Scientific Software Engineering: Mining Repositories to gain insights into BACARDI

Lynn von Kurnatowski [1], Martin Stoffers[1], Martin Weigel[2], Michael Meinel[1], Yi Wasser[2], Kathrin Rack[1], and Hauke Fiedler[2]

[1]German Aerospace Center, Simulation and Software Technology, first_name.last_name@dlr.de

[2]German Aerospace Center, Space Operations and Astronaut Training, first_name.last_name@dlr.de

Abstract— **For Space Situational Awareness, the German Aerospace Center (DLR) develops the software system "Backbone Catalogue of Relational Debris Information" (BACARDI), which allows for keeping track of resident space objects. BACARDI's key features are automated processing services which produce orbit information and products like collision warnings. We present how we applied new methods of software analytics to the BACARDI project.**

BACARDI is an example of a complex software system with large development effort carried out by a team of various specialists. Our goal is to design and implement an efficient software development process, balancing the explorative character of a research project and operational requirements (i.e. tailored from official standards in the aerospace domain). Therefore, we established a software development process for the project where we focus on software quality. We applied methods to structure, communicate, and utilize the diverse skills, knowledge, and experience in the team concisely and precisely. After one year of practical utilization, we analyzed the process based on the repository data. By analyzing these data, we assess and prove the effects of the introduced process on the development of a software, which is used in the aerospace domain.

TABLE OF CONTENTS

1. INTRODUCTION

Software is an important innovation factor and an essential component of modern research. Consequently, development of software is becoming an integral part of research across all scientific fields. "Based on an internal survey from 2005, we know that within the German Aerospace Center (DLR) more than 25% of the personnel costs are spent on software development." [1]

978-1-7281-2734-7/20/$31.00 ©2020 European Union

Nevertheless, software development is often treated as simple tooling to automate tasks. Particularly software properties which contribute to its sustainability like re-usability, maintainability, and extensibility are not prioritized or often not even considered. However, all these properties must be addressed if software artifacts are to be maintained over a longer time period. In consequence researchers of all fields are increasingly faced with software engineering challenges and have to maintain processes for which they have not been trained [1].

The "Backbone Catalogue of Relational Debris Information" (BACARDI) software was initially developed as a demonstrator within a DLR project. During this phase the software was developed without a formal development process in place. Therefore, some factors like maintainability were not considered sufficient during development of the prototype. After the successful development of a prototype, the project has been continued as a long-term project with the goal of developing a functional and production-ready software, but still being a research software with exploratory character. Consequently, a software engineering process has been designed and applied to the BACARDI project to balance the character of a research project and operational requirements.

After the introduction of this process we recognized a first pattern that indicates effects of the established process. Figure 1 shows the number of commits from July 2015 to September 2019. From July 2018 a turning point can be identified by an increase in the total number of commits to the project. However, this is only a first indication. Conclusions cannot be drawn by this chart alone.

In the following paper we want to analyze and evaluate the implemented software engineering methods. Thereby we address the following questions:

1. How consistently was the introduced software engineering process applied during development?
2. Is the introduced software engineering process resulting in more interaction between developers?

The remaining paper is structured as follows:

- First, we discuss what represents the term software analytics (Sect. 2) and give a broad overview of the BACARDI project including the used software engineering process (Sect. 3).
- Next we describe the generation of the data set (Sect. 4) and explain the specific metrics (Sect. 5) used to evaluate the process.
- In the main part of this paper, we present the results (Sect. 6) and discuss the research questions in their context (Sect. 7).
- Finally, we summarize the major findings and indicate

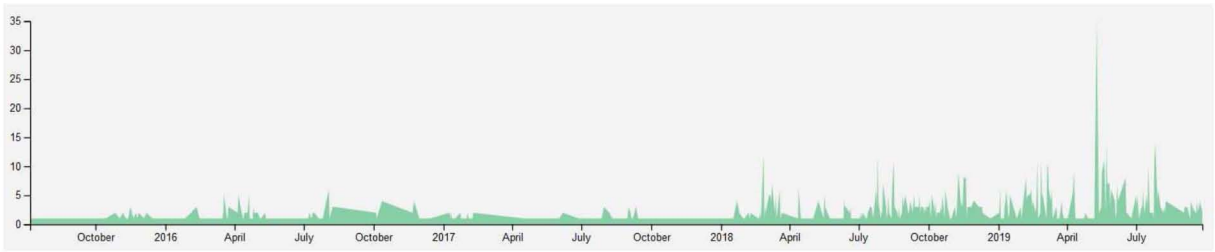

Figure 1. Number of commits from July 2015 to September 2019

future work directions (Sect. 8).

2. Software Analytics

Software development is a highly complex process involving a wide range of responsibilities and people. In addition the complexity of the software itself grows over time. To cope with this different tools are used to support the development process. During the entire software development process all these support tools produce several different types of data. These large amounts of data, which are generated before, during, and after the development of a software, can be analyzed.

In the area of software analytics, there is an increasing trend to utilize and analyze the data generated during the development process. The goal is to optimize the development process of software systems. These heterogeneous data can be used to support development activities more effectively and help to improve the decision-making process for further software development and evolution. The Microsoft Research Asia Group has defined the term software analytics as followed:

"Software analytics aims to obtain insightful and actionable information from software artifacts that help practitioners accomplish tasks related to software development, systems, and users." [2]

In order to evaluate and further optimize the entire development process, it is essential to consider the specific data about a software system, its development process, and participating developers during the analysis. For this purpose we use mining of software repositories.

Mining Software Repositories

Repository Mining is the empirical and systematic analysis of software repositories. Each repository contains a large amount of different data. The term repository does not only include source code repositories, but also issue tracker, commits, merge requests etc. In general, repository mining means to use a large amount of data which is generated during software development. This data can be used to improve the development process of software projects [3].

3. Software Engineering Process in BACARDI

BACARDI is a large-scale Python-based software platform to register and track resident space objects like space debris and satellites. It is a joint effort of DLR's Simulation and Software Technology institute and the German Space Operation Center (GSOC) [4]. The system supports different representations of orbital objects like two-line elements (TLE), osculating orbits, or ephemerides. Thus, it allows for keeping track of objects from different sources with the aim to compile a database with highest completeness of known objects with accurate and precise orbit information orbiting Earth. BACARDI achieves this by processing data from external databases as well as sensor networks all around the globe. Especially the Small Aperture Robotic Telescope Network (SMARTnet^{TM}) [5] provides tracking data from a network of ground-based telescopes operated by DLR, Astronomical Institute of the University of Bern (AIUB), and L3Harris (formerly known as Applied Defence Solution).

As mentioned in the introduction, the new project focus required a change of software development and software engineering strategies used in the BACARDI project. To establish a new software engineering process DLR's Software Engineering Guideline [6] was applied. Addressing all topics in software engineering the guideline points out important aspects for each topic. By diving software into three classes these aspects are prioritized. Since BACARDI is a software with long-term support and will be used in operational environment it is categorized as a class 2 software. According to the guideline, this requires structured requirement management, a maintainable software architecture with focus on constraints and quality, as well as a defined development processes, test automation, and rules for design and implementation. BACARDI is a research project, thus the software engineering process was adapted to avoid a complete cutdown of all research character.

Workflows, Guides and Definitions

To fulfill the guidelines a set of workflows, guides, and definitions were attuned within the development team. The workflows and definitions, which are of interest for the evaluation of the software engineering process, are introduced in the following:

Ticket Workflow: The ticket workflow describes how and where issues are handled within the project. In BACARDI, the workflow is implemented via the GitLab issue tracker.
Git Workflow: The Git workflow describes how the Version Control System (VCS) must be used. In addition, it defines the naming and handling of branches used as baseline and for feature development.
Definition of Done (DoD): The DoD is the main checklist when working on new features or reviewing them. The first half of the DoD supports the main developer of the feature. The second half supports reviewing a new feature. The DoD refers to the workflows and guides listed above.

Sprints

To structure the development in BACARDI, we use sprints as a technique. A sprint marks a short development period with

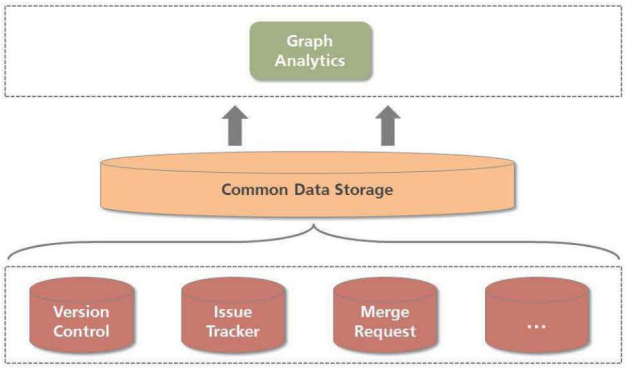

Figure 2. The process of mining software repositories.

a strong focus on previously selected tasks.

A backlog in the issue tracker collects all issues that still need to be addressed. To identify the amount of tasks that can be resolved within one sprint each team member chooses issues which should be treated in the next sprint. Afterwards all team members value the subjectively perceived complexity and add a number between 1 and 8 as comments. Thereby the number 1 means low complexity while 8 indicates a very high complexity. In the following meeting these comments are used to agree on a single complexity weight. Summing up all weighted issues, this yields a total sprint complexity and therefore an estimate of the expected work load. If the total complexity is within reasonable margins, the team has successfully defined the next sprint.

Feature development

During an ongoing sprint, team members pick the issues they want to implement themselves freely. Afterwards, a new feature branch and a GitLab *merge request* is directly created from within the issue (Appendix A). The feature branch is than checkout locally by the team member. The assigned developer implements and tests the feature in iterative steps, while adding new code to the VCS. By pushing commits to the feature branch onto the server a continuous integration system (CI) is carrying out automated tests with coverage, code analyses and builds of code and documentation.

By finishing the work, the developer goes through the first part of the *DoD* checklist to ensure that all applicable tasks were carried out. Afterwards a reviewer is assigned to the merge request, who than checks the second half of the *DoD*. This includes without limitation, to review the new code against code guidelines, to ensure that new tests are sufficient, that all documentation is available and the new code is correct. Comments to the current merge are directly annotated to the merge request and discussed and resolved in place. Once the reviewer considers all discussion as resolved the merge request is approved and assigned to the original developer. Only after a successful review the new feature can be merged into the main development branch. Once the merge was finished the related issue is closed.

4. DATA SET GENERATION

As mentioned in Section 2, we extract the relevant data by using Repository Mining. As shown in Figure 2, we first identify relevant information in these individual repositories and extract the data from them. Since the information about relations between the different repositories, in context of this work, is at least as important as the entities themselves, we selected the Neo4j[2] graph database.

We selected a graph database since relational databases are not suitable for every application purpose and every type of data. Nowadays, data often cannot be integrated into the classical grid of relational databases [7]. Graph databases utilizes a different data representation than relational databases. Entities of the same kind are identified by the same label and can be considered a class of objects. Instances of these labels are representing the nodes of the graph. Attributes can be annotated to labels and are commonly referred to as properties, hence the name "property graph". In contrast to relational databases, properties in graph databases are not limited to a certain data type. To model relationships between labels the graphs edges are used. In graph databases these edges are called relations. Relations are conceptual similar to foreign keys in relational databases. But in contrast to foreign keys, relations are named and can carry additional information, such a weight.

Implementation

The repository of our interest is managed by a single GitLab instance, thus the data was retrieved from its Application Interface (API) [8]. To map the retrieved data to labels, properties and relations, we developed the software GitLab2Graph [9]. Gitlab2Graph reads the GitLab id of the project to be analyzed as well as connection parameters for GitLab and Neo4j from a configuration file.

To automate the transformation process, the software utilizes the packages "python-gitlab"[10] and "py2neo"[11]. The first package manages the connection to the selected GitLab instance and provides access to the GitLab API by defining models for all available data structures. The second package manages the connection to the Neo4j database and provides methods to interact with it. In addition, it implements an Object Graph Mapper (OGM) to model labels including their properties and relations as Python classes. As shown in Figure 3 for an issue entity, both models and the connections to the databases are handled within a "Pipeline" class. This class is derived for each label to separate the processes of requesting data from the GitLab, transforming it and writing the finalized data to the graph database.

Graph Database Model

To ensure that a wide range of metrics can be applied to the resulting graph we decided to map almost all attributes of retrieved entities to corresponding properties at the labels. A shortened example of defined labels, with their important properties, is listed in Table 1.

The GitLab API utilizes the internal database identifiers of retrieved entities to refer to other related entities. To generate the graph representation, the corresponding label in the graph database is queried by the identifier. If the label does not exists in the graph, an empty new label with the identifier is created. Finally the defined relationship between both labels is established. The meta graph in Figure 4 shows that all

[2]**https://neo4j.com**

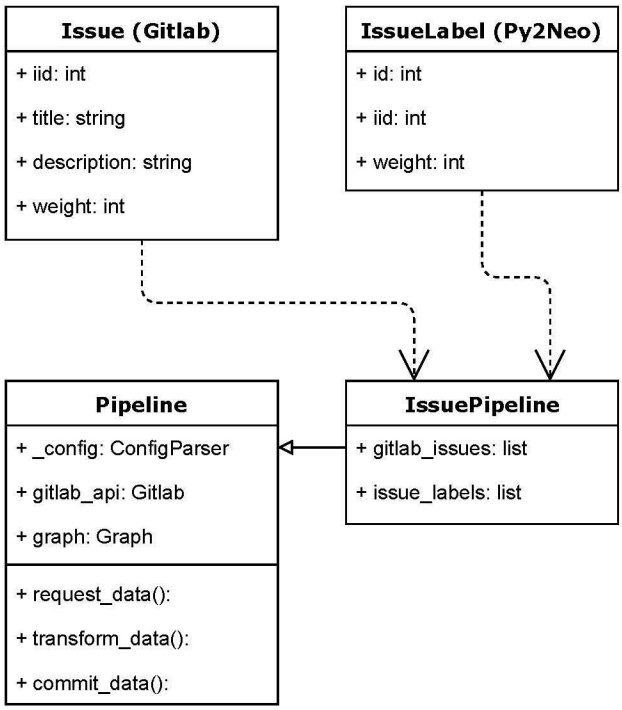

Figure 3. Classes involved creating an Issue label

Table 1. Labels with important properties

Label	Property	Type	Unique
User	id	int	yes
	name	string	
	username	string	yes
Milestone	id	int	yes
	iid	int	yes
	state	string	
	due_date	date	
	start_date	date	
Note	id	int	yes
Issue	id	int	yes
	iid	int	yes
	weight	int	
MergeRequest	id	int	yes
	iid	int	yes
	state	string	
	work_in_progress	bool	
	merge_status	string	
	approved	bool	
	created_at	date	
	merged_at	date	
Commit	id	int	yes
	short_id	string	yes

labels shown in Table 1 maintain relationships to other labels.

5. METRICS

As mentioned in Section 4, the graph database Neo4j was used to store the data needed for the following analysis. To apply these methods on the data set, we use the query language Cypher developed by Neo4j[3]. The analysis methods use the relationships between labels to reconstruct the organization and dynamics of the complex system. The whole graph is used as input.

By using graph queries, we analyze how the software engineering process is applied during development and if the introduced software engineering process results in more interaction between developers. In doing this, we focus on Merge Requests and the Issue Tracker as data artifacts.

The software engineering process was introduced at the beginning of the 3rd quarter 2018. To make a more meaningful statement about the software engineering process, we start to analyze the data artifacts half a year prior (01/01/2018). For the period 01/01/2018 to 06/31/2019 we consider the data quarterly.

Merge Requests

As mentioned in Section 3, feature development should follow a defined process. This includes both development and review. In order to analyze how the feature development process has been accepted, we use the Cypher query shown in Listing 1.

As a first step, the query checks if the general number of merge requests has changed due to the introduction of the process. To do this, the Merge Requests that were created in a specific time period are counted. In the next step, it is checked if the process was followed as described in Section 3. For this the relationships between merge requests and users is analyzed. If the process has been applied correctly, the following relationships must exist between merge request and user:

- CREATED_BY
- IS_ASSIGNED
- WAS_ASSIGNED
- APPROVED_BY
- MERGED_BY

Besides these relationships, the status of the merge request must also be verified. Since no developer is allowed to review his own code, the user which created the merge request must differ from the user who approves and reviews the merge. Since each new feature needs to be defined in an issue to a new feature, we verify that each merge request is related to an issue. Finally, we test if the merge request was correctly closed during the sprint to which it was assigned. Therefore, the date on which the merge request was merged must be less than or equal to the due date of the milestone.

```
MATCH (n:MergeRequest)
WITH*
WHERE
    '2018-01-01' <= n.created_at <=
    '2018-03-31'
WITH*
```

[3]**https://neo4j.com/developer/cypher-basics-i/**

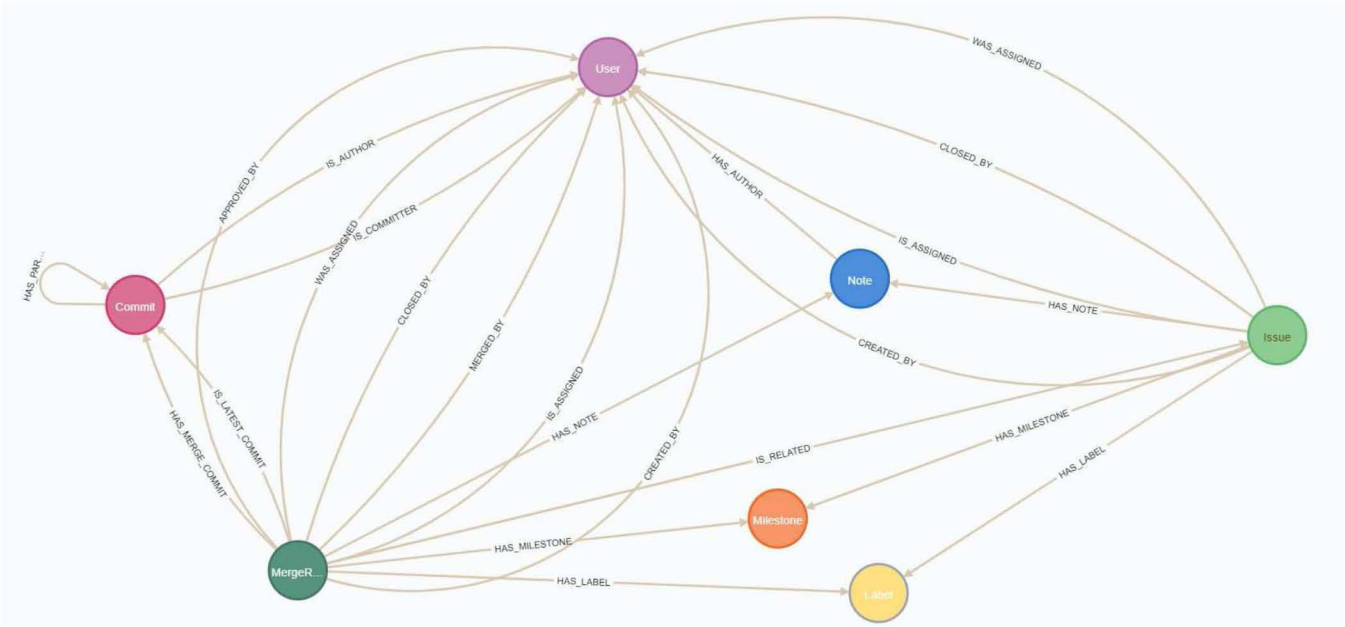

Figure 4. Meta model of the Graph Database

```
WHERE
    ((n)-[:IS_ASSIGNED]->(:User)) and
    ((n)-[:APPROVED_BY]->(:User)) and
    ((n)-[:WAS_ASSIGNED]->(:User)) and
    ((n)-[:MERGED_BY]->(:User)) and
    ((n)-[:CREATED_BY]->(:User))
WITH*
WHERE
    n.approved = true and n.state = 'merged'
WITH*
MATCH (u1:User)<-[:CREATED_BY]-(n)-[:
    APPROVED_BY]->(u2:User)
WHERE
    u1.name <> u2.name
WITH*
MATCH (n)-[:IS_RELATED]->(:Issue)
WITH*
MATCH (n)-[:HAS_MILESTONE]->(m:Milestone)
WHERE
    n.merged_at <= m.due_date
RETURN count(*)
```

Listing 1. **Cypher query to verify that the feature development process has been followed**

Issues

The process described in Section 3 explains that the BAC-ARDI team uses the GitLab issue tracker. With the Cypher query in Listing 2, we verify that the issue tracker is used as described earlier. In a first step, we monitor if the general number of issues has changed as a result of the introduction of the GitLab issue tracker. To do this, we count the issues that were created in a specific time period. In the next step, we check that every issue is annotated with a weight. To accomplish that, we check that the property weight of the issue is not null.

```
MATCH (n:Issue)
WHERE
    '2018-01-01' <= n.created_at <=
    '2018-03-31'
WITH*
WHERE
    n.weight IS NOT NULL
RETURN count(*)
```

Listing 2. **Cypher query to check if the feature development process has been followed**

Interaction

The development team does not only works together across different departments and institutes, but also across several locations, thus the software engineering process should also support interactions within the team. Therefore, we analyze in which way the use of issue tracker and reviews supports the interaction within the team. To achieve this, we count the number of notes that have a relationship to a merge request or to an issue using the queries shown in Listing 3 and 4.

```
MATCH (n:MergeRequest)-[:HAS_NOTE]->(o:Note)
WHERE
    '2018-01-01' <= n.created_at <=
    '2018-03-31'
RETURN count(o)
```

Listing 3. **Cypher query to count the number of notes that have a relationship to a merge request**

```
MATCH (n:Issue)-[:HAS_NOTE]->(o:Note)
WHERE
    '2018-01-01' <= n.created_at <=
    '2018-03-31'
RETURN count(o)
```

Listing 4. **Cypher query to count the number of notes that have a relationship to a issue**

Figure 5. Results Merge Request

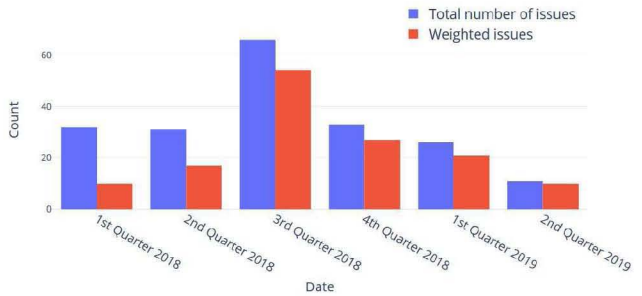

Figure 6. Results Issue Tracker

6. Results

As mentioned in the introduction, we want to evaluate the acceptance of the software engineering process. So we first reviewed the feature development process and the use of the issue tracker through the metrics described in Section 5. The precise data points of the results are listed in Appendix B.

Merge Requests

Figure 5 provides an overview of the results for each query sent to verify if the feature development process was applied. There is a considerable trend that the total number of merge requests rose with the introduction of the software engineering process. The same behavior can be seen in the other metrics. The number of process-conform merge requests, merge requests assigned to an issue, and merge requests closed before the due date of the sprint rose with the launch of the software engineering process to the end of the year 2018. Before introduction of the software engineering process in the second quarter no merge requests are closed before the due date of the sprint.

Since the introduction of the software engineering process, the total number of merge requests converges with the numbers of the other metrics. However, an conspicuous trend is noticeable in the 1st quarter of 2019. The number of merge requests closed before the due date of the milestone has fallen within a quarter. In the following quarter the number of merge requests closed before the due date of the sprint converges to the total number of merge requests but not as much as the other metrics.

Issue Tracker

The bar chart shown in Figure 6 provides an overview of our metric concerning the usage of the Issue Tracker. The total number of issues and the number of weighted issues strongly increased with the introduction of the software engineering process. After the introduction, from the 3rd to the 4th quarter 2018, a decrease of both results is visible. Starting with the introduction of the software engineering process up to the end of the year 2018, the lines of both results are almost symmetrical. From the beginning of the year 2019, the difference between both metrics converges.

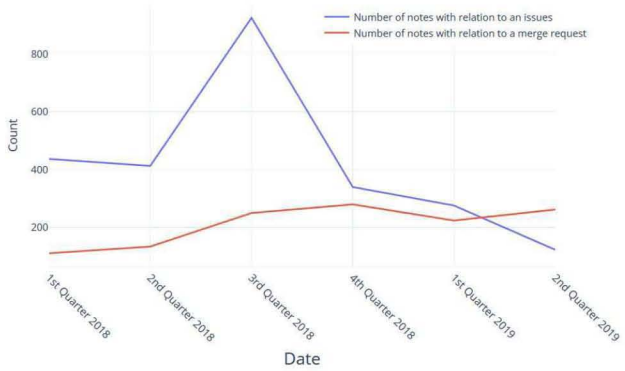

Figure 7. Results Communication

Interaction

The bar chart shown in Figure 7 provides an overview about the results of reviewing the interaction in the team. It is visible, that the interaction with issues grows rapidly with the introduction of the software engineering process. The number of notes in the issues doubled during the 3rd quarter 2019. A major peak is visible. In the following quarter the number of notes in the issues is more than halved. Interaction with the issues is reduced during the same time span. The interaction during merge requests increases slightly at the beginning of the process introduction and then stabilizes. Starting in the 2nd quarter 2019, it is obvious that the amount of interactions with issues and merge requests change in a contrary way.

7. Discussion

In the following Section, we discuss the results in context of the research questions from Section 1.

Continuous compliance of the software engineering process

With the introduction of the software engineering process the total number of merge requests and issues showed a strong increase. At the beginning of the 3rd quarter, when the process was first applied, a noticeable difference between the total number of merge requests and merge requests with a process-conform approval could be observed. This may be linked to a team which is still unfamiliar with the process. Supporting this assumption, there was a great difference between the total number of issues and weighted issues. From the 4th quarter on, a tendency can be recognized that the team gets more familiar with the previously introduced process. This can be noticed by the decreasing difference between the number of issues and weighted issues. The declined difference between the number of merge requests and process-conform merges supports the assumption, too.

However, it should be pointed out that despite a sprint planning, the number of merge requests closed within a sprint does not equal out with the number of process-conform merge requests. Therefore it is still below the desired value. This indicates that the scope of these issues is to broad for a sprint.

Additionally we conclude that the issues were not realistically weighted within the planning phase.

As mentioned before, after introduction of the software engineering process the total number of issues increased considerably. The following decline of the total number of issues indicates that after the introduction of the process the issues are predominantly closed.

As described in Section 3 issues are only weighted during the sprint planning. Thus, a difference between weighted issues and the overall number of issues will always remain. However, the difference between these two numbers gets progressively smaller, since issues already weighted during sprint planning are often not included in the sprint and instead moved into the backlog.

Interaction between developers

With introduction of software engineering process the number of notes added to issues has doubled. This suggests that the team first had to familiarize with the newly introduced methods and workflows. Hence, the process was not immediately adopted. There was much need for clarification and discussion. In the 4th quarter, the extreme decline of interaction in the issues indicates that the team responded positively to the introduced software engineering process. The following marginal decline might be an indication that the concept of sprint planning is positively accepted by the team. Presumably, most of the uncertainties are clarified during the sprint planning, thus fewer discussions occur during work on the issues.

In contrast to the number of interactions within the issues, the number of interactions during the merge requests increased only slightly at start of the software engineering process. This leads to the conclusion that at this point in time the team's main focus was on the familiarization with the process and the building of an infrastructure. Thus work on many issues was neglected. This explains the stagnating quantity of merge requests in the 3rd and 4th quarter 2018. In the 4th quarter 2018, the number of interactions during the merge requests slightly increased and slowly evens out. Therefore, we assume that the software engineering process was fully adopted at this point. In the 2nd quarter 2019 the quantity of interactions in the issues and during the merge requests change in a contrary way. This circumstance, in respect to the declining total number of issues, is an indicator that the process was followed by the team. Issues are constantly processed and the thereby generated code is reviewed.

In order to provide stronger statements on the overall findings, the data needs to be examined over a longer time period. Nevertheless, it can be noted that there is a clear tendency that the process is applied in a proper way. However, external influences on the results cannot be completely excluded.

8. Conclusion and Future Work

In this paper, we presented an approach to use repository mining to analyze the effects of an recently applied and well-defined software engineering process. We applied our approach to an long-running, research oriented software development project. In the interpretation of the results we clearly see effects of the introduced process. Especially, indications that developers needed to familiarize with the new processes and that the process was adopted in the long run were found. As an immediate outcome for the development team we can say that the sprint needs further optimizations to cope with the problem of tasks being moved into the next sprint. We also showed that it is possible to extract information from multiple repositories about how accurate parts of a process were executed by the team. This was only possible due to a strict process definition and the advanced use of tooling. Especially the GitLab ecosystem provided a good base to easily cross-reference different artifacts.

In future research we want to further analyze the type of notes extracted from the repository. This should lead to better insight on the type of interactions between the developers and how code reviews were carried out. Consequently, we want to approve or disprove our assumptions on the communication between developers.

As there are lot of other projects available in our DLR-wide GitLab, we also want to apply this approach to other projects. This requires to adapt the queries for each implemented development process. However, the process we use in the BACARDI project is tailored from the general DLR software guideline and uses general available tools. Thus we are confident to find projects with similar development processes.

APPENDICES

A. BACARDI CHANGE PROCESS

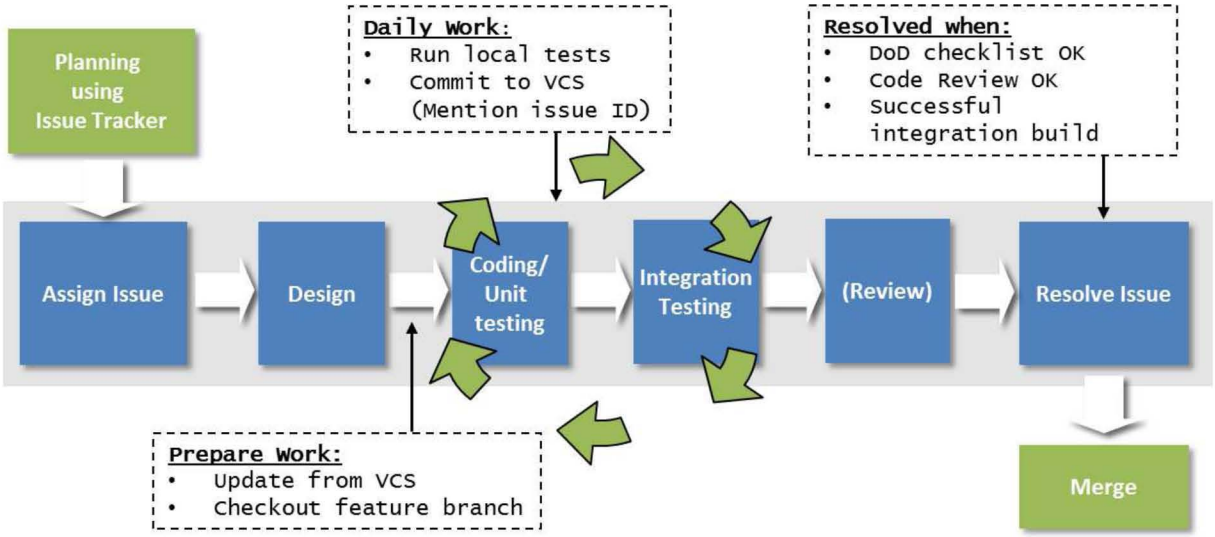

Figure 8. Change process in BACARDI

B. Result Tables

Table 2. Results Merge Request

Metric	01/01/18-03/31/18	04/01/18-06/31/18	07/01/18-09/31/18	10/01/18-21/31/18	01/01/19-03/31/19	04/01/19-06/31/19
Total merge requests	6	9	17	17	13	15
Process conform	0	6	10	12	11	13
Different authors	0	6	10	12	11	13
Related issues	0	5	9	12	10	13
Closed at milestone	0	0	6	12	6	8

Table 3. Results Issue Tracker

Metric	01/01/18-03/31/18	04/01/18-06/31/18	07/01/18-09/31/18	10/01/18-21/31/18	01/01/19-03/31/19	04/01/19-06/31/19
Number of issues	32	31	66	33	26	11
Issue weighted	10	17	54	27	21	10
Average weighting	3.4	3.5	3.5	2.6	3.2	3.2

Table 4. Results Communication

Metric	01/01/18-03/31/18	04/01/18-06/31/18	07/01/18-09/31/18	10/01/18-21/31/18	01/01/19-03/31/19	04/01/19-06/31/19
Number of notes with relation to a merge request	111	134	250	280	224	262
Number of notes with relation to an issue	437	413	925	340	276	123

REFERENCES

[1] T. Schlauch, C. Haupt, M. Meinel, and A. Schreiber, "Analytics and insights about cultivating a software engineering community at dlr," in *2019 IEEE Aerospace Conference*, ser. IEEE Aerospace Conference Proceedings, June 2019, pp. 1–12. [Online]. Available: https://elib.dlr.de/124846/

[2] D. Zhang, S. Han, Y. Dang, J.-G. Lou, H. Zhang, and T. Xie, "Software analytics in practice," *Software, IEEE*, vol. 30, pp. 30–37, 09 2013.

[3] A. E. Hassan, "The road ahead for mining software repositories," in *2008 Frontiers of Software Maintenance*, Sep. 2008, pp. 48–57.

[4] M. Stoffers, M. Meinel, M. Weigel, M. Siggel, H. Fiedler, K. Rack, and Y. Wasser, "Bacardi: A system to track space debris," in *ESA NEO and DEBRIS DETECTION CONFERENCE - EXPLOITING SYNERGIES* -, February 2019. [Online]. Available: https://elib.dlr.de/126572/

[5] H. Fiedler, J. Herzog, M. Prohaska, T. Schildknecht, and M. Weigel, "SMARTnet(TM) - Status and Statistics." International Astronautical Congress 2017 (IAC), 2017.

[6] T. Schlauch, M. Meinel, and C. Haupt, "DLR Software Engineering Guidelines," Aug. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1344612

[7] G. Jaiswal, "Comparative analysis of relational and graph databases," vol. 03, no. 8, pp. 25–27, 2013.

[8] GitLab Inc., "API Docs — Gitlab," Oct. 2019. [Online]. Available: https://docs.gitlab.com/ee/api/README.html

[9] M. Stoffers, "Gitlab2Graph," Oct. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3469386

[10] G. Pocentek, "python-gitlab," Aug. 2019. [Online]. Available: https://github.com/python-gitlab/python-gitlab

[11] N. Small, "py2neo," May 2019. [Online]. Available: https://github.com/technige/py2neo

BIOGRAPHY

Lynn von Kurnatowski *received her M.Sc. in Computer Science from Friedrich-Alexander-University Erlangen-Nuernberg. She graduated with her Master's thesis in collaboration with the German Aerospace Center (DLR) and since 2018, she is a scientific researcher at the Intelligent and Distributed Systems group of DLR Institute Simulation and Software Technology. She now works in the field of empirical and systematical analysis of software repositories with the long-term goal to optimize the general development process of software systems.*

Martin Stoffers *received his M.S. in Computer Science from Leipzig University in 2017. He joined the German Aerospace Center's (DLR) Simulation and Software Technology division the same year, where he works on software engineering and provenance recording within the BACARDI project. In late 2017, he joined the the software engineering group to support software projects of different DLR institutes and train scientists at DLR with focus on practical software engineering and sustainable software.*

Michael Meinel *joined the German Aerospace Center's (DLR) Simulation and Software Technology division for his study of information technology at the university of cooperative education Mannheim in 2004. After receiving his diploma in 2007, he worked at various DLR institutes at different sites for several years. During that time he got in touch with many different research fields. In 2012, he moved to DLR's Berlin site where he joined the software engineering group. Besides supporting software projects of different DLR institutes, he is the lead developer of* F2X*—a versatile, template-based Fortran wrapper written in Python. He is currently acquiring his M.Sc. in IT Security by distant learning.*

Dr. Kathrin Rack *(née Müller) studied theoretical physics at University of Düsseldorf. In 2011 she graduated (M. Sc.) and continued her study in computational biophysics at Forschungszentrum Jülich. As a member of IHRS BioSoft she received her doctoral degree in 2014 from the University of Cologne. Since 2015 she supports the Simulation and Software Technology department (section High Performance Computing) of German Aerospace Centre.*

Hauke Fiedler *received his Diploma degree in General Physics at the Ludwig-Maximilians-Universität in München, his Doctoral Degree in Astrophysics at the Ludwig-Maximilians-Universität in München, and his Master degree in Space System Engineering at TU Delft. At present, he is with DLRs German Space Operation Center (GSOC), leading the Space Situational Awareness Team, and project leader of BACARDI*