



Paul Scheibelmasser, BSc

Development of an open source metadata - analysis / - reporting solution for IT project management systems

Master's Thesis

to achieve the university degree of
Master of Science

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Slany, Wolfgang

Institute of Software Technology

Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Wotawa, Franz

Graz, 02 2024



Paul Scheibelmasser, BSc

Entwicklung einer Open-Source Software für Analyse und Berichterstattung basierend auf der Sammlung von Metadaten aus IT Projektmanagement-Systemen

Masterarbeit

zur Erlangung des akademischen Grades eines
Master of Science
Masterstudium: Software Engineering and Management

eingereicht an der

Technische Universität Graz

Betreuer

Univ.-Prof. Dipl.-Ing. Dr.techn. Slany, Wolfgang

Institute of Software Technology

Vorstand: Univ.-Prof. Dipl.-Ing. Dr.techn. Wotawa, Franz

Graz, 02 2024

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

27.02.2024

Date

Paul Scheidegger

Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

27.02.2024

Datum

Paul Scheidegger

Unterschrift

Abstract

In the field of software maintenance productivity and customer relations are strongly influenced by Service Level Agreements (SLA). Open challenges in this field are the poorly implemented or executed agreements in practice, caused by the complexity of modern IT landscape and the costs of the manual activities. Also, there is a lack of standardization. Varying SLA -definitions and -implementations among platforms are leading to additional effort due to complexity and often the available SLA-information is sparse. Under observation of both, the agile teams in IT companies in practise and the agile Catrobat project from TUGraz, the author implemented a software system which collects and analyses metadata from several issue tracking systems and generates reports with tracked SLA-Metrics. This thesis deals with the scientific and practical landscape of SLA tracking in modern agile software maintenance and development. It proposes a dynamic and highly extendable solution, applicable to development and maintenance in agile and non agile software teams, using state of the art issue tracking systems like Jira, Azure devOps or Github. The system will be applied in a modern IT company and at TUGraz and is meant to create benefits such as transparency in the fields of software -quotation, -planning and -support process quality as well as customer relationships.

Kurzfassung

Im Business-Bereich der Software-Wartung sind Produktivität und Kundenbeziehungen nachhaltig von Service Level Agreements beeinflusst. Der SLA-Bereich weist in diesem Kontext offene Problemstellungen auf. Eine davon ist, dass diese Vereinbarungen aufgrund der Komplexität der modernen IT-Landschaft und der Kosten für die manuellen Tätigkeiten in der Praxis oft unzureichend ausgeführt werden. Auch mangelt es an Standardisierung. Durch sich unterschiedliche Informationen und Umsetzungen von SLA-Tracking auf den Plattformen, entsteht zusätzlicher Aufwand durch Komplexität und spärliche Datenbereitstellung. Unter Beobachtung sowohl von agilen Teams in IT-Unternehmen in der Praxis als auch des agilen Catrobat-Teams der TU-Graz hat der Autor ein Softwaresystem implementiert, das Metadaten aus verschiedenen Issue-Tracking-Systemen sammelt, analysiert und Berichte mit ausgewiesenen SLA-Metriken generiert. Diese Arbeit befasst sich mit der wissenschaftlichen und praktischen Landschaft des SLA-Trackings in der modernen agilen Software-Wartung und -Entwicklung und schlägt eine dynamische und hochgradig erweiterbare Lösung vor, die für agile und nicht-agile Software-Entwicklungs- und -Wartungsteams anwendbar ist und die State-of-the-Art-Issue-Tracking-Systeme wie Jira, Azure devOps oder Github verwendet. Das System kommt in einem modernen IT-Unternehmen sowie in einem Team der TuGraz zum Einsatz und bringt Vorteile wie etwa Transparenz in der Angebotserstellung, der Planung, der Qualität der Support-Prozesse und den Kundenbeziehungen.

Contents

1	Introduction	1
2	Background and Related Work	3
2.1	Service Level Agreements (SLA)	3
2.1.1	Challenges of SLA	4
2.2	Agile Movement - influence on software landscape	4
2.2.1	Agile Maintenance Processes	5
2.2.2	Extreme Programming (XP)	6
2.2.3	Scrum	8
2.2.4	Software Development and IT Operations (DevOps)	10
2.2.5	Kanban	13
2.3	Knowledge and Data-Management as IT productivity Factor	14
2.4	Issue Tracking Systems (Jira, Azure DevOps, GitHub)	15
2.4.1	Jira	15
2.4.2	Azure DevOps	17
2.4.3	GitHub	19
2.5	SLA Management and its Realization in 3rd Level Support Process	20
2.5.1	Fields of application for SLA-Tracking	21
2.6	Summary	21
3	Solution, Architecture & Design	23
3.1	Methodology	23
3.1.1	Research - Observation and ideas	23
3.1.2	Research - Identification of problems	23
3.1.3	Research - Analysis of platforms and data	24
3.1.4	Research Technologies	25
3.1.5	Planning - Specification	27
3.1.6	Execution - Short Description	29
3.1.7	Evaluation - Issues, Limitations, Decisions, Reasoning	31
3.2	Applied Technologies/Concepts	34
3.2.1	REST-Requests	34
3.2.2	REST-Authorization	35
3.2.3	Data-Representation	35
3.2.4	Software-Architecture-Pattern	36
3.2.5	Software Selling Concepts - Open Source Licensing	37

Contents

3.3	Project Catrobat as practical application of the Master Thesis Tool	39
3.3.1	Catrobat - Infrastructure and Data Analysis	40
3.4	Summary	46
4	Documentation & Handbook	47
4.1	Project Details	47
4.2	Interfaces, Neighbour-systems and Limitations	53
4.2.1	Azure-DevOps - Configuration	53
4.2.2	Jira - Configuration	54
4.2.3	GitHub - Configuration	55
4.2.4	Output Reports - Microsoft Excel	56
4.3	Economic Benefits	57
4.4	Summary	58
5	Lessons Learned	59
6	Conclusion and Future Work	61
	Bibliography	63

List of Figures

2.1	Waterfall Software Development Life Cycle(SDLC) illustration . .	5
2.2	Extreme Programming Framework illustration	7
2.3	Scrum Framework illustration	10
2.4	DevOps related areas map	12
2.5	Jira KanBan board implementation	16
2.6	Azure DevOps KanBan board implementation	18
2.7	Azure DevOps Work Item Types(/Ticket Ticket)	18
2.8	Azure DevOps configuration for bug handling	19
2.9	GitHub Kanban-Board plugin	20
2.10	Overview relevant cloud computing areas with SLA-Metrics . .	21
3.1	Language Comparison in Computing Time on tasks	26
3.2	Problematic data quality of customized Jira-Field	33
3.3	Chart of Tickets per Project of Catrobat	40
3.4	Chart of Catrobat tickets created per month	41
3.5	Chart of Catrobat tickets updated per month	42
3.6	Chart of priority Distribution of Catrobat	43
3.7	Chart of Issue-Type Distribution of Catrobat	43
3.8	Chart of state distribution of Catrobat	44
3.9	System-state regarding Blocked-Issues for Catrobat-Jira	45
4.1	Image of the modular structure of the tool	48
4.2	Image of the UML flow of the jiraCrawler	50
4.3	Image of the UML flow of the azureCrawler	51
4.4	Image of the UML flow of the gitCrawler	52

List of Tables

3.1	Application Requirements Table	28
3.2	SLA-Metrics table	29
3.3	Program Structure table	30
3.4	Program Output table	31
3.5	Program Challenges Table	32

1 Introduction

In project management of the IT business sector, the approach has changed in the last decades from long specifications and documents to agile development practices. Modern software development has to deal with continuous delivery and deployment, extreme programming, changing third party dependencies and software landscape as well as new regulations and legal and scientific specifications. These modern requirements created the need for flexible changes, development and handling of many individual customer requests as well as problems inside a development-process of high quality and integrity. Inspired by industrial experiences with widely used project-management- and issue-tracking- systems(PMITS) like Jira, Azure DevOps, Github and comparable tools, the author has analysed business-processes of the IT business sector in the roles of a Developer, Requirements Engineer and Product Owner for more than 6 years. In large IT projects development- as well as maintenance- are organized via SCRUM and teams collaborate via Kanban boards. These boards contain large amounts of tickets for many different projects, customers and applications. Due to dependencies and waiting states, as well as neighbour-systems and priority issues, tickets often remain in a same state for a long time or need to be fixed by a strict deadline. On one hand these time blockers are a challenge for the management, as they have to be checked regularly and on the other hand for the developers who have to deal with the topics several times, for upcoming questions and further progress. Based on the Service Level Agreements(SLA) - contract with the customer, bugs and stories must be recognized, edited and solved in given time frames based on their priority and agreements. Hence if tickets in progress are not updated for a long time, the service level goal may fail and thus the customer's dissatisfaction with the offered quality will rise. The SLA-agreements are negotiated with the customer, they are country and labour law specific and influence the price of offered support service and offered software solutions. To guarantee quality in the software process, as well as maintenance and customer transparency, SLA tracking is applied to projects. Knowing statistics about currently productive projects and past projects, is a key for successful budget planning and calculation for future agreements and customer satisfaction during negotiation. During this master thesis, a system is developed which extracts metadata of project management tools, stores them in a database and creates analysis and reports per project and platform. The goal and benefit is to automatize project controlling in terms of SLA and create awareness and quality feedback for project management.

1 Introduction

Additional to higher and transparent quality standards, SLA-management contains manual activities like browsing and counting tickets, creating reports and controlling performance and billing. This results in periodic costs for each project with high risk of failures. Currently there is no public standard software which fulfills these tasks for multiple platforms. Furthermore, documentation and knowledge distribution is a challenge for long time projects in software industry which may be solved with this software. Through user-centered- and continuous - development software, customers get involved into projects during the development phase and software is directly delivered to and tested by customers. The increased interfaces and integration of customer and third party development lead to the distribution of knowledge and limited segregated documentation for systems. Only the entire system of tickets can provide a complete documentation [Mens et al., 2023].

The following chapters will show the importance of projected management tool metadata and offer solutions for reporting, processing and analysing this data. This master thesis assures quality by pointing out SLA metric violations, collecting and visualizing information of several PMITS and automatizing report generation

2 Background and Related Work

The following chapter will deal with project management methodologies and the software development landscape. It explains how in the last decades the agile manifesto has changed the industry state-of-the-art norm from waterfall to scrum. Whereas waterfall works with exact and long specifications, leading to almost linear development, scrum is based on iterative, flexible, periodic product feedback from customer. Furthermore, the environment of current software development, shaped by DevOps, will be documented and the way project management cooperates via issue tracking systems with customers and their development teams will be demonstrated. Also, it will be shown how data-management and analysis rises productivity and enables maintenance as well as high quality for customer agreements(SLA). The goal is to establish an efficient development process which uses a limited budget and resources optimally. The desired outcome is an optimal final product of high quality and maintainability, which fulfills the customer's requirements [Shaydulin and Sybrandt, 2017].

2.1 Service Level Agreements (SLA)

Service Level Agreements are written contracts between the provider of a service and the recipient of a service. They may include expectations, support scope and time-information such as response- and solve - times as well as measures of service with penalties. They define clear communication channels, expectations and enable providers to measure their quality of service. The agreements are important for good long-term customer-relationships, communication and planability/quality of support. Applied measurements are key performance indicators(KPI) which benchmark(compare against common standards) team performance of help desk [1]. In this context the Service Level Objectives (SLO) are the chosen unit of quality. Regarding on the need of a product, responsiveness, solution time or availability can be chosen metrics inside an SLA. Also special thresholds called Service Level Indicator(SLI) can be agreed. Therefore defined SLO needs to be measured and kept above agreed percentage values like availability above 99% [2]. Also internal measurements exist, like

¹<https://www.atlassian.com/itsm/service-request-management/slas>

²<https://www.atlassian.com/de/incident-management/kpis/sla-vs-slo-vs-sli>

2 Background and Related Work

the Service Level Expectations(SLE) which improve forecast, prioritisation and transparency of quality and can be integrated into Kanban boards [3].

2.1.1 Challenges of SLA

While customers have result-centered perspectives, support providers need to prioritize, manage resources and control efforts: so SLA contracts with response and solution times are often not intuitive for customers. Metrics which are calculated over longer periods lack information about handling of short time problems. Risk management is individual and difficult to plan and forecast. Price building and type of pricing is challenging as maintenance requirements may be much higher than expected or even not necessary in the end. Also services may be complex and only understandable to specialists which creates a conflict between correctness and understand ability while specifying contracts. So high non development related service costs in maintenance, detailed definitions of metrics and services as well as lack of understandable specifications are the challenges of this field [Trienekens et al., 2004].

As product documentation, costs, requirements, dependencies and maintainability are influenced by the organization of the development team and the development decisions, the next chapter will cover the modern IT landscape and organization of software projects.

2.2 Agile Movement - influence on software landscape

Agile Software development is a management approach, resulting from the problem observation of the waterfall model. Development of projects with the waterfall model were characterized by static iteration which are meant to be executed from start to end in the given way. Only in case of problems, bugs or failure a fallback to earlier phases is recommended. As figure 2.1 from ⁴ indicates, a project starts with the definition of all requirements concerning system, software and analysis, inside a specification document. These requirements are separated in loads and specifications. The procedure is independent from the size of a project. Specification documents are the so called 'loads' which describe the must haves of the solution to fulfill the contracts and 'specifications' which define the product and specifics in detail. Then there is design, implementation, testing and productivity including maintenance. As all important facts and solutions must be clear from the beginning, there is a long requirements phase,

³<https://www.scrum.org/resources/blog/kanban-service-level-expectations-and-how-use-them-scrum>

⁴<https://www.turing.com/blog/software-product-development-life-cycle/>

2.2 Agile Movement - influence on software landscape

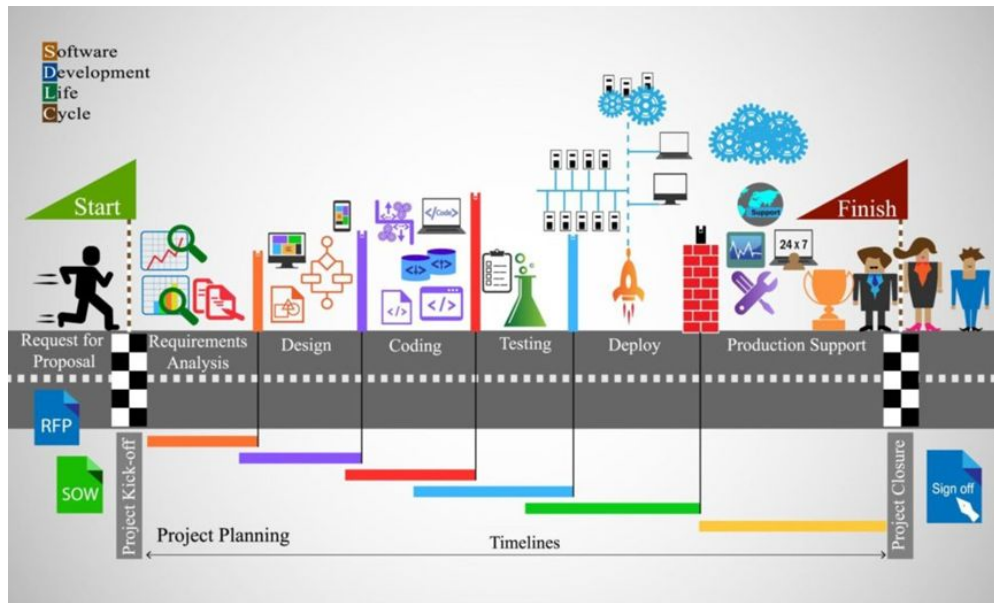


Figure 2.1: Waterfall SDLC framework [turing.com, 2024]

less specification changes during implementation phase and simple estimation and planning of time and resources for a project. A customer sees the final result at the end of the project and is typically not involved in the development and testing phase. In opposite to that, the agile software development offers periodic changes on the requirements, early customer feedback and rapid process of less complexity and rules as waterfall approaches. Agile software development is mostly used in an adapted way in industry. It is based on the spirit principles of agile which are defined in the agile manifesto and worked out by industry experts called the "agile alliance". The agile methodologies of Kanban, Extreme Programming(XP) and Scrum are a selected subset of software development methods the author observed in practice [Shaydulin and Sybrandt, 2017].

2.2.1 Agile Maintenance Processes

Scrum is a framework intended for efficient, additive development of software. Especially in combination with Extreme Programming also code quality and efficiency rise. These agile practices are not only used for development, but also applicable for the field of software maintenance. Researches (Kumar, 2015, page 27 / section 4.1) showed that maintenance benefits from applying agile practices. Compared to maintenance with waterfall approaches the work productivity may be tripled due to improved code quality, fast reactions and higher quality of iterative planning/refactoring [Kumar, 2015].

2.2.2 Extreme Programming (XP)

XP means developing software in a way that a product is continuously developed in cycles of multiple weeks. Then it is seen and reviewed by the customer and another iteration starts with new features and feedback on the system. Figure 2.2 from ⁵ visualizes these concepts. Work packages are handled via epics and underlying user stories and bugs which are realized as tickets in projects management systems in practice. Developers practise pair-programming, two persons exchange knowledge and work on one feature which has to pass tests successfully to be able to get deployed. This leads to low risk of failure and stable systems. Resource as time-, cost- and employee- planning is based on Story Points (SP) which are estimated by the team by using methods such as planning poker. They are a complexity scale and it should be considered that its estimator is maybe not the executor of the story in the end. Stories should be separated in a way that they are done during few person days. A person day means the amount a person works per day due to labour law, that means for average Austrian workers (example calculation for normal-working-hours-model: 38,5 hours divided by 5 workdays = 7 hours 42 minutes = \approx 8 hours. Before picking user stories to be implemented in the next sprint, they are prioritized based on technical blockers, dates and importance to customer. The velocity estimates, based on previous iterations, how many SP can be done in the next sprint and how long the team will need to finish the project with its SP. It is calculated using the finished SP divided by needed sprints and thus adapts resource costs during project run time [Shaydulin and Sybrandt, 2017]. in practice SP are often used to estimate development effort in person days or hours with an empiric factor based on the team's performance (frequent factors are 1.5 or 2). However studies show that this is an unreliable metric. Equivalent to XP, scrum is an individual agile practice. While Scrum focuses on organization, XP focuses on requirements and code. Code reviews and pair programming ensure that only required software and resources remain in the code. Tickets guarantee the match of customer wishes and the resulting feature and the efficiency of the employees is ensured by creating small tasks and avoiding overtime for developers [Tawosi et al., 2022].

⁵https://www.researchgate.net/figure/Extreme-Programming-Cycle_fig1_339915012

Extreme Programming (XP)

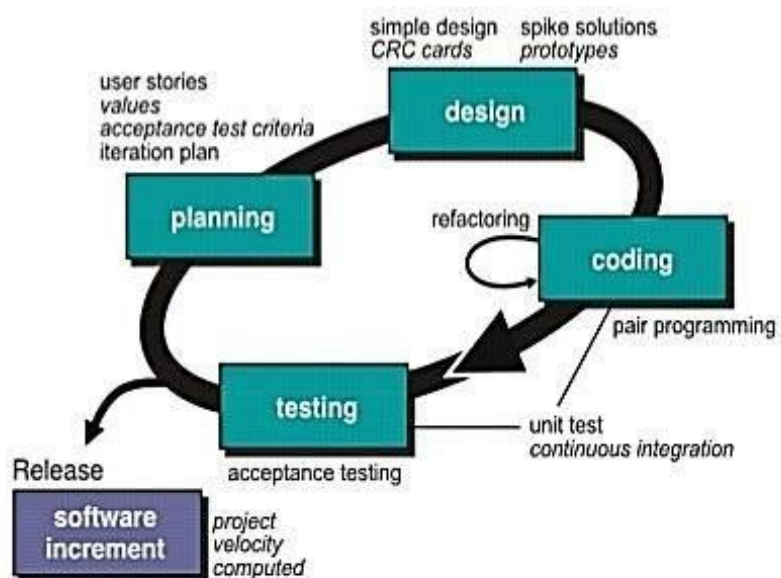


Figure 2.2: XP framework from researchgate

2.2.3 Scrum

Scrum is a management framework which is widely used in IT. Its principles can be looked up at ⁶. Mostly adapted scrum versions are used in practice, but deviations from the theory should be well justified as they can lead to hidden problems in productivity or communication. Characteristics of Scrum are that a product is developed in a process consisting of cyclic incremental phases, predefined roles, regular specific meetings about planning, current state and lessons learned and permanent productive systems which are transparent to customers. Mentioned tools have to be applied in a suggested order to reach the goal of flexible high quality solution with less prone to error and high adaption to customer needs. First of all there is a team consisting of around 5 to 8 persons with predefined roles like product owner, scrum master and the developers consisting of testers, requirements engineers and software developers. As 2.3 visualized, Scrum runs in iterations of 2-4 week duration called sprints with regular meetings at defined time frames, like start, end or daily during the sprints. Before the start of each sprint, features are specified together with the customer and requirements are stored inside the product backlog and separated into tasks inside. Tasks which have all information to be executable, can then be prioritized. Based on prioritization, a product owner selects tasks, estimates them in a meeting called sprint planning which applies a concept called planning poker. Using cards or by speaking out numbers simultaneously. Fibonacci numbers from 1 to 13 and higher numbers are used to determine complexity of individual tasks before putting them into the sprint backlog. Also there is a question mark for discussion about the task. Low SP means that contexts and correlations are simple, whereas big numbers have many interfaces, are difficult to understand or plan and make a lot of effort. Because of that, a task which is 8 SP or higher, is separated in practice in several tasks. With a defined Sprint Backlog a sprint of 2-4 weeks can start. The development team distributes the tasks itself and works on them. Every day in the morning, there is a meeting called Daily where all roles of the team can ask questions, talk about blockers and ask for advise. in practice team members often have to tell what they completed, what they did for other teams and are planning to do. This leads to long Dailies which lose the focus of supporting the development team and become only instruments for economic controlling and meeting points. To guarantee that Scrum is applied correctly, there is the role Scrum master. This role mediates between management- and development-roles and warns if rules are violated or organizational problems occur. Along the sprint, features get implemented, tested and deployed periodically. Finally after every sprint there should be new fixes, values and features for the final product. Before the start of the next iteration, the team optionally can have a meeting called retrospective. Here the team work and performance on the

⁶<https://www.scrum.org/>

2.2 Agile Movement - influence on software landscape

task can be discussed as well as the current state of the product. After all sprints of a project there is an optional meeting called "lessons learned" in practice, where improvements for further corporations can be discussed and problems or feedback can be written down in order to avoid them in the future. At the end there is always a working product to show to the customer, a flexible and well tested development with 4 eyes principle and a solution which fits the customer's needs. Scrum is a concept which is highly adaptive to different branches of industry. This is a strength, but there is also the danger of worsening a process through problematic customization. Adaption of the Scrum process shall only be made if necessary and shall be evaluated constantly. Observed reasons for problems in projects using scrum are unrealistic resource planning regarding time, know-how and backlog. Also extra costs for high quality aims and quick fixes on production side, as well as difficulties in customer communication, like long answering times and missing trust can complicate the process. Furthermore the sprint backlog is not intended to be extended or have changes in the tickets. It need to be consistent so that developers can plan and work independently and precisely [Vijay and Ganapathy, 2014].

in practice small teams and multiple roles may form a Scrum version which is over-bureaucratic and unbalanced. Extra hours and cases where not all Scrum roles can be covered because teams are reduced to three or less people are problematic. It may then happen that people have the role of both a Scrum master and a product owner, or there is no real product owner at all. So the Scrum Master cannot warn the product owner on process violation or the team does not manage the knowledge transfer and requirements properly or works isolation. Things success-full Scrum projects in practice had in common, were the good mood of the members and teamwork as well as realistic and early time planning.

As mentioned, a final product and customer relationship with high quality is essential for Scrum. The focus of the next chapter is how developed software solutions are managed and hosted on customer side.

2 Background and Related Work



Figure 2.3: Scrum framework [Vijay and Ganapathy, 2014]

2.2.4 Software Development and IT Operations (DevOps)

Another concept which became highly relevant through the agile movement is DevOps. There is no generally valid limitation or definition for the term DevOps, but it describes management and execution of software tasks inside companies like software -versions, - deployments, -infrastructure. In theory DevOps concepts are performed by very experienced software engineers, due to high costs and limited availability of workers, these positions may also be outsources or filled with average experienced people. They support agile teams regarding topics like periodic deployments, environments and technical customer collaboration as well as security, platforms, tool specific choices and warning about stability issues. The development side of DevOps is about establishing the best process for customer satisfaction, maintenance, cost and risk by applying concepts visualized in figure 2.4. So the development requires optimized processes which rely on automatising and powerful technologies to ensure quality and efficiency. On process optimization side, frequent releases support more customer inclusion and thus customer-satisfaction. Also important for this effect are quick reactions on feedback and high quality combined with customer transparency, approvals and collaboration. This process requires delivery, managed and automatized by DevOps using deployment pipelines and code quality assuring technologies. Additionally, management of configuration, versioning and dependencies including compatibility and the decision to configure or containerize is handled via DevOps. Important DevOps-concepts for agile projects are continuous delivery(CDel), continuous integration(CI) and continuous deployment(CD), which became crucial for agile software development. CDel implies that only a running productive code is allowed to be pushed into the systems(git main/dev branch). This is assured by automatic software tests which fail if functionality breaks with a code change and prevents the upload of the code. The tests differ in their area of influence

2.2 Agile Movement - influence on software landscape

and are called "unit"-, "migration"-, "system"- tests. CD means to deploy the code directly to an environment (TEST,INT, DEV, PROD). ⁷. The other side of DevOps is the operative aspect which is about structural organisation of cooperation. As an option, DevOps and the product teams can cooperate as individual departments, supporting each other on demand with shared responsibilities. Also cross-functional teams are possible, where the development teams has the full responsibility for both development and DevOps. A third conceptional option is an individual DevOps team taking over actions between development and productivity of software. Finally, operative success is observed in metrics of the DevOps category runtime. Examples are the scalability of software-products as well as the reliability of the features and the availability of the infrastructure and domains. Also ensuring security via chaos engineering and active monitoring of software specifics and standards like certificates, virtualization and dependencies is part of this [Leite et al., 2019].

From the project start and the first requirement phase to delivery and long time support many disciplines, roles, developers and customers may be involved into a product. One way of facing the danger of issues on communication and documentation is to integrate a board into organization defined in an agile method called Kanban.

⁷The usage of environments are project specific, but from the author's experience in practice the following usage can be observed: development(DEV)-environment is mostly used for automatic visualization of the newest code of dev / main branch. Test-Environment is an internal environment for manual or automatic tests with any data and scenario. productive(PROD)-environment is typically a customer server where final product features run on. The customer works and evaluates based on this environment. integration(INT)-environment is an environment with similar code state and data as PROD, except that it is open for customer tests and not in productive use. These concepts make software transparent, testable and enable to perform agile software development with high quality.

12



2.2.5 Kanban

During development of individual tasks to manufacture a product, tasks can have several current states. According to the business, they may be finished, open or in construction with states like "created", "ready for development", "in development", "In review", "done", "productive" and finally "closed". Kanban is a method for organizing projects in a structured way via the previously mentioned workflow-states. By using a board and predefined states, tasks of one or many projects can be managed over full project lifetime. The board is a table with defined states, headers and tasks in form of rectangles with a title on it, inside the underneath columns. As soon as a predefined event happens, a state of a task changes and a task can be moved from one column to another. A ticket called "Fix UI Bug" can for example be moved from "In Development" to "In Review" after a developer fixed a bug. In practice Kanban-Boards are integrated in Project Management Tools(PMT) like Jira or Azure DevOps. They are crucial for building up the workflow of tasks in projects being organized via Scrum. Depending on the platform and setup, the rectangles show information fields like the assignee or SP estimation. Also links to main stories, so called epics, are common as well as tags showing environments on which the tasks are already applied. As studies and practice shows, teams tend to write essential and big topics into the tasks at Kanban board, but prefer to talk directly about details and small topics. At project start, if team members don't know each other or have different working times, the communication frequency and importance of the board is specially high. Via survey-studies Kanban showed in practice to have an impact on the communication during the whole project-lifetime [Oza et al., 2013].

Regarding Agile Setup and Workflow-States in practice, the author worked with two different types of configurations of Kanban board in the past. One was for new application developments based on Java and Javascript-Libraries and hosted on Jira. The other setup for first level support of multiple ".net" web applications hosted via Azure DevOps:

On Jira a new project was created during project start and set in the field "component" of each future task. Then a Kanban-Board with the states "Open", "Todo", "Progress", "Finished", "Reviewed", "Done". Tasks are moved from "Open" to "Todo" by the product owner as soon as all specifics are defined in the requirements. A developer assigns himself to the task and moves the task into the state "Progress". After completion the task gets the state "Finished". Then reviews and tests are performed and a successful task goes into state "Reviewed" and in the case of problems the task gets back into the state "Todo". "Reviewed" tasks are shown to the customer at sprint end and moved to "Done" if they are accepted. On unclear requirement or needed adaption the task can be moved from any state to "Open". In general, tasks are linked to epics like "Initial-UI-Design" or "REST-Service" and estimations are entered as well as

2 Background and Related Work

requirements and acceptance criteria. The benefit of this configuration is that it is usable intuitively, clearly structured and gives focus on development of current sprints. It is challenging to deal with big amount of tasks waiting to be moved from "Review/Tested" to "Done" or "Open" to "Todo" in order to guarantee the overview for individual tasks in "Progress". Also, "Finished" does not tell which environment the code is deployed to. Additionally, as soon as blocked tasks and open dependencies or tasks with final steps from last sprints are on the board, the overview can get lost.

In Azure DevOps there is an overview of several projects inside one Kanban board. Path and security level assure that users only see projects they work on. There are several workflow states which may contain tickets of different projects only separated vertically by their state. So called swim-lanes which are just horizontal lines with a header, enable columns to be split horizontally by individual criteria as well. There are running tasks, tasks blocked by customer and blocked by internal questions, systems or dates. The workflow states are "Backlog", "Active", "Progress", "Finished", "Deployed-INT", "Deployed-PROD", "Closed". Task-requirements get finalized in the column "Backlog" and then a task gets into the state "Active". A developer assigns the task to himself and moves the task to the state "Progress". When completed, the task is moved to "Finished". It remains in this state until it is deployed to TEST or INT environment. Then it is tested and a PROD deployment is scheduled with the customer and executed. After this, a task can be moved to state "Deployed-PROD". Tickets which raise bugs or have problems, can be moved to state "Active" again and state "Deployed-PROD" can be set to state "Closed" at the end of the iteration.

In the next section there will be explained how such configurations and data influence productivity of projects and more about different tools and their chances and configurations will be shown.

2.3 Knowledge and Data-Management as IT productivity Factor

The success/productivity of a project has, in the author's experience as product owner, metrics like sales, profit, customer satisfaction, follow up contracts and added business values. As acquiring a contract and making profit is dependent on optimal estimation during an offer and as estimation bases on numerous historic data and predictions based on information, (meta-)data is a key to project success.

Research about usage of knowledge bases and data management for support projects using ITS, stated that information has strong impact on productivity. By observing the SLA-Metric-Data the quality of the support process and optimal future resource plan ability ensures both competitive support products and

optimal resource usage. So injuring SLA-Agreements leads to loss in customer business relation and further to customer-loss. The amount of time employees spend on tickets can be reduced by maintaining and collecting data, customer information and development documents. Especially first and second level support profits from this data as big parts of raised tickets are repetitive. But also for third level support data helps to predict and solve critical issues. So over all levels effort and thus time can be reduced, quality can be assured and successful long term relationships manifested. [Aglibar and Rodelas, 2022].

As (meta-)data is a broad term, the next section will provide examples of key data regarding projects and their data-provision on different platforms dealing with tickets.

2.4 Issue Tracking Systems (Jira, Azure DevOps, GitHub)

As mentioned, modern software development is executed in teams, with people of different backgrounds and roles with the common aim to cooperate on a well documented long time operative and maintainable solution. Different dependencies, time zones, cultures, degrees of working contracts, opinions and many more factors can complicate the corporation even more. To face these challenges, teams use Issue Tracking Systems(ITS) to handle new features, raised bugs, updates, communication and small documentation. Next to reporting, stakeholders can retrieve operative information like the current state of development, teamwork and blockers for individual components. They also can identify knowledge carriers, find estimations and report or comment the progress. There are different implementations of ITS like "Azure DevOps" of Microsoft, the open source tool "GitHub", or Atlassians "Jira", which may be used individually or parallel for different product lines. What all of them have in common, is the prevention of information overload by using grouping elements like components, tags and labels. They provide histories of added content and handle users, individual access rights and support a huge amount of tickets which are only limited by server storage. [Arya et al., 2019]

2.4.1 Jira

A widespread proprietary software solution is called Jira. It is an ITS implemented by the company Atlassian and offered as product via service amount based licensing. Jira builds up both projects including ticket information and users working on one or many projects in processes with different workflow-states, participants and requirements. Also, tickets are exportable effortlessly and queries are supported as well as integration of features (Visualization, Analytics) to Jira. The process of Jira usage starts by creating a project and

2 Background and Related Work

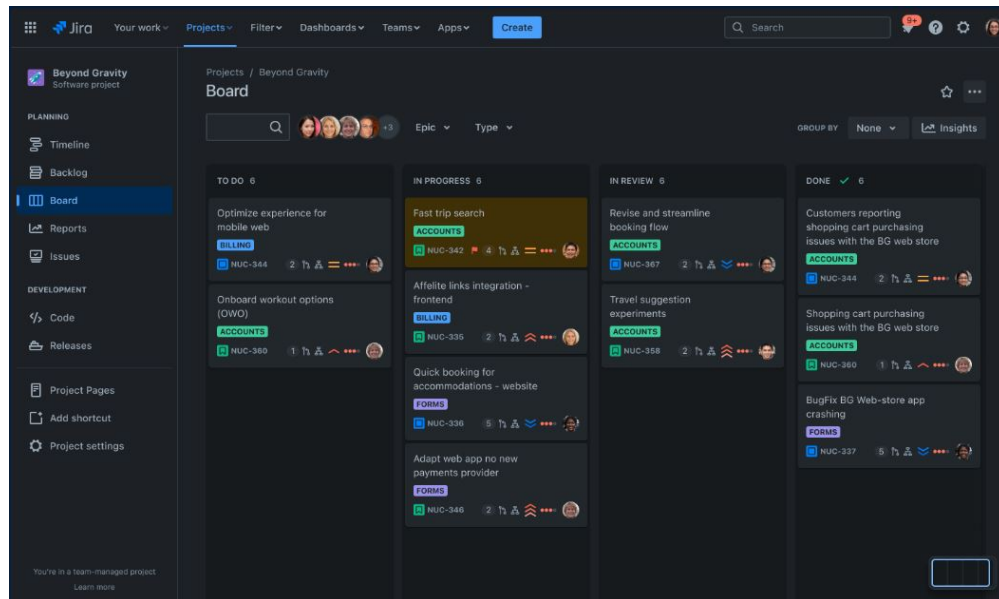


Figure 2.5: Jira example KanBan-Board from [atlassian, 2023a]

privilege employees. A project has a single board with a configurable amount of Kanban-board columns with individual names. The board visualizes tasks and their positions in the work-flow-columns based on states. This can be looked up at figure 2.5. In the backlog-view epics and stories are visualized as well [atlassian, 2023c].

Atlassian defined following definitions for the default issue types: epics are "parent issues" which represent huge topics with a common goal, which need to be chunked in stories and may combine content of stories, bugs and tasks. Hierarchically "child issues" are subordinated to an epic. Epics may be bugs, stories or tasks by default. Bugs indicate functional product issues, stories define work packages and tasks are individual workloads to be executed. For larger tasks, bugs or stories there are also sub tasks which separate them in pieces which can be processed [atlassian, 2023b].

Example: three user stories called "Retrieve Data", "Create Database Schema" and "Migrate Data" and one bug named "Cannot access Data-Provider-Webpage" belong to a single common story named "Implement Person-Data-Interface", which belongs to a single epic titled "Implement new API for company specific data", which is the parent of three other stories as well. As visible in figure 2.5 from ⁸, labels can provide extra information like department belongings, platforms or used technologies.

Also there are numerous integrable tools and features. GitHub commits can

⁸<https://www.atlassian.com/software/jira>

2.4 Issue Tracking Systems (Jira, Azure DevOps, GitHub)

be mapped and visualized per individual task, timelines can be created, color highlighting based on data, evaluations, queries and exports of fields can be executed. While Jira offers better mobile accessibility and less limitation on level and feature integration for tickets, Azure DevOps offers the same basic ITS functionality and support internal repository and pipeline handling [atlassian, 2023a].

2.4.2 Azure DevOps

Another software solution for building up project-details and cooperation of roles over their whole life cycle is Azure DevOps. Again it is an ITS, but implemented by the company Microsoft and offered as cloud-product via service amount based licensing or as private server for companies based on licensing. Azure DevOps has a wide range of features going from work item management for project start, over individual access management, queries, visualization to technical integration like GitHub-repositories, CDel/CI, jobs and tests, individual customized features as well as community features. At DevOps-Boards like those visualized in figure 2.6 (from ⁹), the features are represented board-wise as tabs on the left side. Overview offers the possibility for summarising key data and personal data, creating a wiki with work item links and visualizing workflow charts, important links and sprints, employee or project related individualized and automatically updated charts. Boards separate work items chosen by current filters like sprint and projects in their KanBan workflow state. Work items also show extra information like assignees, tags, labels, name and number of tasks. Swim lanes can be used to provide also horizontal zones. For example in the author's case at third level support: "Intern Blocked", "Customer Blocked" and the swim lane "Processable". Tickets can generally be moved freely on the board with configurable conditions for state transitions. All changes are tracked inside a work items history. The "Repos"-Tab allows to manage GIT-Repositories directly in Azure DevOps and links them to pipelines and test plans. The pipeline tabs handle releases, build, visualize and execute runnable scripts. "Test Plans" enable the team to perform automatic tests on applications, show and guarantee the quality of applications. Finally, in the "Artifacts"-Tab, packages can be managed and shared. Prominent examples are NPM- or NuGet- packages which can be released and from then on accessed by third parties like customers. [Microsoft, 2023a].

Tickets are called work items and have one of the available work-item-types shown in 2.7 (from ¹⁰). They can be created directly on the board and can have project wise restricted view permissions via the "area path" and adaption of the team security configurations. Depending on the chosen configuration of the

⁹<https://azure.microsoft.com/de-de/products/devops/boards/>

¹⁰<https://learn.microsoft.com/en-us/azure/devops/boards/configure-customize?view=azure-devops&tabs=agile-process>

2 Background and Related Work

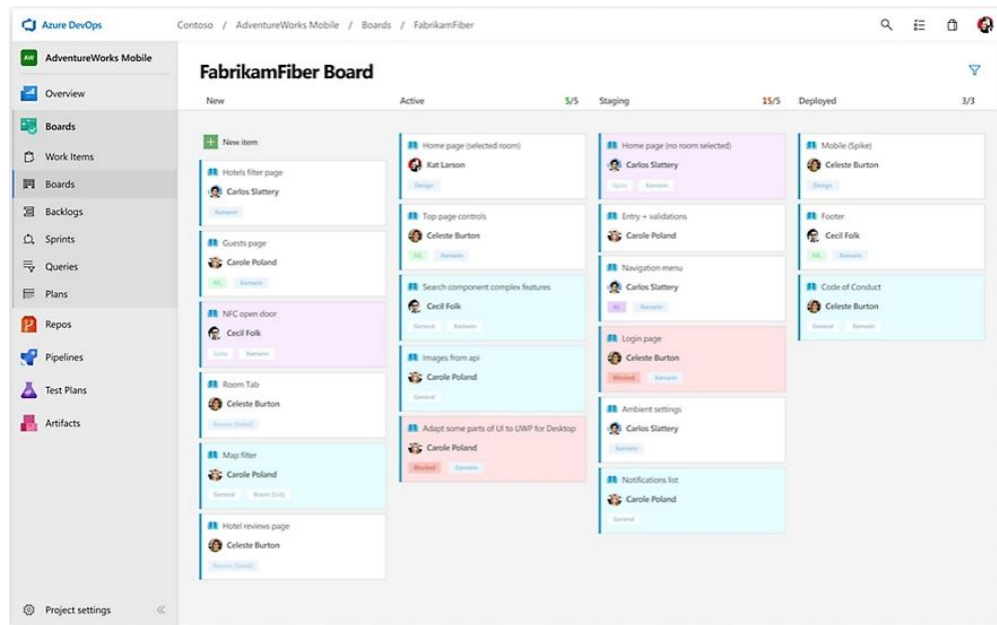


Figure 2.6: Azure DevOps example KanBan-Board from [Microsoft, 2023a]

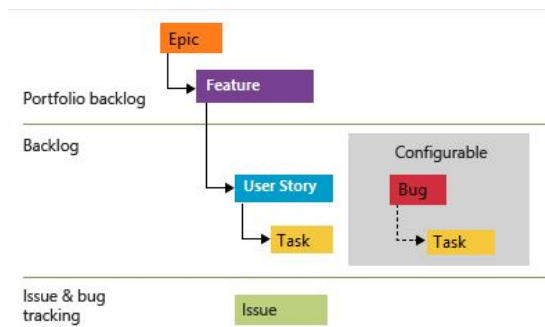


Figure 2.7: Azure visualization of work item types from ¹⁰

“Working with bugs”-setting, shown in figure 2.8 (from ¹¹) bugs can be created in two ways. The former is to create them as part of a work item, which is of the type “User Story”. As latter option, work items can be either user stories or “Bug”s and both can have sub-tasks to track progress and scrum related separations. In both cases work items can be related to each other and can be linked to features which may also have a common epic above. [Microsoft, 2023b].

¹¹<https://learn.microsoft.com/en-us/azure/devops/organizations/settings/show-bugs-on-backlog?view=azure-devops>

2.4 Issue Tracking Systems (Jira, Azure DevOps, GitHub)

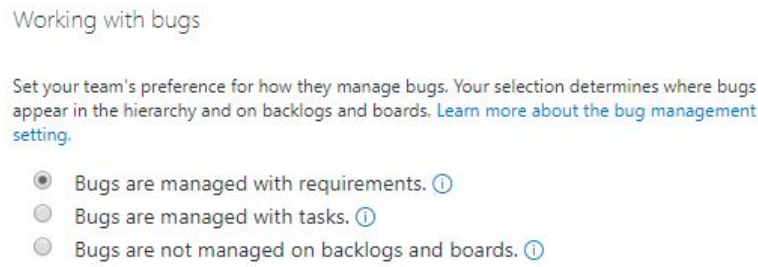


Figure 2.8: Azure configuration of bug visualization from ¹¹

2.4.3 GitHub

GitHub is a software solution for version control and dependency management and has become a standard among software development industry. Contrary to GitHub's predecessors (like SVN), GitHub allows parallel feature development on multiple branches, release-branches with tags, separated branches for environments and merging of branches as well as protection of branches. Furthermore it offers a large scale of functionality, including complex prompt-commands, cooperative development, flow, release-actions and runnable jobs. GitHub was built by C. Wanstrath, P. J. Hyett, T. Preston-Werner, and S. Chacon and acquired by Microsoft. The platform is usable, accessible and hostable for free, but there is also a GitHub's Server with sales strategies like premium versions and features. GitHub is also able to include an ITS which allows to create, manage, control, review and even publish projects. As figure 2.9 from ¹² shows, tasks can be created project wise, Github-collaborator can be invited to specific projects, tasks can be assigned and extra-data can be added. Projects can be managed as KanBan board with the possibility to define own columns and move tickets between columns. Also, with GitHub pages GitHub allows publishing non-static public websites which allows to publish private or educational light weight applications with minimal effort in combination with online databases. For users consuming professional accounts, releases, publication, jobs and other services are available with fewer limitations. Generally, GitHub is free to use as ITS, but in its basic form more lightweight in terms of inheriting, relations, types, task representation and task queries. But it is highly extendable via plugins and customized solutions [Microsoft, 2023c].

2 Background and Related Work

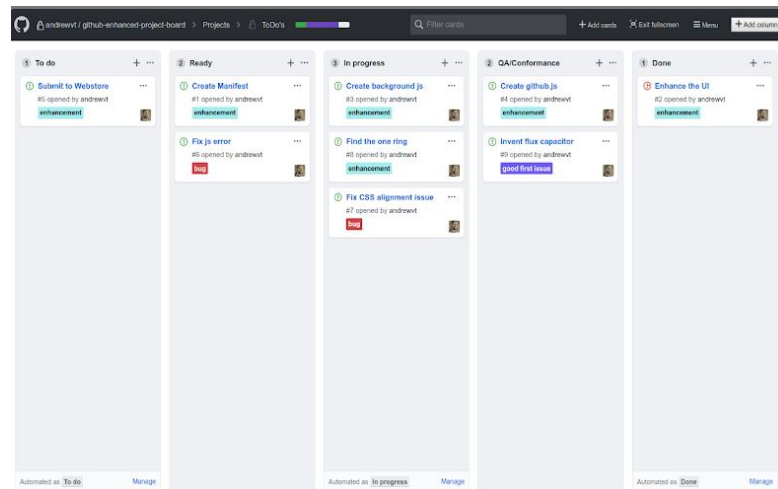


Figure 2.9: Project-board plugin for GitHub from chromewebstore

2.5 SLA Management and its Realization in 3rd Level Support Process

State of the art support processes receive tickets as input which come directly from an application, user input per mail or telephone-hot-lines. Then the support work on tickets is separated in 3 support levels. Employees of the 1st-Level-support are the first contact point for raised tickets and are directly assigned and reachable. They solve clear, repetitive and standard cases and escalate new or complex topics to the 2nd-level of support. The 2nd-level-support consists of employees with larger application specific experience and technical know-how. They cover complicated processes, topics and information retrieval. Typically, 1st-Level and 2nd-Level cover most of the tickets. In case of tickets which need complex analysis, development or certain amount of time and technical overview, employees of the 3rd-Level-Support are contacted. The response, solving and availability of support are dependent on the task's priority and all of this information is defined at the project-SLA-contract. To be noted is that all levels may get in contact with users and customers and that the exact support-process varies from company to company. For estimations, planning and price building it is a common practice to categorize tickets based on their complexity and time effort [Aglibar and Rodelas, 2022].

In the author's case, there is a department which does 1st-Level-support for a huge amount of applications. Inside 2nd- and 3rd- level-support-teams there are certain roles such as product owner, ,developer,etc and an amount of application based on the team size. The first level has predefined answers for standard cases and otherwise collects a predefined amount of information in all other cases, before escalation to 2nd-Level. The author as product owner tries to solve the ticket independently as 2nd-level of support, with application

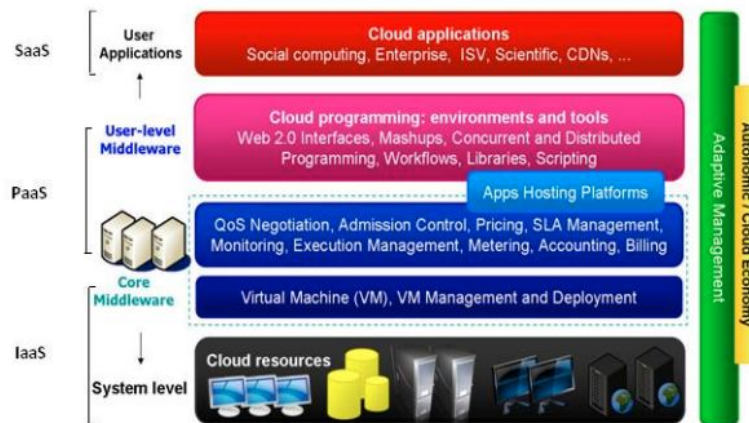


Figure 2.10: Cloud computing layers as instance of complexity of SLA tracking from arxiv

specific knowledge, data querying and similar cases in documentation. For code analysis, code development, special cases or especially complex tasks the development team is consulted as 3rd-Level. As instance before the 1st-Level support, help-pages and chat bots can be installed as level-zero.

2.5.1 Fields of application for SLA-Tracking

The covered metrics of SLA depend highly on the field of application. In modern computing SLA-Contracts are used for business collaboration, risk assessment, dynamic support re-negotiation and cloud computing for Software-as-a-Service(SaaS), Platform-as-a-Service(PaaS) and Infrastructure-as-a-Service(IaaS). Depending on the specific field of application different SLA-Metrics are relevant. For example an IaaS contract may be strict in resource based operating- / processor- time as metrics, while in contrast SaaS may prioritize metrics for individual support with high response time and low error rate in contracts. Figure 2.10 (from ¹⁴) indicates complexity of SLA-Tracking, as one project can have different providers of IaaS, PaaS and SaaS with prizes based on service contracts. There is no standardized dynamic solution for SLA-Metrics, SLA-Tracking and SLA-Contracts established [Wu and Buyya, 2010]. The author uses first-response-time, average-response-time and average-re/solve-time as metrics for the support quality and as indicator for support process quality and contract fulfilment.

2.6 Summary

So modern software development is a team challenge based on the cooperation of many persons, the support of platforms and changing technologies. The

¹⁴<https://arxiv.org/ftp/arxiv/papers/1010/1010.2881.pdf>

2 Background and Related Work

goal is to create secure, maintainable software with restricted resources and achieve customer satisfaction by sticking to proposed contracts (SLA). This requires experienced persons and optimized organisation and cooperation. The performance of development- and maintenance- teams can benefit from agile practices, where persons cooperate in products as roles and project success can be provided by continuous improvements and communication. To handle the complexity of large software projects, employees may get supported by specialist teams like DevOps to cover tasks from local environments and pipelines to customer consulting and dependency management. With principles like sprints, pair programming and Kanban, development teams are enabled to work jointly with clearly assigned work loads, interfaces, goals and verified product quality by code reviews and processes. With CI, CDel and sprint reviews customers can provide feedback early and running products are ensured at any time of the project. The success of a product may be defined by a contract between service provider and recipient called SLA. These contracts contain services and metrics for measuring success and violations. As many platforms are involved into software development, there is no general single standard solution for SLA tracking and it requires individual manual work to balance understandably, effort and correctness of SLA contracts. Relevant data for SLA tracking is the metadata of communication and development platforms like GitHub, Jira and Azure-DevOps. These platforms implement KanBan boards to cover requirements in the form of tickets for development stories and issues(bugs), including process -states, -communication and estimations. A project is not restricted to one provider, there may be different companies and interfaces offering SaaS, Paas or Iaas. In this case there may be several SLA, defining obligatory characteristics for hardware, technologies and individual software products.

The following software tool was an idea created by observation of SLA execution in industry and the above mentioned theory. The following chapter will cover the methodology and basic information about a SLA automatism tool invented and implemented by the author.

3 Solution, Architecture & Design

3.1 Methodology

The implementation of this thesis started with an observation phase, followed by the analysis of systems, their Application Programming Interfaces(APIs) and static pages. Finally technology selection, implementation and evaluation of results were covered. This chapter will document technology and design decisions and the operative organisation of the thesis.

3.1.1 Research - Observation and ideas

During the practice as a product owner in an IT company, the author had to evaluate whether support contracts have an optimal size, how much effort was spent, how the effort changes with the time and the quality based on the SLA. The ticket data was distributed between Azure DevOps and Jira, official ticket states are in Jira while communication and effort planning are in Azure DevOps. Some tickets were directly resolved by another department called first level, others directly solved by product owners without code changes and for the rest GitHub was the central point of documentation with branches and commit messages. The budget, planned tickets and effort for the contract were in a document called offer. The solution was a direct export of the tickets from Jira as an excel report and combining them with the employee time tracking system data and the offer data inside a reporting excel. During these steps it was also required to validate all tasks on Jira via browsing through the user interface and checking whether one of the SLA-Metrics was violated. All of these steps needed to be done manually. So the author analysed the availability of these data via APIs and static URLs.

3.1.2 Research - Identification of problems

The ticket-data is separated on multiple independent systems, the SLA metrics are not considered in Azure DevOps, and for Jira one has to browse ticket by ticket to get information about SLA. Therefore a data collecting service and a report per project as overview is required. Also effort related data is stored in custom fields in Jira and Azure DevOps and the real effort is only visible via GitHub. To give an overview about planned changes and real effort, gathering all metadata is necessary to create an overview in the form of a report.

3.1.3 Research - Analysis of platforms and data

For the already mentioned objectives, the data for the author's projects is distributed on Jira, AzureDevOps and Github. Catrobat uses Jira and a GitHub-Organization. For SLA tracking the ticket data and history of Jira and Azure DevOps is most relevant and the commit messages and numbers of GitHub. The following paragraphs will provide information about the dynamic access of this data on the platforms.

For Jira, Atlassian offers API documentation and there are ¹ jira-java-libraries and a ² python-package directly suggested via Jira. The REST-API-Documentation allows in both cases to request extra data like the task-history via individual requests over the interfaces defined at ³ Jira-Documentation. A connection to Jira can be established by providing the server-name, and basic authentication with a valid jira-user-name and a valid token or password for the given user. The task history data can be retrieved over a static page as unformatted XML. There are libraries in python to format and parse this XML to a JSON, which can be processed by the program. The history of a task with RESULTNUMBER history entries, belonging to the Jira project PROJECTNAME hosted with given country code TOPLEVELDOMAIN(TLDDOMAIN) and with a task identifier PROJECTKEY is accessible over the template link

```
"https://jira.PROJECTNAME.TLDDOMAIN/  
activity?maxResults=RESULTNUMBER&streams=  
issue-key+IS+PROJECTKEY&streams=  
key+IS+PROJECTKEY&os_authType=basic&title=undefined".
```

Regarding Azure-DevOps there is also an API documentation available over ⁴ Azure-API-Documentation which recommends next to .net(C#) and notes also python as client request library. The connection can again be established by naming a server and using a user-name and a valid token or password. As the author's projects use a private instance of a Azure-DevOps-Server, the python packages at ⁵ Git-Azure-API-Docu are still applicable, but individual requests to the API must be executed. For a ticket on the server with the name SERVERNAME and country code TOPLEVELDOMAIN a list of work-items called COMMASEPERATEDLISTOFINTEGERSSMALLERTHANTWOHUNDREDITEMS (CSLIISTTI) for whom the user has the permission to see them, can be retrieved with the template

```
"https://SERVERNAME.ORGANIZATIONDOMAIN.TLDDOMAIN/  
tfs/TfsProjects/_apis/wit/workitems?ids=CSLIISTTI".
```

¹<https://github.com/lesstif/jira-rest-client>

²<https://jira.readthedocs.io/>

³<https://docs.atlassian.com/software/jira/docs/api/REST/5.2/>

⁴<https://learn.microsoft.com/en-us/rest/api/azure/devops/>

⁵<https://github.com/Microsoft/azure-devops-python-api>

GitHub-REST-API-Information can be looked up on ⁶. Also a client access documentation for python is available on ⁷. It has to be considered that organizational GitHub, private GitHub accounts and GitHub via Azure-DevOps require different URLs to access commits. For private users the data can be accessed via ⁸ and ⁹ by using basic authentication with a valid git user and token or password. These links vary for users of repositories hosted by organizations to ¹⁰ and ¹¹ with equal authentication. For Azure-Devops the access pattern is different as GitHub is an integrated service there. The documentation ¹² offers to access the commits in this case via

"https://SERVERNAME.ORGANIZATIONDOMAIN.TLDDOMAIN/organization/project/_apis/git/repositories/repositoryId/commits?api-version=7.1-preview.1".

Finally real personal effort related data has to be exported and inserted via excel exports from the delivery systems. This is necessary due to different realizations in Catrobat, the author's projects and other projects of companies the author worked at.

3.1.4 Research Technologies

To design a configurable dynamic standalone application for gathering of meta data for multiple platforms, with the possibility of extension and given maintainability, the selection of the programming language is crucial for the result. From the author's study related and work related experience C# / Python and Java are the most relevant programming languages for local standalone tools. Python is a programming language from 1989 with the core focus on readability and much functionality in less lines of code. It has a big offer of libraries and is highly integrate-able in applications, but also well suited for standalone applications. Java is a mainly static programming language from 1991 with the idea of pointer safe programming without direct memory management and platform independence due to required usage of the Java Virtual Machine(JVM). Java and C# allow object oriented programming, encapsulation and polymorphous programming. C# is a programming language introduced in 2002 which is like Java pointer safe and manages memory independently, but also offers large build in support of libraries, together with the .net framework. However C# is mostly windows platform dependent and thus may be problematic due to

⁶<https://docs.github.com/en/rest>

⁷<https://gitpython.readthedocs.io/en/stable/tutorial.html>

⁸<https://api.github.com/user/repos>

⁹<https://api.github.com/repos/user/repoName/commits>

¹⁰<https://api.github.com/orgs/orgName/repos>

¹¹<https://api.github.com/repos/orgName/repoName/commits>

¹²<https://learn.microsoft.com/en-us/rest/api/azure/devops/git/commits/get-commits>

3 Solution, Architecture & Design

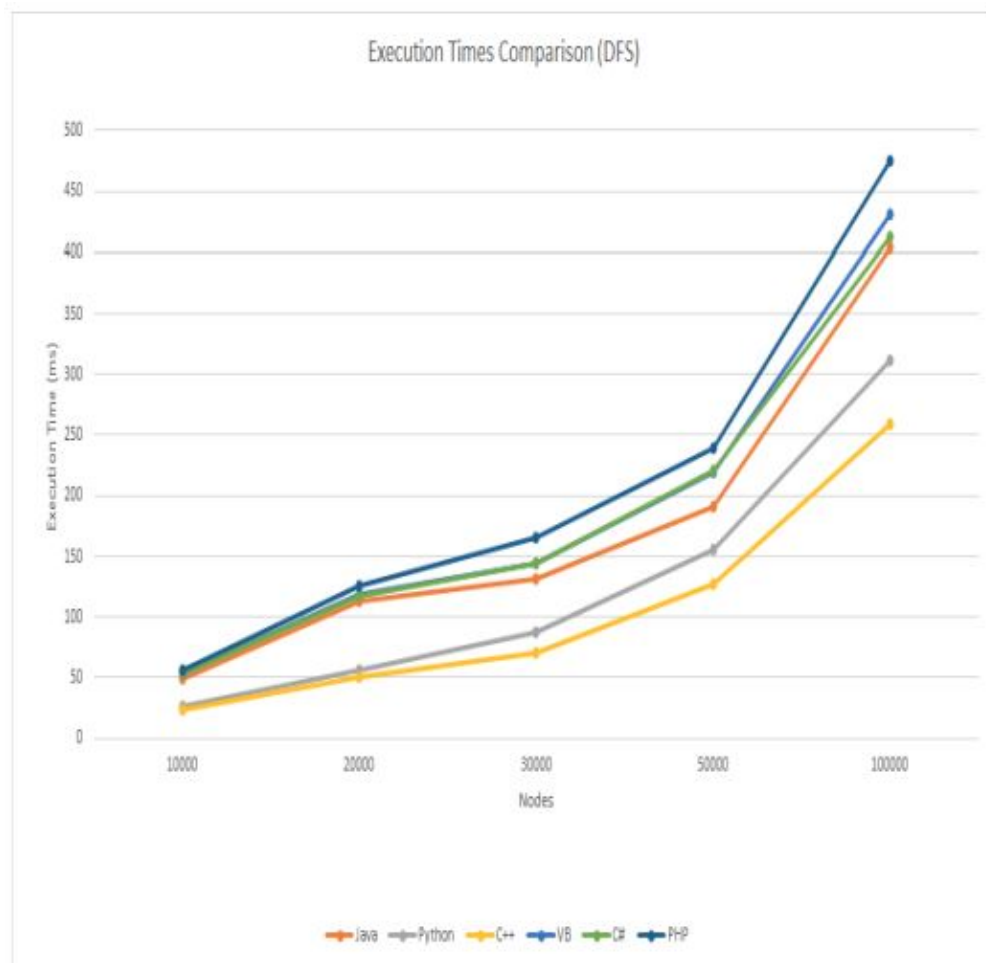


Figure 3.1: Computing time for computing example for different number of nodes [Alomari et al., 2015]

the thread related interrupt handling of windows for real time development
Alomari et al., 2015

As the program processes and reports information with changing structure and interfaces, as it needs to be developed in a short time and represents a new concept, Python is chosen as development language. Python is well suited to process the Big Data in the form of several thousands of tickets including historic- and attached- information. It is supported and well documented for relevant platforms as client language, it provides all libraries for data-processing and is simple to maintain and host. Regarding memory usage and computing time all of these programming languages fit well. However by using specific libraries with good performance, Python even may be faster in computing time for several algorithms than C# and Java according to figure 3.1. The decision for Python as programming language applies to the thesis prototype version of this program, afterwards for a possible business version Java is advisable as

it ensures code quality, is less prone to error for long term development and ensures compatibility with JVM.

To enable rapid development of an application dealing with APIs, empiric trials, exact limitation-tests and thus multiple code-adaption are required. This creates the necessity for a language which is performing fast, maintainable and powerful. For storage ¹³ SQL-Alchemy is used for a MySQL database as library to enable a quick and simple database access with mapping of Python objects and long time storage of task data. For processing data ¹⁴ Python-Pandas is used as library to store big data, process it and report it to excel using ¹⁵ Openpyxl. For REST-requests the package from <https://pypi.org/project/requests/> is used to do simple data crawling on Jira, Azure-DevOps and GitHub. For the task-histories the ¹⁶ package minidome is used to format a challenging xml response and then the ¹⁷ xml-to-dict package is used to convert the JSON, which enables simple processing. Json is explained at ¹⁸. Additionally ¹⁹ matplotlib.pyplot is used to show data as charts and pypi packages like ²⁰ wordcloud to visualize the data distribution for the systems.

3.1.5 Planning - Specification

A standalone Python tool must be implemented. It shall be separated in modules which are based on the individual platforms and technologies. Uniforme naming standards and conventions must be applied. Modules must be configurable individually and functionality shall be structured equally at different modules. Data is retrieved from the individual platforms with REST-requests. Permissions to the user accounts are granted via configured tokens. Server-names and valid application access for users can be assumed. The data must be analysed and statistic plots shall be created using matplotlib.pyplot. There must be a configuration which defines if stored Jira tasks are used via Sql-Alchemy or fresh data is retrieved. A dataclass must be used for retrieved Jira tasks to have a transparent interface in code. As table 3.1 visualizes, the individual platforms need configured minimal information if the module is enabled and required to run successfully. Generally, Python, MySql with credentials and the Python packages are required to run the application. All Jira-Projects must be read out from the Jira platform and reports based on these projects have to be created. By using Pandas-Dataframes, Azure-DevOps and GitHub data shall be collected

¹³<https://www.sqlalchemy.org/>

¹⁴<https://pandas.pydata.org/>

¹⁵<https://openpyxl.readthedocs.io/en/stable/tutorial.html>

¹⁶<https://docs.python.org/3.8/library/xml.dom.minidom.html>

¹⁷<https://pypi.org/project/xmltodict/>

¹⁸<https://www.json.org/json-de.html>

¹⁹https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html

²⁰<https://pypi.org/project/wordcloud/>

3 Solution, Architecture & Design

and SLA metrics have to be calculated per projects. The report must be an excel file with several tabs for overview, Jira, Azure-DevOps, Time-Booking, GitHub and statistic data.

Table 3.1: Application Requirements Table

Platform	Topic	Property	Value
Python	Prerequisite	Version	Python 3.10
Python	Prerequisite	Database	MySQL
Python	Prerequisite	Credential	UserName
Python	Prerequisite	Credential	PassWord
Python	Prerequisite	Dependencies	Install requirements.txt
Jira	Configuration	ServerName	Link
Jira	Configuration	Credential	UserName
Jira	Configuration	Credential	Token / Password
GitHub	Configuration	ServerName	Link
GitHub	Configuration	Information	Account(Org./Private)
GitHub	Configuration	Credential	UserName
GitHub	Configuration	Credential	Token / Password
Azure	Configuration	ServerName	Link
Azure	Configuration	Information	Accessible Tickets(Excel)
Azure	Configuration	Credential	UserName
Azure	Configuration	Credential	Token / Password
MySQL	Configuration	ConnectionString	Entry

Especially the fields of Jira tickets are crucial for SLA-tracking. For the tickets priority, states, timing and estimation are used to calculate the metrics average answer time, response time and solution time. The system analysis focuses strongly on persons and interactions based on fields like reporter, creator, comments, history. Another analysis is the visualization of blocked tasks in the system. This analysis is based on the state and a custom blocked issue field. Table 3.2 shows gathered SLA-Metrics.

Table 3.2: SLA-Metrics table

Name	Calculation-Field
First Response	Task-History-Comments
Average Response	Task-History-Comments
Solution Time	Task-History-States
Timeline	Task-Dates/Task-History-State
Project Progress Data	Task-State / Task-Priorities
System State	Task -Type/-Priorities/-BlockedBy
Statistics	Number of Tasks and Analysis of Metadata

SLA-Metrics Table.

As table 3.3 indicates, the program structure shall be separated in a general module, JiraModule, azureModule, gitModule, reportModule. Modules may contain connection, crawling, configuration, database repository as well as service and individual classes. The configuration file must be of type JSON and in- and out-put must be in separated nested folders.

Milestones for the tool are the successful dynamic access to Jira metadata including projects, task-fields, task-history and customized fields. Subsequently the local storage of this data and analysis of the data. Afterwards the Azure-DevOps access and a mapping between this data and also the GitHub access and project mapping has to be executable completely dynamically. The full multi-platform metadata has to be exported into project related excel files. The result is a runnable standalone Python tool. Optionally the integration into the Catrobat-System and the system of the author's company is also a goal.

There are many plots analysing raw data of the systems in the background, those outputs files which are user-relevant are listed in the table 3.4. There is a chart which shows the number of Jira-tickets which are created per month and one chart which shows the number of updated tickets per month. There are charts showing the data distributions overall Jira-tickets regarding a ticket field. In this way the issue-types(story, bug,...), labels, priorities and states are analysed. Also the project affinity overall tickets and users and their percentage on tickets are evaluated. Finally a network of tasks and their dependency on other tasks is displayed and the raw data of customized Jira fields are analysed via charts and wordclouds.

3.1.6 Execution - Short Description

The author created a object oriented Python application which extracts data from project management systems and visualizes and reports their output in

3 Solution, Architecture & Design

Table 3.3: Program Structure table

Module	Type	Name	Description
JiraModule	JSON	jiraConfig	specific configuration
JiraModule	CLASS	jiraConnector	establish jira connections
JiraModule	CLASS	jiraCrawler	gather data from jira
JiraModule	CLASS	jiraService	jiraModule API
JiraModule	CLASS	jiraRepository	interacts with database
JiraModule	CLASS	jiraStringUtil	offers string functions
JiraModule	CLASS	jiraTaskProjection	data holding object for jira
azureModule	JSON	azureConfig	module specific configuration
azureModule	CLASS	azureConnector	establishes connections to azure
azureModule	CLASS	azureCrawler	gather data from azure
azureModule	CLASS	azureService	API for azureModule
gitModule	JSON	gitConfig	module specific configuration
gitModule	CLASS	gitConnector	establishes connections to git
gitModule	CLASS	gitCrawler	gather data from git
gitModule	CLASS	gitService	API for gitModule
reportModule	CLASS	reportService	API for reportModule
reportModule	CLASS	reportAnalyser	Information
-	-	-AndViewer	-
-	CLASS	main	Programm entry point
-	CLASS	config	manages configurations
-	JSON	baseConfig	basic config
-	TXT	readMe	Store info (facts, links)
-	TXT	requirements	pip install -r requirements.txt

a comprehensive way. The structure of the program is highly dynamic and modular, as the application should be extensible to more platforms and usable for individual contexts. A micro-service pattern is applied and file naming regarding style guides from the author's practice.

Table 3.4: Program Output table

Type	Metric	Name
Report	All	Report_Project_Month_Day_Year.xlsx
Chart.png	Created	createdPerMonth.png
Chart.png	IssueTypes	issueTypes.png
Chart.png	Labels	labelsExceptEmpty.png
Chart.png	Priority	priority.png
Chart.png	Projects	projectsAndTickets.png
Chart.png	States	states.png
Chart.png	Updated	updatedPerMonth.png
Chart.png	InDevelopment	userWithTicketsInDevelopment.png

3.1.7 Evaluation - Issues, Limitations, Decisions, Reasoning

During the analysis and implementation of the tool, data and platform challenges as well as implementation limits and decisions came up. Table 3.5 contains the most time consuming challenges. In the analysing phase inconsistent and sparse customized fields of Jira impeded understanding of fields and progress. As figure 3.2 shows, tag-words are not used uniformly, more than ten fields are not used in over 98% of tickets. User related fields have commonly around one hundred to one thousand different names as labels and for some fields whole text files and scripts are inserted in a normal sized Jira-field which does not even visualize them completely. For the tags uppercase and lower case letters are used, for example "any" and "Any" and for Boolean fields not only "zero" / "one" or "true" / "false" exists but "None", "Non", "n", "No" which describe all the same value. There are also fields which have no clear common meaning in the projects, one custom field contains Arrays, Names, Boolean and further. A faced problem was that qualitative code in Java is more precise about data-types than Python and libraries are more complex to integrate and use. The author faces a library problem which does not allow to retrieve full data of Jira regarding projects and history dynamically. A manual approach worked to retrieve html directly from static pages and parse it, but this is less dynamic and safe in terms of changes. Also big structures in array require more effort to iterate and keep care on types than in Python. So the solution was to use Python as programming language for the tool to enable rapid prototyping of the application and its interfaces. Attached to Jira-tickets, there are individualised fields available. They are retrievable over a REST-requests, independently from their configured Jira-name with the template "customfield_xxxxx". "xxxxx" may be any numeric identifier and depending on the configuration and history, gaps between defined indexes are possible. The API does not offer data on configured custom fields. That makes retrieving this data a complex task. As requests to a

3 Solution, Architecture & Design

non used custom field results in an exception, the solution was finally to make the retrieved fields configurable in the tool. One platform specific challenge occurred for requests of tickets in Azure-DevOps. By default, the maximal number of retrievable work-items per request are limited to an upperbound of ²¹ 200 work-items per request. As described, these settings are configurable at Azure-DevOps, but this cannot be pre-assumed for all applications of the tool. Also only those tickets can be retrieved where an user is allowed to access them. If any not accessible identifier is in the list of requested work-items, an exception will be raised immediately. Also the API offers no possibility to gather information about all tickets which may be accessed for a given user. A feasible solution is that over the query-tabs a list of all accessible work-items can be queried and exported as excel file. The location of this excel file is configurable in the Python tool and also the retrievable number of work-items per request are configurable in the Python tool. A possible future workaround is to do automatic query requests. Also the history tab of Jira was a big challenge. The history is retrievable per task in the form of an xml, containing users, dates, states and further data. For processing it gets transformed to Json and dates, states and comments get extracted and stored into a more compact Json. This information is added as column into the task database. The storing process raises the problem of an appropriate size restriction for the field. As histories may have huge length and the column is chosen as string, the Json data needs to be cut at a defined maximal size and thus cannot be loaded anymore in a valid way. The data is stored as strings because Json format would be stored as binary internally in the database and may bring problems with specifics characters. The solution therefore is to set the field-size of the Json as high as possible so that only tasks with an exception-history in size get ignored. For a future version of the tool, all necessary calculation with the Json can be done pre-hand and only results may be stored. This would create the need for another project-wise storage concept.

Table 3.5: Program Challenges Table

Challenge	Solution
Data Inconsistency	Analyse and filter/catch in code
Libraries and effort in java	Python offers API's and libraries
No uniform Jira-customized-field-indexing	Configure fields by names
Azure-DevOps-Item request -limit/-security	Chunk list of accessible tasks
Store dynamic-length history in DB-Column	Cut at some degree - not optimal

²¹<https://powerusers.microsoft.com/t5/Building-Flows/limit-of-200-items-in-Azure-DevOps-get-query-results-apply-to/td-p/1679461>

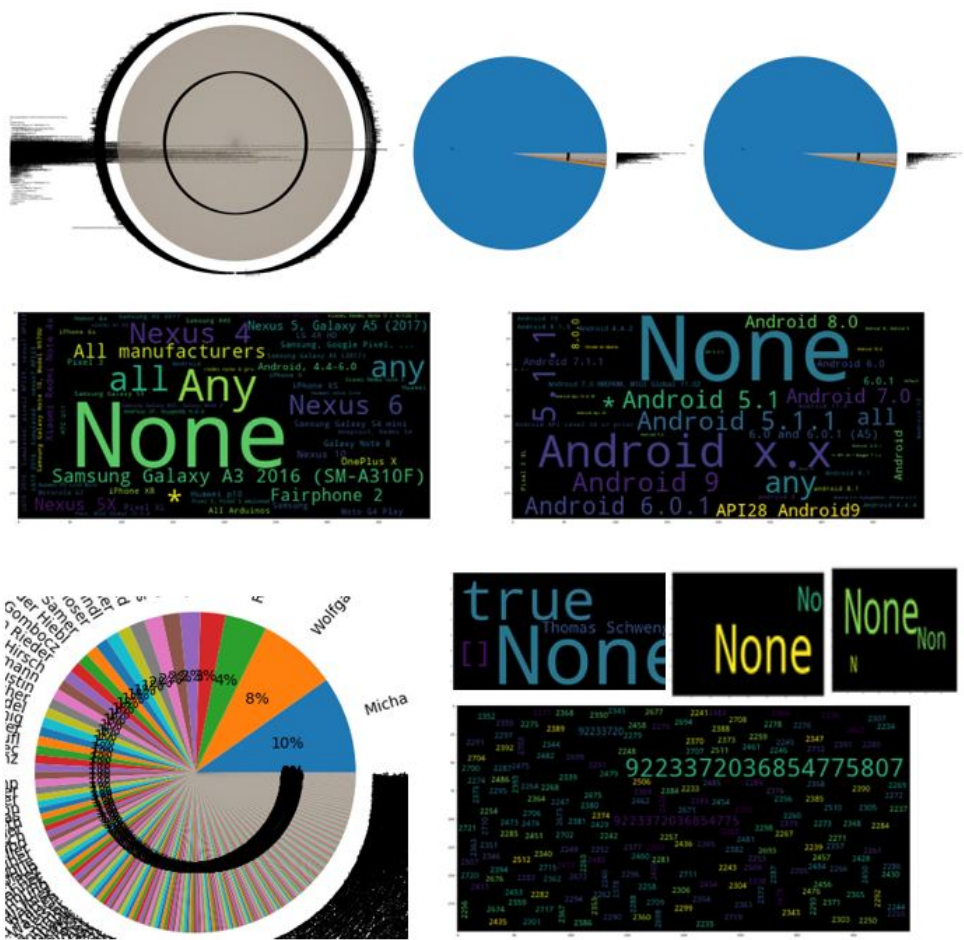


Figure 3.2: Data inconsistency in Jira-Customized field

3.2 Applied Technologies/Concepts

The author's solution consumed and applied various state of the art protocols, concepts and definitions. Big Data is requested from various platforms which work with RESTful Architecture. It is received in the form of XML, converted and stored as JSON and processed inside a Pandas-Dataframe, which allows to finally report results to excel using Python packages. The tool applies a Microservice-Pattern, this ensures expendable reliability and maintainability. To strengthen industry and allow the usage of these tool for the author's company and Catrobat, the tool will be made available via open source licenses. Following chapters will cover the addressed topics in detail.

3.2.1 REST-Requests

As mentioned, Jira, Azure DevOps and GitHub offer REST API's for internal usage and retrieving data from these platforms. Representational State Transfer (REST) is an architecture for software solutions building up on HTTP protocol. It works in a stateless way, so requests with same input get the same output and there is no builtin history or context. REST solutions may return resources having relationships in the form of identifiers to other resources and receiving similar URLs. The architecture is used due to attributes like the powerful performance in describing systems and its scalability in terms of number and supported content of requests. Also simple and qualitatively high implementations are possible and interfaces are clearly document-able, i.e. via a documentation tool called "swagger". A request to a RESTful service has predefined parts. A Request Verb defines the interaction with the requested data. Available are GET to retrieve data only, PUT to completely update, POST to add, DELETE to remove and PATCH to modify/update data. A header supplies crucial extra information like policies, authorization, format, expected response. Finally the body is used to deliver any required business data as JSON, array, document file or text. After submission of a request, a response is triggered which has the same parts as the request and thus contains a status code and optionally data in the body [Sujan et al., 2020].

Returned status codes starting between 100 and 199 contain connection information, 200-299 communicate the success of the operation, 300-399 mean further steps to be done to complete the request, 400-499 mean there is a client side misunderstanding or error and 500-599 communicates that there are problems on server side as <https://restfulapi.net/http-status-codes/> lists. Recognizable for REST-Services is that resources of names entities are accessible under the template URL "BASEURL/NamedEntity/ID". This concept is used for this thesis to gather data [Sujan et al., 2020].

To interact with a REST-Service a client needs to authorize using supported methods.

3.2.2 REST-Authorization

Authorization within the header of a request for RESTful services were handled by applying Basic Authorization or OAuth as authorization-method. Both only may be applied to connections between server and client over HTTPS ²² and for both the request headers need to contain an "Authorization"-property with credentials. If resources are only available for a defined user, OAuth needs to be applied to access them and if any user who is permitted to login, sees the information, Basic Authentication is sufficient. For Basic Authorization credentials are entered in the form of "username:password" encrypted with Base64 as encoding scheme. The described property for Basic Authorization equals the template "Authorization: Basic <encoded_characters>". The server receives these credentials via HTTPS, decodes it and allows or denies access for the user [Sujan et al., 2020].

OAuth is an authorization protocol to enable third-party-client access to a define-able scope of restricted resources by managing tokens. The user who owns the resources is called "Bearer" and may create, manage and share an access-token named "Bearer-Token" with a third party client/application. With these credentials, requests can be submitted by adding the token to the "Authorization:"-property with the template "Authorization: Bearer <Bearer-Token>" [Aaron Parecki, 2024]. For example the author has a Jira, GitHub and Azure-DevOps account and wants to access them with a tool. Jira/AzureDevOps and GitHub offers as feature that the author can create/configure a token and allow the application to retrieve the same data the author would see if he logged into the account.

3.2.3 Data-Representation

Data-Representation has an impact on storage memory consumption and on performance. For storing, processing and transmitting data, various formats have different benefits or disadvantages. Two important formats for data representation used in web applications are the Extensible Markup Language(XML)

²²Hyper Text Transfer Protocol Secure(HTTPS) means a client uses a secure version of the Hyper Text Transfer Protocol(HTTP) with Transport Layer Security(TLS) protocols to establish secure WWW-connections. (See ²³ for more details) Therefore a client is redirected to a server and exchanges information about certificates, metadata, supported protocols and ciphers. They apply a so called handshake to establish a connection and create a session which time-restricted permissions for a client to exchange business data with the server. [Satapathy and Livingston, 2016]

and JavaScript Object Notation(JSON) format. JSON is a format which stores data inside an object by using JavaScript syntax. The base object consists of properties with values or nested JavaScript-Syntax-Objects. These properties can be interpreted as object and accessed for most programming languages by using libraries or native support. In the beginning, JSON lacked optimized searching and loading, but JSPath, "select" and native browser support of loading Json, optimized the performance for its usage. Researches showed that JSON surpasses the processing time consumption of XML in PHP by a factor of four (see Šimec and Magličić, 2014, page 4, end of chapter 3). XML is a format which stores information inside text documents based on nested tags. Programming languages interpret this data in the form of a tree (data structure), which allow performance boosts in search and processing. There are libraries for optimal performance in tree processing, prominent are W3C Document Object Model(DOM),XDM, XPath. XML supports more data types than JSON, comes with libraries for transformation in other data types and querying predefined nodes over simple expressions. So JSON is faster and more light weight to process and transmit, but XML has benefits for complex structures with heterogeneous datatypes and transmission of documents [Šimec and Magličić, 2014]. Another representation for processing of potentially heterogeneous two-dimensional data is a structure(package) for Python called Pandas -Dataframe and is documented at ²⁴. It allows efficient processing on difficult data with changing types, content and gaps and is highly transformable and integrate-able into other Python libraries. The history tabs of Jira-Tasks in the author's tool include comments, persons, time and numbers. So Jira only offers this data as XML-format over its API. To enable simple and faster processing, this data is interpreted via XDM, then formatted and subsequently relevant information is extracted. Finally the data is transformed to JSON which is stored directly as column into the database. So the Json data is later simply processed as column of each task and directly accessed for data processing purposes.

3.2.4 Software-Architecture-Pattern

As already mentioned before, depending on the requirements of the area of application, software needs to fulfill properties like availability, maintainability and further system requirements. To create a common understanding for development and fulfill these requirements, experienced software developers decide to create a system by applying software architecture patterns. The best pattern is dependent from economic as well as technical business values like frequency and costs of deployments, required resiliency, product specifics, the platform and technologies, required functionality and services and also dependent on the development team and its size. Two prominent architecture patterns in the area of web applications are the Monolithic Architecture and the Microservice

²⁴<https://pandas.pydata.org/docs/reference/frame.html>

Architecture. The Monolithic Architecture is the traditional software system. It has collective technologies and libraries, components are connected to each other and the operative-success and building-success ends with a single failure in any subpart of the system. Commonly all business logic runs in a single process and central logic, which simplifies documentation and requires less structured interfaces, both leading to challenges for extensions and maintenance. In summary, Monolithic Architecture are less modifiable, scale able and often complex to understand due to growing sizes and limited maintenance of the technology stack, but also offer simplicity in development, deployment, hosting, testing and required development experience for creation. The Microservice Architecture is an approach to implement software in a way that business logic is chunked into small independent components with well defined autonomous behaviour. Its origin is the Service-Oriented Architecture(SOA) which splits up systems into smallest meaningful independent functionality and hosts them in a distributed way to allow resilience and individual development and deployment on modules. These modules require precisely defined interfaces called Application Programming Interface(API) and individual libraries and resources for storage or access. Modules are kept small in code size to be able to adapt and deploy modules individually, fast and frequently. Overall Microservice Architectures are a modern approach to implement software solutions in a resilient and maintainable. It is suitable for large teams and solutions and well extendable and integrate able. But they are complex to implement, require more configuration effort and complicate development activities like testing and debugging [Salaheddin and Ahmed, 2022].

The tool is built with the basic pattern of a micro-service architecture. Interfaces are uniformly and well defined over service classes with common definitions of functionality. Modules have independent configurations, connection handling, input, output and separated code. To simplify the development process, the complexity and time consumption for setup, a monolithic main class is offered to execute and control the program for local usage. For the final product, the code is compatible to be refactored to a full Microservice Architecture rapidly.

3.2.5 Software Selling Concepts - Open Source Licensing

Similar as in other business fields, the permission to usage, change and publish software may also be affected by licenses. Corresponding with the type of product and the business models of companies, there are 72 usable licenses collected/certified at the Open Source Initiative (OSI). They define conditions for the usage, restrict usable licenses for publishing developments and clarify ownership and allowed services. There is a trend in industry to move away from proprietary software which keeps code in the ownership of companies and as a secret, towards open-source software(OSS) with large communities, collaborations and new business concepts. While for proprietary software the

solution is distributed to create revenue and the source code remains hidden and is understood as a company's or individual's property by law, OSS has distributed code and is often developed by a platform with large teams. 58 licenses of the OSI are valid for OSS software licensing. OSS in this context means both, Free and Open Source Software(FOSS). With the exception of dual licensing which is a mixed form, researches differentiated 8 types of license based business model categorizations. Distinguished by the existence of a copyleft licence "Reciprocal" differ from "Academic" models which does not contain copyleft. "Restrictive" models offer software with conditions for usage, development and further distribution while "Non Restrictive" grant more freedom. "Public" guarantees free changeability, while "Viral" means linked usage. Finally there is the "General Public License" and all non general public related as "Everything else". There is also an informative ²⁵ website to filter licenses based on project requirements. Business Models which grant modifications and distribution with the condition of applying the same licensing type, are reciprocal. They restrict commercial usage dependent on the used license in a weak up to a strong way in terms of conditions for re-licensing. So revenue of Reciprocal Business Models is generated based on services like tools, support, individualization and maintenance. An example for a Reciprocal Business Model is the technology JBoss. Referred to as "BSD"- or "Apache"-Style, business models which may be further developed and integrated into proprietary products are named academic. A well known technology with a "BSD" license is "Enterprise DB". As multiple licenses per product are possible, academic licenses often apply only to core-functionality which are deployed as community edition and further functionality is commercial by using another license. Also mixed forms are possible. There is a business model called Dual where customer can use the software as open-source software, forced to publish modification again or decide to pay for a commercial license instead. "MySQL" is a technology with a dual-license Onetti and Sameer, 2008].

For the publication of this tool, an academic business type is a well fit choice. The Jira and GitHub extraction part and the report generation part may be freely usable as open version and analysis by charts is available in a premium version which can be used via code or by free educative licenses for nonprofit organizations and active students.

²⁵<http://opensource.org/lpc>

3.3 Project Catrobat as practical application of the Master Thesis Tool

Catrobat is an open-source software project, according to an idea of professor Wolfgang Slany, to enforce young people to get playfully into digital business fields. The aim is to teach Software Thinking and Design Thinking to young people world wide, by switching into the role of an app creator. According to ²⁶ Catrobats team-source is the Graz University of Technology, which allows Catrobat to benefit from cooperations cross discipline and research. This helps to consider many views and get feedback and ideas of young people. The project profits from its international character, support and the big development team. It requires only a smartphone, allows various combinations of elements and all services are offered for free. Catrobat is owned by the organization "The International Catrobat Association (Graz,Austria, ZVR 375638318)" and cooperates with "Massachusetts Insitute of Technology", "Berkeley University of California" and the "Play Learn Foundation" to support developing countries to decrease poverty (Starting with India) [The International Catrobat Association, 2024].

Formed by over 1100 contributors and available in more than 180 countries with more than 60 languages, Catrobat has the mission to prepare teenager and adults for an upcoming digital future. The nonprofit project also has a focus on fair and location- and social-status- independent educative chances. The visual programming style of Pocket Code lowers the entry barrier and supports young people to develop a programming mindset. There are less restrictions than in other program languages and thus programming can take place everywhere and by everyone. There is already a trend of raising jobs in the area of Information and Communication Technology(ICT) and also growing digital application and service offer. But still there are people who do not participate and can potentially profit from Catrobat and benefit for their digital future [The International Catrobat Association, 2024].

A part of Catrobat is the app "Pocket Code" which allows to run Catrobat programs native on IOS and Android. In the background the projects use mainly Java, Kotlin, Swift and C to cover creation and translation between Android and IOS. The development is inspired by the agile movement and use Test-Driven-Development, Pair Programming, CI. Also there is a focus on complete documentation, clean code and native testing. At ²⁷ there is a separate website to upload Catrobat apps, encourage the community and get influenced by the available apps. Another aim is to show young people that individual

²⁶<https://catrobat.org/about/>

²⁷<https://share.catrob.at/app/>

3 Solution, Architecture & Design

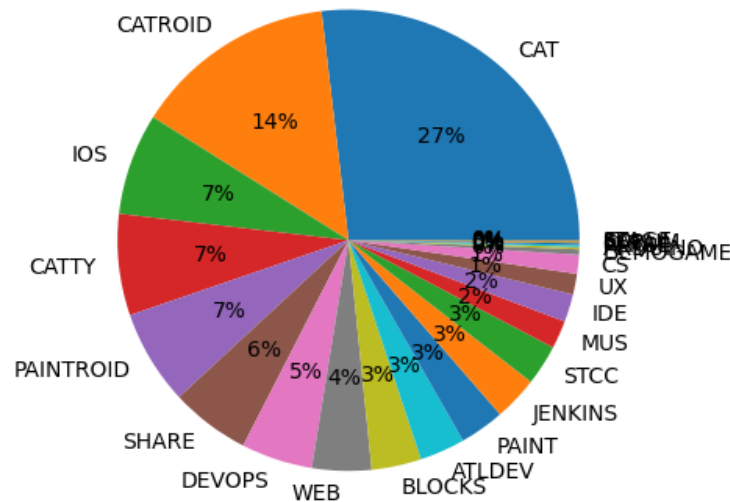


Figure 3.3: Tickets per Project of Catrobat

developers can benefit from each other by sharing code [The International Catrobat Association, 2024].

3.3.1 Catrobat - Infrastructure and Data Analysis

Catrobat has a GitHub organization at ²⁸ and a Jira-Instance at ²⁹, which both cover data of all projects visible on figure 3.3. Additionally, the websites at the "Web" project hosted on ³⁰ are maintained and there are the google play account of Catrobat at ³¹ and the IOS account at ³².

During the implementation of the solution within the master thesis, data analysis of Catrobat-Tickets were performed. The Catrobat-Jira covers over 10000 tickets in total.

Since 2012 an average of 953 tickets are created per year, figure 3.4 shows

²⁸<https://github.com/orgs/Catrobat/repositories>

²⁹<https://jira.catrob.at/secure/Dashboard.jspa>

³⁰<https://catrobat.org/>

³¹<https://play.google.com/store/apps/developer?id=Catrobat>

³²<https://apps.apple.com/us/developer/international-catrobat-association-verein-zur-foerderung/id1117935891>

3.3 Project Catrobat as practical application of the Master Thesis Tool

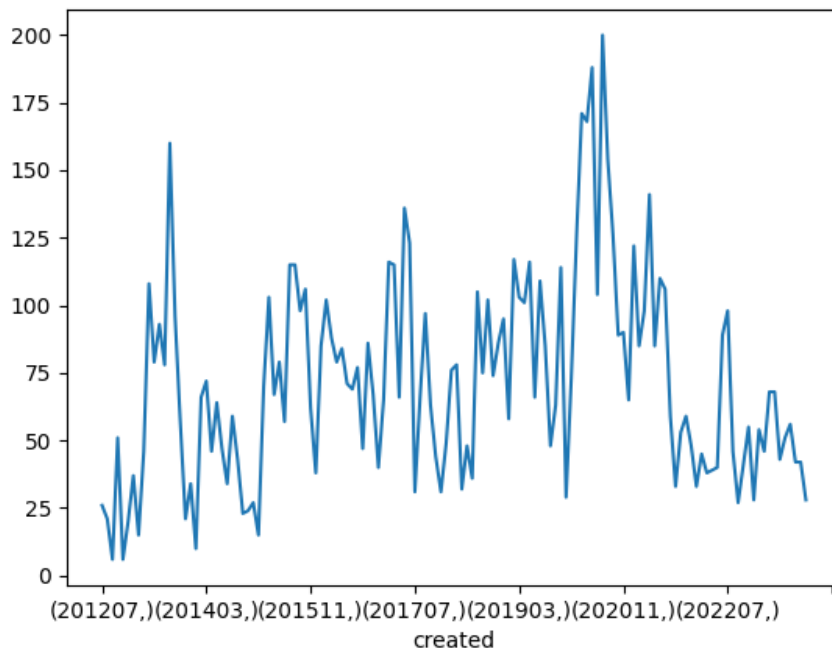


Figure 3.4: Catrobat Tickets created per month (y-axis: tickets, x-axis time[(YYYYMM,)])

their distribution over these years. Regarding active work on these tickets, as figure 3.5 shows, there are no tickets last-updated before 2014 and there are huge spikes where many tickets are updated in 2016 and 2020. This indicates that configurations on the board were applied backwards and the updated date is not a good evidence for progress per time. So there is a tool needed which evaluates the history data of the tickets.

3 Solution, Architecture & Design

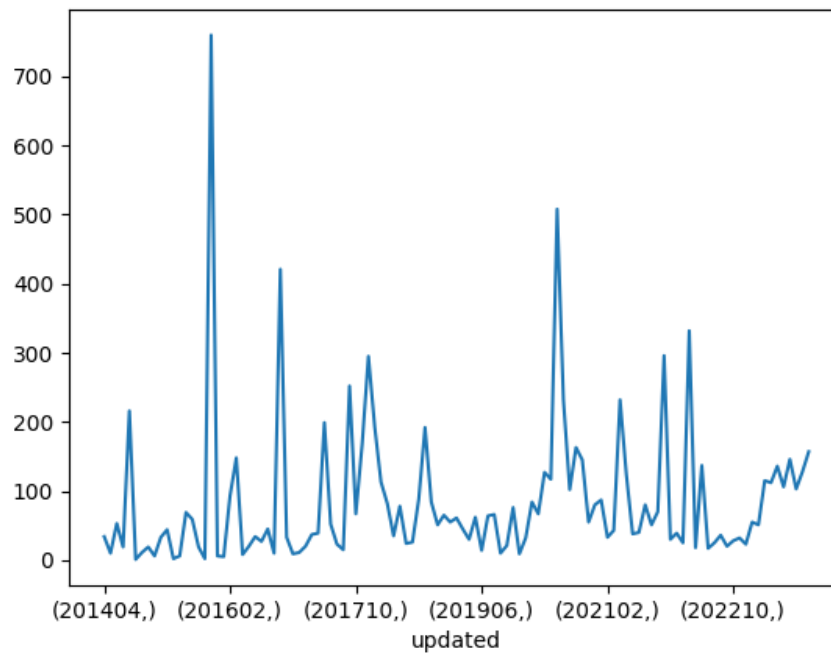


Figure 3.5: Catrobat tickets updated per month (y-axis: tickets, x-axis time[(YYYYMM,)])

The distribution of the tickets into type, priority and state also showed data inconsistency. There are a lot of individual tasks having states which do not match the main states in Catrobat as figure 3.8 indicates and 50% of the tickets have no priority assigned according to 3.6. From the author's experience most employees only set the priorities "Minor/Trivial" and "Major/Critical/Blocker" what means "None" may be interpreted as "Medium" priority. With 43% Bugs are the major task type. This may happen due to the large amount of contributors and high bug finding rate due to Test-Driven-Development. According to figure 3.7 there are less epics and subtasks, this comes from less large developments and that stories are separated fine-granularly or executed by a low number of persons at the same time .

3.3 Project Catrobat as practical application of the Master Thesis Tool

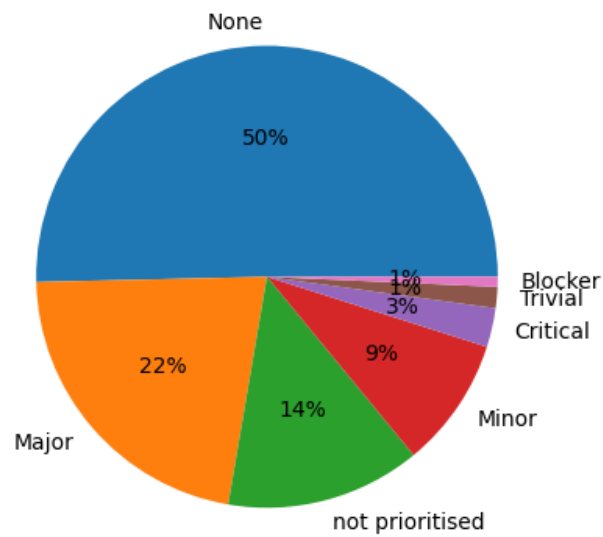


Figure 3.6: Priority distribution over all Catrobat tickets

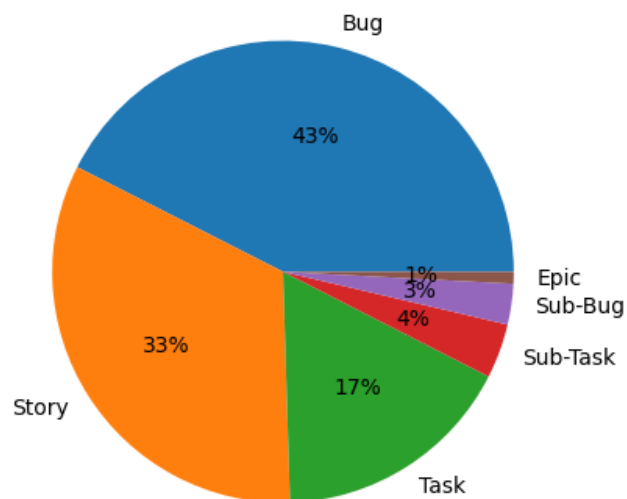


Figure 3.7: Issue-Type distribution over all Catrobat tickets

3 Solution, Architecture & Design

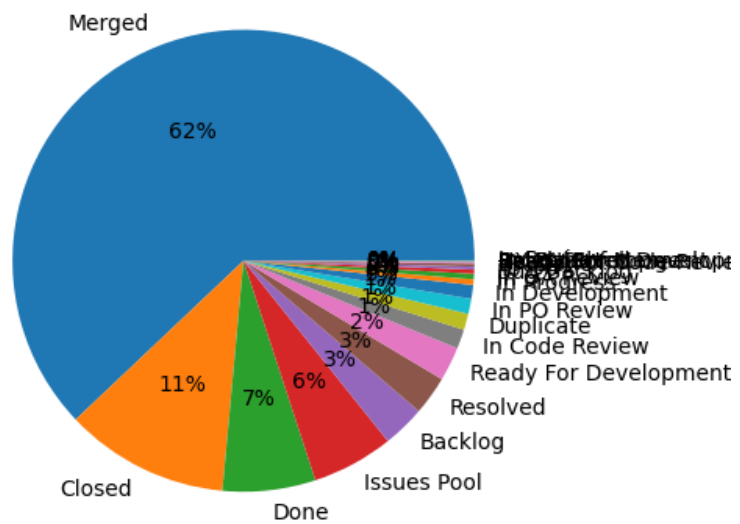


Figure 3.8: State Distribution over all Catrobat tickets

3.3 Project Catrobat as practical application of the Master Thesis Tool

Also the progress of the project and process is analysed via Jira visualization of linked issues. Inside the field "Issue Links" tickets which are developed together, originate from each other or benefit from each other, are linked. They are dependent from each other and may be affecting or waiting on each other. Figure 3.9 indicates many smaller groups of relations at the outer side of the image. This indicates that mostly smaller features get concurrently developed and affect few near tickets. At the middle of the network there is a large group of dependencies mainly between "CATROID", "CATTY", "CAT", "PANTROID" and "Blocks" topics. This indicates that there are complex features where know-how and requirements are distributed across projects, which complicates further tickets in this area. Also some of these related tickets block other tickets which stops progress in this area. It is recommendable that these issues get solved with high priority to prevent problems and circular dependencies in this area. Also the connections between the projects can be seen. The UX-team mostly supports single individual standalone tickets, while "CAT" and "CATROID" or "CAT" and "CATTY" tickets are often related.

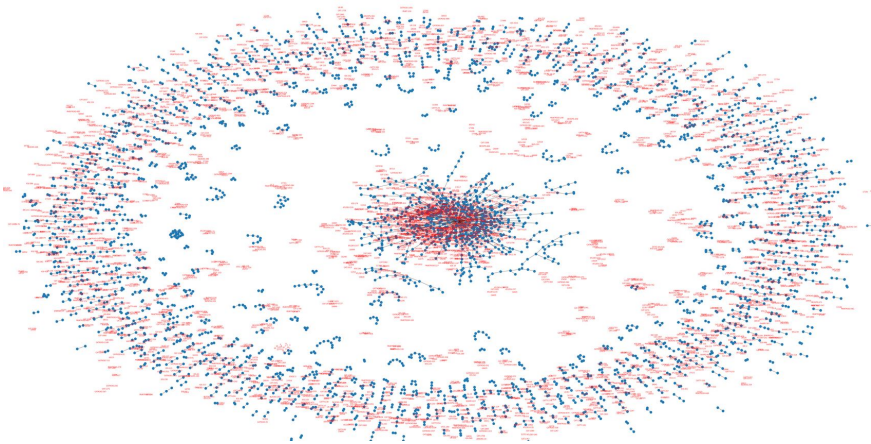


Figure 3.9: Screenshot of blocked task network for Catrobat Jira

3.4 Summary

As visible in this chapter, there are many steps to consider from the observation of an open problem in the field of SLA tracking to the final implementation of a tool which is automatising SLA activities. At the beginning there was the observation of manual activities required for SLA tracking and that some quality related measures do not get tracked for smaller projects due to effort related costs. Also the categorisation of tickets was complicated, as code related information over GitHub / Azure-DevOps was not considered. During research several APIs and some static pages providing extra data were found for multiple programming languages. Distinguishing between platform dependent paths for organization, private users and projects was challenging and took time for analysis. For memory / performance / project-usage reasons Java and Python were analysed more closely. Due to compare and the result of a feasibility test, Python surpassed Java in terms of available APIs and technologies, configuration effort and ability for rapid prototyping, which is highly required for flexible tasks like data retrieval from various platforms. During planning phase necessary data, dependencies and credentials were documented as well as required figures and fields of analysis and relevant metadata. The software was planned with a basic micro-service architecture due to the offered adaptability, modularity and resilience of this architecture. As data representation the XML format was used to retrieve datasets with multiple datatypes and Json was used for storage and processing. The data was requested from REST-APIs and used concepts were analysed. For the authorization requests were performed by using basic authorisation with local configurable usernames and tokens generated inside the platform. For publication, basis functionality is released as open source and extended functionality and analysis is available with a premium academic or educative version. As practical example the organization Catrobat was analysed by research and by code. The organization has Jira, GitHub and personal data. There are more than 1000 contributors and over 10000 Jira tickets available with years of history per ticket. The tool collected information about state, priority and issue-type statistics, analysed the project affiliations of tickets and their relationships to each other. The relations of the tasks also allowed to interpret progress and process patterns of the daily work at Catrobat and to give recommendable solutions on blocks for the system.

After analysing requirements, required concepts for the implementation and the methodology of the creation process, the next chapter will cover all configuration, technical details and specifics about the usage of the tool.

4 Documentation & Handbook

4.1 Project Details

The aim of this tool is to minimize the degree of manual activities and maximize the data for SLA tracking. It allows the management to get continuous project data also for projects with small budgets and enables leaders to extract information of the quality and progress inside the project. The structure and applied patterns the author applied during the implementation of the tool are described in this chapter and visualized in figure 4.1. From a black box view, the pre-configured "main" class may consume azure data as .csv file and produce charts and project reports into the output folder, based on the program configuration.

Applying a Microservice Pattern Architecture, there are 4 different modules, offering SLA services for different platforms over API's. Each web-module consists at least of an independent configuration with platform specific settings, a connector class with functionality to access the platform and a crawler to gather and process data. The Jira crawler maps the data-class called "JiraTasksProjection" directly into the database scheme "project_management_tool" with the table "tasks" by using the functionality of the "JiraRepository" class. The Git and Azure-Crawler directly return an array with all data, while the result of the crawler for Jira gives back the length of the processed projects and enables to retrieve the result table from the database by using the "JiraService". Also the individual dependencies and platform specific packages are only used in the correlating module itself. A different kind of module is the "reportModule" which provides functionality for analysing, plotting and reporting metadata accessible over the API.

To simplify rapid prototyping, testing and configuration during development, the main class is used to apply a Monolithic Application Architecture Pattern by combining all modules functionally inside a single executable system. The application may be started with different kinds of configuration sets. The usage of the database can be disabled, Jira data can be used only from the database and thus crawling can be skipped and reporting as well as the usage of platforms can be modified. Possible usages are firstly Jira or Azure, secondly Jira and GitHub or Azure and GitHub, or lastly simply all platforms. All crawled data is collected in data-frames and split project-wise by configurable predefined

4 Documentation & Handbook

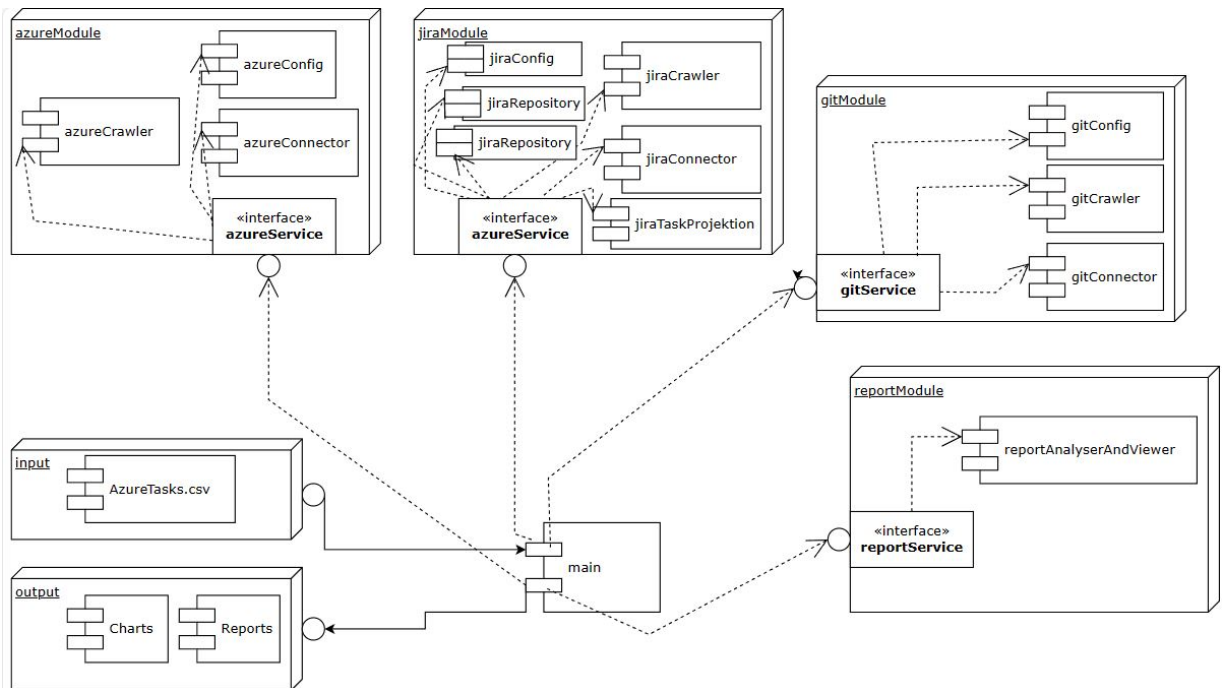


Figure 4.1: Modular structure of the tool

named recognition-fields. Finally one report-file per defined project is created and also various charts which visualize metadata of the individual systems.

While the connector classes simply provide names, passwords, instances and access for the platforms and while service classes are building an interface and call functionality, the crawler classes - dealing with metadata retrieval from platform APIs - are the most complex parts of the tool.

As the flowchart in figure 4.2 shows, the Jira-Crawler is executed if the corresponding configuration variable (IS_JIRA_USED) is enabled. After establishing necessary configurations a variable (IMPORT_New_DATA_FROM_SYSTEM) decides if the existing data is directly used from the database or new data is collected by the crawler and stored in the database. If new data should be crawled, the data crawler calls the "read"-function for all projects where the code collects all tickets inside a list with their project as line-entry in a multi threaded way. The available projects are known through a Jira API node and the number of used threads for ticket-retrieval equals the number of available projects. Finally all tickets are added to the database table and retrieved via getAllTasks to check their existence. In the end the crawler returns the number of tickets in the database and the tickets are gathered over the database and used within a dataframe.

Regarding the Azure-Crawler figure 4.3 shows that the crawler is structured similarly with platform specific adaptations and without storage due to security

considerations. A variable (IS_Azure_USED) defines if the crawler is executed. During execution the crawler reads all projects by collecting available identifiers(ids) from an input file and iterating over the array by building chunks of item ids. It collects a variable ("maximalTasksPerRequest") number of ids, makes a request and appends the task to a list. This is repeated until there are no unrequested IDs left. At the end a list with all visible Azure-Tasks of the user is provided by the crawler function.

As figure 4.4 shows for the Git crawler, a variable called "IS_GIT_USED" defines if the metadata is gathered. Depending on the supplied users and the position of the repositories, "readAllProjectsOfOrganization" or "readAllProjectsOfUser" is called. Currently the organization option is used by default. The crawler iterates through all visible repositories in the organizations GitHub and requests their commits in the form of hashes(SHA). Then for each hash, the corresponding commit messages are retrieved and stored repository-wise in an array. At the end this list is returned from the crawler.

After the documentation of the aim, architecture and core areas of the application, a definition of the exact outputs of the tool is provided as the main purpose of the SLA tool is the generation of SLA templates and meta data analyse.

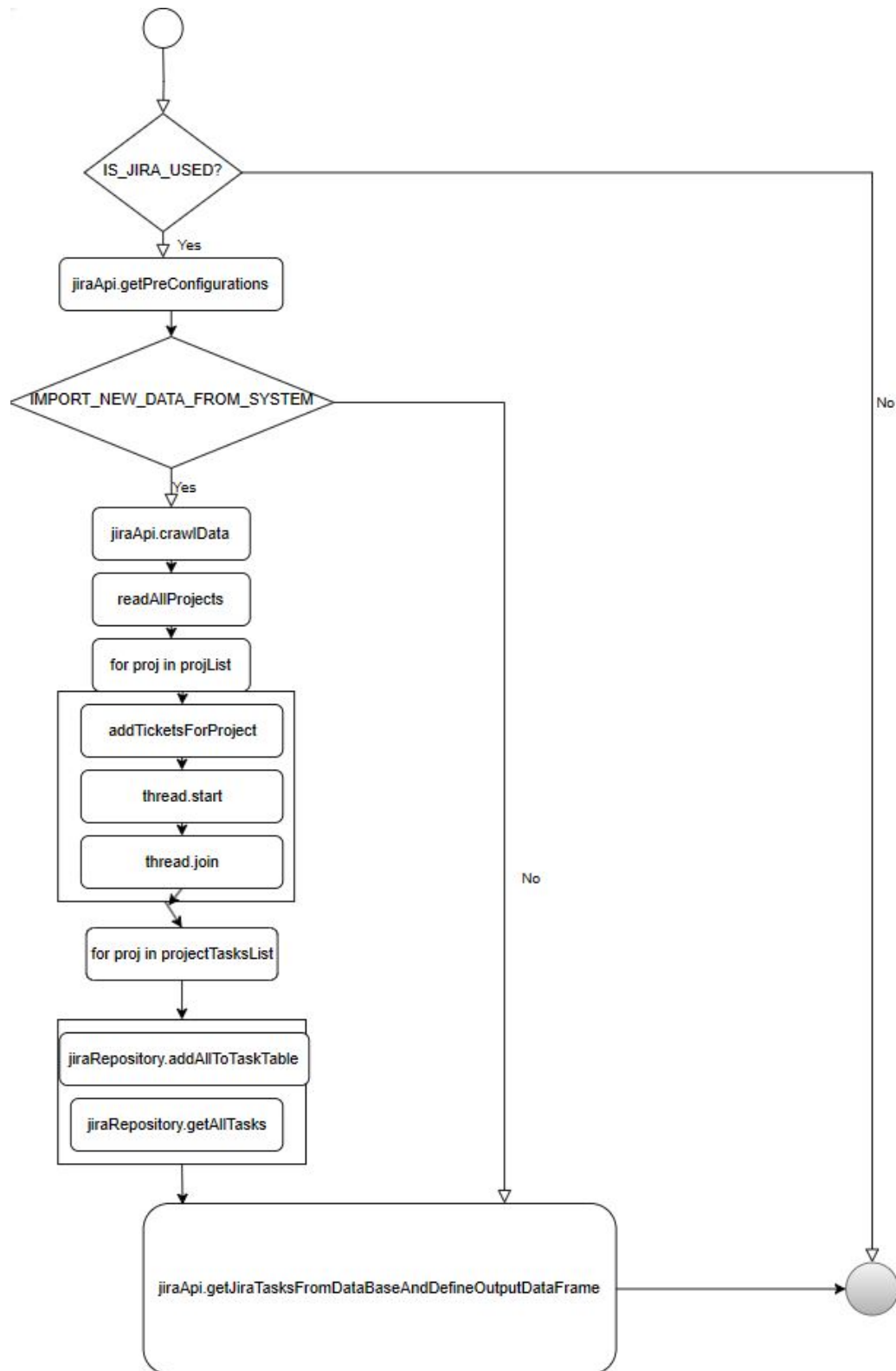


Figure 4.2: UML flow of the jiraCrawler

4.1 Project Details

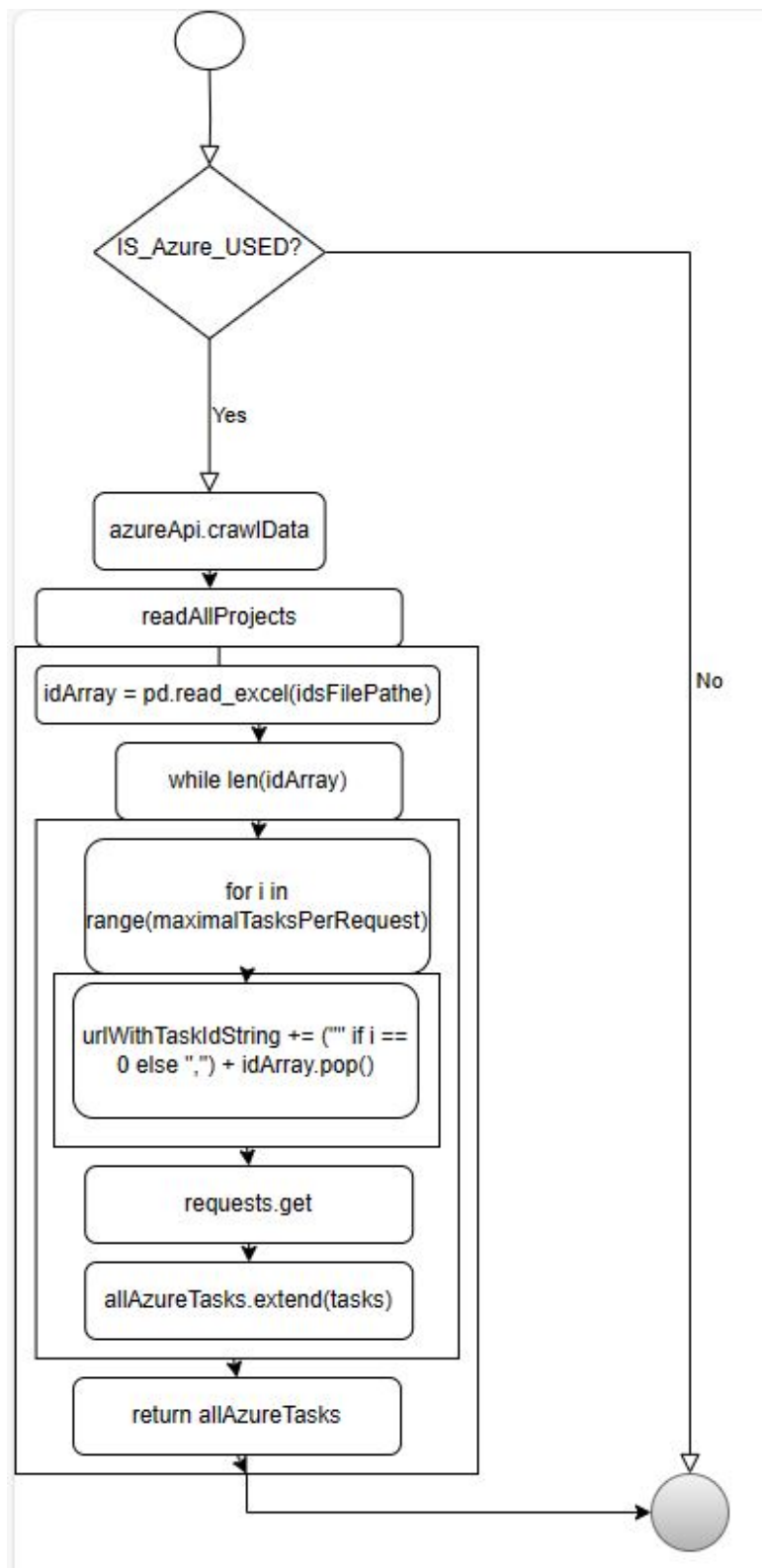


Figure 4.3: UML flow of the `azureCrawler`

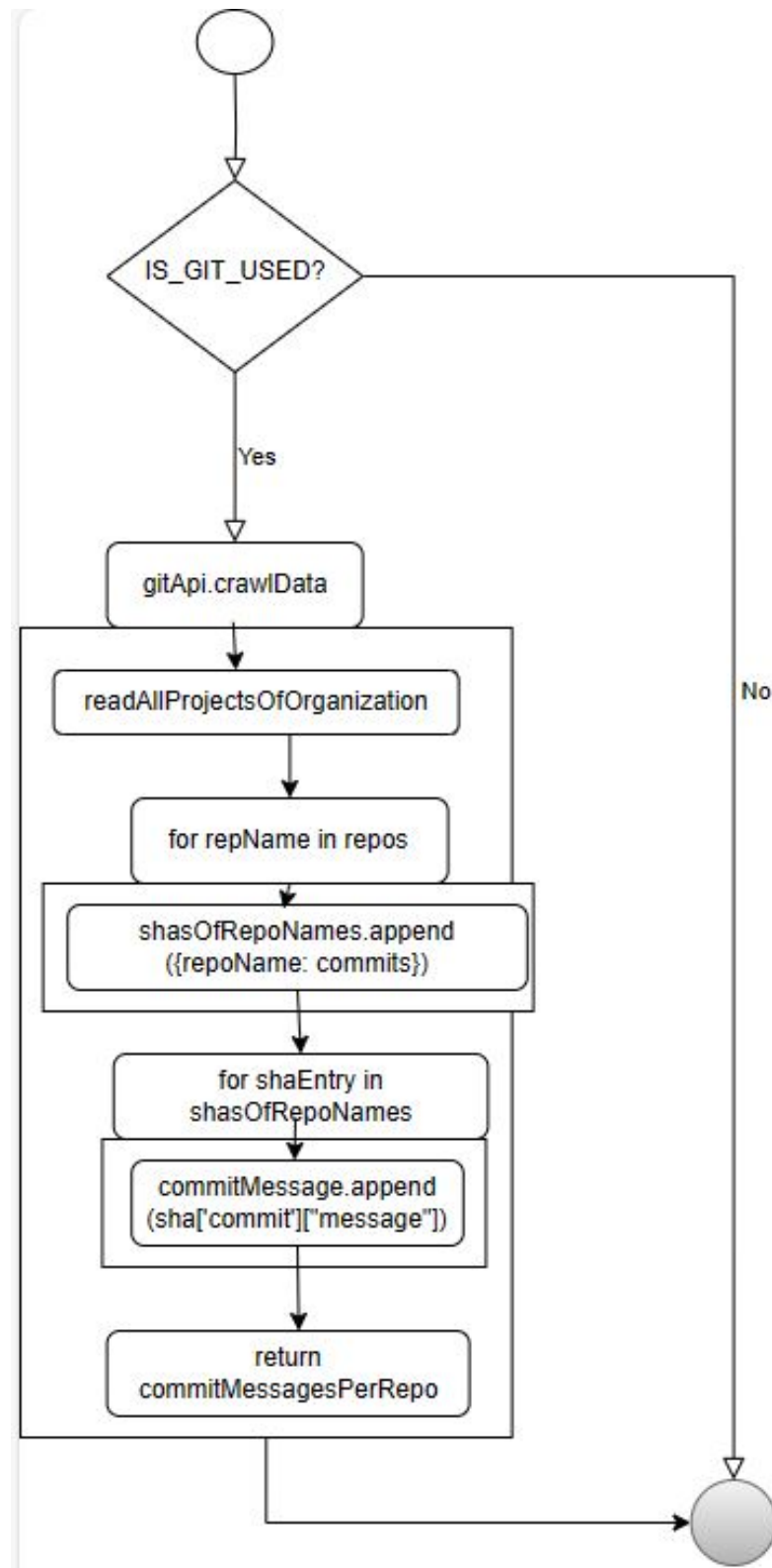


Figure 4.4: UML flow of the gitCrawler

4.2 Interfaces, Neighbour-systems and Limitations

4.2.1 Azure-DevOps - Configuration

Depending on the setup of Azure-DevOps the "Security" tab may be structured differently or be hidden, but Azure-DevOps generally supports the creation of personalized tokens over the UI. A token can be created by navigating and clicking the profile image on the top right corner and navigating to the "Security" option of the unfolded occurred menu. The opened screen is named "User Settings" and has the tab "Personal access tokens" selected by default now. There is a button which shows "New Token", which can be clicked to create a token. On click a dialog pops up and covers the right side of the screen. This dialog allows to define the name, expiry data, organization and scope of the token. Individual features can be allowed or restricted by using options like "Read", "Read & Write" or "Read, Write & Manage". After configuring the permissions of the token and pressing the "create" button, the token secret pops up and has to be copied. It will not be visible again. Finally the "Personal access token"-screen appears again and existing tokens are rendered inside a table including their key data. The configuration page may be directly accessed by navigating to the link-template "AZURE_DEVOPS_BASE_usersSettings\tokens" ("AZURE_DEVOPS_BASE" needs to be replaced with the correct Azure-DevOps instance). The configuration field in the author's tool is named "AzureToken".

As mentioned in chapter 3 regarding limitations, the amount of retrievable work-items per request is per default limited to 200 and the regarding API has security settings leading to a failed request if at least one ticket is not accessible. The amount of tickets in an Azure-Server is only restricted by the data type and thus a large number of tickets can be present and individual requests per ticket are therefore not feasible. The configuration field called "AzureRetrievedItemsPerRequest" allows to define how much work items are covered by the tool per request. The tool therefore requires an input file with an export of all IDs of Azure-DevOps tasks the user is able to access. This export file is create-able over the Azure-DevOps-"Board"-Tab under the sub-menu called "Queries". A query can be added with a one-line condition on the Field "ID" with operator > and value 0. After the execution of the query the exportable fields can be adapted by selecting desired columns on the popup menu of the button "column options". Finally the button "Export to csv" allows the download of the required file. The tool reads this file from a path which is configurable in a tool-field called "AzureInputAccessibleIdsFileName". With this information all available tickets can be gathered successfully.

4.2.2 Jira - Configuration

Depending on the configuration/version of Jira the grouping of settings may vary and tabs may be hidden, but Jira supports the creation of personalized tokens over the UI. To create a token, log in to Jira and navigate to the profile image on the top right corner and click it. Press in the unfolded menu on the "Profile" option and see a configuration manager screen. If there is no tab-menu on the left side, an administrator has to be contacted to validate the account's¹ roles. Otherwise if the tab "Personal Access Tokens" is present, press it and see the final token creation side. This is the second item of the tab-menu on the left screen side. Hit the button "create token" centered in the bottom side of the screen and a dialog pops up. Type a token name into the first field and type an expiring date or deselect the "Automatic expiry" option above the field. After clicking on "Create" a popup shows a long text-sequence on the screen with a copy icon on the right side of the field. Copy and store it, as this code will never be visible again. After hitting the "close" button, there is a table with all valid tokens and their key data on the "Personal Access Token" tab of the profile management section. It is at each time possible to clear all tokens on the "Summary" tab or to "revoke" single tabs in the previously described table in case of security incidents or unnecessary tokens. These pages can also be directly accessed for users which are logged-in and have permissions by navigating to the link-template

```
"JIRA_BASE_PATH/secure/ViewProfile.jspa?SelectTab =
com.atlassian.pats - pats - plugin : jira - user - personal - access -
tokens".
```

In this context "JIRA_BASE_PATH" needs to be replaced with the correct Jira server address. As the tool does not need to distinguish between password and token, the corresponding field in the configuration of the author's tool is called "JiraPassword".

For the feature of historic information about Jira-Tickets, there is a configurable field called "ActivityFieldURLParts" in the configuration of the Jira module. This field contains a pattern to retrieve the XML activity of any task in Jira. It is split up in 4 parts and consumes the exact task data during the request. Only the first part has to be adapted when changing the JiraServer. The first part is a template like

```
"JIRA_SERVER_BASE_PATH/activity?maxResults=",
```

with "JIRA_SERVER_BASE_PATH" as server-address. Appended to this information a number of maximal result rows has to be inserted. 1000 is used as

¹<https://support.atlassian.com/user-management/docs/give-users-admin-permissions/>

default value for the tool, as the standard default value of 20 is too low for tasks with long histories. The second path is followed by the ticket in a template of "projectKey-number" and looks like:

`"&streams=issue-key+IS+"`.

The final two text sequences are

`"&streams=key+IS+"` and `"&os_authType=basic&title=undefined"`,

the project key is inserted between these two URLs. This static link was observed by the author at the activity tab of each ticket in Jira. The data and address can be looked up when clicking the orange icon called "feed" which is located on the right side of the tab bar with the "Activity option" and colored orange with a white inverse wlan symbol inside. An example for the whole string is:

*"https : //jira.FANTASYSERVER.at/activity?maxResults = 1000
&streams = issue - key + IS + MYPROJECT - 49&streams = key +
IS + MYPROJECT&os_authType = basic&title = undefined"*

4.2.3 GitHub - Configuration

To ensure access to the GitHub metadata by using the tool, one has to log into a GitHub account which has access to all relevant projects. Then the profile icon on the right top corner has to be clicked and then the option "settings" inside the popup on the right screen side. Now a screen is visible with a menu on the left and elements on the right side according to the selected tab. The last option on the left side is named "Developer settings" and has to be selected. Finally a screen is rendered, where 3 main options are on the left side of the screen and regarding elements are on the right side. Choose "Personal Access Tokens" and the sub tab "Tokens (classic)". There is now a button with an arrow on the top right side of the screen. Click this button and select "Generate new Token (classic)". On the right side of the screen there is now the possibility to type notes, add an expiring date and define for this token exact permissions for the management of repositories, administration and packages. After clicking the button "Generate token" on the bottom of the page, a popup with a token inside is shown, after saving this token and closing the dialog, all valid tokens can be looked up in a visualized table on the screen. There is also the possibility for users which are logged into GitHub to find the setting directly with the address ² and generate the token. The corresponding field in the configuration of the author's tool is called "GitToken".

²<https://github.com/settings/tokens/>

4.2.4 Output Reports - Microsoft Excel

Excel was chosen as output tool for the project-wise SLA reporting due to its large business distribution and its performance in simple visualization. Each Excel has seven tabs with different platform-data. Namely "Overview", "Jira", "Azure", "Git", "TimeBookingData", "Figures", "Statistics". The tabs are filled optionally on usage, but they are generated for any configuration.

The **"Overview"-tab** covers the name of the project, the number of Jira-tickets inside the project and the amount of stories/bugs among the tickets. Further a list of people who are assigned or mentioned, the earliest created-date among all tickets and the latest. Also there is space left for the user to insert a project description and project contract numbers and ticket -plans / -estimations. All calculations are based on tool data and not in time generated for changes in excel.

The **"Jira"-tab** contains 35 columns with standard and customized Jira-field-names as header and corresponding ticket data on each row. The fields "comments", "labels" and "issuelinks" contain calculated content, due to the length and limited readability of the native data structure. All other fields are directly printed from the Jira tasks. This is the reason why the comment-rows just show a number of comments inside the task to indicate the effort regarding communication per task. For the "issuelink", only the linked numbers are presented in square brackets and for the "labels" the text is shown.

The most important **"Azure"-tab** fields to filter for current work, are the "TeamProject", "AreaPath" and "IterationPath" as they may restrict the visibility of tickets at the Azure-DevOps-Board. For SLA analysis the "priority", "comment-count", "Title", "createDate", "StateChangeDate" and "ChangedBy" are most relevant for indicating the state, progress and effort of the project. The field "CustomerIssueID" is a custom field which contains a link to a ticket in the Customer-Jira. In total there are 21 Azure fields they may be mapped to Jira fields by "CustomerIssueID" or "Title".

For **GitHub** there are two columns, the "Ticket-Number" and the "Commit-Message" column. The field "Ticket-Number" is just the prefix of the corresponding "Commit-Message" - field starting with the pattern "Project-TicketNumber", or an empty text if the pattern is not present.

Another tab is called **"TimeBookingData"** and contains company data with the project related time booking of employees. This data is for most tools exportable as .csv and can simply be copied into the report for evaluations. Most important are the fields "Datum", "Pro - Nr", "Zeit" and "Beschreibung" to evaluate effort related to the project and reasoning for effort. Time booking is mostly done with individual tools. A possible automatism for the future is to retrieve the file over a request from a link-file-share.

There is also a tab which shows excel figures of other tabs. The **"Figures"-Tab** visualizes the statistics and including a chart for states, priorities and labels.

Finally the **"Statistic"-Tab** contains the sum of comments over all tickets, the total number of tickets, the top 4 priorities with their frequency of occurrence and also the top 5 labels and states with their relative frequency.

Now the details of the tool and its output are clarified. The figures which provide metadata are shown at the end of the "Solution, Architecture & Design"-chapter. The next section will evaluate the economic worth of this data and will provide a practical industry context for the tool.

4.3 Economic Benefits

In order to estimate the economic impact of the tool for industrial use as well as the need for the tool and its costs, the author performed a study for project leaders. Each project leader was asked about specific values of his projects to gather information about the potential value of planning / estimation the tool may create. The results were that Jira was used in 89% of the projects, only 33% of the projects have regular SLA tracking and 56% of the projects have data distributed on different platforms. In the survey also the number of tickets per month and the consumed time per project on manual SLA tracking was asked. Using the daily rate the company sells for services, based on the company location and the average project mixtures of different locations, an average daily and hourly rate resulted for manually services. For each project the amount of tickets and the consumed time per ticket were considered to calculate SLA costs related to manual SLA-tracking per month based on hourly rates. The result was 263,94€ for the generation of the reports and 563,21€ per month for data collection. Accordingly per team and month a sales volume of 827,15€ can be acquired by using the tool. To start a project and apply for a budget in the company, sales, fixed costs, variable costs and an economic metric called "return of invest"(ROI) has to be calculated for an idea. With development costs as fixed costs and existing free infrastructure capacities which reduce variable costs and under the assumption the tool runs similar to other standalone tools in the company after 3 years, a ROI of 0,2515 was calculated. With a calculation period of 3 years, the amortization time was 16 months including costs for a service team based on a growing amount of users over three years. All sales were calculated with minimal assumptions and all effort estimated on maximal realistic assumptions. The ROI value is equal to another project the management is interested in. 77% of project leaders are interested to use the tool and with over 20 teams in the company the potential profit per month due to reduced effort and time consumption is in the 5 digit range per month for this company. Further benefits are the statistics about the systems, the documentation effect regarding tickets, the information about processes and interactions and guaranteed warning signs regarding the compliance with the SLA contracts. The survey was made in the project environment of maintenance

and development of web applications in a state of the art company with more than a hundred employees and decades of experience in software services .

4.4 Summary

A dynamic solution for gathering SLA related data and creating SLA-reports is described in the "Documentation & Handbook"-chapter. It is a Python tool, retrieving data from various platforms and supporting highly dynamic configuration. Due to the applied software architecture patterns, it benefits in maintainability, modularity and extensibility from applied Microservice Architecture influences and remains testable, simple in configuration and debugging by being hosted for development purposes in a single sequential program of Monolithic Architecture. The tools collect all available commit messages of a GitHub user or organization, all tasks an Azure-DevOps user can access and all projects and their tasks in Jira. Also metadata of these platforms is gathered and analysed by the tool. The reporting module creates pie-charts, relationship networks, wordclouds and excel templates with one tab per platform and automatic statistics in other tabs. There is one report created per Jira project and the platforms are mapped by configurable matching identifier fields or names. The structure of the program and complex parts like the data crawlers are visualized by using the Unified Modelling Language(UML) standard. The crawlers can be enabled or disabled by configuration variables and gathered during operation data from individual platform APIs by using REST requests. The Jira crawler also stores all tickets with project tags inside a local database. It is configurable whether new data is retrieved regularly or the stored data is used. This and the multi-threaded operation of the crawler have a big impact on the execution speed of the program. The chapter also contains the most important configuration instructions and documentation of output fields in the template. Jira, GitHub and Azure-DevOps use user specific tokens, which can be generated and modified in terms of permissions over the UIs of the platforms. It must be considered that the token may expire by given dates and the setting menus of the platform may vary or change with the platform-version in use. The output data of the tool is finally evaluated regarding its economic and project value for software industry. It turned out to indicate a beneficial project process and operative- as well as higher- management is interested in the productive use of the solutions.

5 Lessons Learned

Planning and researching is the most important phase of the master thesis. Issues with libraries and potential features which are not found, increase the effort during implementation dramatically. Sub-sequentially projects should have a researching phase of at least a quarter of the practical thesis time. Especially between the different platforms "Jira" and "Azure-DevOs", limitations and the structure of the web pages were very different and limitation was hard to find. In the end sources for any challenge support or documentation were available in the internet and the problem could have been identified earlier or even avoided. Selecting Python over Java as programming language was a good decision, as modifications and implementations are much faster and simpler to configure. Gathering data from different APIs is very prone to error, due to less code restrictions compared to Java. It was important to use Python in a way that changing metadata and types as well as the amount of data does not cause issues. Storage mapping and transformation of the data by using Python packages was the key to that. Also a good decision was the local storage and mapping of Jira tasks. During the implementation process, the platform was several times offline due to maintenance when the tool wanted to access it. In this context also good error messages and careful storage of documentation of used libraries are helpful. It is important that software patterns and technologies need to be applied early on as complexity rises with the amount of functionality. Also the solution must be adapted to the concept of the platforms and structure of the APIs. Finally it is crucial to directly document all changes to individual lines of code, as knowledge decreases with the growth of the program and time passed. On the whole the implementation of the tool, as well as the documentation and writing process of the master thesis worked very fluently, but the mentioned learned concepts may reduce debugging and consumed time if a similar task will be executed by the author.

6 Conclusion and Future Work

Considering the most important platforms for Scrum based maintenance teams, the author developed a tool for optimizing the process of SLA tracking in software projects. As the field of Service Level Agreements (SLA) lacks standardization and readability for complex systems, the tool is published as open source software with the aim of forming a community for a standard solution. This minimizes the manual effort in a way that projects with small budgets may also gather SLA and that result reports and system analysis can be standardized. Furthermore, the tool gives information about the process and the progress of individual projects and systems, including data consistency, history and data relationships. With the benefit of a Microservice Architecture, the tool is modular maintainable, may be processed in parallel or in the cloud and is well structured. It deals with GitHub, Jira and Azure DevOps as API and analyses systems by creating charts, wordclouds and networks from the system and automatic excel reports per project. The tool was successfully applied at the organization Catrobat and in a mid sized company in the private IT sector. Requirements from these fields were to deal with over 1000 contributors on one page or collect data of multiple projects from various platforms and evaluate them project-wise. An industry user study indicated a large business worth for the tool by analysing the effort reduction for SLA tracking regarding costs and project potential. Metrics like effort through task-comments, required code-changes, linked tasks and know-how transfer between persons can be reported. Especially for support contract negotiation and project quality control, the project creates new possibilities and solves open challenges. During the implementation process the author documented many API related experiences regarding limitations, platform specifics and libraries. Applied to the project Catrobat and projects in the author's company, the tool is already in use for both, profit- and nonprofit- software industry.

The tool will be published by a creative common license to enable both Catrobat and private software industry to benefit from this master thesis. The project team of Catrobat will analyse their boards and processes based on the output of the tool and the author will apply the software in industry projects. The author plans to create a common cluster service where individual tools may be hosted independent of their technology by dynamic configurable platforms. There will be a simple web application which offers express documentation to execute

6 Conclusion and Future Work

this service for a project. Project leaders decide on their own if and where they apply the tools. The SLA tool will be the first hosted individual software and will be the central point for SLA related information. Developments on the tool must be uploaded to be shared with the whole community. Further platforms may be optionally usable for the SLA report and the metrics may be standardized and extended. The tool may be hosted by a webservice with a full microservice architecture and may be integratable into platforms like Jira. The import of Azure data was implemented in a feasible way for the time frame, optimally, automatic queries must be sent to Azure-DevOps as REST request to read accessible tickets and only retrieve relevant tickets of projects. As soon as only relevant data of projects is covered and the data is filtered in Azure-DevOps, the Azure crawler and GitHub crawler may also store the data inside the database. Additionally the tool has potential as documentation software and cross platform search engine. Advanced password checks for projects which are subject to security restrictions may also be possible as specific know-how search across platforms and projects. Also automatic documentation files or release logs based on GitHub commits are possible. Altogether the tool deals with an area of growing business importance and small degree of standardization. It also has multiple possible interfaces to business relevant topics like searching, documentation and software quality. Issue tracking systems are spread all over IT industry and the tool is implemented dynamically and extensible for the most used systems. The tool has a good chance to be further developed and may be applied to many contexts in industry.

Bibliography

- Aaron Parecki, o. (2024, January). *Oauth authorization* @ONLINE. <https://oauth.net/2/> (cit. on p. 35).
- Aglibar, K., & Rodelas, N. (2022, February). *Impact of critical and auto ticket: Analysis for management and workers productivity in using a ticketing system*. <https://arxiv.org/abs/2203.03709> (cit. on pp. 15, 20).
- Alomari, Z., Halimi, O. E., Sivaprasad, K., & Pandit, C. (2015). Comparative studies of six programming languages. (Cit. on p. 26).
- Arya, D., Wang, W., Guo, J. L., & Cheng, J. (2019). Analysis and detection of information types of open source software issue discussions, 454–464. <https://doi.org/10.1109/ICSE.2019.00058> (cit. on p. 15).
- atlassian, J. S. (2023a, December). *Jira documentation azure compare link*. <https://www.atlassian.com/software/jira/comparison/jira-vs-azure-devops> (cit. on pp. 16, 17).
- atlassian, J. S. (2023b, December). *Jira documentation issue types link*. <https://support.atlassian.com/jira-cloud-administration/docs/what-are-issue-types/> (cit. on p. 16).
- atlassian, J. S. (2023c, December). *Jira documentation link*. <https://www.atlassian.com/software/jira> (cit. on p. 16).
- Kumar, B. (2015). The sway of agile processes over software maintainability. *International Journal of Computer Applications*, 109, 25–29. <https://doi.org/10.5120/19152-0581> (cit. on p. 5).
- Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A survey of DevOps concepts and challenges. *ACM Computing Surveys*, 52(6), 1–35. <https://doi.org/10.1145/3359981> (cit. on pp. 11, 12).
- Mens, T., Roover, C. D., & Cleve, A. (Eds.). (2023). *Software ecosystems*. Springer International Publishing. <https://doi.org/10.1007/978-3-031-36060-2> (cit. on p. 2).
- Microsoft, A. (2023a, December). *Azure devops framework* @ONLINE. <https://azure.microsoft.com/de-de/products/devops> (cit. on pp. 17, 18).
- Microsoft, A. (2023b, December). *Azure devops work item type* @ONLINE. <https://learn.microsoft.com/en-us/azure/devops/boards/configure-customize?view=azure-devops%5C&tabs=agile-process> (cit. on p. 18).
- Microsoft, A. (2023c, December). *Github docu* @ONLINE. <https://docs.github.com/de> (cit. on p. 19).

Bibliography

- Onetti, A., & Sameer, V. (2008). Licensing and business models. https://www.researchgate.net/publication/5127733-Licensing_and_Business_Models (cit. on p. 38).
- Oza, N., Fagerholm, F., & Münch, J. (2013). How does kanban impact communication and collaboration in software engineering teams? *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013 - Proceedings*. <https://doi.org/10.1109/CHASE.2013.6614747> (cit. on p. 13).
- Salaheddin, N., & Ahmed, N. (2022). Microservices vs. monolithic architectures [the differential structure between two architectures]. *MINAR International Journal of Applied Sciences and Technology*, 4, 484–490. <https://doi.org/10.47832/2717-8234.12.47> (cit. on p. 37).
- Satapathy, A., & Livingston, J. (2016). A comprehensive survey on ssl/ tls and their vulnerabilities. *International Journal of Computer Applications*, 153, 31–38. <https://doi.org/10.5120/ijca2016912063> (cit. on p. 35).
- Shaydulin, R., & Sybrandt, J. (2017). To agile, or not to agile: A comparison of software development methodologies. (Cit. on pp. 3, 5, 6).
- Šimec, A., & Magličić. (2014). Comparison of json and xml data formats. https://www.researchgate.net/publication/329707959-Comparison_of_JSON_and_XML_Data_Formats (cit. on p. 36).
- Sujan, Y., R. S. H., & Nagapadma, D. R. (2020). Survey paper: Framework of rest apis. <https://api.semanticscholar.org/CorpusID:235344060> (cit. on pp. 34, 35).
- Tawosi, V., Moussa, R., & Sarro, F. (2022). On the relationship between story points and development effort in agile open-source software, 183–194. <https://doi.org/10.1145/3544902.3546238> (cit. on p. 6).
- The International Catrobat Association, c. (2024, January). *Catrobat project information @ONLINE*. <https://developer.catrobat.org/> (cit. on pp. 39, 40).
- Trienekens, J., Bouman, J., & Zwan, M. (2004). Specification of service level agreements: Problems, principles and practices. *Software Quality Control*, 12, 43–57. <https://doi.org/10.1023/B:SQJO.0000013358.61395.96> (cit. on p. 4).
- turing.com. (2024, January). *Software-product-development-life-cycle @ONLINE*. <https://www.turing.com/blog/software-product-development-life-cycle/> (cit. on p. 5).
- Vijay, D., & Ganapathy, G. (2014). Guidelines to minimize cost of software quality in agile scrum process. *International Journal of Software Engineering & Applications*, 5. <https://doi.org/10.5121/ijsea.2014.5305> (cit. on pp. 9, 10).
- Wu, L., & Buyya, R. (2010). Service level agreement (sla) in utility computing systems. <https://arxiv.org/ftp/arxiv/papers/1010/1010.2881.pdf> (cit. on p. 21).