However, this time only value of the key *deletions* from the JSON file is extracted using the python script and inserted into *nloc_deleted* column of the database table.

4.5.6 M11 - Change rate

The value of this metric is calculated using the metrics M7, M8 and M10. It is the ratio of sum of NLOC that are added, changed and deleted since the previous version over the NLOC of previous version. The metric NLOC changed is not instrumented in this experiment. Hence, it is calculated as sum of NLOC that are added and deleted since the previous version over the NLOC of previous version. The calculated value is inserted into *change_rate* column of the database table.

4.5.7 M14 - Cyclomatic complexity

This metric is measured in the code analysis activity under the commit stage (sonar_build) of the deployment pipeline. The data required to calculate the metric are source code files from the version control repository. The metric value is extracted from the Sonarqube application using API call. The URL used for API call is given below .

```
http://192.168.56.15:9000/api/measures/component?
component=com.harish:welcome-webapplication&metricKeys=complexity
```

A JSON response is returned upon the API call as given below where, key "value" holds the value of cyclomatic complexity i.e. 5. This value is inserted into the column *cyclomatic_complexity* of the database table.

```
{
    "component": {
        "key": "com.harish:welcome-webapplication",
        "name": "welcome-webapplication",
        "description": "pom provides dependency",
        "qualifier": "TRK",
        "measures": [
            {
                "metric": "complexity",
                "value": "5"
            }
        ]
    }
}
```

4.5.8 M25 - Number of failed test cases

This data for this metric collected similar to metric M7. The only change is in the URL of the API call as given below. The *metricKeys* value used in the URL

is “test_failures”. There are multiple places where the testing takes place in a deployment pipeline such as automated unit tests in commit stage, automated acceptance testing and manual testing in test stage. For this metric, only the automated unit and integration test cases are included.

```
http://192.168.56.15:9000/api/measures/component?
component=com.harish:welcome-webapplication&metricKeys=test_failures
```

A JSON response is returned upon the API call as given below where, key “value” holds the value of Number of failed test cases i.e. 0. This value is inserted into the column *failed_ test_ case* of the database table.


```
{
    "component": {
        "key": "com.harish:welcome-webapplication",
        "name": "welcome-webapplication",
        "description": "pom provides dependency",
        "qualifier": "TRK",
        "measures": [
            {
                "metric": "test_failures",
                "value": "0",
                "bestValue": true
            }
        ]
    }
}
```


Calculating only the failed test cases would not give much insight regarding the testing activity. Rather, it is good to calculate both failed test cases and total number test cases executed. Hence, we also calculated the total number of test cases (unit and integration test cases) by using below given URL. The total passed test cases can be calculated by simply subtracting the number of failed test cases by total number of test cases.

```
http://192.168.56.15:9000/api/measures/component?
component=com.harish:welcome-webapplication&metricKeys=tests
```

The above mentioned URL returns a JSON response as given below where, key “value” holds the value of total number of test cases i.e. 5. This value is inserted into the column *test_ case* of the database table.

```
    {
    "component": {
        "key": "com.harish:welcome-webapplication",
        "name": "welcome-webapplication",
        "description": "Parent pom providing dependency and plugin management for
        "qualifier": "TRK",
        "measures": [
            {
                "metric": "tests",
                "value": "5"
            }
        ]
    }
}
```


4.5.9 M26 - Number of passed acceptance tests

This metric is measured in the testing activity under the test stage of the deployment pipeline. The data for this metric collected similar to metric M25. The only change is in the URL of the API call as given below. The name of the component(project name) is changed i.e. *welcome-webapplication.testing.testng*. Both number of total acceptance test cases and total number of failed acceptance cases is extracted from the Sonarqube. The Number of passed acceptance test cases is calculated by subtracting the total test cases by failed test cases. Finally, the values for total test cases and failed test cases are inserted into the column *acceptance_ test_ case* and *failed_ acceptance_ test_ case* respectively.

```
http://192.168.56.15:9000/api/measures/component?
component=com.harish:welcome-webapplication.testing.testng
&metricKeys=tests

http://192.168.56.15:9000/api/measures/component?
component=com.harish:welcome-webapplication.testing.testng
&metricKeys=tests_failures
```

4.5.10 M18 - Production deployment

This metric is measured in the release activity under the release stage of the deployment pipeline. A python script is used for inserting a time-stamp in *production_ deployment* column of the database table only after the release stage completes successfully in the deployment pipeline.

4.6 Discussion

Section 4.5 provides the answers to the RQ2 i.e. how the metrics can be operationally measured. Below is a summary of the data collection results for each of the metric that was measured.

The metrics M2, M3, M7, M8, M10, M11, M14, M19 and M25 are measured in the compile activity under the commit stage of the deployment pipeline. Data for metrics M2, M3 and M19 are collected from the version control repository i.e GitLab. The data required to calculate the metric M7, M8, M10, M11, M14 and M25 are the source code files present in the version control repository and data for these metrics recorded in the Sonarqube application.

The metric M26 is measured testing activity under the test stage of the deployment pipeline. The data required to calculate this metric are source code files and data for this metric is recorded in the Sonarqube application. Metric M18 is measured in the release activity under the release stage of the deployment pipeline.

The experiment is carried out for a week and in this period deployment pipeline is executed for 65 times. Out 65 times, 32 times deployment pipeline successfully completed all the stages and 33 times the pipeline failed at various stages. All the 11 metrics (M2, M3, M7, M8, M10, M11, M14, M19, M18, M25 and M26) are measured only when the deployment pipeline executes all the 10 stages successfully. If the pipeline fails at any stage, few of the metrics are not measured. For example, if the deployment pipeline fails at *build* stage, only metrics M2, M3, M7, M8, M10, M11, M14, M19 and M25 are measured. Whereas, metrics M26 and M18 are not measured as these metrics are calculated in the sonar_test and release stage of the pipeline respectively. It is important to note that measurements which already took place in a failed deployment pipeline is also captured and stored.

Data for each of the measured metrics is stored in a MySQL database table mentioned in the section 4.2.6 and this table serves as data source for visualising the metrics. The detailed information regrading the visualisation of each of the measured metric is given in the next chapter i.e. Data Visualisation.

This section will also discuss on how easy and difficult was to operationalise the metrics. For the mentioned eleven metrics, the instrumentation procedures are mentioned in the section 4.5. The vagrant files and ansible playbooks were difficult to create. This was done to automate the creation of environments, as well as the installation and configuration of software used in the experiment. However, after creating vagrant files and ansible playbooks, the experiment setup was completed in a matter of minutes. A replication package is created to recreate this experiment setup and is placed in a public repository which is available at URL: `https://doi.org/10.5281/zenodo.6998873`.

The data to calculate metrics M2, M3, M8, M10, M11, M18 and M19

were taken from the GitLab. But, data to calculate the metrics M7, M14, M24 and M26 were not available in GitLab. Hence, it was required to setup Sonarqube application to collect these metrics. Except for metric M18, all other metrics required an API call to extract the data from either GitLab or Sonarqube. Hence, it was required to develop a python script to perform the API calls to extract the data. Furthermore, because the data returned by the API call was in JSON format, data prepossessing was required to extract metrics data after the API call. This process of extracting and pre-processing of data adds a small amount(10-15 seconds) of overhead to the deployment pipeline.

Among all the metrics, metric M25 (Number of failed test cases) was the most difficult to instrument. Because, if a test case fails, the build of the product will also fail. Hence, it was not possible to collect the data regarding test cases i.e. number of failed/passed test cases. To overcome this problem, we tweaked the command used in the sonar_build stage to allow the build to pass even if there are test case failures. Similarly, it is done for the metric M26(Number of passed acceptance tests) also to calculate the number of passed/failed acceptance test cases in sonar_test stage. However, this one lead to adding more overhead to the deployment pipeline. Because, both stages sonar_build and build works similar, only change is that sonar_build stage will pass even though there is test case failure and in case of build stage it will fail.

In this study, the metrics M4, M5, M6, M9, M12, M13, and M15 are not operationalized. This is due to time constraint of this study and also the data required to measure these metrics were not readily available in the current experiment setup.

Chapter 5

Instrumentation - Data Visualisation

In this chapter, a rapid review is performed as described by Cartaxo et al.[4] to find the types of chart that can be used to visualise the data collected in chapter 4. A survey research is conducted to select the best possible chart type to visualise the metrics by creating dashboards in a data visualisation tool.

5.1 Rapid review

5.1.1 Search strategy

Rapid review (RR) is performed using the scientific repository, ACM digital library with ACM Guide to Computing Literature database. This database was queried to get all the relevant papers that discuss about the data visualisation. To fetch relevant papers, some search query trials were executed using keywords that related to data visualisation. Running these trials was the first step taken to identify studies and prevalent synonymous words related to data visualisation. The goal of these search query trials us can be summed up as follows-

1. To find the papers related to "Data visualization"

2. To find keywords related to ‘Data visualization”

The remainder of this section describes the steps taken to obtain the RR’s final corpus. This corpus was then used as a starting point for identifying and extracting relevant data. The results of the information identification, extraction, and classification allowed for the synthesis of the findings reported in this study. These activities, along with their sequential order are depicted in Figure 3.1.

To obtain the final corpus for the RR, the following filters are applied sequentially-

F1 Firstly, selected the papers related to the topic ”visualization”. The prevalent closely connected words related to “visualization” used in the scientific literature are identified as “Dashboard” and “Analytics”. So, the filtered papers should have either Visualization, Dashboard or Analytics in their title and abstract.

F2 Secondly, out of all those papers selected in the F1, shortlisted the papers based on content of the abstract of the paper. The abstract of the paper should contain the keyword Visualization or Dashboard or Analytics.

F3 Thirdly, selected the papers only between the duration of 2020 and 2021. The reason for selecting the papers only published in the last 2 years is to include only the recent studies. Also, only for 2 years of duration there were in total 861 results were returned which is a good numbers of papers for a rapid review.

F4 Fourthly, selected papers only related to the research article. The reason for selecting only research articles is to retrieve the latest strong validated [1] outcomes from the research community. Thus, these research papers provide a strong foundation for the content analysis.

The final search query with all of the above filter is-

"query": Title:("visualization" OR "dashboard" OR "analytics") AND Abstract:("visualization" OR "dashboard" OR "analytics")

[1] By the community itself through peer-review.

"filter": Article Type: Research Article,Publication Date: (01/01/2020 TO 12/31/2021)

5.1.2 Study selection

Candidate papers obtained after applying the filter mentioned in step 5.1.1 needed to be reviewed to determine their suitability for the study's goal, which was to identify the various types of charts/plots used to display the data.

The content analysis of each candidate paper was divided in two sub-steps. These two sub-steps are described next-

Abstract

The first sub-step was to review only the abstract of each paper to check the written language and the type of study. Based on the objectives of this step, the inclusion (IC) and exclusion (EC) criteria were defined as follows:

IC1 Papers written in English are included.

EC1 Papers related to systematic literature reviews are excluded

EC2 Papers related to extended abstract are excluded.

Content

After completing the first sub-step, the second sub-step required a thorough review of each retained candidate paper. The purpose of this comprehensive review was to determine whether the candidate paper's findings were supported by empirical evidence and related to the purpose of this study. The goal of this sub-step was to add the following conditions to the IC and EC criteria-

IC2 Contains papers that deal with empirical research. Research investigations classified as "empirical" are those whose conclusions

are backed up by data gathered from field studies or tests carried out in restricted settings. The fundamental reason for selecting this kind of research was to look for insights into how charts were instrumented as part of data visualisation.

IC3 Papers with any charts/plots used to visualise the data and also any tools used to display the charts/plots using a dashboard.

5.2 Data extraction

The goal of this step is to look through the retained articles for the information needed to answer the RQ4. The various types of charts extracted from the papers that were included in the final corpus as part of the RR. In addition, the various dashboard tools used to display data using charts are extracted.

5.3 Results

This section presents the results of having executed the methodology's steps as presented in Section 3.1.

Only empirical papers were chosen for the content analysis because the empirical study is based solely on the collected data and does not make any assumptions or theories.

5.3.1 Result - Search strategy

The automated search in the ACM database corresponds to the first step. In this step, the filters F1, F2, F3, and F4 are applied in an accumulative manner. Table 5.1 shows the results of having applied these filters. Post applying these filters, 861 papers were selected for the second step: i.e. *Study selection.*

Step	Filters	Selected papers
1	**F1**	21852
1	**F1+F2**	14924
1	**F1+F2+F3**	1440
1	**F1+F2+F3+F4**	861
2	**F1+F2+F3+F4+IC1**	850
2	**F1+F2+F3+F4+IC1+EC1**	832
2	**F1+F2+F3+F4+IC1+EC1+EC2**	815
2	**F1+F2+F3+F4+IC1+EC1+EC2+IC2**	694
2	**F1+F2+F3+F4+IC1+EC1+EC2+IC2+IC3**	12

Table 5.1: Research papers selected for each filter, inclusion and exclusion criteria

5.3.2 Result - Content analysis

The content analysis is performed in two sub-steps:

- Firstly, content was analysed based on the abstract of the paper. Papers were included and excluded based on the inclusion (IC1) and exclusion criteria (EC1, EC2) as mentioned in Section 5.1.2. Post applying inclusion and exclusion criteria, 815 papers were retained for the next sub-step.
- Secondly, content was analysed based on the entire content of the paper. Papers were included based on the inclusion criteria (IC2, IC3) as mentioned in Section 5.1.2. Post applying these criteria, 12 papers were retained for the next step: i.e. *Data Extraction*.

Table 5.1 also shows the results of applying the exclusion and inclusion criteria, which resulted in a corpus of 12 papers.

5.3.3 Result - Data extraction

This section describes the different types of charts extracted from the papers which made into final corpus as part RR are listed in the table 5.2. The different types of dashboard tool used for visualisation which are extracted from the papers is listed in the table 5.3.

Chart Type	Citations
Radar chart	**[5] [35] [3]**
Line chart/Lines-plot	**[20] [8] [38] [3] [15]**
Heat map	**[20] [29] [7] [38] [15]**
Area chart	**[29]**
Pie chart	**[29] [8] [43] [3] [15]**
Bar graph/Bar chart/Bar plot	**[29] [11] [38]**
Stacked bar chart	**[7]**
Sunburst graph	**[7] [15]**
Doughnut chart	**[37]**
Line-node graph	**[43]**
Scatter-plot	**[38]**
Histograms	**[38]**
Bubble chart	**[38]**
Box-plot	**[38]**
Radial chart	**[38]**
Spider chart	**[38]**
Circular heat map	**[38]**
Timelines	**[38]**
Trees	**[38]**

Table 5.2: Chart types

Dashboard Tool	Citations	Free	Open-source
Kibana	**[29] [38]**	Yes	Yes
Grafana	**[29] [38]**	Yes	Yes
Web application	**[7]**	Yes	Yes
ThingsBoard	**[38]**	Yes	Yes
Plotly	**[38]**	Yes	Yes
Tableau	**[38]**	Yes	No
IBM Watson IoT platform	**[38]**	Yes	No
Power BI	**[38]**	Yes	No
Gephi	**[38]**	Yes	No

Table 5.3: Dashboard Tools

Out of 12 papers which made into the final corpus, only one paper [38] contains the descriptions for the chart types. The descriptions extracted from the paper is given below.

5.3.4 Descriptions of chart type

- **Bar chart/Bar graph/Bar plot :** Bar chart displays the relative frequency or relative density of different items.
- **Scatter-plot:** Scatter-plot are used to visualise data instances as points in 2D and 3D spaces.
- **Histograms:** Histograms are similar to bar-plots but they offer a more fine-grained approach to reveal the parameter distribution over time, while the range of values is set into a series of intervals.
- **Bubble chart:** Bubble chart is used for specification of higher dimension pictures in a smaller dimension by projecting the third dimension on a 2D chart.
- **Box-plots:** Box-plots are specific plots that show typical quantiles of the distribution of data and for single dimension measures of central tendency and dispersion of data.
- **Radial chart:** Radial chart shows the different extents of the relative values that each variable can take.
- **Spider chart:** Spider chart shows the achieved level by different dimensions.
- **Heat map:** Heat map illustrates the spatial distribution of a variable on a map.
- **Circular heat maps:** Circular heat maps are used to compactly depict the change of a variable over a month.
- **Timelines:** Timelines are useful data elements that are utilized to understand the evolution of a one or more variables and the behavior of changing trends or patterns over time from an analytics perspective point of view.
- **Trees:** Trees are special data structures used for representing scenarios with hierarchical decomposition.

5.4 Mapping of chart type with metric

After the data extraction performed as mentioned in the section 5.3.3. The main challenge is to map the suitable charts for each of the metric listed in the table 3.2. After analysing the nature of the data that each metric holds, it turned out that there are more than one chart that can be used to visualise each metric. For example, metric “Number of failed test cases” can be displayed using either “Pie chart” or “Doughnut chart”. This led to consider the various chart types that could be used to display each metric.

The mapping of suitable chart types for each of the metric is mentioned in table 5.4. The column "Chart type 1", "Chart type 2" and "Chart type 3" gives the type of charts that can be used to visualise the metric. If a column holds value as "-", it means the chart type is not defined for the metric.

Metric Name	**Chart Type 1**	**Chart Type 2**	**Chart Type 3**
Number of commits by individual	Stacked bar chart	Stacked area chart	Stacked line chart
Number of commits by team	Bar chart	Line chart	Area chart
Merge your own code(MYOC)	Bar chart	Line chart	-
Early branching(EB)	Bar chart	Line chart	-
Merge-Often(MO)	Bar chart	Line chart	-
Nloc	Bar chart	Line chart	-
Nloc added	Bar chart	Line chart	-
Nloc changed	Bar chart	Line chart	-
Nloc deleted	Bar chart	Line chart	-
Change rate	Bar chart	Line chart	-
Token count	Bar chart	Line chart	-
Parameter count	Bar chart	Line chart	-
Cyclomatic complexity	Bar chart	Line chart	-
Effective cyclomatic complexity	Bar chart	Line chart	-
Production deployment	Bar chart	Line chart	-
Source code commits per day	Bar chart	Line chart	Area chart
Number of failed test cases	Stacked bar chart	Stacked area chart	-
Number of passed acceptance tests	Stacked bar chart	Stacked area chart	-

Table 5.4: Mapping metrics with chart type

A survey is conducted to avoid bias and to select the best suitable chart from among all possible charts for each metric. The details of the survey are explained in the following section.

5.5 Survey

As mentioned in the section 5.4, there are more than one chart types is available for visualising each metric. It is important to select a "best suitable" chart type for each of the metric. To find the best suitable chart type to visualise the each of the metric, a "self-administered questionnaires" survey research[2] is carried out.

[2]Survey research is just one method for gathering data to gain insight into people or problems under research.

The methodology of the survey is adapted based on the paper by Kasunic et al [24].This section explains in detail regarding the steps taken to conduct survey process. The methodology used for survey research is shown in 5.1. The survey research process carried out as part of the study involves 5 stages.

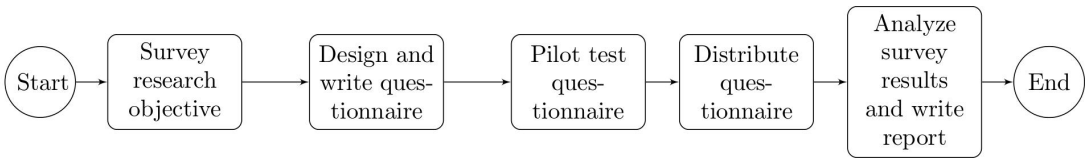


Figure 5.1: Survey research methodology

5.5.1 Survey research objective

The objective of the survey is to find the "best suitable" chart type to visualise the metric among all possible charts for each metric as mentioned in table 5.4.

5.5.2 Design and write questionnaire

The survey objective and internal questions are translated into carefully worded questionnaire items in this step. The questionnaire is mainly divided into 2 sections.

1. Questions regarding the best suitable chart for each metric.
2. Questions related to information about the participant.

The questions related to chart type for metric is done by grouping metrics which can be displayed with the same type of charts. This grouping is shown in table 5.5. One question is prepared for each of the groups to cover all the metrics in the survey. The participants are provided with options for each question as the chart type and also allowed to pick multiple options. Participants were given the opportunity to choose an additional option as "Other" in addition to the predetermined list of answer options, allowing them to offer an alternative chart type.

Group	**Metric Name**	**Chart Type**
Group 1	Number of commits by individual	Stacked bar chart/Stacked area chart/Stacked line chart
Group 2	Number of commits by team Source code commits per day	Bar chart/Line chart/Area chart

Group 3	Merge your own code(MYOC) Early branching(EB) Merge-Often(MO) Change rate Effective cyclomatic complexity	Bar chart/Line chart
Group 4	Nloc Nloc added Nloc changed Nloc deleted Token count Parameter count Cyclomatic complexity Production deployment	Bar chart/Line chart
Group 5	Number of failed test cases Number of passed acceptance tests	Stacked bar chart / Stacked area chart

Table 5.5: Group metrics based on chart type

Personal attributes such as highest degree, area of study, work experience, domain knowledge, profile of the participant is also asked. This information is mainly collected to analyse the survey result based on the profile of the participant.

5.5.3 Pilot test questionnaire

A pilot test of survey is carried out with small set of target audience to test the implementation of the survey. This is done in order to expose problems or weaknesses in the questions, questionnaire layout, process or technology. The questionnaires are then altered based on the feedback given from the audience who are participated in the pilot test which lead to creation of final version of the survey.

5.5.4 Distribute the Questionnaire

The final version of web-based questionnaire is distributed to the participants via e-mail. The survey is also shared on social media in order to invite a larger audience to participate. To increase response rates, the response window's duration is kept at about 4 weeks.

5.5.5 Survey results

In this section, results of the survey is discussed. A total of 55 responses are received as part of the survey. Figure 5.2 shows the number of responses received based on the profile of the respondent. It is interesting to see that, 11, 13 and 4 participants are developer, team lead and tester respectively. Only 3 participants are manager or higher level. 24 participants i.e 44 % of the participants are from the profile "Other".

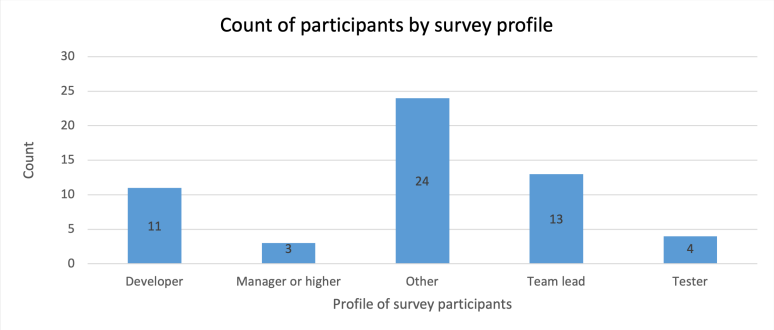


Figure 5.2: Survey responses based on profile of respondent

Number of commits by individual

The survey result for this metric is shown in figure 5.3. For the metric Number of commits by individual, 40% of the participants have selected the *Stacked bar chart* and 25% of the participants have selected the *Stacked area chart* and 18% for the Stacked line chart. 9% of the participants selected both stacked bar and area chart and 6% selected the stacked bar and line chart. The stacked bar chart is common in participants who selected more than one chart.

All the 4 participants, who are *tester* voted for the chart type *Stacked bar chart*. 5 participants each who are *Team lead* voted for *Stacked bar chart* and *Stacked are chart*. All 3 managers has selected 3 different chart types for displaying the metric. Where as 5 developers out of 11 voted for the *Stacked bar chart*. It is evident from the survey result that the best suitable chart type for displaying the metric "Number of commits by individual" is *Stacked bar chart*.

> The best suitable chart for visualising the metric "Number of commits by individual" is *Stacked bar chart*.

Number of commits by team

The survey result for this metric is shown in figure 5.4. 33% of the total participants has selected Bar chart and Area chart each. only 5% of the participants selected

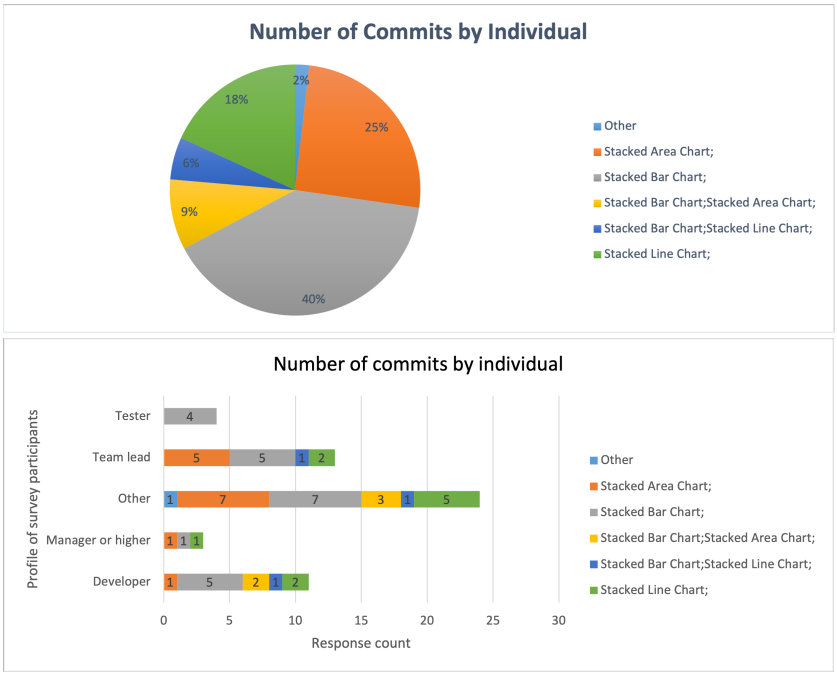


Figure 5.3: Survey response for metric - Number of commits by individual

both Bar and Area chart. 11% of the participants selected both Bar and Line chart. 4% of the participants selected all the 3 charts i.e., Bar, Line and Area chart. In most of the cases, the Bar chart is selected the most.

On one hand, 2 out of 3 managers has selected bar chart as best suitable chart for displaying the metric number of commits by a team. On the other hand, 4 developers out of 11 has explicitly selected area chart and only one developer selected bar chart. 5 team leads selected bar chart and 4 team leads selected the area chart. However, there are more number of participants selected the bar chart when selecting more than one chart. As a result, it appears that the bar chart is best suited to visualise the metric number of commits by a team.

The best suitable chart for visualising the metric "Number of commits by a team" is *Bar chart.*

NLOC(Number of lines of code)

The survey result for this metric is shown in figure 5.5. 74% of the participants chosen line chart and 20% of chosen bar chart. Only 4% of the participant selected both bar and line chart. All 3 managers, 10 out of 11 developers, 10 out of 13 team leads and 3 out of 4 testes selected line chart as the best suitable chart to visualise the metric number of lines code(NLOC). Therefore, it appears that the line chart is the most effective way to represent the metric NLOC.

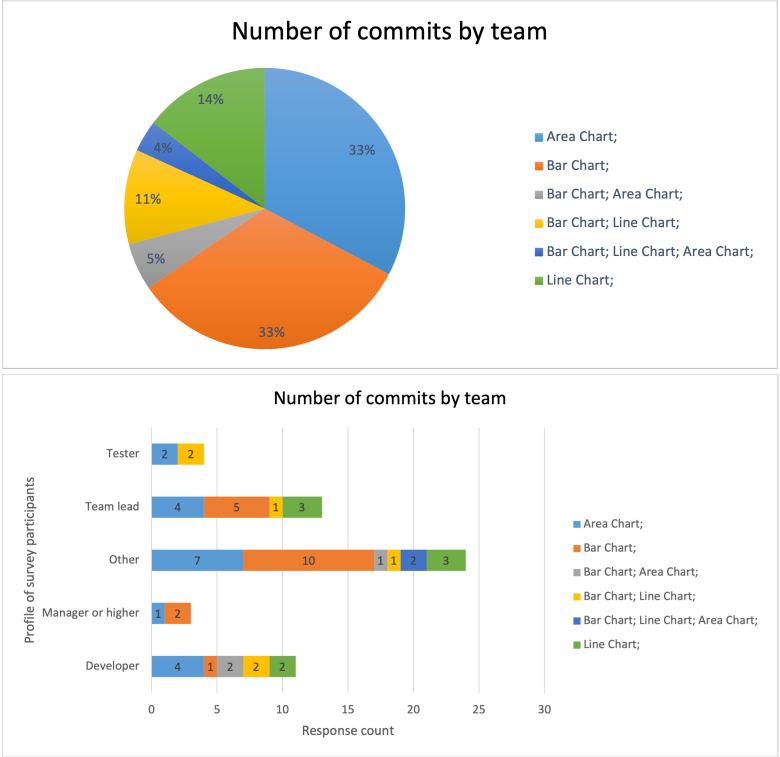


Figure 5.4: Survey response for metric - Number of commits by team

> The best suitable chart for visualising the metric "Number of commits by a team" is *Line chart*.

Change rate

The survey result for this metric is shown in figure 5.6. 47% of the participants selected line chart and 38% of the participants selected bar chart. only 11% of the respondents selected both bar and line chart. It looks overall line chart is the best chart for visualising the metric change rate. This can also seen in the responses based on the profile of the respondents. 2 managers, 9 team leads selected the line chart over bar chart. 3 testers out of 4 selected line chart as well. Only the developers have preferred bar chart instead line chart.

> The best suitable chart for visualising the metric "Change rate" is *Line chart*.

Number of failed test cases

Stacked bar chart has been selected by 82% of the total participants as shown in 5.7. whereas only 13% of participants selected stacked area chart. The responses

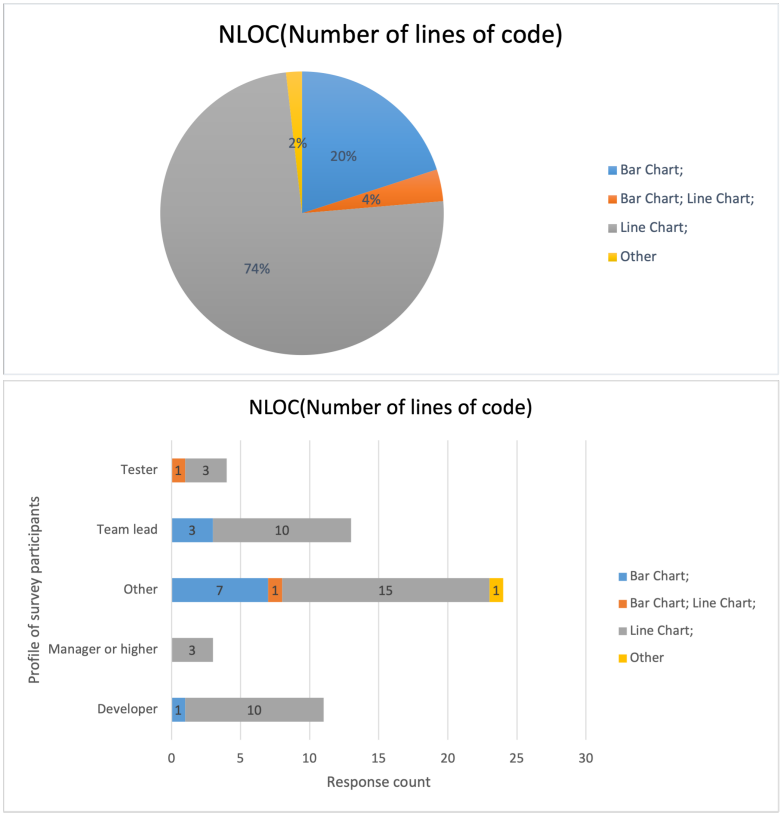


Figure 5.5: Survey response for metric - NLOC

based on participants profile shows that all the profile has selected stacked bar chart as the best suitable chart to display the metric Number of failed test cases.

> The best suitable chart for visualising the metric "Number of failed test cases" is *Stacked bar chart.*

Number of test cases failed for latest run

The survey result for this metric is shown in figure 5.8. 51% of the respondents selected pie chart and 31% of the respondents selected doughnut chart. 16% of the participants voted for both pie and doughnut chart. 2 out of 3 managers selected the doughnut chart over pie chart. However, 9 team leads out of 13 voted for pie chart. also the developers and testers voted more for the pie chart compared to doughnut chart. Majority of the profile is voted for pie chart as the best suitable chart to visualise the metric Number of test cases failed for the latest run.

> The best suitable chart for visualising the metric "Number of failed test cases" for latest run is *Pie chart.*

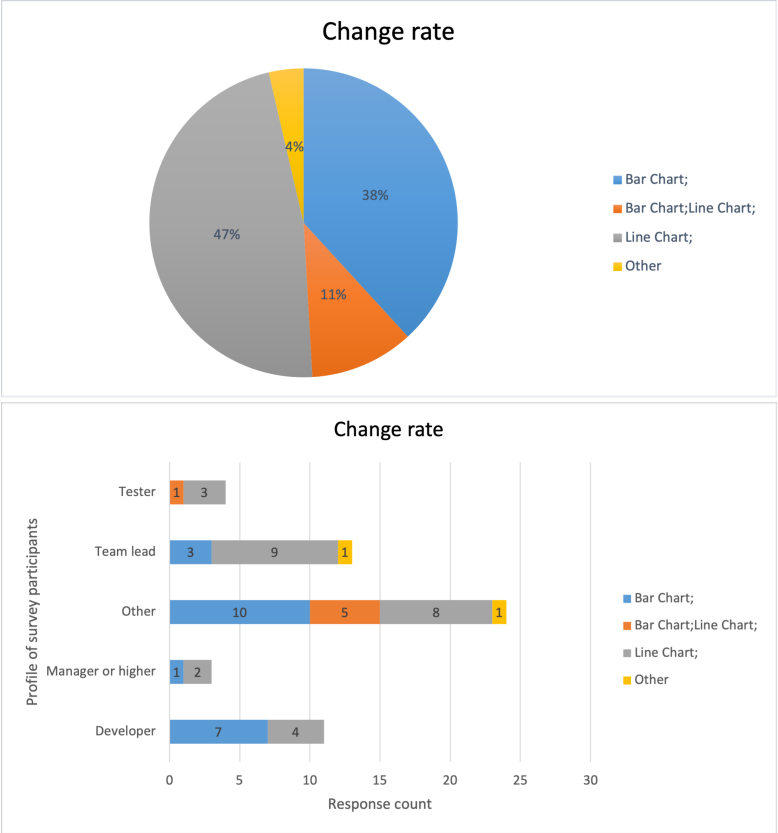


Figure 5.6: Survey response for metric - Change rate

Number of commits by individual on a single day in a project

The survey result for this metric is shown in figure 5.9. 56% of the respondents selected pie chart and 26% of the respondents selected doughnut chart. 18% of the participants voted for both pie and doughnut chart. 2 out of 3 managers selected the pie chart over doughnut chart. 9 team leads out of 13 voted for pie chart. also the developers and testers voted more for the pie chart compared to doughnut chart. Majority of the profile is voted for pie chart as the best suitable chart to visualise the metric Number of commits by individual on a single day in a project.

> The best suitable chart for visualising the metric "Number of commits by individual" on a single day in a project is *Pie chart*.

The summary of the survey result with the best suitable chart for each of the metric is mentioned in the table 5.6. This result is used as the reference for all the metric which are part of respective group.

Metric Name	**Group**	**Chart Type**
Number of commits by individual	Group 1	Stacked bar chart

Number of commits by team	Group 2	Bar chart
Change rate	Group 3	Line chart
Nloc	Group 4	Line chart
Number of failed test cases	Group 5	Stacked bar chart
Number of failed test cases for latest run	Group 6	Pie chart
Number of commits by individual on a single day in a project	Group 7	Pie chart

Table 5.6: Mapping metrics with chart type based on the result of survey

5.6 Implementation

In this section, implementation of visualisation for each measured metrics are explained. Figure 5.10 shows the overview of the dashboard created using the data collected for metrics.

The tool used for visualisation is Grafana and instrumentation of Grafana is given in section 4.2.7. The data source for each metric is MySQL table. The charts selected for displaying the metrics data is entirely based on result of survey as given in the section 5.5.5. The experiment is carried out over a 9-day period. The implementation of each of the metric is explained below.

5.6.1 Number of commits by individual

The stacked bar chart is used for displaying this metric as show in in figure 5.11a. It shows the number of commits done by the each individual in a team. From chart it can seen that three developers (developer1, developer2, developer3) have done commit operation to the VCS. The SQL query used for extracting the data from table is given below.

Similarly, number of commits done by individual on single day using a pie chart as shown in figure 5.11b. The data is displayed for only the latest where the deployment pipeline has triggered. The SQL query used for extracting the data from table is given below.

```
 SELECT
  date(committed_date) AS time,
  commiter_name AS metric,
  count(commiter_name) AS value
FROM metrics
GROUP BY commiter_name, DATE(committed_date)
ORDER BY date(committed_date) ASC
```

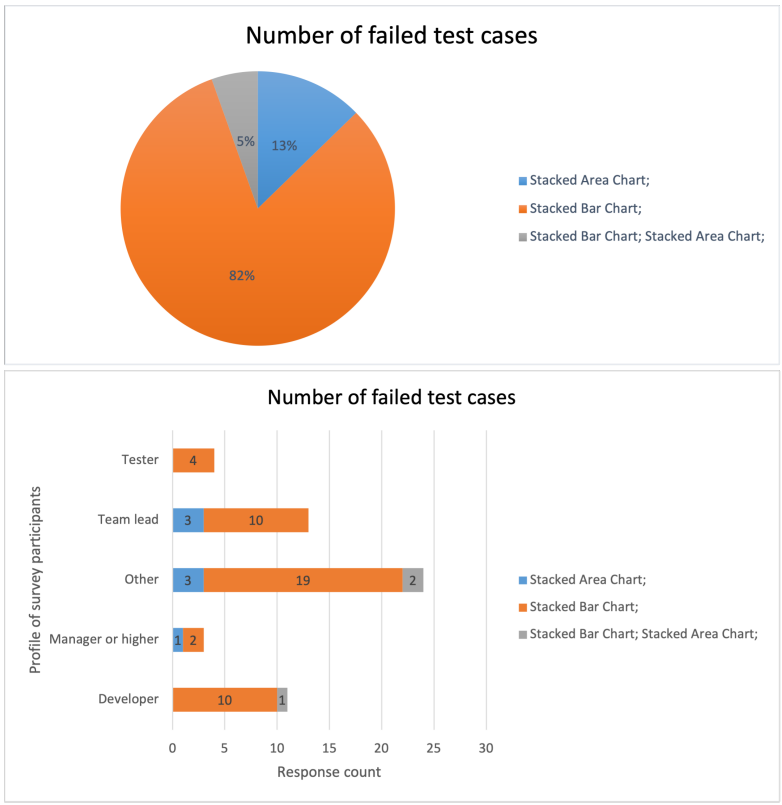


Figure 5.7: Survey response for metric - Number of failed test cases

This metric allows you to evaluate an individual developer's performance. The developer's performance is defined as the time it takes to complete a specific activity (i.e. code requirements). A way to assess his/her performance is by looking at the number of commits he/she does over certain period of time. This can be analysed from the figure 5.11a. Out of three developer, developer3 is not committing the code regularly and developer1 and developer2 are committing their code every day with multiple commits on each day.

5.6.2 Number of commits by team

It is implemented similar to metric number of commits by individual but only change is that the total number of commits is calculated for all members in the development team. It is done by aggregating the commit data based on date. The figure 5.12 shows the data for the metrics number of commits by team. The SQL query used for extracting the data from table is given below.

```
SELECT
  date(committed_date) AS time,
  count(commiter_name) AS "Commit"
FROM metrics
GROUP BY DATE(committed_date)
```

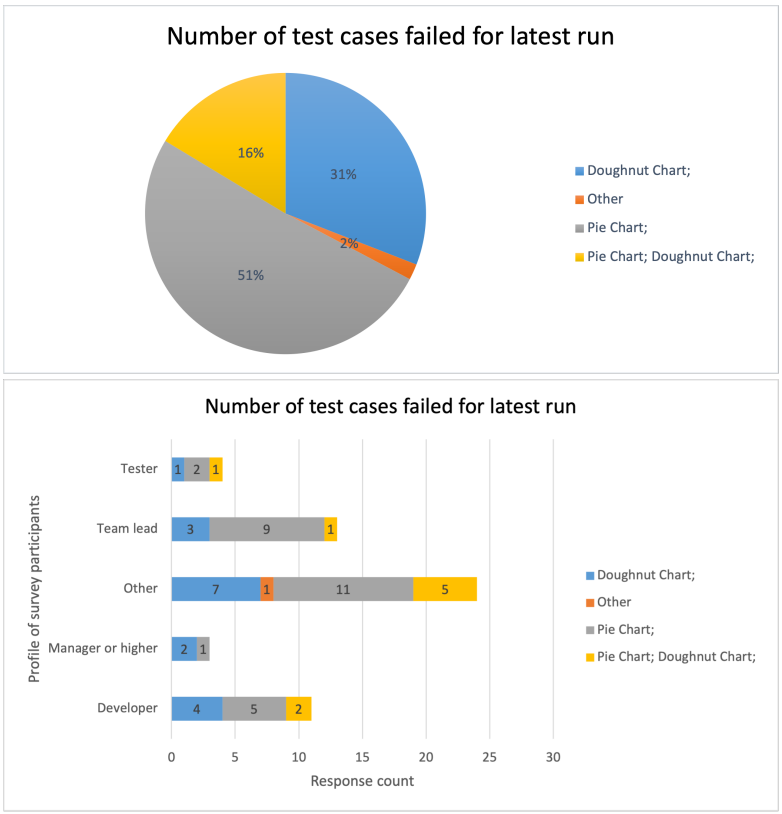


Figure 5.8: Survey response for Number of failed test cases for latest run

```
ORDER BY date(committed_date) ASC
```

This metric also allows to assess the performance similar to metric number of commits by individual. whereas this metric does it for a group of developers. From figure 5.12, It is simple to see how the team works on the project on a daily basis. For example, on 08/03, the team only made two commits. However, the team made more commits on 08/04, totaling 12.

5.6.3 NLOC

The line chart is used for displaying this metric as show in in figure 5.13. It displays the number of lines of code contained in the source code (i.e. product) for each time the pipeline triggers by using the SQL query as given below. The chart helps in understanding how the source code under development is evolving on each commit by the developers.

```
SELECT
  committed_date AS time,
  nloc AS "NLOC"
FROM metrics
ORDER BY committed_date ASC
```

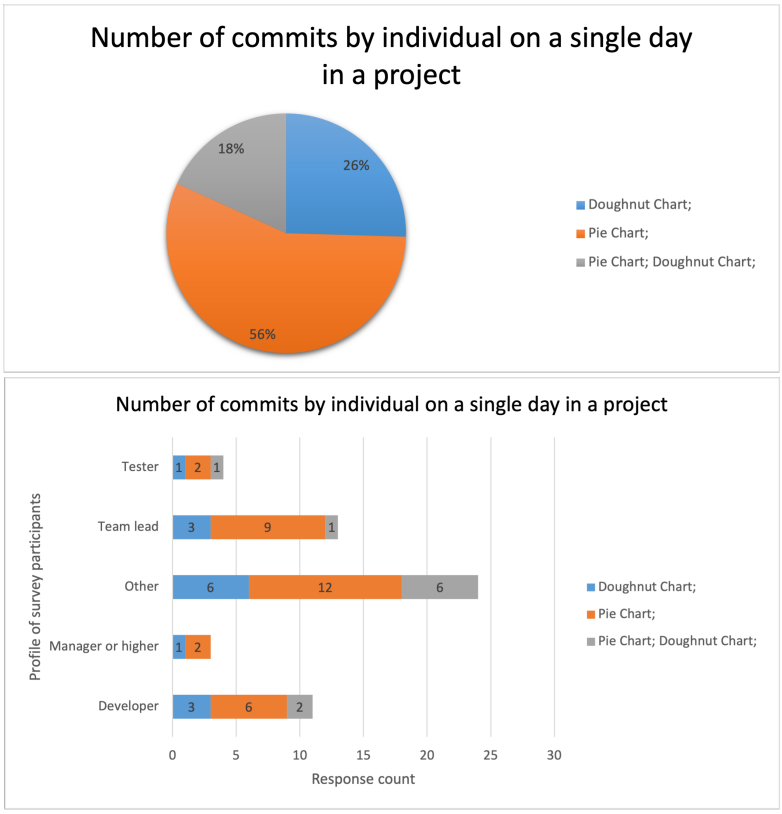


Figure 5.9: Survey response for Number of commits by individual on a single day in a project

5.6.4 NLOC added

This metric is displayed similar to metric NLOC using a line chart as shown in. figure 5.14. However, only the nloc that have been added since previous version is displayed. The chart gives more insight on how much new codes are getting added to the source code on each commit.

5.6.5 NLOC deleted

This metric is displayed similar to metric NLOC using a line chart as shown in. figure 5.14. However, only the nloc that have been deleted since previous version is displayed. The chart gives more insight on how much of source code is getting changed on each commit.

5.6.6 Change rate

Figure 5.16 depicts the Change rate measured for 9 days using a line chart. The source data is extracted from the Change_rate column of the database table. This

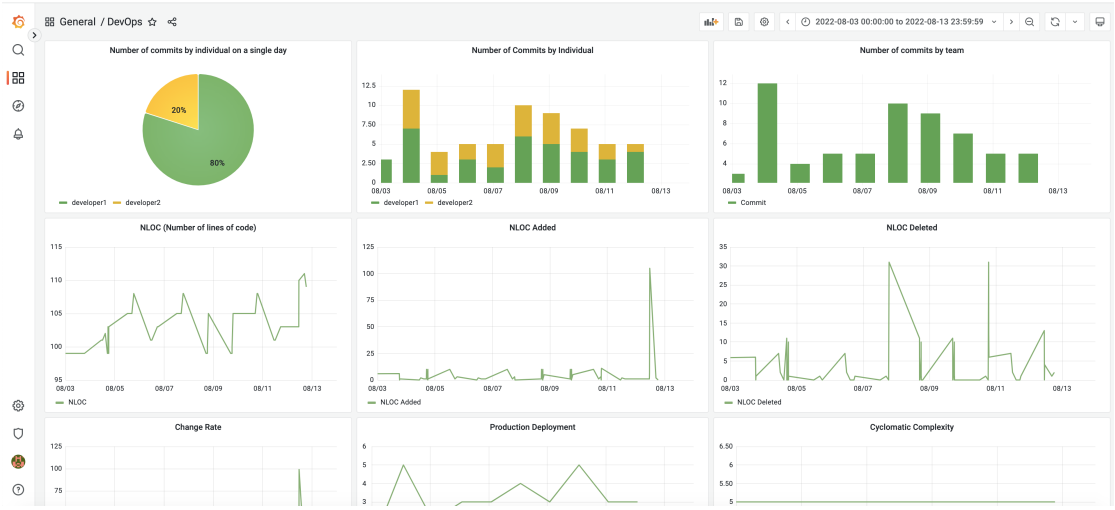
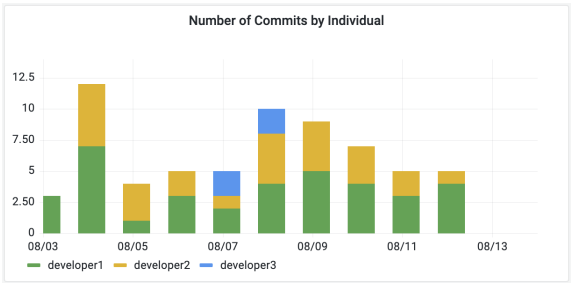
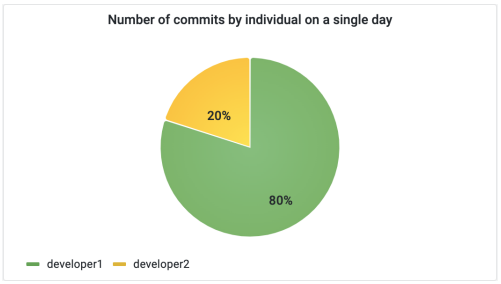


Figure 5.10: Dashboard overview



(a) Number of commits by individual



(b) Number of commits by individual on a single day

Figure 5.11: Number of commits by individual

chart shows how the code base under development evolves with each developer commit.

5.6.7 Number of failed test cases

For this metric both passed and failed test cases which includes both unit and integration test cases. Stacked bar chart is used to display both passed and failed test cases as depicted in the figure 5.17. A pie chart is also created to display only the results of test cases that were executed as part of the most recent deployment pipeline instance as shown in the figure 5.19. For efficient visualisation, the failed and passed test cases are shown in red and green color. The SQL queries used for creating these charts are given below. This chart allows to visualise how well the code is written with respect to the test cases. If there are more number of the failed test case, shows low quality of the work committed by developers. If there are no failures, it indicates that no errors found in the source code and high quality of work done by the developers.

Figure 5.12: Number of commits by team

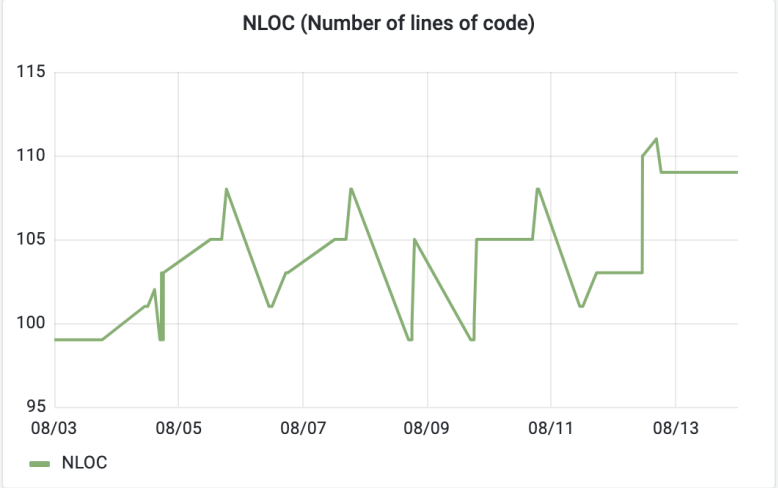


Figure 5.13: NLOC

```
SELECT
  committed_date AS time,
  test_case-failed_test_case AS "Test case passed",
  failed_test_case AS "Test case failed"
FROM metrics
ORDER BY committed_date ASC

SELECT
  committed_date AS time,
  failed_test_case "Test case failed",
  test_case-failed_test_case AS "Test case passed"
 FROM metrics
ORDER BY committed_date DESC LIMIT 1
```

5.6.8 Number of passed acceptance test

The implementation of visualisation for this metric similar to the metric number of failed test cases as depicted in figure 5.18.Instead of showing only the passed

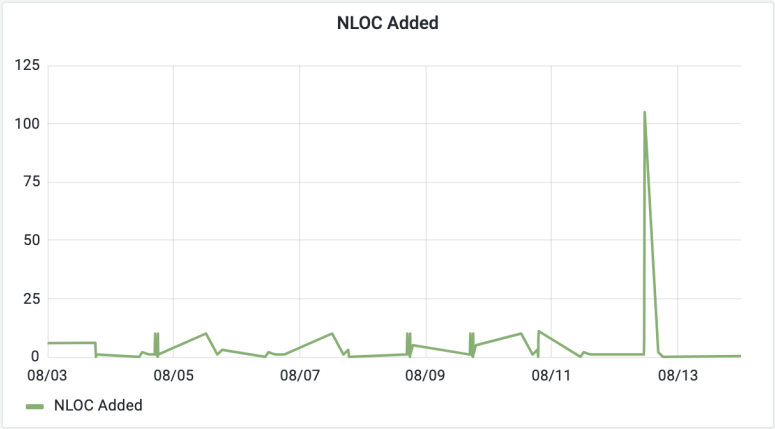


Figure 5.14: NLOC added

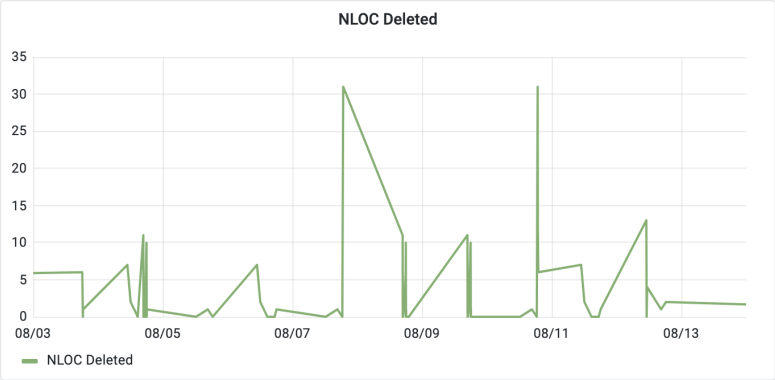


Figure 5.15: NLOC deleted

acceptance tests, passed acceptance is also displayed. Also, a pie chart is created to display only the results of acceptance test cases that were executed as part of the most recent deployment pipeline instance as shown in the figure 5.19b.

5.6.9 Cyclomatic complexity

This metric is displayed using a line chart as shown in figure 5.20.

The metrics NLOC, NLOC added, NLOC deleted, Change rate, and Cyclomatic complexity are source code-based metrics used to assess the complexity of a software product. Knowing the complexity also allows to predict the effort required to maintain the product. Charts implemented in the dashboard for these metrics provide additional information about the maintainability of the software under development.

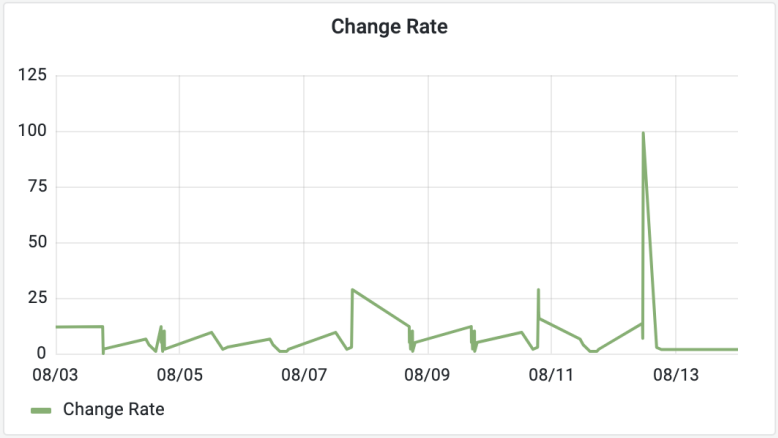


Figure 5.16: Change rate

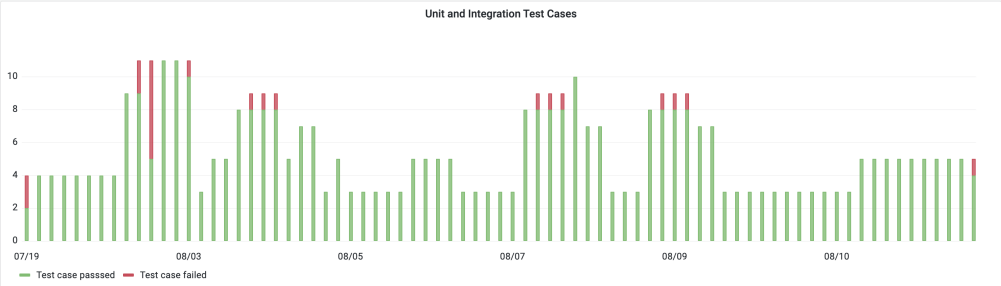
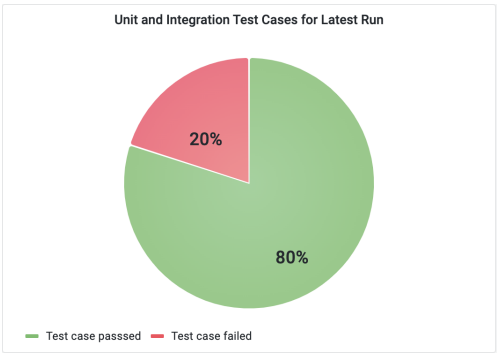


Figure 5.17: Number of failed test cases

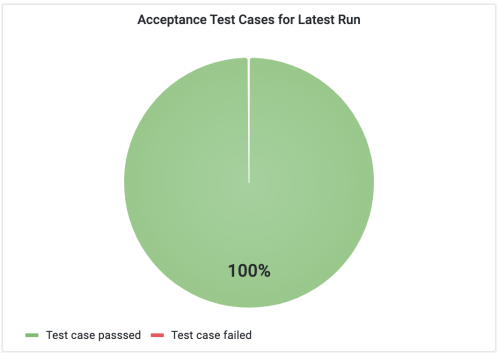
5.6.10 Production deployment

This metric is displayed using a line chart as shown in figure 5.21. This chart helps the in analysing how many times a product is released the production environment. For example, on 08/03, only one time the product is released and on 08/04, five times the prodcut is released to the production environment.

Figure 5.18: Number of passed acceptance tests



(a) Number of failed test cases



(b) Number of passed acceptance tests

Figure 5.19: Test case result for latest run

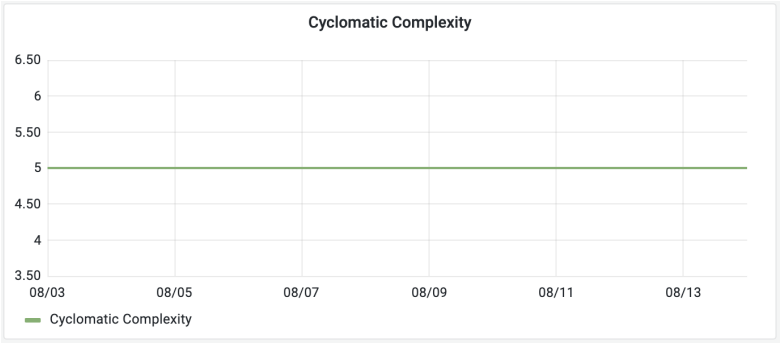


Figure 5.20: Cyclomatic complexity

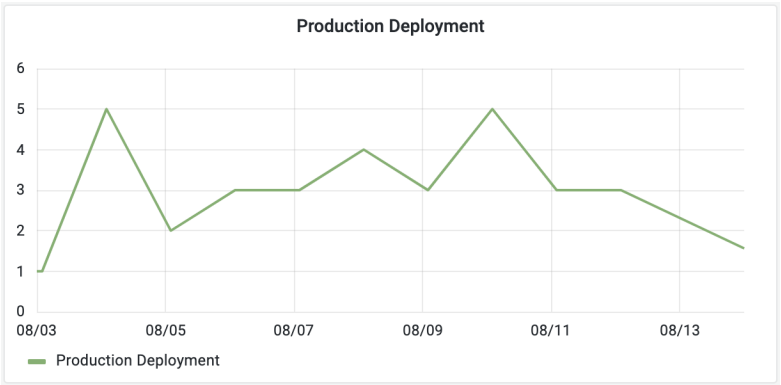


Figure 5.21: Production deployment

Chapter 6

Threats to validity

This chapter discusses the threats to the validity of the thesis, specifically whether any internal, external, or construction elements may have biased the findings. Analysing the threats to validity is crucial since the researcher reading this thesis will be aware of the many biases involved, allowing them to better their own study by using more effective countermeasures to avoid such threats.

6.1 Construct validity

When the various layers of the construct have not been specified, making it hard to provide a clear operational description, or when insufficient reasons have been provided for its content and scope, it is said that the construct of a research study is inadequately defined.

It is concerned on how well the RR process was designed to address the research questions(RQ1, RQ2 and RQ3) with respect to the metrics related to a continuous delivery . The RR was designed following the guidelines provided by Kitchenham et al. in [25] about how to perform a SLR. These guidelines were adapted according the properties that characterise a RR as explained Cartaxo et al. in [4].

Nevertheless, the the study suffers of some limitations. There is always a risk that a potentially relevant piece of work (paper or book)

is excluded by the exclusion criteria. In this RR the main factors are related to the time span for which works are included (2017-2021) and type of works (only research papers). Having initially inspected the numbers of works obtained for the period 2011-2022, it was decided to take only from the last five years (70% belong to this period) to observed the most recent results with reasonable grounds to believe no relevant works would be missed. Books were left aside to complete the RR in a timely manner. It is acknowledged that not including books represents a limitation to the study. However, a quick analysis of the books (based on title) revealed that most of them (exceptions are [16] and [44]) are technology-oriented books rather general research-oriented books. This gives certain level of confidence that not relevant content was missed for the purposes of the study [1].

Similarly, a RR performed to answer the research question RQ4. Only papers for 2 years (2020-2021) are included to find the chart types to visualise each of the measured metric. Though the time frame is very less. However, according to Cartax [4], limiting the search with date is a good practice in RR to speed up the RR process. There was good amount of papers found for the two years of time span and content extracted from these papers was adequate to answer the research question. This increases the likelihood that no information relevant to the study's objectives was overlooked. To avoid the bias in making the selection of chart to display the measured metrics, a survey is conducted.

The survey's closed-ended questions posed a threat to construct validity. As a result of closed-ended questions, the richness of the responses may be affected. However, to minimize this, an additional option was provided to participant to write their answers apart from the predetermined set of options. Also, the benefits of closed-ended questions outweighed the drawbacks: Closed-ended questions are more straightforward to answer and analyse [24]. Hence, we focused on closed-ended questions. The survey was designed following the guidelines provided by Mark et al. in [24] about how to perform a survey research. Also to minimize the bias in the questions and options

[1]Reminder: to observe what the scientific community has done up to now to determine what are the metrics to be measured in the area of *continuous delivery*.

given as part of survey, a pretest is carried with a limited number of participants before conducting an actual survey.

6.2 Internal validity

Internal validity refers to a study's ability to demonstrate a causal connection between the experiment and the observed result [42]. It also comprised of the internal decisions made by the author that consequently affects the results obtained.

Firstly, it is concerned on how the RR is conducted, in particular on how the quality selection, data extraction and synthesis were made, and whether there are factors that might have caused some degree of bias on the overall process. To mitigate the threads on these factors, two stakeholders have performed the selection and quality appraisal of the papers according to the defined inclusion and exclusion criteria. Thus, this represents a step forward in turning the RR into a SLR. Other characteristics of the RR that have been extended to be aligned with the manner in which SLRs have to be performed were: the stakeholders profiles (practitioner and researcher), and the manner in which the results are reported (research paper). In case of disagreement among a selected paper, a discussion was made to reach consensus about the final decision. Thus, it is trusted that not only the fact that multiple stakeholder perform (some) of the steps, but also their different profiles have helped to reduce the bias.

Secondly, certain tools are used, and some instrumentation are done for measuring the metrics, and they are specified in the chapter 4. The tools used in the thesis are mainly chosen only because they are free and open-source. The choice of tools and their instrumentation are also important factors in the internal validity.

Thirdly, tools and instrumentation are employed to analyse the measured metrics, as detailed in chapters 4 and 5. The tool used for this was chosen primarily because of the reference given in the original study, where that tool can be found via the RR. The type of chart used to visualise each metric is determined by the survey results, as mentioned in section 5.5.

6.3 External validity

External validity is the degree to which a study's findings can be generalised to other measurements, contexts, or populations [42]. In other words, can the findings of the study be generalised?

The structure of the thesis, i.e., the procedure by which the RR is done to determine the metrics related to the continuous delivery, then deciding which metric can be measured using the data generated by the deployment pipeline, and then the instrumentation, measurement, and visualisation of the metrics using a software product is a sound research methodology. Here, each input to a process is backed by solid evidence from the empirical papers. Consequently, the research methodology could be generalised to solve a wide variety of challenging problems.

Additionally, the instrumentation and the tools used to measure and visualise the metrics are all free and open-source software. Therefore, this procedure could be generalised in any such software environment.

Chapter 7

Related works

In this chapter, an overview of the research works related to this thesis is provided.

In the research paper titled "Defining metrics for continuous delivery and deployment pipeline" published in 2015 [27] aim at addressing the metrics that are suited for supporting continuous delivery and deployment through a descriptive and exploratory single case study. The metrics for the pipeline is divided into two categories: (1) Metrics on the Implementation Level - Development time, Deployment time, Activation time, and Oldest done feature and (2) Metrics on the Pipeline Level - Features Per Month, Releases Per Month, Fastest Possible Feature Lead Time. The paper discuss about the implementation of deployment Pipeline of the Project is provided in a general way. However, there is no adequate information on how the metrics are operationally measured and also tools used for the instrumentation of the metrics are also not available. Whereas in this thesis, we have performed an RR and a content analysis to obtain the metrics related to continuous delivery, and some of the metrics using the data generated by the deployment pipeline are made operational.

The second work corresponds to Lehtonen et al. [28] who have defined metrics meant to be measured using data obtained from the execution of tools. Some of the proposed metrics "seem" to be semantically equivalent to Accelerate's: i.e. *Fastest Possible Feature Lead*

Time, and *Releases Per Month* seem to measure the same as *Delivery Lead Time* and *Deployment Frequency*, respectively. However, as the definitions of these metrics (on both works) are not formally provided it is impossible to precisely determined whether they are measuring exactly the same or not.

Chapter 8

Conclusion

This thesis focuses on finding the state-of-the-art metrics related to a continuous delivery and valuating which ones can be measured using data produced by the execution of the deployment pipeline. To achieve that, an elaborated Systematic Literature Review is done to find the metrics related to a continuous delivery, and only the empirical papers were considered for the review. Additionally, what these metrics allow us to conclude is also documented. None of the paper which made into final corpus explained on the operationalisation of the metrics, to address this, a experiment is carried out by creating a deployment pipeline and measured some of the metrics.

Additionally, a RR is performed to find the types of charts available to display the measured metrics. A web-based survey is conducted to find the best suitable chart type to visualise the metrics measured. A dashboard is created to visualise the measured metrics using the best suitable chart that are resulted from the survey result. Also, a replication package is developed to re-create the experiment setup.

8.1 Future works

The future works specified in this section could be carried out as an extension to this study. The future works include the following tasks.

1. Out of the 18 collected metrics in the SLR that can be measured using data produced by the execution of the deployment pipeline, the instrumentation and measurement are taken only for ten metrics. One of the important future works includes instrumentation and measurement of the remaining metrics which are M4, M5, M6, M9, M12, M13 and M15.

2. The product used in this experiment is a java based code using maven as the build automation tool. The experiment can also be carried out using other programming languages or automation build tool.

3. Implementing the metric "delivery lead time" [28] to measure the efficiency of the deployment pipeline by measuring the performance of stages and each jobs within a stage. This is a very important metric which allows to analyse how much overhead instrumentation adds to the actual pipeline execution.

Bibliography

[1] Alpernas, K., Feldman, Y. M. Y., and Peleg, H. The wonderful wizard of loc: Paying attention to the man behind the curtain of lines-of-code metrics. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (New York, NY, USA, 2020), Onward! 2020, Association for Computing Machinery, p. 146–156.

[2] Bai, X., Li, M., Pei, D., Li, S., and Ye, D. Continuous delivery of personalized assessment and feedback in agile software engineering projects. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training* (New York, NY, USA, 2018), ICSE-SEET '18, Association for Computing Machinery, p. 58–67.

[3] Biswas, P., Saluja, K. S., Arjun, S., Murthy, L., Prabhakar, G., Sharma, V. K., and Dv, J. S. Covid-19 data visualization through automatic phase detection. *Digit. Gov.: Res. Pract. 1*, 4 (jul 2020).

[4] Cartaxo, B., Pinto, G., and Soares, S. *Rapid Reviews in Software Engineering*. Springer International Publishing, Cham, 2020, pp. 357–384.

[5] Castillo-Segura, P., Fernández-Panadero, C., Alario-Hoyos, C., Muñoz Merino, P. J., and Delgado Kloos, C. A cost-effective iot learning environment for the training and assessment of surgical technical skills with visual learning analytics. *J. of Biomedical Informatics 124*, C (dec 2021).

[6] Çalikli, G., Staron, M., and Meding, W. Measure early and decide fast: Transforming quality management and measurement to continuous deployment. In *Proceedings of the 2018 International Conference on Software and System Process* (New York, NY, USA, 2018), ICSSP '18, Association for Computing Machinery, p. 51–60.

[7] Ceccarini, C., Mirri, S., Salomoni, P., and Prandi, C. On exploiting data visualization and iot for increasing sustainability and safety in a smart campus. *Mob. Netw. Appl. 26*, 5 (oct 2021), 2066–2075.

[8] Celepkolu, M., Wiggins, J. B., Galdo, A. C., and Boyer, K. E. Designing a visualization tool for children to reflect on their collaborative dialogue. *Int. J. Child-Comp. Interact. 27*, C (mar 2021).

[9] Chakraborty, M., and Kundan, A. P. *Grafana*. Apress, Berkeley, CA, 2021, pp. 187–240.

[10] Chen, L. Continuous delivery: Huge benefits, but challenges too. *IEEE Software 32*, 2 (Mar 2015), 50–54.

[11] Chong, S., Lee, Y. H., and Tang, Y. W. Data analytics and visualization to support the adult learner in higher education. In *2020 The 4th International Conference on E-Society, E-Education and E-Technology* (New York, NY, USA, 2020), ICSET'20, Association for Computing Machinery, p. 126–131.

[12] Claps, G. G., Berntsson Svensson, R., and Aurum, A. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology 57* (2015), 21–31.

[13] Do, H., Rothermel, G., and Kinneer, A. Empirical studies of test case prioritization in a junit testing environment. In *15th International Symposium on Software Reliability Engineering* (Nov 2004), pp. 113–124.

[14] Feitelson, D., Frachtenberg, E., and Beck, K. Development and deployment at facebook. *Internet Computing, IEEE 17* (07 2013), 8–17.

[15] Feng, M., Zheng, J., Ren, J., and Liu, Y. Towards big data analytics and mining for uk traffic accident analysis, visualization amp; prediction. In *Proceedings of the 2020 12th International Conference on Machine Learning and Computing* (New York, NY, USA, 2020), ICMLC 2020, Association for Computing Machinery, p. 225–229.

[16] Forsgren, N., Humble, J., and Kim, G. *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*, 1st ed. IT Revolution Press, 2018.

[17] Gallaba, K., and McIntosh, S. Use and misuse of continuous integration features: An empirical study of projects that (mis)use travis ci. *IEEE Transactions on Software Engineering 46*, 1 (2020), 33–50.

[18] Gill, G. K., and Kemerer, C. F. Cyclomatic complexity density and software maintenance productivity. *IEEE transactions on software engineering 17*, 12 (1991), 1284–1288.

[19] Herbst, N., Bauer, A., Kounev, S., Oikonomou, G., Eyk, E. V., Kousiouris, G., Evangelinou, A., Krebs, R., Brecht, T., Abad, C. L., and Iosup, A. Quantifying cloud performance and dependability: Taxonomy, metric design, and emerging challenges. *ACM Trans. Model. Perform. Eval. Comput. Syst. 3*, 4 (aug 2018).

[20] Herodotou, C., Maguire, C., McDowell, N., Hlosta, M., and Boroowa, A. The engagement of university teachers with predictive learning analytics. *Comput. Educ. 173*, C (nov 2021).

[21] Hilton, M., Tunnell, T., Huang, K., Marinov, D., and Dig, D. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (New York, NY, USA, 2016), ASE 2016, Association for Computing Machinery, p. 426–437.

[22] Humble, J. Continuous delivery : [reliable software releases through build, test, and deployment automation], 2011.

[23] Humble, J., and Farley, D. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[24] Kasunic, M. Designing an effective survey.

[25] Kitchenham, B., Budgen, D., and Brereton, P. *Evidence-Based Software Engineering and Systematic Reviews*. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series. CRC Press, 2015.

[26] Kochhar, P. S., Thung, F., Lo, D., and Lawall, J. An empirical study on the adequacy of testing in open source projects. In *2014 21st Asia-Pacific Software Engineering Conference* (Dec 2014), vol. 1, pp. 215–222.

[27] Lehtonen, T., Suonsyrjä, S., Kilamo, T., and Mikkonen, T. Defining metrics for continuous delivery and deployment pipeline. In *SPLST* (2015).

[28] Lehtonen, T., Suonsyrjä, S., Kilamo, T., and Mikkonen, T. Defining metrics for continuous delivery and deployment pipeline. In *Proceedings of the 14th Symposium on Programming Languages and Software Tools* (2015), CEUR Workshop Proceedings, pp. 16–30.

[29] Liu, J.-C., Yang, C.-T., Chan, Y.-W., Kristiani, E., and Jiang, W.-J. Cyberattack detection model using deep learning in a network log system with data visualization. *J. Supercomput. 77*, 10 (oct 2021), 10984–11003.

[30] Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., Kuvaja, P., Mikkonen, T., Oivo, M., and Lassenius, C. Devops in practice: A multiple case study of five companies. *Information and Software Technology 114* (2019), 217–230.

[31] Mamun, M. A. A., Berger, C., and Hansson, J. Correlations of software code metrics: An empirical study. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement* (New York, NY, USA, 2017), IWSM Mensura '17, Association for Computing Machinery, p. 255–266.

[32] Masek, P., Štůsek, M., Krejčí, J., Zeman, K., Pokorny, J., and Kudlacek, M. Unleashing full potential of ansible framework: University labs administration. vol. 426.

[33] Morales, J. A., Yasar, H., and Volkman, A. Implementing devops practices in highly regulated environments. In *Proceedings of the 19th International Conference on Agile Software Development: Companion* (New York, NY, USA, 2018), XP '18, Association for Computing Machinery.

[34] Mostafa, S., and Wang, X. An empirical study on the usage of mocking frameworks in software testing. In *2014 14th International Conference on Quality Software* (Oct 2014), pp. 127–132.

[35] Nurwidyantoro, A., Shahin, M., Chaudron, M., Hussain, W., Perera, H., Shams, R. A., and Whittle, J. *Towards a Human Values Dashboard for Software Development: An Exploratory Study.* Association for Computing Machinery, New York, NY, USA, 2021.

[36] Pal, T. Topo pal's devops metrics and measurements playlist, 2022.

[37] Pham, S., Avelino, M., Silhavy, D., An, T.-S., and Arbanowski, S. Standards-based streaming analytics and its visualization. In *Proceedings of the 12th ACM Multimedia Systems Conference* (New York, NY, USA, 2021), MMSys '21, Association for Computing Machinery, p. 350–355.

[38] Protopsaltis, A., Sarigiannidis, P., Margounakis, D., and Lytos, A. Data visualization in internet of things: Tools, methodologies, and challenges. In *Proceedings of the 15th International Conference on Availability, Reliability and Security* (New York, NY, USA, 2020), ARES '20, Association for Computing Machinery.

[39] Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., and Männistö, T. Devops adoption benefits and challenges in practice: A case study. In *Product-Focused Software Process Improvement* (Cham, 2016), P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, and T. Mikkonen, Eds., Springer International Publishing, pp. 590–597.

[40] Shepperd, M. J. *Foundations of software measurement.* Prentice Hall, 1995.

[41] Singh, C., Gaba, N. S., Kaur, M., and Kaur, B. Comparison of different ci/cd tools integrated with cloud platform. In *2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence)* (2019), pp. 7–12.

[42] Slack, M., and Draugalis, J. Establishing the internal and external validity of experimental studies. *American Journal of Health-System Pharmacy 58* (11 2001), 2173–2181.

[43] Sun, B., and Pang, C. A visual analytics approach to explore potential anomalous behavior in corporate communication. In *2021 The 5th International Conference on Compute and Data Analysis* (New York, NY, USA, 2021), ICCDA 2021, Association for Computing Machinery, p. 119–128.

[44] Swartout, P. *Continuous Delivery and DevOps – A Quickstart Guide: Start your journey to successful adoption of CD and DevOps, 3rd Edition.* Packt Publishing, 2018.

[45] Than, P. P., and Phyu, M. P. Continuous integration for laravel applications with gitlab. In *Proceedings of the International Conference on Advanced Information Science and System* (New York, NY, USA, 2019), AISS '19, Association for Computing Machinery.

[46] Toh, M. Z., Sahibuddin, S., and Bakar, R. A. A review on devops adoption in continuous delivery process. In *2021 International Conference on Software Engineering Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM)* (Aug 2021), pp. 98–103.

[47] Trochim, W., and Donnelly, J. *The Research Methods Knowledge Base.* Cengage Learning, 2006.

[48] Trochim, W. M. K., and Donnelly, J. P. Research methods knowledge base.