Collecting Data from Continuous Practices: an Infrastructure to Support Team Development

Ana Filipa Nogueira*†, Emilien Sergeant†, José C. B. Ribeiro‡, Mário A. Zenha-Rela* and Antoine Craske†

* CISUC, University of Coimbra, Portugal
† DOSI, La Redoute S.A., France
‡ CIIC, Polytechnic Institute of Leiria, Portugal

***Abstract*—Through software analytics, raw data with low value originates information that is valuable and able to provide insights, enabling the support of claims that would otherwise not be possible to verify. The software development ecosystem has plenty of sources that can help understanding the quality of processes and products but, to reach that goal, it is necessary to collect and store the data. This paper describes an infrastructure to allow the collection, storage and analysis of data from software repositories. The scope of the research is an industrial case study, which encompasses several specificities: tools and work methodology. The current solution is able to collect information from the continuous delivery & deployment pipeline, which includes data sources such as the source code repository (SVN), the static analysis tool (SonarQube), the continuous integration server (from Jenkins jobs) and the continuous testing tool (an in-house tool called Cerberus). Future work also includes the implementation of components that will allow the collection of unstructured data from the bug-tracking system and incident management tool. As stated in the literature, correlating the history of issues and incidents will allow the team to address, or at least identify, areas of improvement.**

***Index Terms*—CI/CD Pipeline, Mining Software Repositories, Machine Learning, Software Analytics, Software Quality**

I. Introduction

Software Analytics as a means to predict or gather information about software development activities is an engaging research field that aims to provide insights and solutions, not only for the technical topics, but also for organizational issues. In general, one's aspiration is to guide teams to enhance the quality of the products being developed and processes being executed ([1], [2], [3], [4]). In the scope of software development, the ability to perform change impact analysis is especially relevant since team members will be able to receive direct feedback from their work and at the same time forecast or identify support for other tasks such as test-case selection and prioritization [5].

Mining Software Repositories (MSR) is among the new techniques used to perform change impact analysis [6]; this technique makes use of historical data available in software repositories, such as issue tracking systems and source code repositories. Menzies and Zimmermann write about the common vision shared in the literature [4]: "data from software projects contains useful information that can be found by data-mining algorithms". As such, commit messages and bug-reports are examples of artifacts from which knowledge related to change impacts can be extracted. For instance, Ying *et al.* [7] claim that a developer may benefit from patterns in the history change of a system while performing maintenance tasks. Therefore, well-known applications scope include recommendation systems, and effort and defects predictors for software development. New fields of research include: detectors for unexpected or bug-prone code, indicators for developer's mood by applying sentiment analysis to comments, and explorers for complex spaces. As for data science in the industry, what was seen in the past as cutting-edge research is now part of daily operations of IT companies [4], [8].

With the current shift concerning automated delivery and deployment, and when moving from manual and bureaucratic processes to almost full automation, all those tools pose as data sources. Nowadays, delivery and deployment are continuous practices supported by tools [9], and the capture of events is more straightforward. In addition to mine source repositories and bug tracker systems, the spectrum of sources may include: code review tools [10], Continuous Integration/Continuous Delivery (CI/CD) tools or automated tools for continuous testing. Even though there are principles and practices transversal to companies that implement continuous practices, a company's specific characteristics influence how these practices are implemented [11], including the business domain, the type of product being developed, or the impact on the customers (e.g., some companies report that frequent releases may lead to an increase on customer churn rate).

In order to gain visibility of all the events generated by continuous practices it is necessary to implement an infrastructure to support the data collection and storage. This paper reports on a real case study, implemented for the IT department of La Redoute[1], which is an *e-commerce* company. The infrastructure presented aims to collect data from several data sources available in the continuous pipeline: source code repository, static analysis tool, CI/CD server and continuous testing tool. The paper is the continuation of previous work

DOI reference number: 10.18293/SEKE2019-192

[1]https://www.laredoute.fr/

[12]; it described the ongoing research and motivation to apply machine learning techniques to improve the quality of processes and products developed in La Redoute's IT department. A particularity of the architecture presented in this paper is that it is emerging during the digital transformation of the company, i.e., new projects moving to micro-services cloud-based architecture, in opposition to monolithic solutions implemented in the past. The infrastructure being developed is also supported by this new technological stack which includes components such as Kafka[2], Kubernetes[3], ElasticSearch, Logstash and Kibana (from the Elastic stack[4]). Considering some the advantages of software analytics and MSR, this infrastructure will support the team's need to anticipate issues (*"Is this commit considered problematic?"*) and to understand the best and worst practices in place, both technical and social (*"Do people on my team need training?"*).

The rest of paper is organised as follows: Section II overviews relevant literature; Section III describes the main requirements and the architecture being implemented, and finally, IV concludes this paper while identifying future directions.

II. Related Work

This section overviews literature that takes advantage of repositories mining in order to find patterns and to extract knowledge about the aspects influencing the software development and evolution.

Ball *et al.* [13] proposed a framework to delve into the relationships between several components of the software development process: requirements, technological stack, software development and the development's team organizational characteristics. In the scope of C++ case studies, the authors derived a VCS-related metric: connection strength determined with basis on how probable it is that two classes are modified together. Additionally, Ball *et al.* highlighted the existence of valuable contextual information that could be leveraged to understand how a component evolved from one release to another; examples include code modified, date and time of modification, and the author. The assumption is that automated analysis can also be applied to contextual data.

Hipikat tool [14] was conceived by Čubranić and Murphy as a means to aid newcomers to an open-source project, which don't have the same support net when compared to traditional *in-house* teams. The authors use the concept of *implicit group memory* that infers links between archived artifacts produced in the source repository, issue-tracking systems, communication channels and online documentation. The *implicit group memory* is then used to recommend artifacts, from the archives, possibly relevant to the task assigned to the newcomer.

Zimmermann *et al.* [15] present the tool ROSE – a plugin for Eclipse IDE[5] – that guides the programmer. An association rule mining algorithm was employed to mine the *version histories* providing the developer means to: i) "suggest and predict likely changes"; ii) "prevent errors due to incomplete changes"; and iii) "detect coupling undetectable by program analysis". As the tool is based on the files' version history, it is possible to observe another type of coupling that is not code-based: coupling between items that are not applications or programs.

Canfora and Cerulo [16] proposed a method to infer the list of impacted source files that will be impacted by a change request. The method makes use of information from the source repository and the changes requests (Bugzilla[6]) and via information retrieval algorithms, the technique makes the link between the new change request and the historical revisions impacted by similar requests. The evaluation of the method was implemented in four open source projects, and the positive results range from 30% to 78%. The prediction of the effort required to test is another possible application pointed by the authors that will help both developers and project managers.

Ren *et al.* developed Chianti [17], a tool to support change impact analysis of Java programs, integrated in the context of Eclipse IDE. For this tool in particular, regression and unit tests, and corresponding executions, are the artefacts of interest, i.e., the tool reports on how tests' behaviour are influenced by changes. For each change on a test's behaviour, the tool also determines the "affecting changes".

Ying *et al.* [7] developed an approach that relies on data mining techniques to identify patterns on the change history of the base code; specifically, the approach searches for patterns among the changes observed on the set of files that commonly change together. Based on their approach, the authors report that history change patterns support recommendation systems that indicate additional code that should be modified when a developer is doing a modification of the source code. This work is aligned with work done by Zimmermann *et al.* [15] even if applying different algorithms.

Zanjani, Swartzendruber and Kagdi [6] also presented an approach called *InComIA* to understand the change impacts of an incoming request. They aim for connections between historical data, provided in task management applications (Mylyn[7]), and histories and contextual information available in commits (via source code repositories). A *corpus* of source code entities – methods and files – was created by applying several techniques: information retrieval, machine learning and source code analysis. For an incoming request, its text is used to query the *corpus* and as a result, the tool returns a list of the most prone to change entities.

TARMAQ is an algorithm developed by *Rolfsnes et al.* [5] for mining *evolutionary coupling* (a driver for change impact analysis); it was empirically validated on six projects – 2 industrial and 4 open source – and the authors claim better results when compared with ROSE tool [15], as it worked better for heterogeneous systems. *Evolutionary coupling* aims to understand how systems evolved during their lifecycle,

[2]https://kafka.apache.org/documentation/
[3]https://kubernetes.io/
[4]https://www.elastic.co/
[5]https://www.eclipse.org/ide/
[6]https://www.bugzilla.org/
[7]https://www.eclipse.org/mylyn/

the goal is to pick data that changed "together" and mine the connections between the entities that were modified. The authors focused at connections between files, although several granularity levels would be possible (methods or variables).

Chatley et Jones [18] presented a tool called *Diggit* which is able to generate code review comments in an automated fashion; the authors used historical changes from a Git repository, using a mining algorithm to pinpoint directions on eventual changes. The tool was integrated in a development team inside an industrial case study, and the authors also reflected on the impacts of adapting an academic research into the real world to be used by developers. The authors pinpoint potential both for commercial and open-source projects (which may include a higher number of developers and number of commits during the time). Motivation for their work encompasses not only the assurance of commits' quality but also the support to improve developers' competences.

Following a different perspective, Mens and Goeminne [19] studied the social component applied on the open-source community: work methodologies, cooperation, communication and information sharing. The main motivation was to understand how communities impact the evolution of a software product, some of the events that were analysed by authors are also true for industrial case studies, examples, departure of a key developer or the handover of a project to another team.

Murgia *et al.* [20] also adopted a more social approach by mining issue reports to gather emotional information about the software development. The authors focused on the issue tracking system of Apache Software Foundation[8] and were able to observe emotions such as sadness, joy and gratitude, through a human observation of the reports. The motivation for analysing this type of information is on the fact that the lack of happiness or safety may lead developers to fall behind. Moreover, to support the importance of context in the quality, Bird *et al.* [21] report about the impacts of organizational aspects, which are seen as strong indicators of quality, refuting that geographically teams are producing worst products. Menzies and Zimmermann [4] also agree that social factors are promising quality predictors.

Another level of repositories mining is to mine repositories of repositories (RoRs). For instance, Sowe *et al.* [22] studied the types of projects being developed in the open source community, for that purpose the authors used metadata available in the RoRs FLOSSmole[9].

Our research addresses a industrial case study, i.e., the scope include projects developed by an IT department composed by almost 110 people from Development and Operations. The underlying motivation for the implementation of the infrastructure described in this document is to have means to support and improve products, processes and competences. Even though there are a few studies reporting about industrial case studies, the majority of the aforementioned research focuses on open-source case studies. Nevertheless, those are a representative sample of complex projects both in the technical and social perspectives.

[8]https://www.apache.org/

[9]https://flossmole.org/

III. Architecture

A. Continuous Delivery & Deployment Pipeline

Figure 1 depicts a general view of the delivery & deployment pipeline implemented by the IT team; it is supported mainly by Jenkins[10], integrated with the other tools so as to allow the build, static analysis, deploy and testing of the components being delivered and deployed.

The pipeline is triggered immediately upon a commit on the version control system, SVN[11]. Jenkins listens for modifications on the code repository and initiates the build of the component after each new commit. In the same job instance, after the build, the component is submitted to SonarQube [12], which, in turn, executes the static analysis (reports about technical debt, test coverage, complexity) and enforces quality rules that should be respected. If the build and quality rules are valid, the project is deployed in a first non-production environment QA – *Quality Assurance*, so that developers can perform their first tests in a machine other than theirs. A successful deployment triggers a functional test campaign, ensuring non-regression, implemented through an *in-house* tool named Cerberus [23]. The global output of the test campaign – OK or KO – is used to decide if the component should be installed in the following environments in the pipeline. After QA, the team can move their components to the next environments – UAT and PROD. UAT or *User Acceptance Tests* is a pre-production environment allowing to perform more tests in a confined environment.After the deploy of a component in the UAT environment, the corresponding automated test campaigns will be triggered as well. For Production, it is also possible to execute automated tests.

B. Architecture

Altogether, the proposed architecture (Figure 2) collects structured data about: the commits, the Jenkins jobs status for build, deployment and test campaigns; SonarQube measurements and the summary for test campaigns executions.

As mentioned in the Introduction, the architecture described in this paper is implemented in a Kubernetes cluster, which is an "open-source system for automating deployment, scaling, and management of containerized applications". This type of infrastructure eliminates most of the manual work required to set up and modify an infrastructure, in opposition to traditional Virtual Machines. Thence, if necessary, a new resource for data collection, treatment or analysis can be easily set up and integrated in the infrastructure.

The entry or starting point of the data collection process is the Kafka component – identified as "Data pipeline". Kafka is a distributed streaming platform that allows, among other features, to implement the streaming of data pipelines in real-time. For this specific architecture, the jobs in the Jenkins

[10]https://jenkins.io/

[11]https://subversion.apache.org/

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

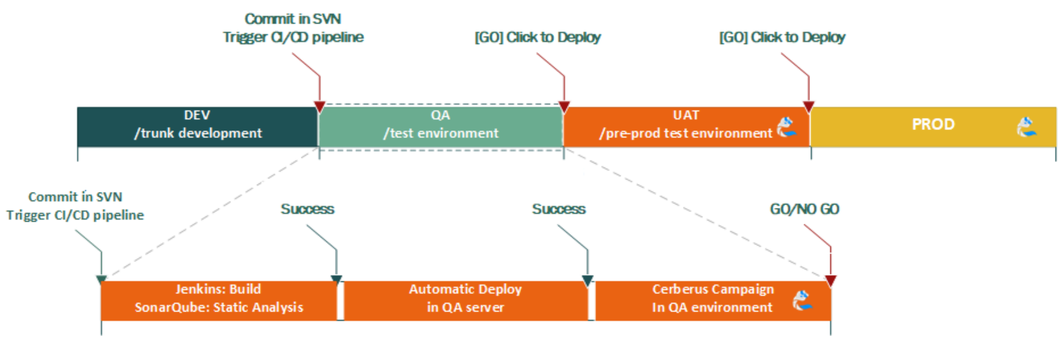

Fig. 1. Overview of development pipeline supported by Jenkins, SVN, SonarQube and Cerberus.

pipeline send data to a specific Kafka topic called "*jenkins*". A *topic* defines a stream that holds specific types of events or messages. An advantage is that any consumer can pick a message or event in a topic and treat it according to their specific use case. The potential of streaming the events triggered in the continuous pipeline is huge, for instance, the "*jenkins*" topic is also supporting some release engineering tasks, since the team is able to extract statistics about the team's velocity and the time necessary to reach production environments.

In this architecture, the consumer of the "*jenkins*" topic is referred as "Data filter", which is a Logstash service that reads the messages and extract the meaningful data for this research. Logstash is an *Extract, Transform,* and *Load* (ETL) component. The "Data filter" will parse and load the useful data into a component implemented in Java, which in turn is responsible for correlating the data received with metrics from: i) the code quality analysis and ii) the result of the functional test campaign. This step will promote the construction of a complete data set that will serve the purpose of the system. The data will then be stored inside a special component, the "Data Repository", which is an ElasticSearch engine that stores data as JSON documents. ElasticSearch allows quick searches and integrates very well with other technologies simplifying both data analysis and visualization tasks.

The second half of the architecture ("Decision Making Process"), represents the components that can consume data from the "Data repository". It is expected to use Machine Learning techniques to predict the severity level of the events happening in the continuous pipeline. For that purpose, a first TensorFlow[13] component is being implemented in the Kubernetes cluster. Additionally, it is important to provide means to visualize the data, and for that purpose we expect to implement useful dashboards. To begin, Kibana is also available in the cluster, facilitating the access and visualization of the data available in "Data Repository".

With basis on this architecture, the next step is to implement all the processes and components that will analyse the data and provide valuable insights to users, either developers or managers. For that purpose, future work will need to ensure some basic requirements [24] [18]:

- *Short execution times*: if the analysis takes too long, developers will feel tempted to abort it and to move on.
- *Short number of false positives*: a high number of false positives may lead to the abandonment of the framework.

C. Other data sources

Other data sources included in the "Data repository", but that are currently being manually loaded include:

- *Developers experience*: the different developer categories are used as input. Developer's data needs to be managed carefully to avoid exposing personal details. Future work may include information about the managers and team organization in order to understand the organizational impacts.
- *Impact of the project*: currently being loaded from an excel file; however, there is an ongoing project to implement a database repository which will ease the management of this type of information. Also, it is necessary to ensure that the owners of each software component share accurate information about the impacts and severity for the business.

A next step for this architecture includes the development of specific connectors to gather non-structured data from: i) Mantis – the bug tracking system for non-production issues; ii) *iTop* – system used to report issues detected in PROD environment; and iii) logs – any type of execution logs.

D. Early Observations

Even though is to soon to comment on the data collected and the insights that are possible to infer, it is interesting to report a curious observation. Linear regression was applied to two variables: i) the result of the Jenkins build job, and ii) the time of the day that a commit was done. An unexpected high number of build jobs, which are triggered by the commit, failed in the hour after the lunch break. A possible research

[13]https://www.tensorflow.org/

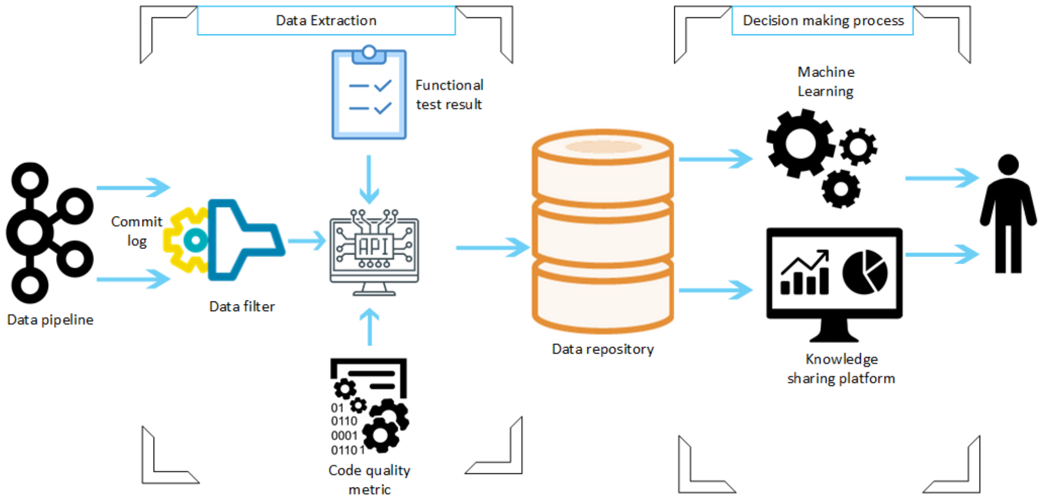

Fig. 2. Architecture to collect data from the CI/CD pipeline.

direction: What are the impacts of interrupting the "thinking flow" of a developer?

IV. Conclusions and Future Directions

Even though software analytics opens several possibilities for "discovering, verifying, and monitoring the factors that affect software development", there are several underlying issues [4]. The following paragraphs describe how those issues relate to the work described in this paper.

Even though the described architecture is prepared to collect data from several sources in the pipeline, there are factors that are not visible in those components nor collected in an automated manner. For that purpose, two surveys were conducted to obtain information that is not available in the tools used, but that is more related to the social component and team's characteristics. This type of activity is time-consuming, but the surveys results are expected to uncover not only the points of improvement, but also the good practices implemented by the company.

The context may be another issue, as it may differ among projects, even when handled by the same team. Currently, we are restricting the research scope to new projects that have a high-level of readiness to integrate a micro-services architecture. Nevertheless, the legacy code, which is distributed across a panoply of projects using different technological stacks, represents a big part of the scope, and it may hinder bad practices that are propagated and need to be addressed by the team. This is a topic that needs to be monitored closely in order to understand how can we transfer the knowledge acquired from a project – that fits the standards of the new architecture – to a legacy project.

A frequent goal of this type of research, which focuses on the software development data, is to have actionable insights, i.e., how can we use the information available to take some real and concrete actions. This work is expected to provide information that can help teams reduce the overhead of manual tasks, raise automatic alerts in case of need, and also identify potential needs in terms of training. These actionable actions will impact the continuous practices and drive business value, as an increase on the quality of processes and products is foreseen.

References

[1] H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *J. Softw. Maint. Evol.*, vol. 19, pp. 77–131, Mar. 2007.

[2] T. Menzies and T. Zimmermann, "Software analytics: So what?," *IEEE Softw.*, vol. 30, pp. 31–37, July 2013.

[3] M. A. de F. Farias, R. Novais, M. C. Júnior, L. P. da Silva Carvalho, M. Mendonça, and R. O. Spínola, "A systematic mapping study on mining software repositories," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, (New York, NY, USA), pp. 1472–1479, ACM, 2016.

[4] T. Menzies and T. Zimmermann, "Software analytics: Whats next?," *IEEE Software*, vol. 35, pp. 64–70, Sep. 2018.

[5] T. Rolfsnes, S. D. Alesio, R. Behjati, L. Moonen, and D. W. Binkley, "Generalizing the analysis of evolutionary coupling for software change impact analysis," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, pp. 201–212, March 2016.

[6] M. B. Zanjani, G. Swartzendruber, and H. Kagdi, "Impact analysis of change requests on source code based on interaction and commit histories," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, (New York, NY, USA), pp. 162–171, ACM, 2014.

[7] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Trans. Softw. Eng.*, vol. 30, pp. 574–586, Sept. 2004.

[8] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "The emerging role of data scientists on software development teams," in *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, (New York, NY, USA), pp. 96–107, ACM, 2016.

[9] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access*, vol. 5, no. Ci, pp. 3909–3943, 2017.

[10] X. Yang, R. G. Kula, N. Yoshida, and H. Iida, "Mining the modern code review repositories," *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16*, pp. 460–463, 2016.

[11] M. Leppnen, S. Mkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. Mntyl, and T. Mnnist, "The highways and country roads to continuous deployment.," *IEEE Software*, vol. 32, no. 2, pp. 64–72, 2015.

[12] A. F. Nogueira, J. C. B. Ribeiro, M. Z. Rela, and A. Craske, "Improving la redoute's CI/CD pipeline and devops processes by applying machine learning techniques," in *11th International Conference on the Quality of Information and Communications Technology, QUATIC 2018, Coimbra, Portugal, September 4-7, 2018*, pp. 282–286, 2018.

[13] T. Ball, J.-M. K. Porter, and H. P. Siy, "If Your Version Control System Could Talk ...," in *ICSE Workshop on Process Modeling and Empirical Studies of Software Engineering*, 1997.

[14] D. Cubranic and G. Murphy, "Hipikat: recommending pertinent software development artifacts," *25th International Conference on Software Engineering, 2003. Proceedings.*, vol. 6, pp. 408–418, 2004.

[15] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, (Washington, DC, USA), pp. 563–572, IEEE Computer Society, 2004.

[16] G. Canfora and L. Cerulo, "Impact analysis by mining software and change request repositories," *Proceedings - International Software Metrics Symposium*, vol. 2005, no. Metrics, pp. 261–269, 2005.

[17] Xiaoxia Ren, B. Ryder, M. Stoerzer, and F. Tip, "Chianti: a change impact analysis tool for Java programs," *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pp. 664–665, 2005.

[18] R. Chatley and L. Jones, "Diggit: Automated code review via software repository mining," *25th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2018 - Proceedings*, vol. 2018-March, pp. 567–571, 2018.

[19] T. Mens and M. Goeminne, "Analysing the evolution of social aspects of open source software ecosystems," *CEUR Workshop Proceedings*, vol. 746, no. January 2011, pp. 1–14, 2011.

[20] A. Murgia, P. Tourani, B. Adams, and M. Ortu, "Do developers feel emotions? an exploratory analysis of emotions in software artifacts," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, (New York, NY, USA), pp. 262–271, ACM, 2014.

[21] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? an empirical case study of windows vista," in *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, (Washington, DC, USA), pp. 518–528, IEEE Computer Society, 2009.

[22] S. K. Sowe, L. Angelis, I. Stamelos, and Y. Manolopoulos, "Using repository of repositories (rors) to study the growth of F/OSS projects: A meta-analysis research approach," in *Open Source Development, Adoption and Innovation, IFIP Working Group 2.13 on Open Source Software, June 11-14, 2007, Limerick, Ireland*, pp. 147–160, 2007.

[23] Cerberus 2011-2019, "An open source, user friendly, automated testing tool." https://www.cerberus-testing.org/index.php/en/. Online; accessed 01 March 2019.

[24] C. Calcagno and D. Distefano, "Infer: An automatic program verifier for memory safety of c programs," in *Proceedings of the Third International Conference on NASA Formal Methods*, NFM'11, (Berlin, Heidelberg), pp. 459–465, Springer-Verlag, 2011.