Bachelor Thesis

Software Development Optimization through Integrated Information Management: Conceptual Framework and Practical Implementation

for the Degree of

Bachelor of Science

submitted to the Department of Mathematics, Natural Sciences, and Computer Science at the Technische Hochschule Mittelhessen (University of Applied Sciences)

by

Benjamin Wirth

October 2, 2024

Referee: Prof. Dr. Dennis Priefer

Co-Referee: Kevin Linne

Declaration of the use of Generative AI

In accordance with the recommendation of the German Research Foundation (DFG - Deutsche Forschungsgemeinschaft)[1] and that of the journal Theoretical Computer Science[2] I (the author) herby declare the use of generative AI.

During the preparation of this work I used ChatGPT 4 in order to improve readability and language, only. After using ChatGPT 4, I reviewed and edited the content as needed and take full responsibility for the content of this thesis.

Declaration of Independence

I hereby declare that I have composed the present work independently and have not used any sources or aids other than those cited, and that all quotations have been clearly indicated.

Gießen, on October 2, 2024 Benjamin Wirth

1 DFG Formulates Guidelines for Dealing with Generative Models for Text and Image Creation: https://www.dfg.de/en/news/news-topics/announcements-proposals/2023/info-wissenschaft-23-72

2 Declaration of generative AI in scientific writing: https://www.sciencedirect.com/journal/theoretical-computer-science/publish/guide-for-authors

This bachelor thesis deals with the conception, prototypical development and introduction of a software solution that optimizes the software development process through the central aggregation and visualization of process-related data. In today's digitalized world, software is becoming increasingly important, while the development of software is a complex undertaking. The multitude of tools used and the large amount of information make it difficult to maintain an overview and make well-founded decisions in a timely manner.

The developed software addresses this problem by normalizing, aggregating and presenting data from different sources in a user-friendly way. The concept is based on extensive literature research and the analysis of existing software solutions in this area.

As part of the practical implementation at Janitza electronics GmbH, the impact of the software on the development process is examined. A positive influence on the company's development process is identified, particularly with regard to the improved overview and efficiency. This work thus lays a foundation for further research and development in the area of process optimization through integrated information management in software engineering.

Contents

1 Introduction

This thesis deals with the conception, prototypical development and introduction of a software solution that is intended to make the software development process more efficient through the aggregation and central visualization of data. This chapter presents the motivation for the work, formulates the research questions and explains the methodological approach to developing and introducing the software and answering the research questions.

1.1 Motivation

Software development has evolved into a significant, important, and highly complex process that encompasses many different sub-disciplines, all aiming to create high-quality software [BP20, p. 6]. These disciplines include various software development methodologies such as Agile, Scrum, each offering different approaches to project planning and execution [Sar24]. Additionally, quality management and risk management play a crucial role in ensuring that the software is not only functional but also reliable and secure. Other important areas include requirements management, which captures the needs and expectations of users, and software architecture, which defines the structure and design of the software [BP20, p. 21-32].

Software has become an essential component of nearly every aspect of daily life, serving a fundamental role in contemporary society. The market value of software solutions continues to increase, driven by ongoing digitalization and the growing demand for technological solutions. Figure 1.1 demonstrates a significant increase over the last few years. This growth is evident in various sectors, including critical infrastructures such as energy supply, healthcare [Car16], and transportation [Lov21]. The failure or malfunction of software in these areas can have severe consequences, highlighting the importance of high-quality software [MG24].

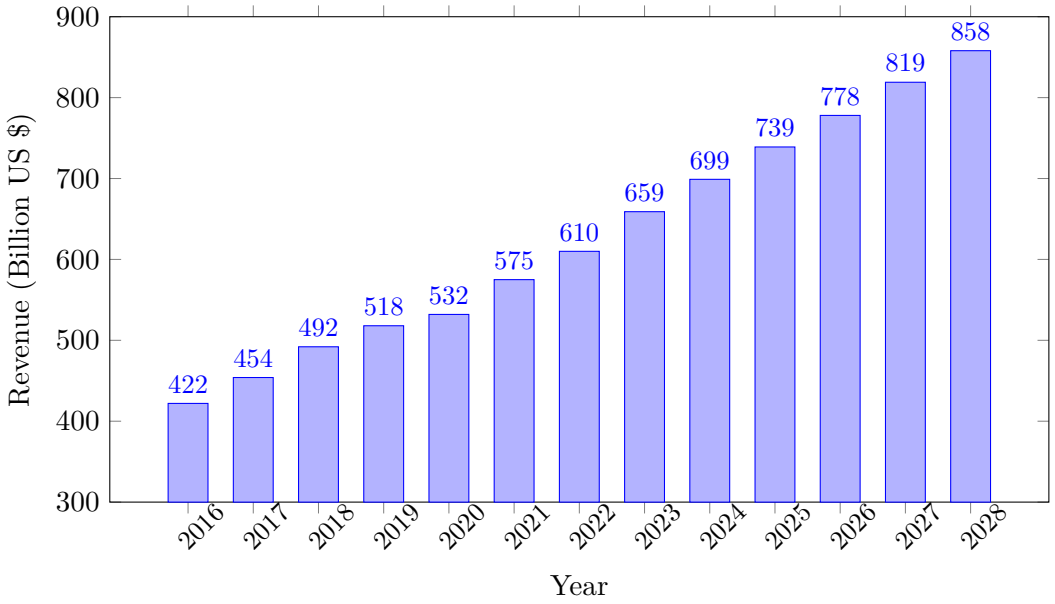

Figure 1.1: Worldwide Software Revenue from 2016 to 2028 in Billion US $ [Tav23]

However, as software development processes become more intricate, the amount of information generated and required to manage these processes has increased. Developers and project managers are often overwhelmed by the sheer amount of data coming from various tools and platforms used in different stages of development [Ant12]. These tools and practices, encompassing project tracking, version control, continuous integration, testing, code review and many more produce a diverse array of information, including bug reports, build statuses, feedback and many more. Managing and integrating this information efficiently is a significant challenge.

The complexity of modern software development means that relevant information must be timely and accessible to the right stakeholders. Developers need up-to-date information on code changes, test results, and deployment statuses. Project managers require insights into progress, resource allocation, and potential risks. Without effective aggregation and distribution of this information, critical data can be missed, leading to inefficiencies, miscommunications, and errors [Coi14].

Despite the availability of numerous tools to support the development process, challenges lie in integrating and managing the overwhelming amount of information from disparate sources. Inefficient information management can result in delayed decision-making, overlooked issues, and ultimately, a decline in software quality [And02].

1.2 Goal of this work

The aim of this thesis is to develop a concept for software that enhances the development process by consolidating information from various tools and practices utilized throughout the lifecycle. This consolidated information will be presented in a clear and visually coherent manner, tailored to the needs of the relevant stakeholders. The software must be flexible and generic enough to handle different types of information and their structures, allowing for implementations based on this concept to be adaptable to various development environments.

The following research questions arise from these objectives:

RQ1 What does a concept for software that consolidates and visualizes information from various tools in the software development process look like?

RQ2 What considerations and steps are necessary for a successful introduction and integration of such software within an organization?

RQ3 To what extent can the implementation of the proposed concept improve the efficiency and effectiveness of the software development process?

1.3 Methodology

This thesis follows a deductive and quantitative research approach. The thesis focuses on the hypothesis that software that aggregates and processes differently structured information generated during the software development process offers added value for software developers and other people involved in the software development process.

At the beginning of the thesis, a basic overview of the relevant topics is provided in order to create the necessary understanding. In particular, this includes modern methods in software development, data integration and information design, which form the thematic core of this thesis. In addition, relevant ISO standards in the field of software development are explained.

Subsequently, a comprehensive literature review serves as the basis for the software concept to be developed. Reference software solutions are also identified that can be assigned to the areas of data integration and information design in order to analyze and identify possible solutions. The findings from the literature research and the analysis of the reference software form the basis for the requirements definition of the software, which is documented using modern software development methods. The resulting software concept represents the well-founded answer to research question 1.

Since the introduction of the developed software in a company is also the subject of this thesis, a brief introduction to modern methods of software introduction is given. An in-depth literature review on this topic forms the basis for the software implementation concept, which answers research question 2.

The methodology for the quantitative answer to research question 3 is first determined by a literature review and then developed on this basis.

An exemplary implementation of the software based on the software concept is carried out at Janitza electronics GmbH. Accordingly, the introduction of the software will also be based on the developed concept for software introduction. The methodology previously defined for the third research question will also be carried out there in order to provide a final answer to the third research question.

1.4 Delimitation

The work focuses exclusively on the software itself, without an in-depth analysis of the specific data to be displayed in the software or which data could offer the greatest added value. Due to time constraints, no detailed analysis of the data to be displayed is carried out. Instead, for the implementation of the software at Janitza electronics GmbH, the opinions of the employees are used to determine the selection of the data to be displayed. Further literature research on this aspect will therefore not be carried out.

1.5 Structure of the Thesis

The thesis is structured into a total of six chapters.
The first chapter provides an overview of the topic and situates the work within its relevant context to enhance understanding of the background. Research questions are defined here, alongside the methods chosen to address them.
Chapter two provides an overview of the most important topics required to understand this work. These topics are presented in order to provide a basic understanding, as they will also subsequently form the core of the software concept, the introduction concept and the evaluation concept.
The third chapter focuses on elaborating the software concept, including integration and evaluation strategies aimed at determining whether the software has added value or improved the software development process in any way.
Chapter four involves implementing these concepts within Janitza electronics GmbH. Initially, a proof-of-concept is implemented. Subsequently, the software is integrated into the software development process according to the introduction strategy, and then

evaluated based on the evaluation framework.
The fifth chapter evaluates the developed software. The first step is to examine the extent to which the prototype solution implemented at Janitza electronics GmbH meets the requirements defined in the concept. After that the results of the evaluation of the software are presented and appropriate conclusions are drawn.
The sixth and final chapter provides a summary of the work carried out. Alternative approaches and possible future steps are also discussed. A critical reflection of the work, including an evaluation of the methods used and their implementation, is also provided. Finally, an outlook is given to round off the work.

2 Background

The following chapter provides an overview of the key topics required to understand this thesis. The focus is on current, modern and widespread practices in software development as well as on the central topics of data integration and information design, as these areas are an integral part of the software to be developed. In addition, relevant ISO standards that are used in software development are presented.

2.1 Software Development

The following section will provide a brief introduction to the topic of software development. It will focus on the different development methods and the tools used in the software development process.

2.1.1 Software Development Methods

Numerous concepts, methods and guidelines for software development have been developed. These can be divided into different categories: Concepts such as Scrum focus on project management, while others such as DevOps provide specific recommendations for action in order to deliver software efficiently and reliably. According to a study from 2022 [Git22], the DevOps and Scrum methods are used particularly frequently in software development, which is why they are examined in more detail here.

DevOps

DevOps is an approach that optimizes the collaboration between development, IT operations and business in order to be able to react more flexible and faster to market requirements. The central pillars here are corporate culture, automation, lean principles, measurement and knowledge sharing. The corporate culture, which is characterized by subsidiarity and leadership by example, plays a central role. Automation is achieved through CI/CD pipelines, while lean methods promote efficiency and value creation. Measurable results and a focus on added value increase business benefits. A "blame-free

culture" enables effective sharing of knowledge and experience [Hal20, p. 5ff].
There is also a particular focus on short feedback loops that enable rapid feedback to developers. Feedback here refers to the continuous evaluation and improvement of the software through direct feedback from operations, users, automated tests and, in particular, the results of automated pipelines. These short feedback loops allow developers to react quickly to problems and make continuous improvements. This not only leads to higher software quality, but also to faster adaptation to changing requirements and thus to increased business benefits [Hal20, p. 13].

Scrum

Scrum is an agile, lightweight approach to project management for software development. It structures the work in short, manageable units called sprints, which typically last two to four weeks. The requirements for the product are recorded in a product backlog as user stories. User stories describe what the product should be able to do from the customer's perspective. At the start of a sprint, specific tasks from the product backlog are transferred to the sprint backlog and implemented during the sprint. Short daily meetings, the Daily Scrum Meetings, are used to exchange information within the team. At the end of each sprint, there is an executable product increment or verifiable result. Important roles in Scrum are the Product Owner, who defines the requirements and priorities, the Development Team, which builds the product, the Customer, who ensures that the product meets customer expectations, and the Scrum Master, who guides the team within the Scrum framework [BP20, p. 31f].

The methods described here, DevOps and Scrum, are just two examples from a wide range of approaches to software development. In practice, there are many other methods and concepts that can be used depending on the project requirements and team dynamics, such as Kanban, Waterfall, V-Modell and many more [BP20, p.23ff]. However, what can often be observed across the board are central concepts such as user stories and feedback loops, which are integrated into many of these methods [Pre15, p. 6f][Han17, p. 42ff]. These elements contribute significantly to the successful and efficient implementation of software development projects by promoting communication and continuous improvement [Han17, p. 45].

2.1.2 Software Tools used in Software Development

In software development, various software tools play a crucial role in increasing the efficiency and quality of development processes. Recent statistics from Jetbrains show the prevalence and use of different software tools in the industry (see Figure 2.1).

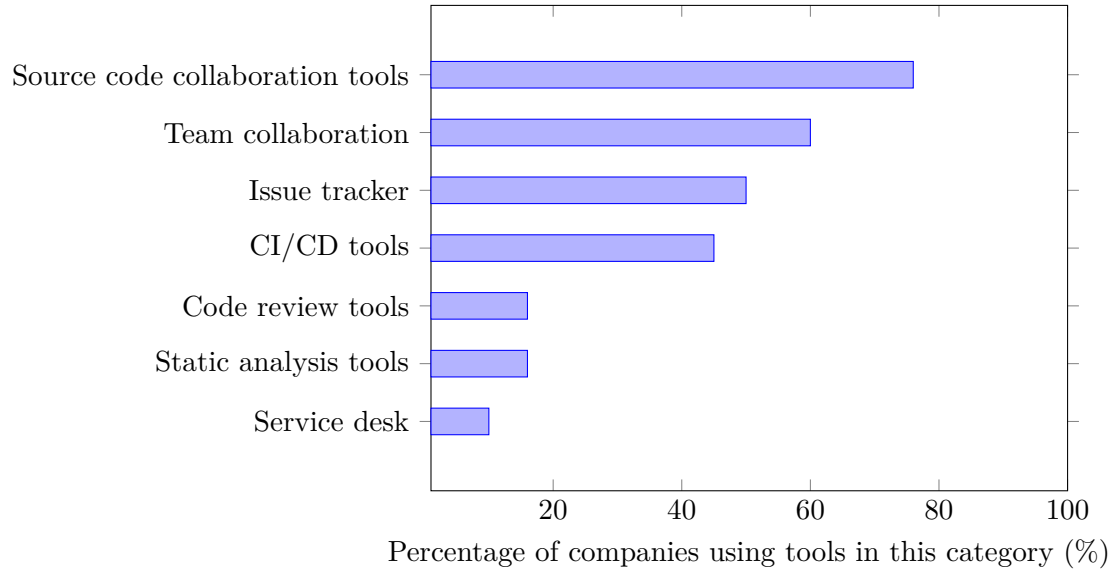

Figure 2.1: Usage of different tools used in software-development in 2023 [Jet23].

The data shows that most developers use a wide range of specialized tools to optimize collaboration, project management and quality assurance [Jet23].

2.2 User-centered design

User-centered design describes an approach to the development of interactive systems that focuses on the needs and requirements of the user in order to ensure that the software is as usable and appropriate as possible. This approach is defined in ISO 9241-210, which outlines the principles and practices for implementing user-centered design in interactive systems [ISO19, p. 5]. Overall, the process can be divided into several steps, which are shown in Figure 2.2.

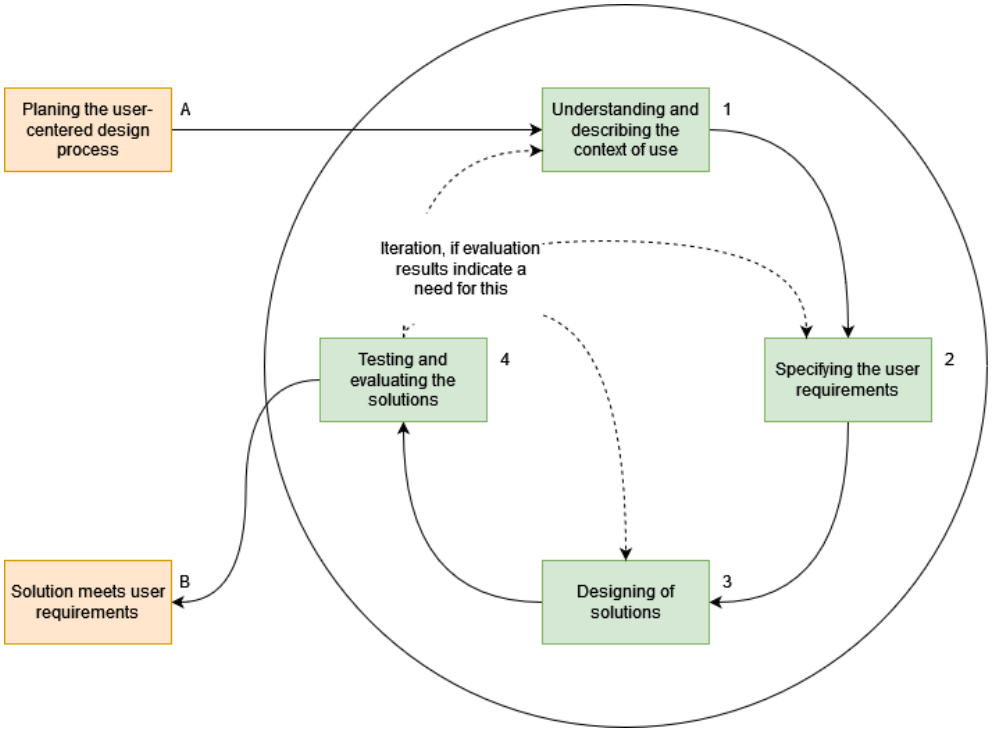

Figure 2.2: Illustration showing the whole process of UCD [ISO19, Based on Fig. 1].

2.2.1 Steps of User-centered design

A. **Planning the user-centered design process**
User-centered design requires comprehensive planning for all phases of the product life cycle. This includes in particular the conception, design, implementation and testing phases [ISO19, p. 13f].

1. **Understanding and describing the context of use**
The first step involves a detailed analysis of the application's context of use. This includes a thorough examination of all relevant influencing factors of the system environment. The aim is to gain a precise understanding of the conditions under which the system will be operated. A comprehensive description of these

conditions ensures that the end product meets the specific requirements and needs of the users [ISO19, p. 16f].

2. **Specifying the user requirements**
   In the second step, the specifications of the usage requirements and the definitions of the requirements are determined on the basis of the information collected in the first step [ISO19, p. 17f].

3. **Designing of solutions**
   The third step involves the design of solution concepts that fulfill the previously identified usage requirements. This phase includes the development of prototypes and design sketches, which are then iteratively tested and evaluated to ensure that they meet the defined requirements. The design solutions should take into account both functional and aesthetic aspects and be characterized by user-friendliness and efficiency [ISO19, p. 18ff].

4. **Testing and evaluating the solutions**
   The fourth step involves the systematic evaluation of the design solutions by users to check their suitability for use. This is done to ensure that the developed solutions meet the needs and expectations of the users. The iterative approach allows problems to be identified and rectified at an early stage, thereby continuously improving the usability of the end product [ISO19, p. 26ff].

2.2.2 Tools used in User-centered design

In the individual steps of user-centered design, various modern tools can be used to document specific information [Rog23, sec. 10, sec. 11]:

- **Personas**
  A persona is a fictitious but detailed representation of a typical user or user group of a product or service. It is based on extensive user research and data analysis to give the development team a precise and empathetic understanding of the target group. Personas help to illustrate the needs, behaviors, goals and challenges of users, which supports the development of user-centric solutions [Rog23, sec. 11.5.1].

- **User scenarios**
  A user scenario is a narrative description of a situation in which a persona achieves a specific task or goal with a product or service. It serves to describe the context of use and the persona's interactions with the system in detail. User scenarios are closely linked to user stories, as they are based on the specific needs and goals of the persona and embed the user story in a realistic context of use. The

combination of persona and user story enables a more comprehensive understanding of user requirements and application scenarios, which supports the development of user-friendly and targeted solutions [Rog23, sec. 11.5.2].

- **User stories**
  User stories are a simple informal explanation of a feature from the perspective of an end user. The role, function and benefit are always specified in more detail.
  Example:

  As a chef, I would like to set a timer by voice with my voice assistant so that I don't have to put my knife down.

  User stories are a popular tool due to their user-centered focus. Instead of just processing tickets, developers focus on solving actual problems. This encourages a critical and creative approach to the question of which solutions are most effective in helping users achieve their goals. However, it is important to note that not all requirements can be defined by user stories. This method is particularly suitable for requirements that can be directly perceived by people. Non-functional requirements, on the other hand, cannot be mapped in this way [Rog23, sec. 11.3].

2.3 ISO 25000

The ISO 25000 series of standards, also known as SQuaRE (System and Software Quality Requirements and Evaluation), is a comprehensive collection of standards dealing with the quality requirements and evaluation of software and systems. The series of standards consists of several parts, each of which covers different aspects of software and system quality [ISO23]. Some of these parts are of direct relevance to this thesis, as they have an active impact on the concept of the software to be developed. The most important parts are briefly presented below.

2.3.1 ISO 25010

The ISO 25010 standard, one of the most central elements of the ISO 25000 series, defines a detailed quality model for software products. This model comprises a total of eight different main characteristics:

- **Functional Suitability**
  The functional suitability describes the extent to which the software fulfills the specified tasks and requirements. It also includes the correctness of the results provided and the appropriateness of the functions for the intended tasks.

- **Reliability**
  Reliability here refers to the ability of the software to function error-free under defined conditions. This includes the maturity of the system, i.e. error-free operation over longer periods of time, availability, fault tolerance in the event of faults and recoverability after faults.

- **Usability**
  Usability measures how easily and efficiently the software can be learned and used by certain users. The focus here is particularly on learnability, operability and protection against user errors, as well as accessibility for different user groups.

- **Performance Efficiency**
  Performance efficiency evaluates the performance of the software in terms of time behavior and resource usage. The focus is on the response time of the software to requests and how efficiently system resources such as CPU and memory are used.

- **Compatibility**
  The compatibility refers to the ability of the software to interact with other software products, i.e. the extent to which it is able to communicate and work together with other software products. In addition, a certain "coexistence" is evaluated, e.g. the ability of the software to run alongside other software products on the same hardware without influencing each other.

- **Security**
  Security describes the protection of the software against unauthorized access and data manipulation. Important aspects here are confidentiality, integrity, the traceability of user interaction and the authentication of users.

- **Maintainability**
  Maintainability assesses how easy it is to maintain and further develop the software. Certain key aspects here are the modularity of software components, reusability, the ability to diagnose (analyzability), modifiability and testability.

- **Portability**
  Portability measures the ability of the software to be installed and operated on different environments. This also includes the adaptability to different environments and the installability of the software.

This quality model allows for a systematic and thorough evaluation of a software product's quality. Therefore, it is crucial to consider these characteristics throughout the software development process. All eight points should be factored in during the planning phase, as this is where important non-functional features are established [ISO23].

2.3.2 ISO 25012

The ISO 25012 standard defines a detailed quality model for data used in software. The standard defines 15 dimensions, which can be divided into two main categories: inherent[1] and system-dependent[2] Data-Quality. Some of these dimensions can also be assigned to both categories, as shown in Figure 2.3.

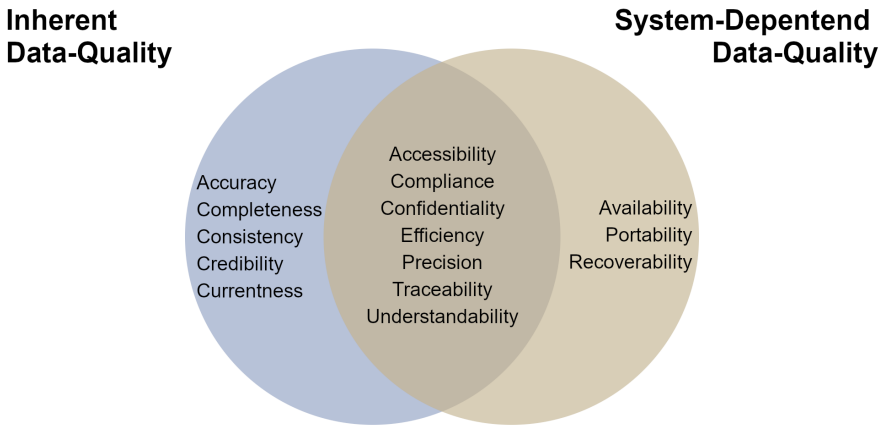

Figure 2.3: Venn diagram showing blue for inherent data quality dimensions, orange for system-dependent data quality dimensions, and the overlap for dimensions belonging to both categories.

- **Accuracy**
  The correspondence of the data with the real world.
- **Completeness**
  The extent to which data has all necessary parts.
- **Consistency**
  The degree to which data is consistent internally and externally. Internally consistent data does not contain contradictions within itself, and externally consistent data aligns with related data sets.
- **Credibility**
  The extent to which data is regarded as true and credible by users. Credible data comes from trusted sources and is perceived as reliable.
- **Currentness**
  The extent to which data is up-to-date.

1 Regardless of the use-case
2 Dependent on the context of use

- **Accessibility**
  The degree to which data is easily retrievable and usable by authorized users.

- **Compliance**
  The extent to which data complies with relevant standards, rules, and regulations. Compliant data meets all legal and organizational requirements.

- **Confidentiality**
  The degree to which data is protected from unauthorized access. Confidential data is safeguarded to maintain privacy and security.

- **Efficiency**
  The extent to which data management operations perform optimally with minimal resources. Efficient data processing minimizes delays and costs.

- **Precision**
  The level of detail in the data. Precise data is specific and granular enough for the intended purpose.

- **Traceability**
  The extent to which data lineage can be traced to its origins. Traceable data allows users to track data back to its source.

- **Understandability**
  The degree to which data is comprehensible and interpretable by users. Understandable data is clearly defined and easily interpretable.

- **Availability**
  The degree to which data is present and ready for use when needed. Available data can be accessed without excessive delays.

- **Portability**
  The extent to which data can be transferred from one system or context to another without loss of quality. Portable data is easily adaptable to different environments.

- **Recoverability**
  The degree to which data can be restored in case of loss or corruption. Recoverable data is backed up and can be recovered quickly and completely.

The quality model enables a systematic and comprehensive evaluation of data in the software. Consideration of the dimensions of ISO 25012 is essential, especially for data-intensive applications whose functionality depends heavily on the data used. ISO 25012 should therefore be taken into account when planning software and presenting data to the user [ISO08].

2.4 Data integration

Data integration is a central process in modern data processing that makes it possible to combine different data sources into a coherent and usable data base [Sta17, p. 23ff]. This integration is of crucial importance as it forms the basis for well-founded decisions, efficient business processes and progressive analyses [Sta17, p. V].

Before working with data from different data sources, the data must first be standardized. This can be divided into a total of three steps (see Figure 2.4).

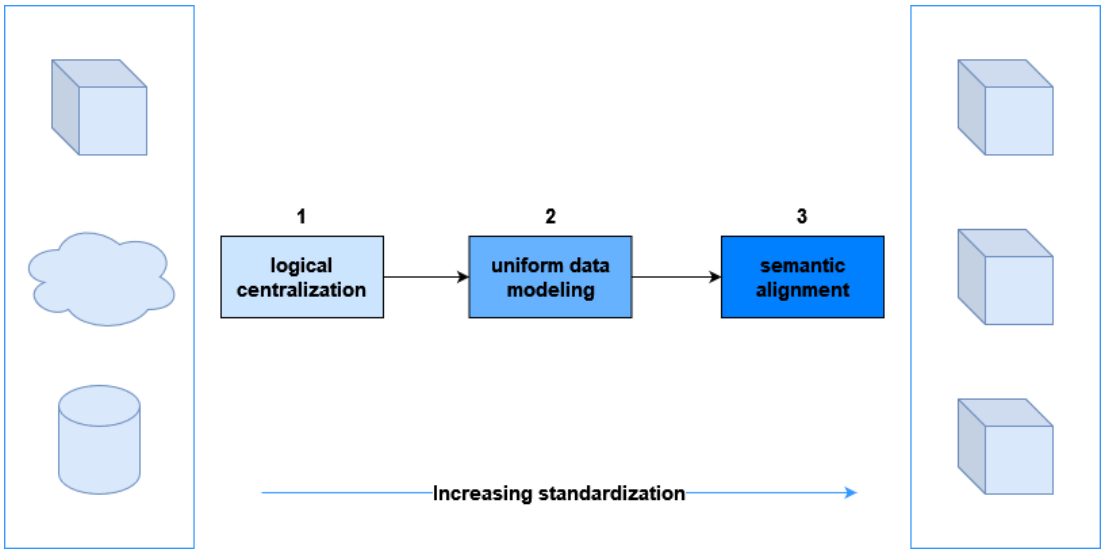

Figure 2.4: Illustration showing the increasing standardization of data [Sta17, Based on Fig. 4.2].

1. The first step is logical centralization. This involves collecting the data from the various data sources in a central system. This system then represents a central access point for all users.

2. The second step, uniform data modeling, is about structurally aligning the data. This includes standardized concepts, terms and methods to consistently classify and describe data.

3. The final step, semantic alignment, involves defining the meaning of terms and data in a standardized way so that everyone involved has the same understanding of the content used.

Once this standardization of the initially differently structured data has taken place and a uniform data structure is now available, the next steps in data integration can be carried out (see Figure 2.5).

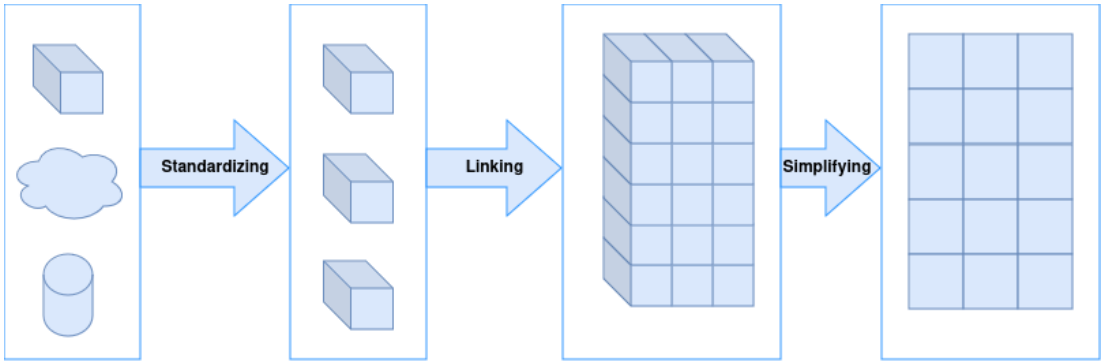

Figure 2.5: Illustration showing the whole process of data integration [Sta17, Based on Fig. 4.3].

Standardized data with a uniform structure enables the creation of a high-quality data collection. This standardization facilitates the integration and linking of data, which in turn offers the opportunity to gain new insights. By linking data, complex relationships can be uncovered that would not be recognizable if viewed in isolation. This linked data can be effectively visualized and presented, for example through the use of dashboards or other forms of presentation that enable a simplified yet informative presentation of the results [Sta17, p. 23ff].

2.5 Information design

Information design is the process of planning and designing information to make it understandable, accessible and effectively communicable. It integrates various disciplines such as graphic design, typography, user experience and data visualization to present complex data and content in a way that is easy to grasp and interpret. The aim of information design is to optimize the transfer of information and help users to understand the information presented efficiently and correctly [Bla17, p. XIf].
Dashboards are a widely used tool for the clear and quick presentation of information [Few06, p. 12] [Wex17, p. 3ff]. However, when developing such dashboards, several key aspects must be taken into account to ensure their effectiveness and user-friendliness:

- **Personalize Dashboards**
  Personalization of dashboards increases the relevance of the data presented to individual users. Tailoring the information to the specific needs and interests of users encourages their engagement and optimizes the effective use of data [Wex17, p. 338ff] [Few06, p. 30ff].

- **Clarity**
  A dashboard must present data clearly and concisely to enable users to quickly and precisely absorb information. Unclear or cluttered dashboards make data analysis more difficult. This also includes things such as avoiding useless decoration, unnecessary variety in the presentation of data or the inconsistent arrangement of data [Few06, p. 51f, 56f, 58f]

- **Usage of colors**
  The use of color should be targeted and considered. Colors have different visual and psychological effects; warm colors attract more attention, while cooler colors are less noticeable. In designs, such as dashboards, colors can be interpreted as carrying meaning in different areas, which can lead to misunderstandings, especially for about 10% of men and 1% of women who are colorblind. Therefore, it is crucial to choose colors that are understandable to all user groups [Few06, p.61f] [Wex17, p. 390ff].

- **Interactivity and drilldown**
  Interactive dashboards allow users to filter and click to get more detailed information and explore specific data. Drill-down capabilities provide the ability to switch from aggregated to detailed data views, allowing for deeper analysis and a better understanding of the underlying data. These features promote targeted and efficient data analysis and support informed decision making [Few06, p. 82f] [Wex17, p. 44].

2.6 Change management

Change management is a systematic approach to designing and supporting change processes within organizations. It is primarily concerned with the human aspects of change and aims to ensure the acceptance and successful completion of change projects [Hag19, p. 13ff].

2.6.1 Lean Change Management

Lean change management is a contemporary methodology that applies the principles of lean management to change management in order to enable more efficient and sustainable adjustments within organizations [Kus22, p. 433].

The aim of lean change management is to optimize change processes through continuous improvement, transparent communication and close involvement of employees. This methodology is particularly relevant in software development, an area characterized

by rapid technological developments and constantly changing customer requirements [Geo23]. The iterative nature of lean change management allows for flexible and continuous adjustments, which is of central importance in software development [Lit16].

The essential core aspects of lean change management can be summarized as follows:

- **Feedback driven:** Feedback from users is continuously collected and actively integrated into the development process. An open communication channel must be maintained at all times in order to be able to respond to feedback promptly [Lit16, p. 35ff][Geo23, p. 33f].
- **User centricity:** The focus of development is on the needs of end users in order to increase acceptance of the changes [Lit16, p. 23ff][Geo23, p. 20ff].
- **Empowerment of users:** Employees and users are involved in the change process and are given responsibility and authority to identify problems and propose solutions. This promotes both acceptance and commitment [Lit16, p. 114ff][Geo23, p. 41].
- **Iterative planning and implementation:** Changes are implemented in short, manageable cycles. This enables regular evaluation of progress and adaptation of the strategy based on the knowledge gained. Iterative planning supports a flexible response to new challenges and changes [Lit16][Geo23, p. 33f][Kus22, p. 434].

The principles of lean change management are particularly suitable for the introduction of new software in existing processes [Geo23, p. 7].

2.6.2 Introduction types

There are different approaches to introducing changes to processes (and therefore also the introduction of software, for example). A distinction is made between the big-bang approach, the step-by-step approach and the parallel approach.

- **Big-Bang Approach**
  With this approach, the change is introduced in a single step, which means that the entire process is changed for everyone involved at the same time. This approach carries a significant risk, as unexpected problems or errors can have a major impact and require immediate problem solving.
- **Step-By-Step Approach**
  This approach implements the change step by step, starting with selected sub-areas or departments. The introduction takes place in several phases, with each phase being based on the experiences and findings of the previous phase. This enables

gradual adaptation and problem identification, which minimizes risks and errors during the process.

- **Parallel Approach**
  With the parallel approach, the new solution is operated simultaneously with the existing solution. This usually leads to higher costs, as both systems have to be maintained at the same time. The advantage of this approach lies in the possibility of continuous comparison and ensuring an alternative system in the event of problems with the new solution (Fallback level).

Depending on the type of project, the type of implementation must be planned and assessed [Kus22, p. 213f].

2.7 Empirical methods in software engineering

Empirical research methods are used to test hypotheses by either proving or disproving them [Fel19, p. 3, p. 5]. In the field of software engineering, they are used in particular to analyze facts and measure the effects of certain measures or phenomena using collected data [Shu08, p. 11f]. Within empirical research, a distinction is made between quantitative and qualitative methods [Fel19, p. 4].

Qualitative methods aim to understand the "why" of a phenomenon by attempting to provide deeper explanations for observed events. These approaches focus on non-numerical data, such as texts, interviews or observations, which are interpreted in their entirety [Fel19, p. 4f].

In contrast, quantitative methods strive to quantify phenomena through measurable, numerical data. They allow information to be recorded in the form of numbers and thus enable comparisons, statistical evaluations and the testing of hypotheses on a quantitative level. These two methodological approaches are complementary and each contribute in their own way to the understanding and validation of research results [Fel19, p. 5].

Three central empirical methods have become established in software engineering: Surveys, case studies and experiments [Fel19, p. 5]. These are briefly discussed in more detail below:

- **Surveys**
  A survey is a systematic approach used to gather data about various topics, including individuals or projects, with the aim of describing, comparing, or understanding their knowledge, attitudes, and behaviors [Fel19, p. 6].

Typically, a survey is conducted retrospectively after a tool or technique has already been in use for a certain period of time [Pfl95]. Data is typically gathered qualitatively or quantitatively through interviews or questionnaires aimed at a representative sample of the target group. The data obtained is then analyzed in order to draw descriptive and explanatory conclusions that can be applied to the population as a whole [Con03, p. 19].

- **Case studies**
  A case study in software development is an investigation in which one or more real examples of a particular topic are examined more closely. It is an attempt to understand how the topic works in its real environment, especially when the boundaries between the topic and its environment are not clear [Fel19, p. 6]. This type of research is often called "typical research" because it relates to real projects and thus provides practical insights. Case studies are used to observe projects or tasks and collect data to track certain things or find out how different factors are related. Unlike experiments, which are conducted under controlled conditions, a case study is more about observing how things develop in the real world. The aim is often to make predictions, for example how many errors might occur in a test, and statistical methods are used to do this [Con03, p. 18].

- **Experiments**
  Experiments are methods that aim to investigate causal relationships between different variables of a phenomenon [Fel19, p. 6]. As a rule, one or more independent variables are deliberately manipulated, while other variables are kept constant in order to measure their influence on the dependent variables. This makes it possible to test, validate or refute hypotheses. In controlled experiments, the assignment of subjects to the various treatments is random, whereas in quasi-experiments this assignment is not random due to the characteristics of the subjects or objects. Replication experiments are conducted to verify or compare the findings of previous experiments in varying contexts. [Con03, p. 18].

3 Concept

In this chapter, a concept for a software is developed that supports software developers in their daily workflow through information aggregation and processing and thus aims to optimize the software development process as a whole. This concept addresses research question 1.
Furthermore, strategies for the introduction of the software in companies and for the evaluation of its influence on the software development process are developed, creating the basis for answering research question 2 and research question 3.

The procedure for developing the software concept is based on the principles of user-centered design (see section 2.2), which has proven to be successful in practice [Vre02]. First, the context of use is analyzed in order to determine the specific requirements for the software. Based on these requirements, it is then evaluated whether in-house development makes sense. If this is the case, a software concept is created that serves as the basis for implementation. Otherwise, the most suitable available software solution is identified.
In contrast to classic user-centered design, however, the development of several prototypes is avoided. Instead, the software concept is deliberately designed to be so generic that it is still possible to incorporate user feedback and make appropriate adjustments during the implementation and introduction phase.

3.1 Understand & describe the context of use

In this step, the application context of the software is first analyzed. In addition, a detailed examination of the core aspects that the software must take into account is carried out in order to design the functionality and architecture accordingly. Existing software solutions are also analyzed on this basis in order to identify potential design approaches.
Based on these findings, personas and user scenarios (see section 2.2.2) are developed in order to realistically describe and define the context of use. These methods are central components and are often used in the context of user-centered design and are intended to help the developer to better understand and include the user and the context.

This approach also specifically focuses on the end user in order to ensure a high level of benefit from the software.

3.1.1 Context of use

The scope of the software to be developed is primarily aimed at modern software development teams working in dynamic and iterative environments. Current statistics show that practices such as DevOps and Scrum are becoming increasingly widespread and are used intensively in these teams [Git22][Hus09][Dig]. Modern software development is also characterized by the use of a variety of different software tools, which are essential for overcoming the many challenges (see section 2.1.2).

For the software to be developed, this means that particular attention must be paid to pipelines for build and deploy processes in the context of DevOps. In addition, the software must be designed in such a way that it meets the requirements of modern structured teams working according to Scrum and offers appropriate added value.

3.1.2 Core aspects of the software

As explained in section 3.1.1, numerous software tools are used in modern software development. This multitude of tools leads to a fragmented information landscape in which relevant data and status information is scattered across different systems. This fragmentation makes it difficult for teams to maintain a comprehensive overview of the current project status, make quick decisions and identify potential problems at an early stage. The topic of data integration, which deals with this problem, is discussed in detail in section 2.4. Another key aspect is the information design of the software. It is not enough to simply aggregate the information; it must also be presented in a form that meets the various requirements. Details of these requirements are explained in more detail in section 2.5.

There are therefore two key requirements for the software to be developed: First, the aspects of data integration must be taken into account, which includes sourcing, normalizing, linking and simplifying the data. Secondly, the presentation of the data must be designed in such a way that it is comprehensible, clear and easily accessible.

3.1.3 Examination of existing software solutions

As part of the analysis of reference software, software solutions are identified that are used in software development and address the topics of data integration and information

design. Particular attention is paid to the functional and non-functional features as well as the solution approaches that these software solutions offer for solving specific problems. This analysis forms the basis for the subsequent decision as to whether new software should be developed or an existing software solution should be used.

Grafana

Grafana is an open-source platform for visualizing and monitoring data, designed especially for the analysis of time series data. It allows users to create and customize dashboards to display and monitor metrics and logs from various data sources. Grafana supports a variety of data sources, including Prometheus, InfluxDB, Elasticsearch and many others, and offers extensive features for creating interactive charts, graphs and tables [Gra]. The platform is widely used in the IT and DevOps community to monitor and analyze system and application metrics [Gra23], and enables the integration of alerting systems for proactive problem identification and resolution. Grafana is characterized by its flexible, extensible architecture and a user-friendly interface that enables complex data analysis to be performed efficiently and intuitively.
The flexible structure of Grafana allows a wide variety of information to be visualized, including data from all common tools frequently used in software development.
The aspects of information design are fully covered by Grafana. The ability to create individual dashboards and configure them flexibly enables the independent fulfilment of the criteria “Clarity” and “Usage of Color”. The drill-down functionality can also be implemented in theory, but sometimes requires some detours. For example, additional dashboards may have to be created for more detailed information on individual metrics, which can be accessed by clicking on the metric.
A significant weakness identified in test implementations is that when querying data from external APIs, the external APIs are requested again for each update of the dashboard and for each individual user. This quickly leads to the rate limits of the external APIs being exceeded. To avoid this problem, an additional backend would be required to handle data retrieval and storage. When implementing this backend, attention must be paid to the aspects of data integration[Gra].
Figure 3.1 shows an example of a dashboard created in Grafana that illustrates various options for displaying figures and metrics. The example shown includes pie charts, ring charts and bar charts in various forms.

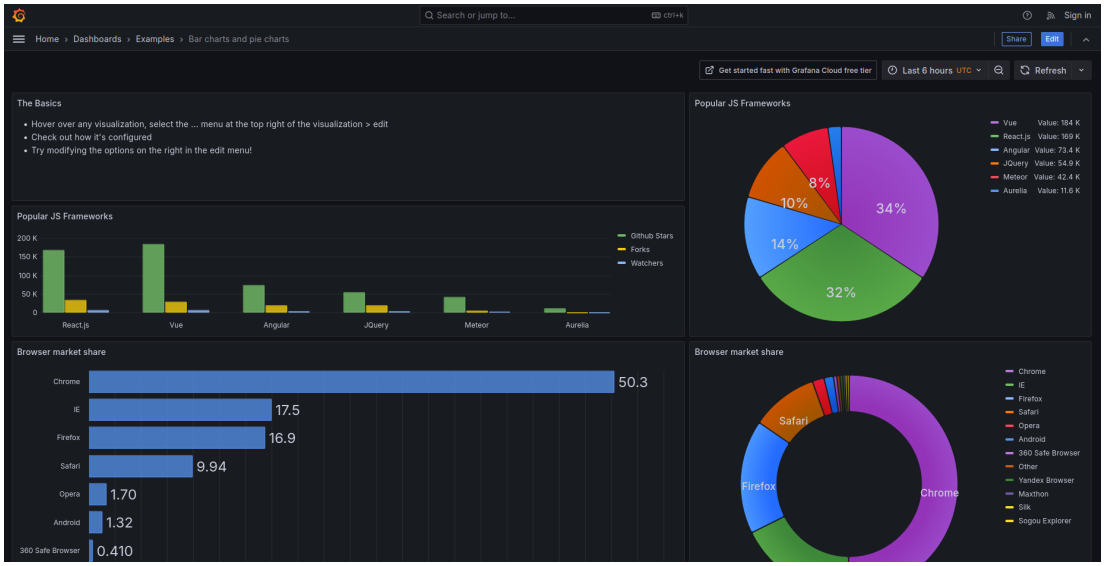

Figure 3.1: Example of a dashboard in Grafana [Gra].

Monitoror

Monitoror is a simple open source web application that is specially designed to monitor various tools. These include Jenkins, GitLab and GitHub. The main function of Monitoror is to monitor pipelines and build jobs, which is a typical use case in the DevOps area. The information is displayed on clear dashboards consisting of individual tiles. These tiles display both metrics and the progress of build jobs, providing a clear and easy-to-understand visualization of the monitored processes.
Monitoror is strongly designed for fixed applications such as Jenkins, GitLab and GitHub. The flexible addition of new metrics and data sources is therefore not easily possible. Due to this specific structure, Monitoror is not generic enough to ensure the display of differently structured information from different software tools in software development [Ale].
Figure 3.2 shows an example dashboard from Monitoror, which enables immediate detection of faulty or problematic areas. The color scheme contributes significantly to highlighting the sources of errors.

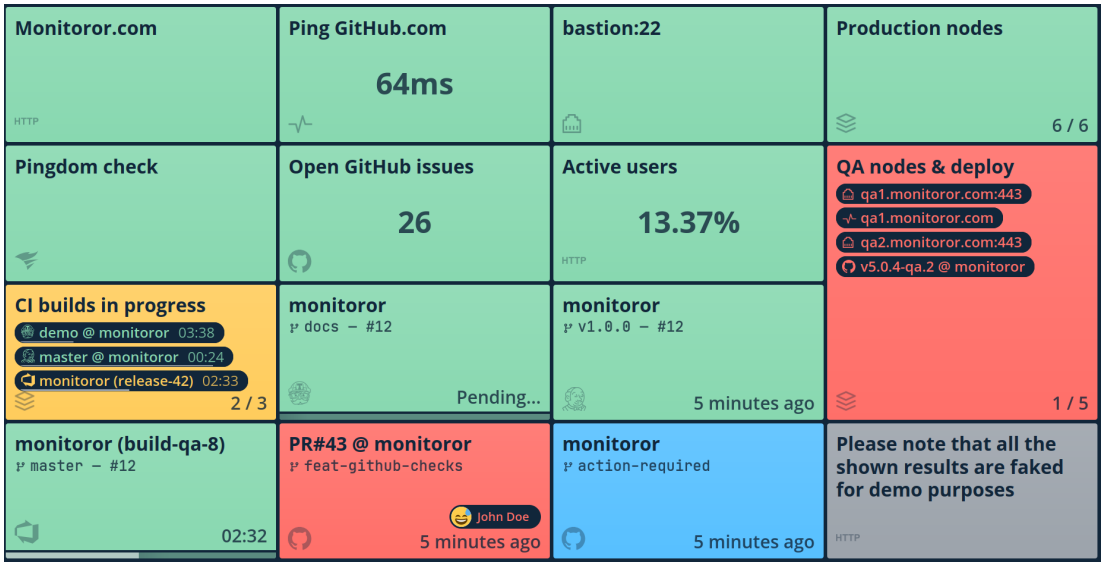

Figure 3.2: Example of a dashboard in Monitoror [Ale].

3.1.4 Personas

Based on the analysis, personas are developed for better visualization. Attention is paid to representing all relevant groups of people in a modern software development team. The personas created therefore represent different user groups that will use the new software.

Niklas (see figure 3.3) represents a new employee in the company who has relatively little prior knowledge. The software is particularly important for him as it enables him to get to know the tools used in the company better. The software should also help Niklas to plan his day-to-day work more efficiently and quickly become aware of errors or upcoming tasks. These functions of the software are crucial for shortening the induction period for new employees and increasing their productivity right from the start. By using the software in a targeted way, Niklas can understand the operational procedures and processes more quickly and integrate more seamlessly into the existing team.

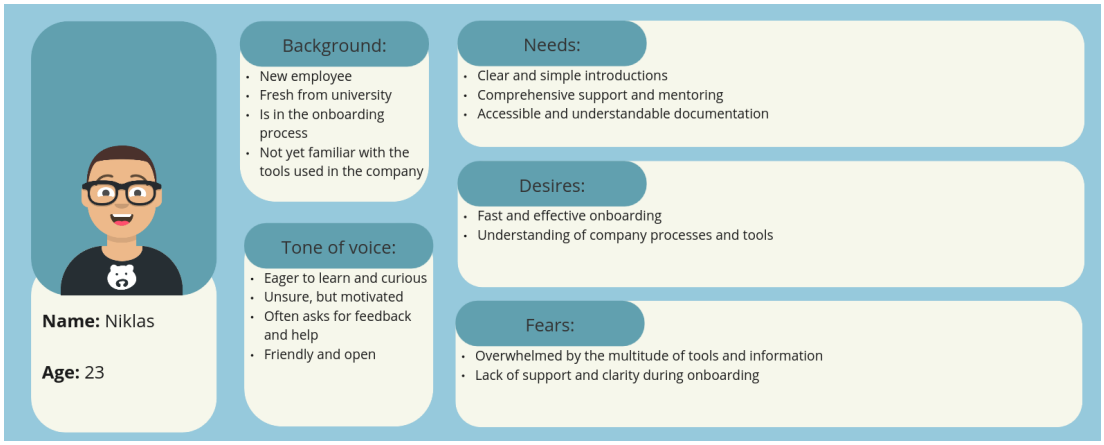

Figure 3.3: Representation of the persona "Niklas"

Jonathan (see figure 3.4) represents a long-standing employee with extensive experience. He strives to integrate the software effectively into his day-to-day work and is generally open to using it. It is particularly important to him that the software only displays relevant information that directly affects him. This allows Jonathan to plan and prioritize his day-to-day work more efficiently and maintain a clear overview of upcoming tasks and potential challenges. This targeted information processing is essential in order to maximize the work performance of experienced employees and make the best possible use of their expertise.

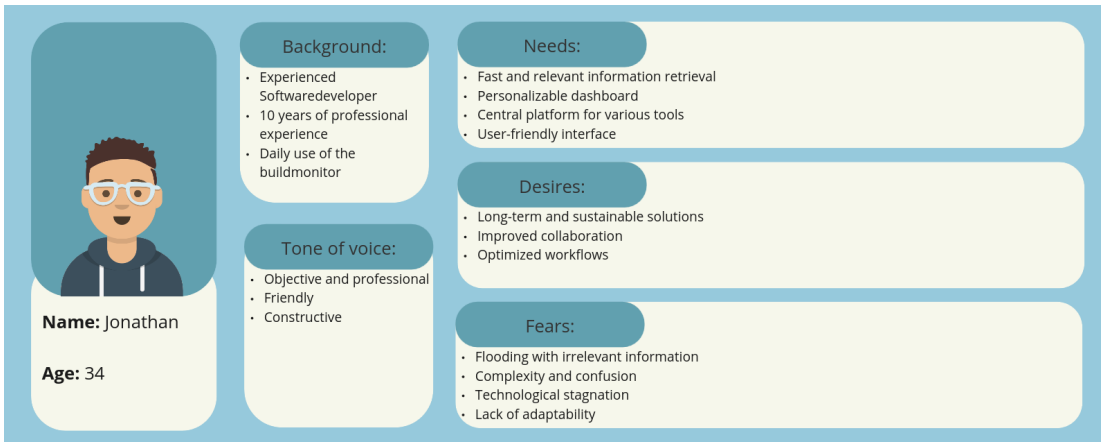

Figure 3.4: Representation of the persona "Jonathan"

Luis (see figure 3.5) represents a very experienced software developer who has only been working for the company for a short time. For him, it is crucial that the metrics and information in the software are transparent and comprehensible. A key concern for Luis is to minimize the overhead in the software development process to ensure a more effective and efficient way of working. His focus is on high efficiency, which should also be reflected in the software used. This includes a clear and concise presentation of relevant data and the provision of tools that optimize the development process and

reduce unnecessary complexities. By using such software, Luis can make the most of his expertise and contribute to increasing productivity within the company.

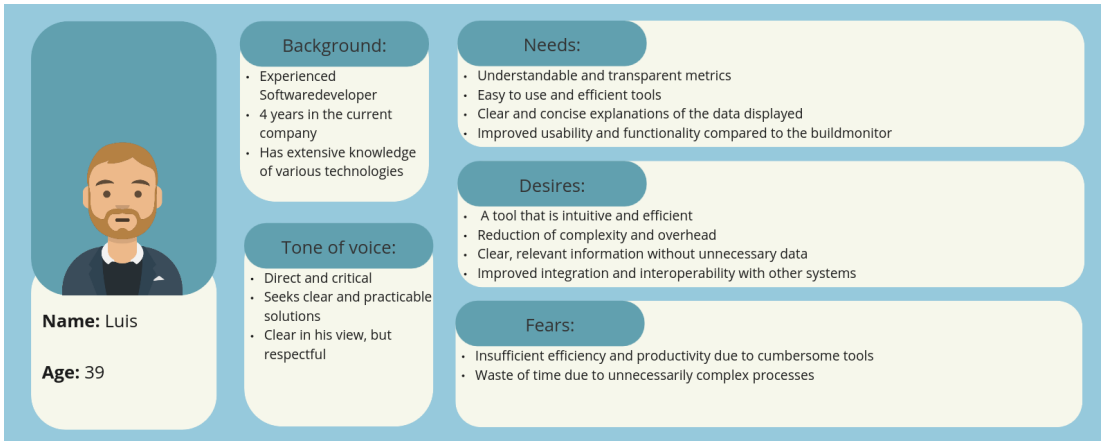

Figure 3.5: Representation of the persona "Luis"

Brenda (see figure 3.6) represents an experienced developer in a leadership position who has been with the company for a long time and is familiar with all the tools used. Her focus is on analyzing aggregated team metrics to better assess where action is needed within the team or department. For Brenda, it is very important that the software uses robust and up-to-date technologies and meets high security standards, as it works with very sensitive data. These requirements are essential to ensure the integrity and confidentiality of the data. By using such software, Brenda can make informed strategic decisions and increase the efficiency of the team.

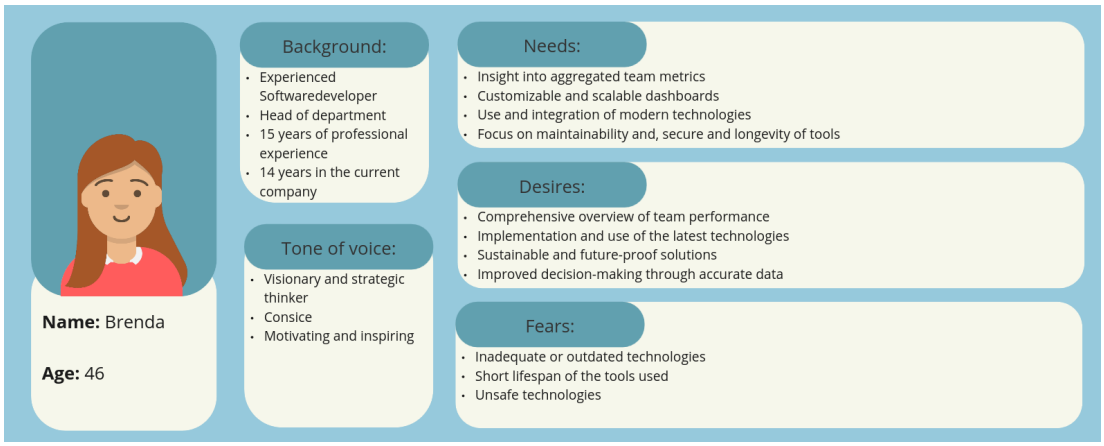

Figure 3.6: Representation of the persona "Brenda"

3.1.5 User scenarios

In the following, user scenarios are developed based on the personas defined in section 3.1.4. These scenarios are intended to illustrate the various application and interaction

contexts in which the defined user profiles could interact with the system or product. The aim is to create a sound basis for further system design and evaluation by taking into account the specific needs and behaviors of the personas.

Efficient Daily Error Management and Task Organization for Niklas

Niklas opens the software in the morning and logs in. The software immediately shows him an overview of the current errors for which his team is responsible. The errors are presented clearly and comprehensibly so that Niklas can quickly see which problems need to be solved urgently. Thanks to the intuitive user interface, he can work directly on the problems in a targeted manner. Niklas uses the software to retrieve the necessary information about the errors and take appropriate action. By processing the problems efficiently, he is able to resolve them quickly and then move on to other tasks. The software has helped him to organize his working day in a structured and effective way, which facilitates his integration into the team and increases his productivity.

Streamlined Code Commit Review and Error Detection for Niklas

Niklas commits source code to a repository and then opens the software to check the status of his commit. In the software, he can see directly whether his commit has led to errors. The software aggregates the results from various tools, such as pipelines and static code analyses, and displays them clearly. As Niklas is not yet familiar with all the tools used in detail and is still in the familiarization phase, it is particularly advantageous for him that he does not have to go through every single tool to identify potential problems. Instead, they immediately receive all relevant results in the software and can react to any errors in a targeted manner. This integration of the results in a central view saves him time and makes it easier for him to find his way around, which speeds up his integration into the team and increases his productivity.

Prioritization of tasks by Brenda based on team metrics

Brenda, as head of department, opens the software and the view of the individual teams and immediately notices that a large number of complaints and code reviews are open for a specific team. The aggregated metrics in the software clearly show that these tasks are currently a high priority. Based on this insight, Brenda recommends that the team pause work on the new feature for the time being and focus on processing the open complaints and code reviews. By making this decision, she optimizes resource allocation and ensures that urgent tasks are prioritized to improve team performance and minimize potential risks. Using the software allows Brenda to make such strategic

decisions efficiently and based on data, increasing the team's effectiveness while ensuring the quality of work.

3.2 Specify functional & non-functional requirements

The functional and non-functional requirements are defined on the basis of the analysis of the context of use. Accordingly, the following aspects form the basis on which the requirements for the software are built:

- The central aspects of ISO 25000 (see section 2.3)
- The core components of the concept of data integration (see section 2.4)
- The important aspects of information design with regard to the presentation and creation of dashboards (see section 2.5)
- The findings and proposed solutions of the software analyzed in section 3.1.3
- The personas and user scenarios (see section 3.1.4 and 3.1.5)

3.2.1 Functional Requirements

Functional requirements are now defined and recorded with the help of user stories.

US1 As a user, I want to be able to see all the current information from the tools I use during software development that concerns me on a dashboard so that I can plan my day-to-day work better.

US2 As a user, I would like to see all the current buildjobs from my CI/CD Tool, to quickly identify failed buildjobs.

US3 As a user, I would like to be forwarded directly from each metric to the corresponding place in the tool in order to minimize unnecessary interaction with the tool.

US4 As a user, I would like to be able to mark a favorite dashboard that opens directly when the software is called up so that I spend less time starting the software.

US5 As a user, I would like to be able to join a team so that all team-relevant information is displayed on the team dashboard.

US6 As a user, I would like to be able to click on a metric to get all the necessary information about the individual data.

US7 As a user with personnel responsibility, I would like to be able to view team-aggregated metrics in order to better plan where there is an acute need for action.

US8 As an administrator, I would like to be able to freely configure the dashboards for the users in order to be able to respond to the wishes and requirements of the users.

US9 As an administrator, I want to be able to add new metrics and data sources without much effort in order to be able to continuously expand the functionality of the software.

US10 As an administrator, I would like to be able to define individual thresholds when adding metrics, above which the metric is displayed as a warning or error in order to better symbolize the need for action for users.

US11 As an administrator, I would like to be able to determine different algorithms that are used when calculating the values of metrics in order to better take into account the different types of metrics.

3.2.2 Non-functional requirements

The non-functional requirements for the software are described and documented in detail below. These requirements relate to the quality attributes of the system and are crucial for its successful implementation and use.
As it has not yet been decided whether a ready-made software solution will be implemented or custom software developed, general requirements are highlighted in blue, requirements for ready-made software in magenta and specific requirements for custom software are highlighted in orange.

Usability

- **Requirement 1** - Comprehensive documentation on the use of the software must be provided in order to offer users a first point of contact for reference purposes in the event of uncertainties.
- **Requirement 2** - The structure of the dashboards must be in line with the principles of information design (see section 2.5).

Compatibility

- **Requirement 3** - The software have to be compatible with all common software products in the categories listed in section 2.1.1. It must be able to receive and process data from these software products.
- **Requirement 4** - A relational database is to be used, as this enables the relationships between metrics, teams and users to be optimally represented.
- **Requirement 5** - The database structure should be designed in a generic way in order to take appropriate account of the different structures of the information.

Security

- **Requirement 6** - The software have to support OAuth2 to ensure a high level of security and enable the use of company accounts.
- **Requirement 7** - The development team that develops the software should be large enough to ensure continuous further development of the software, in particular to be able to fix security gaps and other problems promptly.
- **Requirement 8** - When developing the front end, back end and database, modern and up-to-date frameworks should be used that are actively developed further.

Maintainability

- **Requirement 9** - The architecture of the backend should correspond to modern software architecture patterns in order to ensure a high level of maintainability.
- **Requirement 10** - The business logic of the application should have a test coverage of more than 75%.
- **Requirement 11** - A pipeline is to be implemented that enables the direct deployment process of the application in the live environment. This pipeline should also include testing and linting of the software.

Functional Suitablity

- **Requirement 12** - The software should not contain significantly more functions than necessary.

Credibility

- **Requirement 13** - The data must originate directly from the original data source.

Currentness

- **Requirement 14** - The data should be updated regularly and automatically, at least once a day.

Confidentiality

- **Requirement 15** - Access to the data should be restricted to authorized users only.

Traceability

- **Requirement 16** - The traceability of the data to the origin of the information must always be guaranteed. Exceptions are only permitted in special cases.

Understandability

- **Requirement 17** - Data should always be provided with detailed and meaningful descriptions.

Portability

- **Requirement 18** - Data must also be made available in JSON format independently of the dashboard.

Accessibility

- **Requirement 19** - All data must be stored on the same system.

3.3 Justification for Own Software Development

This chapter examines whether an existing software solution meets the requirements or whether an own software development is preferable. For this purpose, the software solutions presented in section 3.1.3 are analyzed in detail and evaluated based on the requirements defined in section 3.2. If none of the existing software solutions meet the requirements, the decision in favor of an own software development is justified in this chapter. Both the shortcomings of the existing solutions and the advantages and potential of individual development are presented. The aim is to create a sound basis for selecting the optimal software strategy.

3.3.1 Grafana

Grafana could in theory be considered as a front end for the application, but it is not suitable as the sole solution. A separate backend is required to make the application functional. The main reason for this is the need to retrieve data from external APIs, which will definitely be the case in the application. This data must be stored persistently in some form by a separate application. If this is not taken into account and Grafana is used directly to retrieve data from the external APIs, this would quickly lead to rate limiting issues as each loading of a dashboard triggers API calls. This also scales poorly with a larger number of users. In addition, the APIs would have to be queried continuously to ensure that the data on the dashboard is updated automatically.
Consequently, Grafana could only be considered as a pure front-end, if at all. However, the question arises as to whether the use of Grafana would not be oversized for this application. Non-functional requirement 12 states that the software should only contain the functions that are actually needed. In addition, the use of Grafana would create a dependency on external software and thus on a third-party company.
As the requirements for the frontend of the software are not overly complex, it is decided not to use Grafana.

3.3.2 Monitoror

Monitoror is quickly ruled out as a possible software solution because, among other things, the following requirements are not met:

- **User-Story 1, 2, 4, 8**: Monitoror does not offer any functionality for the specific assignment of metrics and information to individual persons or teams. The system only provides a single dashboard.

- **User-Story 9**: There is no way to add new metrics or new data sources without a lot of effort and changes to the source code.
- **Requirement 7**: Monitoror was developed by two developers, with the last changes to the main branch being made on 26.07.2020 [Ale24]. This indicates a possible discontinuation of active development and maintenance.

Although Monitoror is not considered a suitable software solution, valuable insights can be gained for the further course of the work. In particular, the presentation of the information is extremely simple and clearly laid out. The information is presented in the form of individual boxes, which correspond to the principles of information design (see section 2.5).

3.3.3 Decision

As Grafana could only be considered as a front-end solution, but appears oversized due to its comprehensive range of functions and Monitoror does not sufficiently fulfill both functional and non-functional requirements, the development of a custom software solution is being considered.

3.4 Software Concept

The conceptual designs for the front end and back end are developed on the basis of the requirements (see section 3.2) and the general context of use (see section 3.1). Figure 2.3 shows the pictorial representation of the conceptual software structure, which consists of the backend (see section 3.4.1) and the frontend (see section 3.4.2).

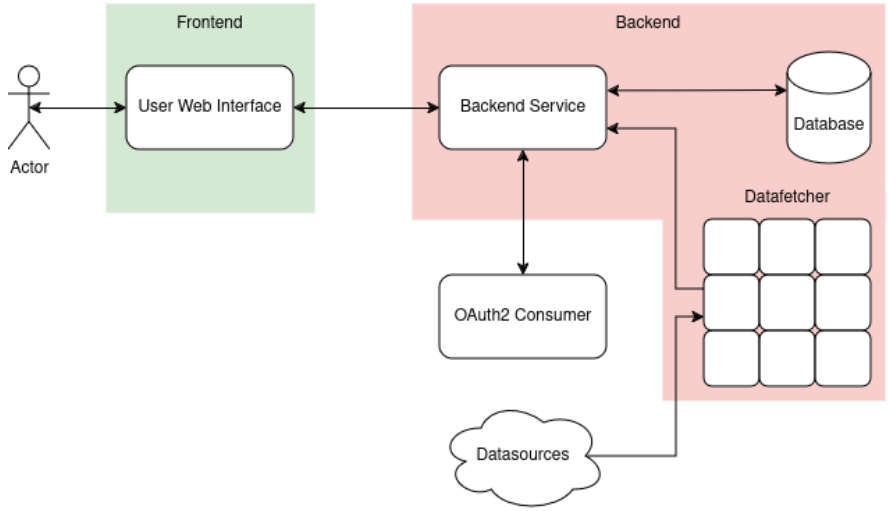

Figure 3.7: Pictorial representation of the conceptual software structure

3.4.1 Conceptual design of the backend application

The functions of the backend are divided into a total of three main components:

Database

The database has a relational structure, but also supports a generic data structure. The structure of the database is shown in Figure 3.8. The tables highlighted in cyan represent the real entities, while the tables highlighted in red represent m:n relationships (junctiontables). In general, the developed database schema enables the creation of metrics and the assignment of individual data records, which are assigned to these metrics, to users, repositories and other relevant entities. The metrics are then assigned to the corresponding dashboards via junctiontables. It should be noted that, for example, there are three different junctiontables between the entities "Dashboard" and "Team", each of which has a different semantic meaning. This differentiation makes it possible, for example, to distinguish between the display of team member metrics, the metrics of the team repositories and the build jobs of the team repositories on the dashboards.

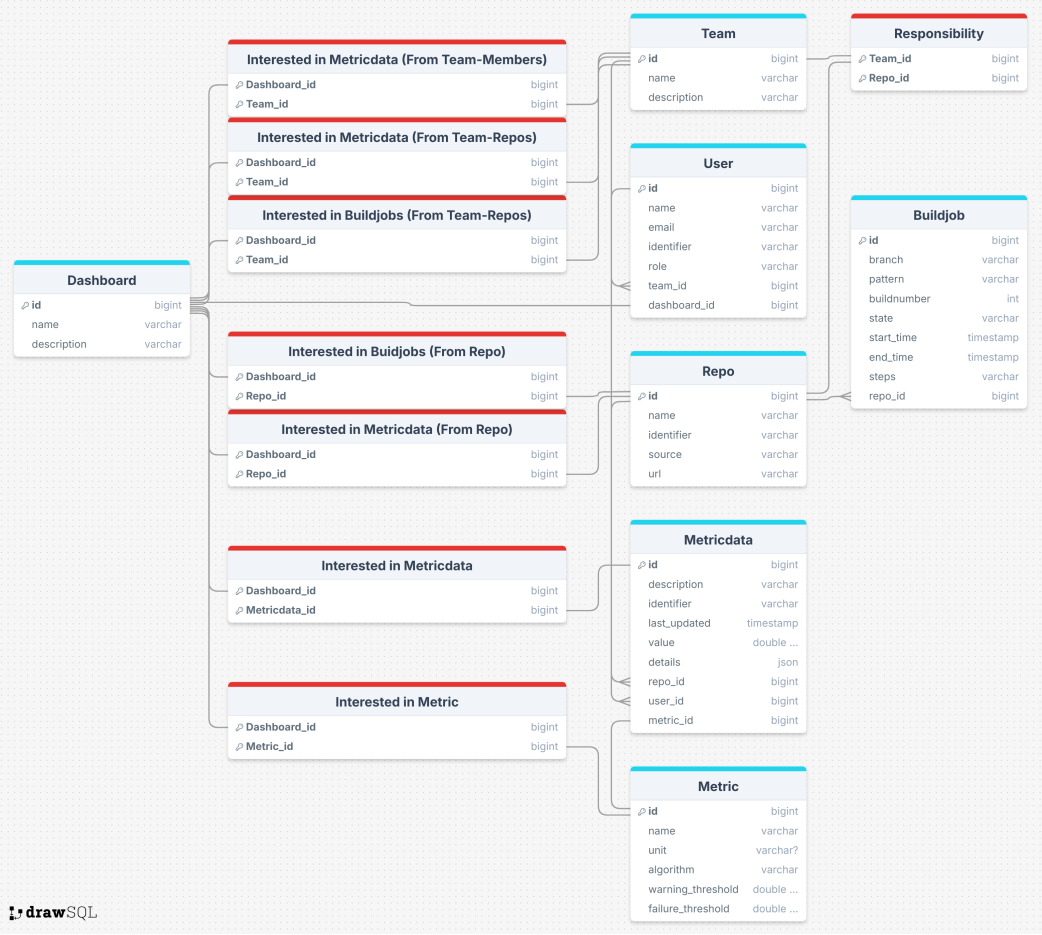

Figure 3.8: Entity-Relationship diagram of the database-schema

Backend Service

The backend service forms the central unit of the backend and enables general communication via a REST interface. It receives data and stores it accordingly in the database. The routes required for communication with the frontend are secured by OAuth2, while the routes required for communication with the data fetchers are protected by HMACs. This ensures both the integrity of the data and the authenticity of the sender.
A complete overview of all REST routes, including a brief description and details of the security measures, can be found in the appendix under point A.

Datafetcher

The data fetchers are independent programs that retrieve data from various sources, adapt it to the specified structure of the database and then transfer it to the backend service using suitable REST routes. The data is assigned to the users and repositories in the backend service.
The separation of the data fetcher from the backend service is necessary in order to reduce the complexity of the backend service. Integrating the fetcher functionality into the backend service would require a generic structure that enables access to different data sources and adapts the data according to the database structure, which would mean a considerable amount of additional work.
This architecture allows individual data fetchers to be developed independently of the source code of the backend service, which then send data to the backend service autonomously.

3.4.2 Conceptual design of the frontend application

Wireframes are created for the front-end concept, which illustrate the basic functions of the software. Particular emphasis is placed on the principles of information design (see section 2.5). Dashboards are particularly suitable for displaying the relevant information. Care should be taken to ensure that these dashboards are clearly structured and not overloaded. The choice of colors should enable a quick interpretation of the current metrics, taking into account the challenges of color blindness. In addition, specific wireframes should be developed for the drill-downs of the metrics to provide detailed views.

Figure 3.9 illustrates the wireframe for the design of the dashboards. A horizontal navigation bar is integrated in the upper area, which offers conventional navigation options within the software. Three buttons are placed directly below this bar, which

make it possible to favor the current dashboard, select another dashboard or hide the display of build jobs. In the section below, there are two main areas that are intended for displaying the metrics and the build jobs. Both elements, metrics and build jobs, are visualized in the form of cards, which are shown in more detail in Figures 3.10 and 3.11.

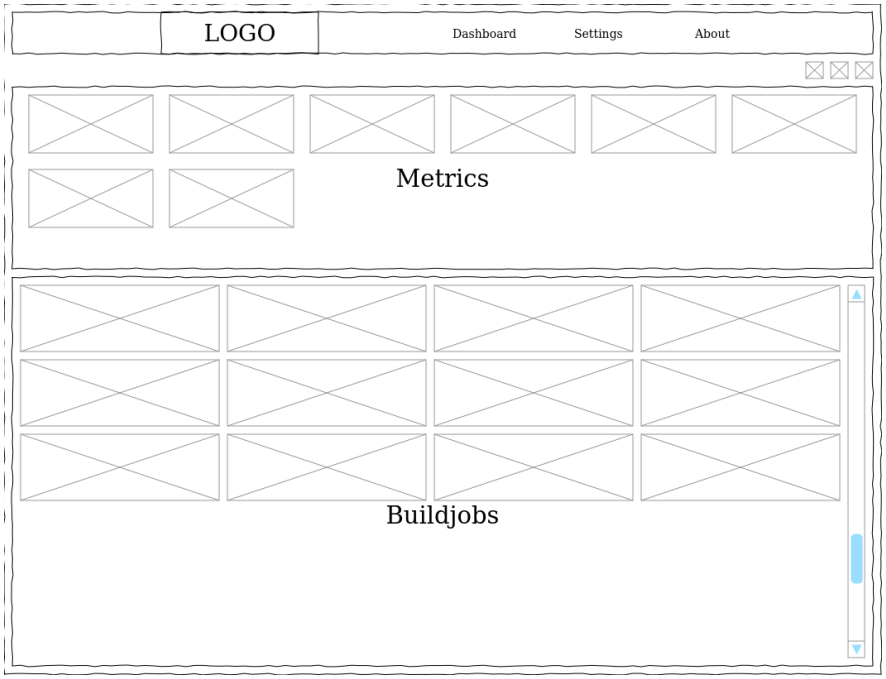

Figure 3.9: Wireframe of the structure of the dashboards

Figure 3.10 illustrates the structure of a metric card. The name of the metric is displayed on the left-hand side, while the current value of the metric is displayed on the right-hand side. When implementing the metric cards, it is essential to consider the principles of information design (see section 2.5) and to select suitable colors for the metric cards and for the various status states accordingly.

Figure 3.10: Wireframe of the structure of the metric cards

Figure 3.11 shows the structure of a build job card. The name of the repository in which the build job is executed, the type of pipeline, the associated branch and a timestamp are displayed on this card. The build number is displayed in the bottom right-hand corner, while the status of the build job is displayed in the top right-hand corner. For the status display in particular, it is important to observe the principles of information design (see section 2.5) and to use suitable colors to differentiate the statuses.

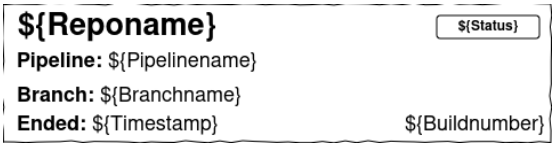

Figure 3.11: Wireframe of the structure of the buildjob card

Figure 3.12 shows the structure of the detailed view of a metric. All metric data is presented in a table. The threshold values above which a metric is flagged as a warning or failure are shown above the table to provide additional context for interpreting the metric value.

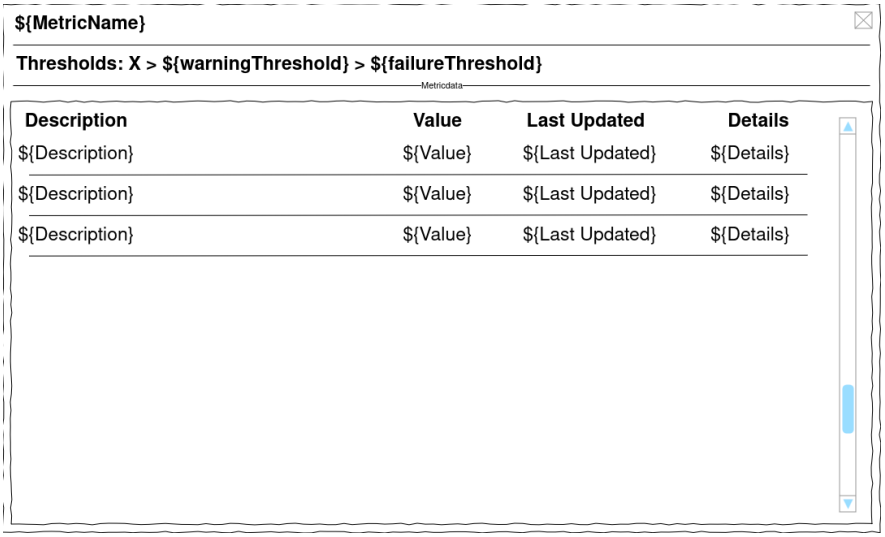

Figure 3.12: Wireframe of the structure of the metric details

Figure 3.13 shows the structure for configuring dashboards. There are two buttons in the top right-hand corner that allow you to either load an existing dashboard for configuration or create a new one. Seven boxes are displayed in the lower area, each representing the seven link tables in the database. Each box shows the entries of the associated table, including repositories, teams, metrics and individual data. There is a search field above the entries that allows users to search for specific information. By selecting an entry, the corresponding information is displayed on the dashboard. In addition, each box has a button in the top right-hand corner that opens a help window and provides a brief explanation of how to configure the respective box.

Figure 3.13: Wireframe of the structure of the dashboard configuration

3.5 Change Management Concept

This section explains the concept of introducing the software in more detail.

3.5.1 Change Management Strategy

Modern software development teams are already familiar with iterative processes (see sections 2.1.1 and 3.1.1). An iterative approach is therefore also chosen for the introduction of the software. The principles of lean change management (see section 2.6.1) serve as the basis, as these harmonize well with modern development methods. Particular emphasis is placed on continuous user feedback. The end user's perspective was already taken into account during the development of the software concept, and this user-centered approach is to be continued during the introduction of the software.

The software is introduced step by step in accordance with the step-by-step approach (see section 2.6.2), with implementation being carried out in teams within the company. A continuous, open communication channel enables user feedback to be recorded and integrated promptly. In this way, errors, bugs or necessary adjustments can be identified and implemented more quickly. The active involvement of users also promotes greater acceptance of the software, which further supports the gradual introduction process.

3.5.2 Strategic team selection for the software launch

As the software is to be introduced iteratively, it is first necessary to determine the order in which the software will be introduced among the various development teams (or individuals in the case of smaller companies). It is recommended that teams with particularly experienced developers and a long period of service are selected first, as this will allow a higher level of feedback to be generated during implementation. In addition, care should be taken to ensure that the general acceptance of the software within the first teams is initially high in order to minimize resistance to the introduction [Kot96, p. 41ff]. In the further course of the project, the software is gradually introduced to other teams until finally all participants are actively using the software.

3.5.3 Training and Support Plan

To ensure maximum support during the introduction of the software and to maintain a high level of user acceptance and satisfaction, even in the event of errors, it is necessary to provide suitable documentation. This documentation should enable users to look up the operation of the software. In addition, a continuously open communication channel should be set up in order to be able to respond promptly to feedback, problems and suggestions for improvement.

3.6 Evaluation Concept

An empirical quantitative method is used to evaluate the influence of the software on the software development process and on the way users work (see section 2.7). According to Pfleeger, questionnaires are particularly suitable in this context, as the software is introduced in accordance with the implementation concept (see section 3.5) and the evaluation can be carried out after the software has been in use for a certain period of time [Pfl95].

By using questionnaires, it is possible to make statements about a larger number of software developers. According to Conradi, for example, the opinions of 100 software developers can be inferred if 25 of them have completed a questionnaire on a new process. The aim of such questionnaires is therefore to provide generalizable results [Con03, p. 29].

In addition, questionnaires are particularly suitable for this work, as they are an efficient and time-saving method in the context of a time-limited research project [Shu08, p. 23].

When creating the questionnaires, the following aspects are taken into account in accordance with the recommendations of Groves et al. in order to capture the respondents precise attitudes [Gro09, p. 362ff]:

- The attitude object is clearly defined.
- Double-barreled questions are avoided.
- If necessary, the intensity of the attitude is measured by separate items.
- Closed questions are used to record attitudes.
- Five- to seven-point response scales are used, with each scale point clearly labeled.
- The scales are started with the least popular end.

The final questionnaire can be found in the appendix under point B.

When conducting the questionnaire, it must be ensured that it is completed anonymously. This serves to avoid distortions in the answers that could arise if participants deviate from their actual opinion due to social desirability or fear of negative evaluation [Gro09, p. 256f].

4 Implementation

This chapter covers the exemplary implementation of the software based on the previously developed concept (see section 3). The software, named "DevMon"—a combination of the words Development and Monitoring—is implemented and introduced as an example in the company "Janitza Electronics GmbH".

4.1 Janitza electronics GmbH

Janitza electronics GmbH, based in Lahnau, Mittelhessen, employs around 400 people and is active in the field of energy management. It offers comprehensive solutions ranging from measuring devices to software named "GridVis" that enables the configuration of the measuring devices and evaluates the collected data. The software is currently developed by around 40 employees in a total of 8 teams.

Buildmonitor

The “Research and Development - Software” department currently uses a self-developed tool called “Buildmonitor”, which already covers similar functions as the planned new software. The Buildmonitor is used to collect and display relevant information and pass it on to the developers.

A major limitation of the Buildmonitor is the fact that the dashboard is neither configurable nor customizable. This means that all teams have to use the same view, making it impossible to distribute information to specific people. In addition, new metrics can only be added by making direct changes to the source code.

Technologically, the Buildmonitor is based on a React application that displays data from a Firebase application. This data is stored in a Firestore database, whereby the structure of the database is not generic. Each metric has its own predefined table, which means that both the source code and the database structure have to be adapted to add new metrics.

In order to address the challenge of targeted information distribution, the role of the “Monitor-Pate” was introduced. This task is rotated between the teams as part of sprints. The main responsibility of the Monitor-Pate is to monitor the Buildmonitor on a daily basis and to forward relevant information via Microsoft Teams to the responsible persons or teams.

Figure 4.1 shows the structure of the Buildmonitor, with the various metrics shown in the upper section and build jobs from Jenkins integrated in the lower section using an iFrame. Table 4.1 lists the available metrics and provides a brief explanation of each metric.

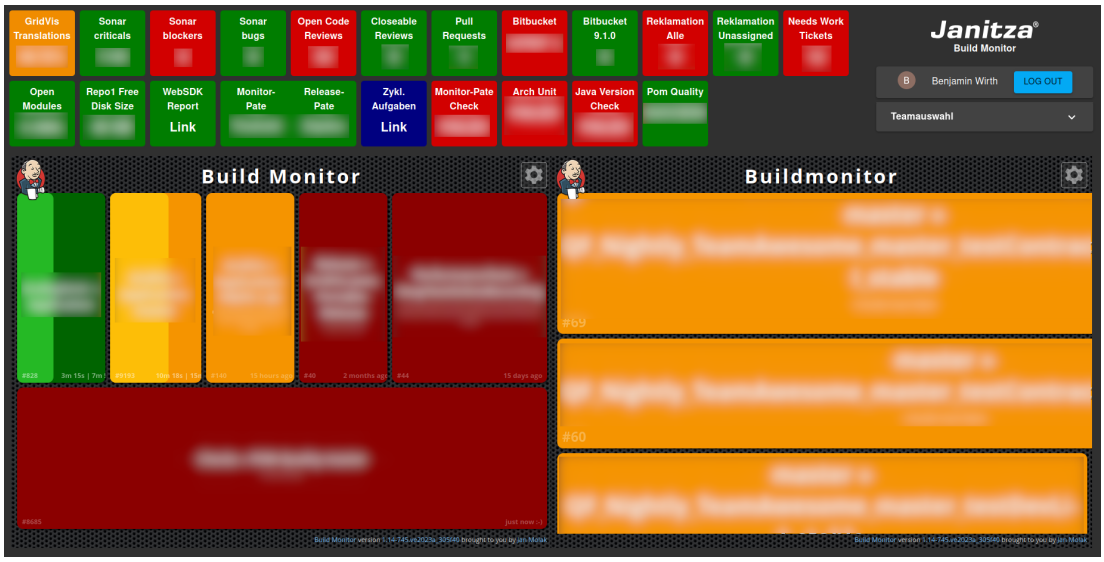

Figure 4.1: Example screenshot showing the interface of the "Buildmonitor" (sensitive data has been made unrecognizable).

Metric	Source	Description
GridVis Translations	Internal translation server	Translation progress broken down by GridVis module
Sonar Criticals	SonarCloud	SonarCloud issues classified as "Critical"
Sonar Blockers	SonarCloud	SonarCloud issues classified as "Blockers"
Sonar Bugs	SonarCloud	SonarCloud issues classified as "Bugs"
Open Code Reviews	Fisheye & Crucible	Currently open Code-Reviews
Closeable Reviews	Fisheye & Crucible	Code-Reviews that meet the requirements to be closed
Pull Requests	Bitbucket Cloud	Currently open Pull-Requests
Reklamation Alle	Jira	Currently open Jira issues that are classified as complaint
Reklamation Unassigned	Jira	Currently open Jira issues that are classified as complaint and got no assignee
Needs Work Tickets	Jira	Jira issues where information is missing (e.g. no registered version or no assigned sprint)
Open Modules	Bitbucket Cloud	Maven modules that are "open" (ready to be edited)
Repo1 Free Disk Size	Internal server	Current free disk space on the Repo1 server
Monitor-Pate Check	Bitbucket Cloud	A check to see whether the current monitor-pate has already become active today (similar to a dead man's switch)
Arch Unit	Bitbucket Cloud	Architecture check - turns red for architecture violations
Java Version Check	Bitbucket Cloud	Checks whether a newer Java version is available than the one currently in use.
Pom Quality	Bitbucket Cloud	POM quality check - turns red if POM files with "errors" exist

Table 4.1: Metrics used in "Buildmonitor"

4.2 Technology Stack

This section describes the technological stack of the software in detail. Both functional and non-functional requirements (see section 3.2) were taken into account.

4.2.1 Version Control

During development, version control is carried out via "Bitbucket Cloud", as this platform is already established in the company. The "Feature-based Branching Strategy" is used to ensure a structured and clear development of the software components and individual features.

4.2.2 Data-Persistence

Based on the non-functional requirements (see section 3.2.2), the implementation of the database is realized with PostgreSQL. PostgreSQL is selected because it is a relational database system, as required in Requirement 4. In addition, PostgreSQL supports the JSONB data format, which enables a flexible and generic data structure in order to support the different structures of the information, as specified in Requirement 5. These features make PostgreSQL a suitable choice for implementing the defined requirements. Moreover, PostgreSQL is the most widely used database system [Sta23][Sta24], which increases the likelihood that many employees are familiar with it. This widespread use positively impacts maintainability, as it facilitates easier support and development due to the larger pool of skilled professionals and resources available.

4.2.3 Backend-Service

The backend is implemented using SpringBoot, which is a suitable choice for several reasons:

- The structure of SpringBoot encourages the use of the "controller service repository" architectural pattern, which improves the maintainability of the code and thus meets Requirement 9.
- The use of JPA[1] considerably simplifies the entire communication with the database.
- The author already has a solid knowledge of the Java programming language.
- As SpringBoot is already in use in the company, this contributes to better maintainability of the software, as several employees are able to further develop and maintain the software after the initial implementation.

1 Jakarta Persistence API - Interface for transferring data to database systems

- SpringBoot is the most commonly used Java web framework [Jet23][Sta23][Sta24], which means that new employees are very likely to already have prior knowledge and can immediately apply their knowledge in practice.

The standard HTTP methods GET, POST and PUT are used for all REST routes. However, dashboard subscriptions are an exception. Here it is essential that new data is sent to the open dashboards as soon as it arrives. Server-Sent Events (SSE) are implemented for this purpose, as they are easy to handle and are well suited to this requirement.

4.2.4 Data fetchers

The data fetchers are programmed in JavaScript and executed with Node.js. JavaScript is a logical choice, as it is the most widely used programming language in the world [Jet23][Sta23][Sta24]. A dedicated data fetcher is created for each data source. These data fetchers are divided into two categories: those that are executed hourly and those that are executed daily. The execution of the data fetcher is scheduled via separate Bitbucket pipelines. The foundation for the data fetchers (or rather the metrics) is based on the metrics provided by the software "Buildmonitor" that is already used by Janitza electronics GmbH. In total, the following data fetchers are implemented for the initial use of the software:

- **Daily**
  - **Bitbucket repositories:** This fetcher retrieves all repositories from Bitbucket and sends the data directly to the backend, where it is either recreated or updated.
  - **Team Repository Responsibilities:** This fetcher extracts the team responsibilities for repositories from an online Excel spreadsheet and transmits this information to the backend.
- **Hourly**
  - **FeCru Code Reviews:** This fetcher extracts code reviews from the code review portal FishEye & Crucible. Two metrics are fed: Firstly, the number of open code reviews and secondly, the number of code reviews that meet the requirements for closure.
  - **Jira Issues:** This data fetcher extracts data from the Jira ticket management system and feeds three metrics: The number of all open complaints assigned to an agent, the number of all open complaints without an agent and the

number of tickets with errors, such as missing sprint assignments or missing solution versions.

- **SonarCloud Issues:** This data fetcher retrieves data from the code analysis tool SonarCloud and records the metrics of the SonarCloud blockers, i.e. particularly serious errors in the source code. It would also be possible to record other types of issues if this is considered useful.
- **SonarCloud Coverage:** This data fetcher also retrieves data from the SonarCloud code analysis tool and captures the SonarCloud coverage metric for each repository. Other potential metrics could also be considered, such as the number of lines of code, code duplications and other relevant metrics.
- **GridVis Translation:** This data fetcher retrieves data from the company's own translation server and records the metrics of the missing translations.
- **Bitbucket Cloud Pull-Requests:** This data fetcher retrieves data from Bitbucket Cloud and captures all open pull-requests.
- ***Deployment server diskspace:*** No datafetcher is used directly here; instead, the deployment server reports the available disk space independently via webhook and sends it to the backend.

The metrics are created and configured in the software on a separate page. Once a metric has been created in the web interface, it is linked to the data fetcher by storing the corresponding metric ID in the data fetcher.

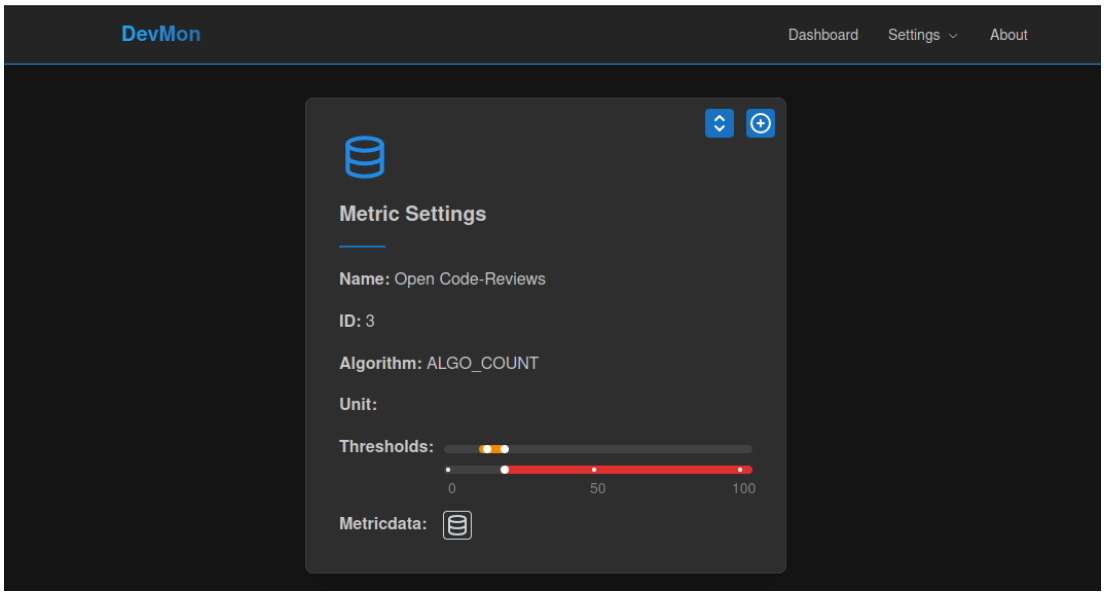

Figure 4.2: Example configuration of the "Open Code-Reviews" metric

4.2.5 Webhooks

Webhooks are set up in all repositories to enable the display of build jobs on the dashboards. These webhooks automatically forward data from running pipelines and transmit them to the backend service via the corresponding REST interfaces. In this way, updates, such as error messages or interruptions to build jobs, are also transmitted to the service.

4.2.6 Security

Various security mechanisms are implemented within the SpringBoot application:

- All REST routes that are relevant for the front end are protected by OAuth2 (Spring Security), with Bitbucket being used as the OAuth2 consumer. This means that only logged-in users from the Janitza workgroup have access to these routes and therefore to the frontend.
- The REST routes that are relevant for the data fetchers and the webhooks are secured by HMACs. These routes can only be used if a specific secret is used.
- Cross-Origin Resource Sharing (CORS) is implemented and configured to control how resources on the server are shared with external origins, further enhancing security by restricting which domains can access the application's resources.

In addition to the security implemented by Spring Security, an Apache reverse proxy is used to further restrict access to the frontend and backend. Access is only possible for devices from the internal Janitza network or via VPN. In addition, the IP range of Bitbucket Cloud is unlocked to enable the data fetchers to reach the backend and transmit the data.

4.2.7 Frontend-Service

The frontend is implemented using React, which is a suitable choice for several reasons:

- React is used as the main framework for the "GridVis" software developed within the company. As a result, many developers already have in-depth prior knowledge, which facilitates the maintainability of the application and allows more people to be involved in the further development and maintenance of the software after initial implementation.

- React is the most commonly used web framework among professional developers [Sta23][Sta24]. There is therefore a high probability that new employees will already have prior knowledge of React, which makes it easier for them to quickly familiarize themselves with the software and contribute productively.

The React Component Library Mantine is used to support front-end development, as it enables the efficient creation of components and thus saves development time. Mantine is also available free of charge as an open source solution.
The frontend itself does not contain any business logic, but merely represents the resources provided by the backend via REST or SSE[1].
Figure 4.3 presents the final implementation of a dashboard, using a software development team as an example. The dashboard clearly visualizes the information relevant to the team, including open code reviews, tickets that still need to be processed and critical issues from SonarCloud. In addition, only the build jobs relating to the repositories for which the team is responsible are displayed in the lower section.

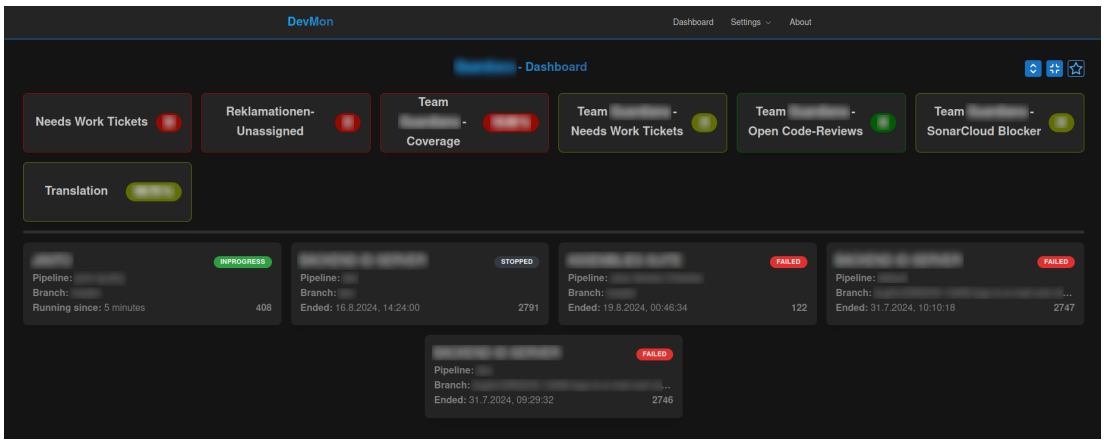

Figure 4.3: Dashboard of a software development team

Figure 4.4 illustrates the detailed view of a metric. It is clear that the display is automatically adapted to the specific details of the metric. The “Status” is not implemented here as a separate column in the database table, but is stored in the “Details”. Each piece of information in this view is accessible with a mouse click and leads directly to the corresponding source.

1 Server Sent Events - The server independently sends data to the “subscribers”

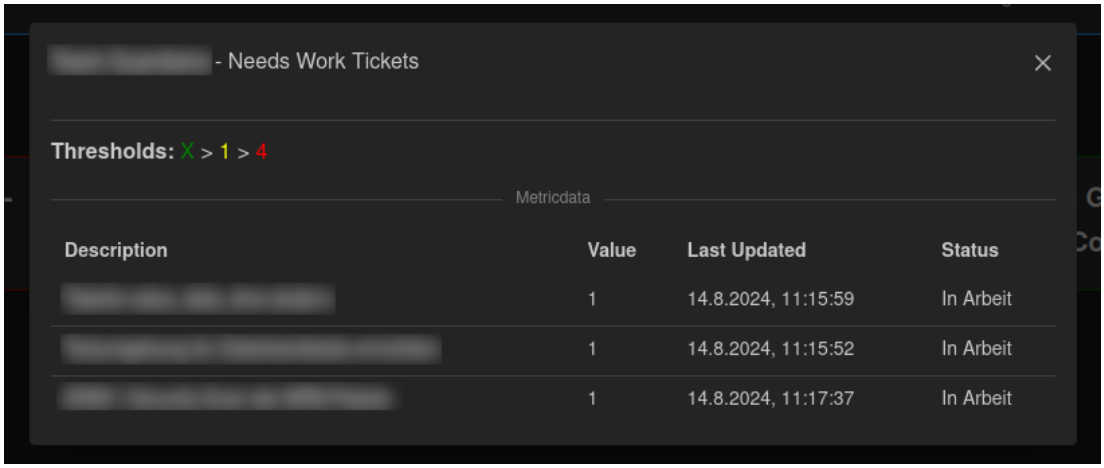

Figure 4.4: Detailed view of a metric

4.3 Testing Strategies

As requirement 10 specifies that the business logic of the application must have a test coverage of at least 75%, particular attention must be paid to this aspect. For this reason, Test-Driven Development is initially applied during development, in which tests are first created before the corresponding methods are programmed.
If errors occur during use or development, specific tests are first developed to cover these errors. The affected methods are then adapted or refactored in order to rectify the identified problems.
A total of 210 tests are available to check the complete functionality of the implemented methods.

4.4 Pipelines

According to requirement 11, different pipelines are configured in Bitbucket Cloud to automate different development and deployment processes:

- One pipeline is executed with every pull request to test the software for errors and check them by linting.
- Another pipeline is activated when changes are made to the dev branch. Here too, the software is checked by tests and linting.
- The third pipeline is triggered when changes are made to the main branch. First, the software is tested and linted. If these tests are successful, the backend and frontend are built separately and packaged in Docker images. These images

are then uploaded to the company's own Artifactory server. Finally, a script is executed on the production server that downloads and starts the new Docker images. If only changes are made to the data fetchers, the build and deployment steps for the frontend and backend are skipped.

4.5 Documentation

In accordance with requirement 1, both a user manual and documentation of the REST interface of the backend are created. The user manual is created in the company's internal wiki using Confluence and explains the basic operating elements, the first login, the metrics and the dashboards. The instructions are illustrated with screenshots. An exemplary excerpt can be found in the appendix under point C.1.
The documentation of the REST interface is created in accordance with the OpenAPI specification and provided with SwaggerUI in a web interface. An example screenshot of this documentation can be found in the appendix under point C.2. The REST interface is particularly useful for administrators, as it provides a clear overview of the available routes. In addition, the SwaggerUI is secured by OAuth2, which makes it possible to try out the routes directly within the SwaggerUI.

4.6 Software Introduction

This section describes the introduction of the software in the company on the basis of the change management concept developed in section 3.5.

4.6.1 Realize the change management strategy

The first step is to carefully select a suitable team for the initial introduction of the software. As described in the underlying concept, care is taken to ensure that the selected team includes both experienced software developers and employees who have been with the company for some time. This team actively uses the software over a period of two weeks before other teams are included in the process. The entire process is carried out in an iterative approach, which is described in detail in the concept.
Maintaining a continuously open communication channel is an essential part of the change process. This is realized via the "Microsoft Teams" communication platform used in the company to ensure that feedback and suggestions from employees can be recorded immediately and without delay.

4.6.2 Determining the fall-back strategy

During the introduction of the new software, the old Buildmonitor remains in operation. This ensures that the old Buildmonitor can be accessed at any time in the event of problems. It also enables the teams to make direct comparisons in order to analyze the effects of the new software on the development process compared to the previous work with the old Buildmonitor.

4.6.3 Creation of required dashboards

First, the teams must be created via the web interface (see figure 4.5). A separate dashboard is then created and configured for each team. This is done via a separate page for the dashboard configuration (see figure 4.6).

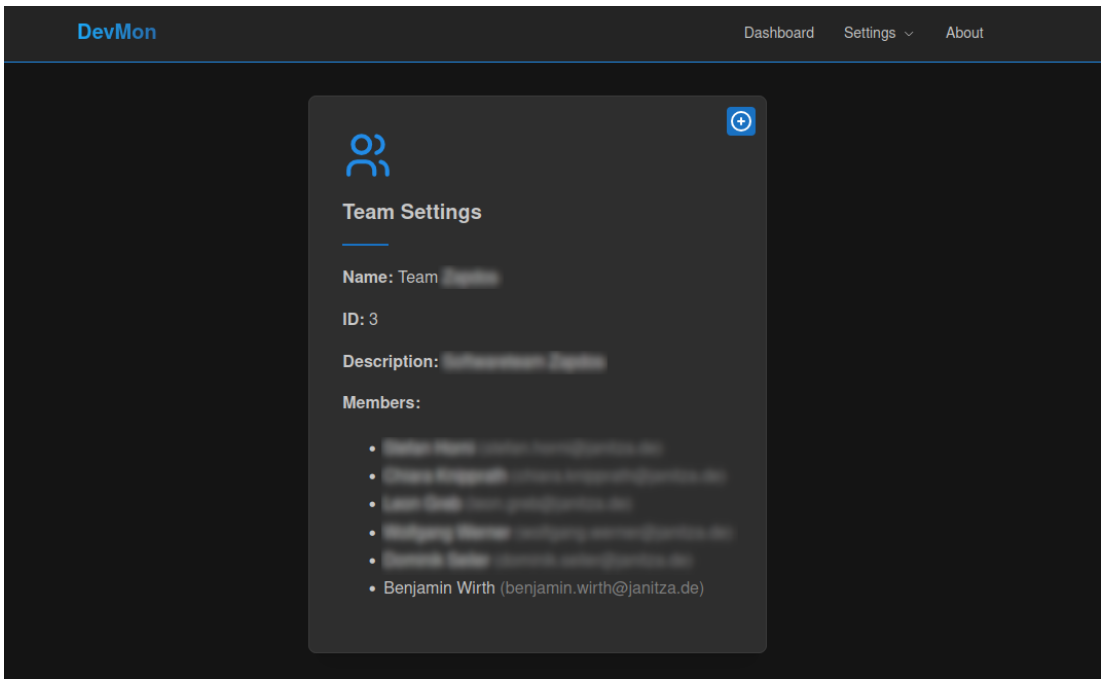

Figure 4.5: Teams can be created, configured and viewed via an extra page

Figure 4.6 illustrates the exemplary configuration of a team dashboard. The individual tabs on the dashboard setting page represent the corresponding database tables, in particular the junction tables (see section 3.4.1 - The junction tables are marked in red).

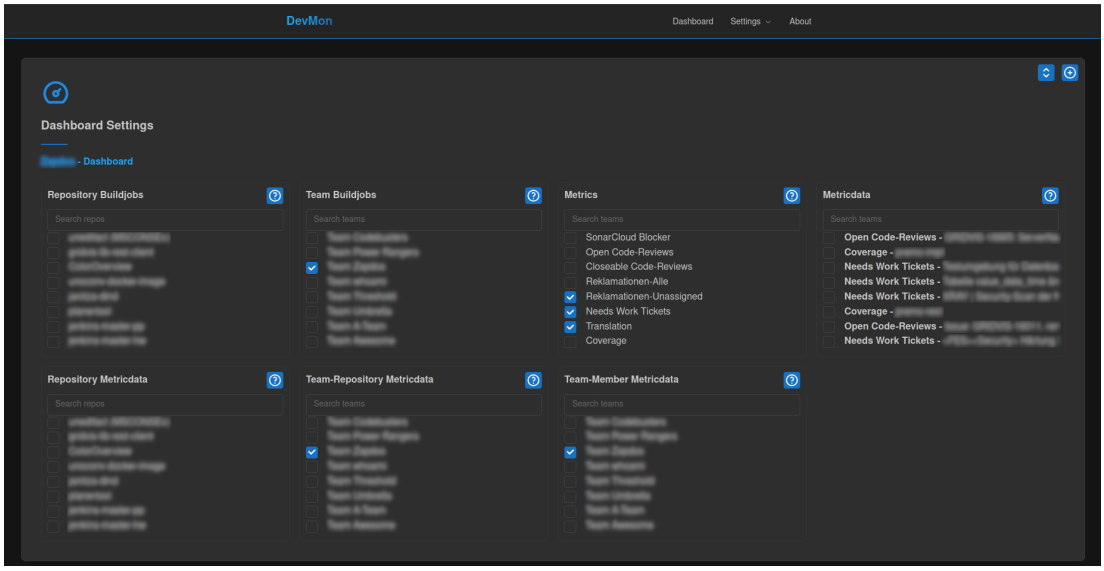

Figure 4.6: Dashboards can be individually configured via an extra page

4.7 Determining the influence of the software on the software development process

Once the software has been fully implemented and productively introduced in all software development teams at Janitza electronics GmbH, the extent to which the software influences the development process will be investigated. The evaluation concept from section 3.6, which is based on the use of questionnaires, will be implemented to collect the relevant data.

The questionnaires are provided via Microsoft Forms and allow users to take part in the survey anonymously and once only. The questions follow exactly the pattern defined in the concept and were only translated into German.

The evaluation and analysis of the collected data is carried out in section 5.2, where the results are discussed in detail with regard to the influence of the software on the development process.

5 Implementation Review

This chapter evaluates the implemented "DevMon" software (see chapter 4). First, the extent to which the software meets the requirements of the originally developed concept (see chapter 3) is examined. The results of the questionnaires conducted 4.7 are then analyzed in order to evaluate and discuss the influence of the software on the software development process.

5.1 Alignment between Concept and Implementation

This section examines the extent to which the implemented "DevMon" software (see chapter 4) complies with the concept (see chapter 3) and the requirements defined therein. The functional requirements are checked first, followed by an analysis of the fulfillment of the non-functional requirements.

5.1.1 Review of the functional requirements

The functional requirements were defined in section 3.2.1 on the basis of user stories. Each of the user stories is checked with regard to its fulfillment. A complete tabular representation of the review can be found in the appendix (see section D.1). An explanation with reference to the implementation is provided for each user story. The result of this analysis shows that the implementation of the "DevMon" software completely fulfills all functional requirements based on the concept.

5.1.2 Review of the non-functional requirements

The non-functional requirements are documented in section 3.2.2. Each requirement is checked for fulfillment. A comprehensive tabular overview of the review can be found in the appendix (see section D.2). An explanation of fulfillment and implementation is provided for each requirement. The analysis shows that all non-functional requirements are fulfilled by the "DevMon" software.

5.2 Questionnaire analysis

The evaluation of the completed questionnaires is presented in this section. A detailed overview of the response distributions can be found in the appendix under point E. A total of 17 employees of Janitza Electronics GmbH took part in the survey.

5.2.1 Frequency of use

The majority of participants stated that they use DevMon several times a week or even daily. The reason given for less frequent use was that DevMon is not yet fully integrated into the daily workflow. It was also reported that DevMon is predominantly used independently; however, use in a team, for example in daily meetings, is not widespread.

5.2.2 Influence on the software development process

The majority of respondents reported that the use of DevMon ensures a better overview of the development process. In addition, DevMon helps most participants to react more quickly to problems such as complaints or build errors. The targeted presentation of information was perceived as a time-saver overall.
The statement that DevMon has improved the way we work was rated positively by half of the respondents, while the other half remained mostly neutral. Only two people stated that the use of DevMon had not improved their working methods.

5.2.3 Concrete Effects

When asked whether DevMon had improved work efficiency, more than half of the respondents answered that there had been a positive change. The remaining participants stated that efficiency had not changed. No one reported any negative effects. With regard to communication, it was reported that hardly any changes had taken place within the teams, but a majority stated that communication with other teams had improved. This assessment is also confirmed in the free text responses.

5.2.4 Demographic Questions

In the demographic questions, around a third of respondents stated that they had been with Janitza for less than one year, while another third had been with the company for more than one year and less than three years. The final third stated that they had

been with the company for more than three years. All participants worked in the field of software development, with one person stating that they worked as a student trainee in software development.

5.2.5 Open questions

In the free text responses, most respondents said that the tool is generally very good and has great potential. However, it was noted that the full integration of the tool into everyday working life was still lacking in some areas. Several people also again positively emphasized the targeted provision of information.

5.2.6 Discussion

Overall, the completed questionnaires indicate that DevMon has a positive influence on the software development process. The majority of respondents already use the software on a daily basis and report increased efficiency and a better overview of the development process.

A comprehensive analysis of the responses shows that the more frequently DevMon is used, the more positively efficiency and overview are rated. People who have regularly integrated the tool into their daily work have a more positive effect compared to those who use it less frequently.

It can be deduced from this that the integration of other important metrics, such as cyclical tasks, could be useful. It can also be expected that the longer the tool is used, the better its integration into everyday working life will become, even among less active users.

However, it should be noted that the questionnaire was only completed by 17 people, while around 50 people currently use the tool. The reason for this discrepancy remains unclear and should be taken into account in the further evaluation. It should also be noted that all respondents work in the field of software development and that the survey was conducted exclusively at Janitza Electronics GmbH.

6 Conclusion

The final chapter summarizes the results of the work. After the final answers to the research questions, the key findings and the resulting conclusions are presented. In addition, the next steps are explained and an outlook on possible future developments is given.

6.1 Summary

The first chapter of the thesis introduces the topic of the thesis and then formulates three research questions. The first research question examines how a software solution must be conceptually designed in order to aggregate and visually prepare data from various tools used in the software development process. The first part of the third chapter deals with the development of this concept. For this purpose, specific requirements were developed that served as the basis for the creation of a detailed software concept. This concept is based on a comprehensive literature research, the analysis of existing software solutions in this area, as well as widely used and current ISO standards. The subsequent prototype development of software based on this concept at Janitza electronics GmbH also went smoothly. The software could be developed without any adjustments to the original concept, which confirms the practicality and robustness of the concept. However, it should be noted that the implementation only took place in one company, which represents a possible external threat to validity. Despite this limitation, the first research question was successfully answered.

The second research question deals with the introduction of software into a company and its integration into existing processes. In the middle part of the third chapter, a concept for software introduction is presented that is based on various approaches from the relevant literature. The result is an iterative introduction process that involves the continuous integration of user feedback in order to increase acceptance of the software during introduction. The successful introduction of the developed software at Janitza electronics GmbH shows that the proposed introduction model works well in practice. However, it must also be taken into account here that the implementation only took place in one company, which indicates a potential external validity risk. Nevertheless,

the second research question could be clearly answered based on the positive experience gained during the introduction of the software.

The third research question addresses the influence of the implementation of the created software concept on the software development process. To answer this question, a questionnaire was developed based on relevant literature and completed by the end users of the software. However, as only 17 people had answered the questionnaire by the end of the thesis processing period, a general statement is only possible to a limited extent. In addition, the implementation was only carried out in one company, which indicates a possible threat to external validity. A conclusive answer to the third research question is therefore only possible to a limited extent within the scope of this work. Another aspect to consider is the relevance of the metrics displayed on the dashboards. The usefulness of the software depends to a large extent on the selection and presentation of these metrics. In order to answer the third research question conclusively, additional implementations and further investigations into the metrics displayed on dashboards would be required.

In summary, it can be said that the software concept developed forms a solid basis for a framework which, with the correct selection and use of suitable metrics, has the potential to positively influence the software development process and thus improve the day-to-day work of software developers.

6.2 Evaluation

In this section, the methodology, potential threats to validity and the results of the individual research questions and hypotheses are comprehensively evaluated.

RQ1: What does a concept for software that consolidates and visualizes information from various tools in the software development process look like?

To answer the research question, a combined methodology of literature research and the analysis of existing software solutions was chosen. Scientific publications, conference papers, statistics, reference books and ISO standards were consulted as part of the literature research. Access to relevant literature proved to be largely unproblematic. However, analyzing existing software solutions proved to be more challenging, as only a few comparable approaches could be identified. Grafana and Monitoror were used as examples, but their applicability to the specific problem area was limited. In retrospect, it can be seen that the software analysis had less influence on the concept development, while the literature research played a central role. In particular, the

data integration paradigms described in the literature provided a solid basis for the design. The application of and orientation towards practical approaches such as user-centered design and corresponding methods (personas, user scenarios, user stories) also significantly supported the development of the concept.

The subsequent implementation of the software solution based on the developed concept ran without any major challenges, which underlines the quality of the concept. A possible external threat to validity could arise from the implementation of the software in a specific company context. However, this is considered to be low, as the concept is based on generally recognized scientific principles and standards.

RQ2: What considerations and steps are necessary for a successful introduction and integration of such software within an organization?

To answer the research question, a literature review was chosen as the central methodology, which focused on change management in the context of software engineering. Reference books, scientific papers and statistical data were used to develop a well-founded concept for the introduction of the software. The implemented software was then implemented and introduced at Janitza electronics GmbH in accordance with this concept, which confirmed the practical effectiveness of the concept.

The implementation was carried out step by step, gradually integrating the software into the teams and continuously collecting feedback via open communication channels. This iterative approach, initially targeting a team with high adoption, allowed for early improvements and optimization of the software before the ever-widening rollout. As a result, teams with initially lower adoption showed increased usage and a positive attitude towards the software as they benefited from the improvements.

During the implementation, minor adjustments were made based on feedback, such as the graphical adaptation of various elements or the arrangement of metrics. The effectiveness of this iterative, feedback-oriented approach underlines its consistency with the principles of lean change management.

A potential external threat to validity could be that the software has only been introduced in one company so far. However, this risk is considered to be low, as the implementation concept is based on generally applicable specialist literature and current statistical data.

RQ3: To what extent can the implementation of the proposed concept improve the efficiency and effectiveness of the software development process?

In order to answer research question 3, a theoretical concept first had to be developed to serve as the basis for the analysis. The originally defined deductive and quantitative approach of the thesis restricted the choice of suitable methods to empirical-quantitative procedures. In order to determine an adequate methodology, a comprehensive literature search was carried out, focusing in particular on reference books and scientific articles on the subject of empirical research in the field of software engineering. Based on this research, the questionnaire survey method was selected to answer the research question. The questionnaire was designed with current best practices in mind.

However, it must be noted that it is difficult to provide a universally valid answer to the research question. The data collected originates exclusively from users within Janitza electronics GmbH, which means that the results can only be applied to this company. This poses a potential threat to external validity. In addition, the chosen methodology of the questionnaire survey could be critically questioned. Questionnaires are susceptible to various bias effects, such as social desirability, where participants tend to give more socially acceptable answers [Pod03]. Alternatively, other methods, such as in-depth interviews or detailed observation of work processes before and after the introduction of the software, could also have been considered. However, the limited time frame of the work must also be taken into account, which would have made a more in-depth investigation using more time-consuming methods such as interviews or observations more difficult.

Another critical point is that the usefulness of the software depends heavily on the metrics displayed on the dashboards. As the selection and definition of these metrics was only a marginal topic of the study, further research is required in this area in order to conclusively clarify the third research question.

To summarize, there is a potential threat to both internal and external validity when answering this research question. Accordingly, there is no real final answer to this research question.

Hypothesis: Software that aggregates and processes differently structured information generated during the software development process offers added value for software developers and other people involved in the software development process

Three research questions were answered to test the hypothesis. The basis for proving the hypothesis is the developed software concept as well as its implementation and the successful introduction of the software in a company. The hypothesis that software that

normalizes, aggregates and visualizes information from the software development process can positively influence this process is supported to a certain extent by the results of the questionnaire from the third research question. The answers of the respondents indicate that a positive influence on the software development process is possible through the use of the software.

Here too, however, it should be noted that the influence of the software on the software development process depends to a large extent on the selection of information and metrics displayed on the dashboards. The quality and relevance of this data is crucial for the effectiveness of the software. Similar to the third research question, there is also a potential external validity threat, as both the implementation of the software concept and the subsequent survey were conducted exclusively in one company. Furthermore, it could be discussed whether the third research question was necessary or whether the hypothesis could be sufficiently proven without it.

Finally, similar to research question 3, the hypothesis can at least be proven for the company Janitza electronics GmbH.

6.3 Further Approaches

This work has taken the approach of continuously updating and replacing data so that the dashboards always display the most up-to-date data. Another potentially valuable approach could be to capture and store time-based data. By implementing a time-based database, such as *Prometheus*, metrics could be monitored over longer periods of time. This would make it possible to recognize trends and react to long-term developments in the data.

The introduction of time-based data storage could not only improve the monitoring of metrics, but also support deeper analysis. For example, correlations between different metrics could be examined to identify patterns and relationships. Such information would be valuable for proactive measures and strategic decisions, for example to optimize development processes or identify potential problems at an early stage.

In addition, historical data analysis could help to fine-tune thresholds or algorithms used to evaluate metrics. In the long term, this approach would provide a broader and deeper insight into the performance and dynamics of systems, which could have a positive impact on both the efficiency and quality of software development.

The focus of the software would therefore shift slightly. Instead of collecting information centrally and forwarding it to the responsible developers, the focus would be more on monitoring metrics, performance and long-term trends. Nevertheless, an implementation

that supports both approaches - centralized information forwarding and continuous metrics monitoring - in a single software solution would be feasible. This would enable holistic and flexible use that meets both current and long-term requirements for monitoring and analyzing development processes.

6.4 Next Steps

Based on the generic concept for a software solution developed in this thesis, further investigation could focus on which metrics and information should be displayed on the dashboards in order to optimally increase the efficiency of work processes. It would be important to consider factors such as team structure, team size and the specific areas of responsibility of the teams. Such an analysis would help to optimize the use of the software in a targeted manner and adapt the presentation of information more closely to the individual requirements of the teams. In this context, best practices and potential anti-patterns should also be included in the analysis. These further steps provide a conclusive answer to research question 3 and thus comprehensively confirm the hypothesis.

Another critical next step would be the development of a comprehensive backup strategy, as the software handles a large volume of data, and any system failure could have severe consequences. A well-designed backup plan would ensure data security and minimize the risk of data loss, further enhancing the reliability and robustness of the software solution.

6.5 Outlook

This work lays the foundation for software that supports the software development process through data integration and information design by preparing and visualizing relevant information in a targeted and centralized manner for software developers. The software concept presented here offers a generic solution that can be flexibly adapted to different use cases and tools used in the software development process.

For future research, it would be useful to investigate in detail which specific information and metrics should be displayed on the dashboards in order to achieve a significant improvement in the efficiency and quality of work of the developers.

Bibliography

[Ale] Alexandre Demode und Jean-Sébastien Didierlaurent: Monitoror — Unified monitoring wallboard, URL https://monitoror.com

[Ale24] Alexandre Demode: Monitoror-Releases (2024), URL https://github.com/monitoror/monitoror, original-date: 2019-01-11T14:59:26Z

[And02] Anderson, Kenneth M.; Sherba, Susanne A. und Lepthien, William V.: Towards large-scale information integration, in: *Proceedings of the 24th international conference on Software engineering - ICSE '02*, ACM Press, Orlando, Florida, S. 524, URL http://portal.acm.org/citation.cfm?doid=581339.581403

[Ant12] Antunes, Bruno; Cordeiro, Joel und Gomes, Paulo: Context Modeling and Context Transition Detection in Software Development:, in: *Proceedings of the 7th International Conference on Software Paradigm Trends*, SciTePress - Science and and Technology Publications, Rome, Italy, S. 477–484, URL http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0004084204770484

[Bla17] Black, Alison H.; Luna, Paul; Lund, Ole; Walker, Sue; Lund, Ole und Spiekermann, Erik (Herausgeber): *Information design: research and practice*, Routledge, London New York (2017)

[BP20] Brandt-Pook, Hans und Kollmeier, Rainer: *Softwareentwicklung kompakt und verständlich: Wie Softwaresysteme entstehen*, Springer Fachmedien Wiesbaden, Wiesbaden (2020), URL http://link.springer.com/10.1007/978-3-658-30631-1

[Car16] Carroll, Noel und Richardson, Ita: Software-as-a-Medical Device: demystifying Connected Health regulations. *Journal of Systems and Information Technology* (2016), Bd. 18(2): S. 186–215, URL https://www.emerald.com/insight/content/doi/10.1108/JSIT-07-2015-0061/full/html

[Coi14] Cois, Constantine Aaron; Yankel, Joseph und Connell, Anne: Modern DevOps: Optimizing software development through effective system interactions, in: *2014 IEEE International Professional Communication Conference (IPCC)*, IEEE, Pittsburgh, PA, S. 1–7, URL https://ieeexplore.ieee.org/document/7020388/

[Con03] Conradi, Reidar (Herausgeber): *Empirical methods and studies in software engineering: experiences from ESERNET*, Nr. 2765 in Lecture notes in computer science, Springer, Berlin Heidelberg (2003)

[Dig] Digital.ai: 17th State of Agile Report, URL https://digital.ai/resource-center/analyst-reports/state-of-agile-report/

[Fel19] Felderer, Michael und Travassos, Guilherme Horta: The Evolution of Empirical Methods in Software Engineering (2019), URL https://arxiv.org/abs/1912.11512, version Number: 4

[Few06] Few, Stephen: *Information dashboard design: the effective visual communication of data*, O'Reilly & Associates, Sebastopol, CA, 1. aufl Aufl. (2006)

[Geo23] Georg, Stefan; Paegle, Lelde und Heiler, Chris: *Anwendung des Lean-Change-Management-Approachs in Zeiten der digitalen Transformation: Ein praxisorientiertes Vorgehensmodell*, essentials, Springer Fachmedien Wiesbaden, Wiesbaden (2023), URL https://link.springer.com/10.1007/978-3-658-42266-0

[Git22] GitLab: The GitLab 2022 Global DevSecOps Survey Thriving in an insecure world (2022), URL https://learn.gitlab.com/dev-survey-22/2022-devsecops-report

[Gra] Grafana Labs: Grafana OSS and Enterprise | Grafana documentation, URL https://grafana.com/docs/grafana/latest/

[Gra23] Grafana Labs: Observability Survey 2023 (2023), URL https://grafana.com/observability-survey-2023/

[Gro09] Groves, Robert M.; Fowler, Floyd J.; Couper, Mick; Lepkowski, James M.; Singer, Eleanor und Tourangeau, Roger: *Survey methodology*, Wiley series in survey methodology, Wiley, Hoboken, NJ, second edition Aufl. (2009)

[Hag19] Hagemann, Michael: *Changemanagement für Praktiker: Mindset, Infrastructure, Capability*, Schäffer-Poeschel Verlag, Stuttgart [Freiburg], 1. auflage Aufl. (2019)

[Hal20] Halstenberg, Jürgen; Pfitzinger, Bernd und Jestädt, Thomas: *DevOps: Ein Überblick*, essentials, Springer Fachmedien Wiesbaden, Wiesbaden (2020), URL https://link.springer.com/10.1007/978-3-658-31405-7

[Han17] Hanschke, Inge: *Agile in der Unternehmenspraxis*, Springer Fachmedien Wiesbaden, Wiesbaden (2017), URL http://link.springer.com/10.1007/978-3-658-19158-0

[Hus09] Hussain, Zahid; Slany, Wolfgang und Holzinger, Andreas: Current State of Agile User-Centered Design: A Survey, in: David Hutchison; Takeo Kanade; Josef Kittler; Jon M. Kleinberg; Friedemann Mattern; John C. Mitchell; Moni Naor; Oscar Nierstrasz; C. Pandu Rangan; Bernhard Steffen;

Madhu Sudan; Demetri Terzopoulos; Doug Tygar; Moshe Y. Vardi; Gerhard Weikum; Andreas Holzinger und Klaus Miesenberger (Herausgeber) *HCI and Usability for e-Inclusion*, Bd. 5889, Springer Berlin Heidelberg, Berlin, Heidelberg (2009), S. 416–427, URL http://link.springer.com/10.1007/978-3-642-10308-7_30, series Title: Lecture Notes in Computer Science

[ISO08] ISO/IEC JTC 1/SC 7: Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Data quality model (2008), URL https://www.iso.org/obp/ui/#iso:std:iso-iec:25012:ed-1:v1:en

[ISO19] ISO/TC 159/SC 4: Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems (2019), URL https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-2:v1:en:

[ISO23] ISO/IEC JTC 1/SC 7: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model (2023), URL https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-2:v1:en

[Jet23] Jetbrains: The State of Developer Ecosystem 2023 (2023), URL https://www.jetbrains.com/lp/devecosystem-2023/

[Kot96] Kotter, John P.: *Leading change*, Harvard Business School Press, Boston, Mass (1996)

[Kus22] Kuster, Jürg; Bachmann, Christian; Hubmann, Mike; Lippmann, Robert und Schneider, Patrick: *Handbuch Projektmanagement: Agil – Klassisch – Hybrid*, Springer Berlin Heidelberg, Berlin, Heidelberg (2022), URL https://link.springer.com/10.1007/978-3-662-65473-6

[Lit16] Little, Jason: *Lean Change Management: innovative Ansätze für das Management organisationaler Veränderung*, Happy Melly Express, Erscheinungsort nicht ermittelbar (2016)

[Lov21] Lovelace, Robin: Open source tools for geographic analysis in transport planning. *Journal of Geographical Systems* (2021), Bd. 23(4): S. 547–578, URL https://link.springer.com/10.1007/s10109-020-00342-2

[MG24] Mejía-Granda, Carlos M.; Fernández-Alemán, José L.; Carrillo-de Gea, Juan M. und García-Berná, José A.: Security vulnerabilities in healthcare: an analysis of medical devices and software. *Medical & Biological Engineering & Computing* (2024), Bd. 62(1): S. 257–273, URL https://link.springer.com/10.1007/s11517-023-02912-0

[Pfl95] Pfleeger, Shari Lawrence: Experimental design and analysis in software engineering. *Annals of Software Engineering* (1995), Bd. 1(1): S. 219–253, URL http://link.springer.com/10.1007/BF02249052

[Pod03] Podsakoff, Philip M.; MacKenzie, Scott B.; Lee, Jeong-Yeon und Podsakoff, Nathan P.: Common method biases in behavioral research:

A critical review of the literature and recommended remedies. *Journal of Applied Psychology* (2003), Bd. 88(5): S. 879–903, URL https://doi.apa.org/doi/10.1037/0021-9010.88.5.879

[Pre15] Preussig, Jörg: *Agiles Projektmanagement: Scrum, Use Cases, Task Boards & Co*, Nr. 270 in TaschenGuide, Haufe-Lexware, Freiburg, 1. auflage Aufl. (2015)

[Rog23] Rogers, Yvonne; Sharp, Helen und Preece, Jenny: *Interaction design: beyond human-computer interaction*, Wiley, Hoboken, New Jersey, sixth edition Aufl. (2023)

[Sar24] Sarkar, Tiyas; Moharana, Bhimasen; Rakhra, Manik und Cheema, Gagandeep Singh: Comparative Analysis of Empirical Research on Agile Software Development Approaches, in: *2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, IEEE, Noida, India, S. 1–6, URL https://ieeexplore.ieee.org/document/10522134/

[Shu08] Shull, Forrest; Singer, Janice und Sjøberg, Dag I. K.: *Guide to advanced empirical software engineering*, Springer, London? (2008)

[Sta17] Stahl, Reinhold und Staab, Patricia: *Die Vermessung des Datenuniversums*, Springer Berlin Heidelberg, Berlin, Heidelberg (2017), URL http://link.springer.com/10.1007/978-3-662-54738-0

[Sta23] Stack Overflow: Stack Overflow Survey 2023 (2023), URL https://survey.stackoverflow.co/2023/

[Sta24] Stack Overflow: Stack Overflow Survey 2024 (2024), URL https://survey.stackoverflow.co/2024/

[Tav23] Tavva, Naveen und Nguyen, Phuong-Ha: Market Insights Report (2023), URL https://de.statista.com/statistik/studie/id/102693/dokument/software-report/

[Vre02] Vredenburg, Karel; Mao, Ji-Ye; Smith, Paul W. und Carey, Tom: A survey of user-centered design practice, in: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, Minneapolis Minnesota USA, S. 471–478, URL https://dl.acm.org/doi/10.1145/503376.503460

[Wex17] Wexler, Steve; Shaffer, Jeffrey und Cotgreave, Andy: *The big book of dashboards: visualizing your data using real-world business scenarios*, Wiley, Hoboken, New Jersey (2017)

List of Figures

List of Tables

A REST routes

Users			
GET	/user/details	OAuth2	Returns the details of the logged in user
GET	/user/favoriteDashboard/	OAuth2	Returns the favorite dashboard of the logged in user
PUT	/user/favoriteDashboard/{dashboardId}	OAuth2	Sets the favorite dasbhoard for the logged in user
Teams			
GET	/team	OAuth2	Returns all teams
GET	/team/{teamId}	OAuth2	Returns info for the specified team
POST	/team	OAuth2	Creates a new team
PUT	/team/{teamid}/repos	HMAC	Overwrites the repos for which the team is responsible for
POST	/team/{teamid}/repos/{repoId}	OAuth2	Assignes a responsible team for a repo
DELETE	/team/{teamid}/repos/{repoId}	OAuth2	Cancels the assignment of a responsible team for a repository
POST	/team/{teamid}/repos/{userId}	OAuth2	Assigns a user to a team
DELETE	/team/{teamid}/repos/{userId}	OAuth2	Removes a user from the team
Repositories			
GET	/repo	OAuth2	Returns all repos
GET	/repo/{repoId}	OAuth2	Returns the specified repo
PUT	/repo/{repoId}	HMAC	Updates an existing repo

POST	/repo/	HMAC	Creates a new repo
DELETE	/repo/{repoId}	HMAC	Deletes a repo
Buildjobs			
GET	/repo/{repoId}/buildjobs	OAuth2	Returns the buildjobs from a repo
POST	/repo/{repoId}/buildjobs	HMAC	Creates a new buildjob for a repo
Metrics			
GET	/metric	OAuth2	Returns all metrics
GET	/metric/{metricId}	OAuth2	Returns the specified metric
POST	/metric	OAuth2	Creates a new metric
DELETE	/metric/{metricId}	OAuth2	Deletes the specified metric
Metricdata			
GET	/metric/{metricId}/data	OAuth2	Returns the metricdata for the specified metric
PUT	/metric/{metricId}/data	HMAC	Updates the metricdata for the specified metric
POST	/metricdata/{metricdataId}/user/{userId}	OAuth2	Assigns a metricdata to a user
DELETE	/metricdata/{metricdataId}/user/{userId}	OAuth2	Cancels assignment of a metricdata to a user
POST	/metricdata/{metricdataId}/repo/{repoId}	OAuth2	Assigns a metricdata to a repo
DELETE	/metricdata/{metricdataId}/repo/{repoId}	OAuth2	Cancels assignment of a metricdata to a repo
Dashboards			
GET	/dashboard	OAuth2	Returns all dashboards
POST	/dashboard	OAuth2	Creates a new dashboard

POST	/dashboard/{dashboardId}/ • TeamUserMetricdata/{teamId} • TeamRepoMetricdata/{teamId} • TeamBuildjobs/{teamId} • SingleMetricdata/{metricdataId} • RepoMetricdata/{repoId} • RepoBuildjobs/{repoId} • MetricMetricdata/{metricId}	OAuth2	Defines what is to be displayed on a dashboard.
DELETE	/dashboard/{dashboardId}/ • TeamUserMetricdata/{teamId} • TeamRepoMetricdata/{teamId} • TeamBuildjobs/{teamId} • SingleMetricdata/{metricdataId} • RepoMetricdata/{repoId} • RepoBuildjobs/{repoId} • MetricMetricdata/{metricId}	OAuth2	Removes the specified entry that is to be displayed on a dashboard.
GET	/dashboard/{dashboardId}/subscribe	OAuth2	Subscribes to the specified dashboard
GET	/dashboard/{dashboardId}/allMetricdata	OAuth2	Returns all metricdata from the specified dasbhoard
GET	/dashboard/{dashboardId}/allBuildjobs	OAuth2	Returns all buildjobs from the specified dashboard
Webhooks			
POST	/webhook/[...]	HMAC	Routes for webhooks from specific services can be created if needed.

Table A.1: List of all REST routes

B Questionnaire - Impact of the software

Frequency of use How often do you use the software in your day-to-day work? (If you look at it together as a team, simply add up the usage)	• Never • Less frequent • Once a week • Several times a week • Daily
If you ticked "Once a week", "Less frequent" or "Never", what stops you from using it more often?	Free text answer
How is the software integrated into your everyday life?	• I look at my dashboard/s independently. • We look at the dashboard together as a team in the daily. • Other (free text answer)
Influence on the software development process The software gives me a better overview of the development process.	• Do not agree at all • Do not agree • Neutral • Agree • Fully agree

The software helps me to react more quickly to problems (issues, complaints, build errors, errors in the source code).	• Do not agree at all • Do not agree • Neutral • Agree • Fully agree
The centralized and targeted display of information, build jobs and metrics saves me time.	• Do not agree at all • Do not agree • Neutral • Agree • Fully agree
The software has improved the way I work.	• Do not agree at all • Do not agree • Neutral • Agree • Fully agree
Concrete effects	
How has your efficiency at work changed since the software was introduced?	• Strongly deteriorated • Slightly worsened • Unchanged • Slightly improved • Greatly improved

Have the communication channels within your team changed as a result of the software?	• Strongly deteriorated • Slightly worsened • Unchanged • Slightly improved • Greatly improved
Have the communication channels to other teams changed as a result of the software?	• Strongly deteriorated • Slightly worsened • Unchanged • Slightly improved • Greatly improved
Demographic questions	
Length of service at Janitza	• <= 1 Year • >1 Year and <= 3 Years • >3 Years
Jobdescription	• Softwaredeveloper • Tester • Data Analyst • Scrum Master • Product Owner • Other (Free text answer)
Open questions Are there any other remarks or comments you would like to share?	Free text answer

Table B.1: Questionnaire - Impact of the software

C Documentation

C.1 User Manual

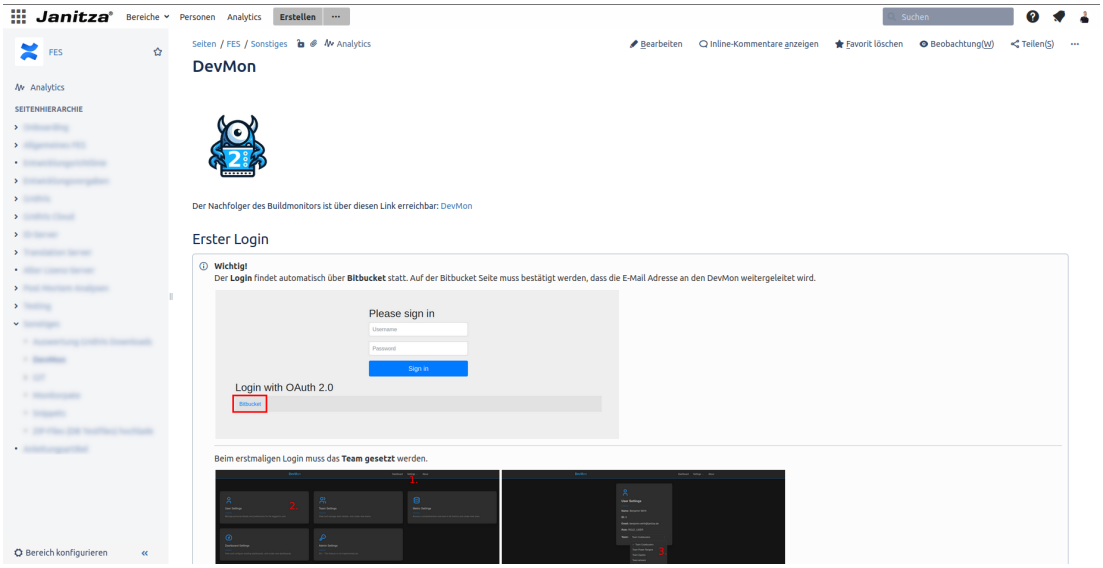

Figure C.1: Excerpt from the user manual in Confluence

C.2 REST Documentation

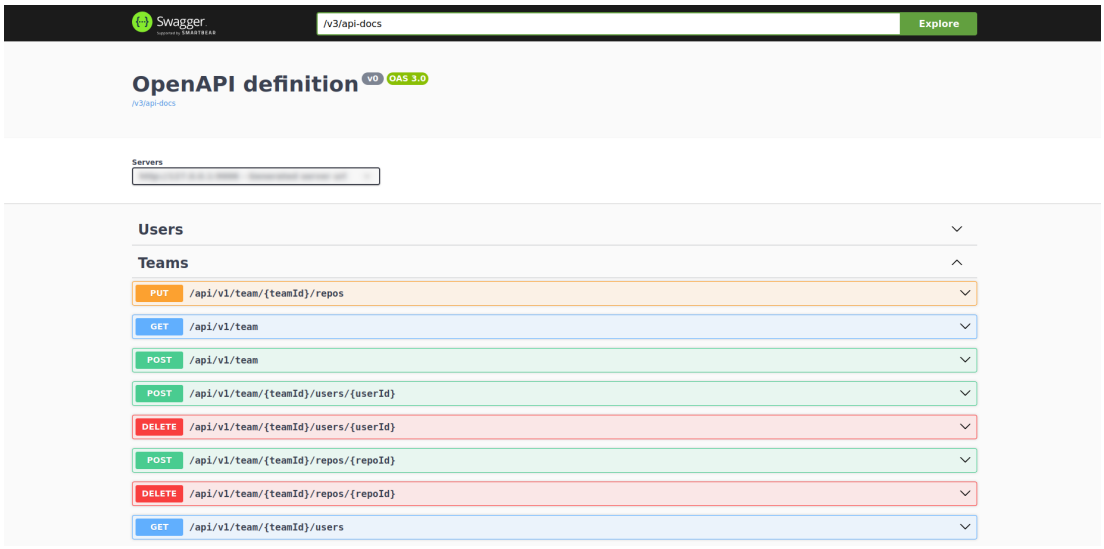

Figure C.2: Excerpt from the REST Documentation realized with Swagger

D DevMon evaluation

D.1 Functional requirements

Table D.1 presents a review of the individual user stories and shows the extent to which the "DevMon" software fulfills these requirements.

User-Story	**Evaluation**	**Fulfilled ?**
User-Story 1	• "DevMon" presents the information clearly with the help of dashboards. • The generic structure of the backend makes it possible to send a wide variety of data to the backend.	✓
User-Story 2	• Bitbucket Cloud, Janitza's build tool, was connected to "DevMon" via a webhook, allowing new build jobs to be displayed on the dashboards in real time.	✓
User-Story 3	• Where possible, a URL is stored for each piece of information, which redirects directly to the corresponding source.	✓
User-Story 4	• Dashboards can be marked by clicking on a favorite icon so that the favorite dashboard is automatically loaded when "DevMon" is called up.	✓
User-Story 5	• Users can join teams and all relevant information is automatically displayed on the dashboard of the respective team.	✓

User-Story 6	• Clicking on a metric opens a pop-up window that provides detailed information on that metric.	✓
User-Story 7	• Metrics are aggregated by team so that, for example, the coverage of Team A and Team B can be viewed separately. • This differentiation also applies to other metrics, such as open Jira tickets.	✓
User-Story 8	• Dashboards can be freely configured via the web interface according to the requirements of users and teams.	✓
User-Story 9	• The Datafetcher concept enables the integration of new data sources using simple JavaScript files. • Changes to the backend source code are not required. • No adjustments to the database are necessary either.	✓
User-Story 10	• Metrics are provided with threshold values for warning and error states. • The status of a metric, i.e. whether there is a warning or an error, is visualized using clearly assignable colors (green, yellow, red).	✓
User-Story 11	• Different calculation algorithms can be defined for metrics, which are used to evaluate the values. • For example, a distinction is made between the number of data per metric and the average of all values within a metric.	✓

Table D.1: Tabular representation of the review of "DevMon" with regard to the fulfillment of the functional requirements

D.2 Non-functional requirements

Table D.2 presents an assessment of the individual non-functional requirements. Each requirement is evaluated and briefly described.

Requirement	**Evaluation**	**Fulfilled ?**
Requirement 1	• Comprehensive documentation is available in the company's internal wiki and is accessible to all users. During creation, particular emphasis was placed on detailed descriptions and the use of illustrations. • There is also separate documentation for the REST interfaces, which is provided using Swagger.	✓
Requirement 2	• The dashboards are configurable and can be adapted to individual or team-related requirements. • They are clearly laid out and avoid an overload of information. • Appropriate colors (green, yellow, red) are used to facilitate interpretation. • Clicking on the metrics opens pop-up windows (drill-downs) that display detailed information on the respective metrics.	✓
Requirement 3	• The database has been structured and built to support a generic approach, enabling it to store and manage different types of information.	✓

Requirement 4	• PostgreSQL is a relational database that therefore supports the management of relationships between data. This allows teams, users, metrics and other entities to be efficiently linked together.	✓
Requirement 5	• The tables were deliberately designed according to a generic approach; the Metricdata table in particular is designed to be flexible. • PostgreSQL supports the *jsonb* column format, which enables the storage of JSON data and therefore offers a high degree of flexibility when managing different data types.	✓
Requirement 6	• Spring Security supports authentication using OAuth2. • Bitbucket Cloud was implemented as an OAuth consumer, corresponding to the technologies used in Janitza.	✓
Requirement 7	• Not applicable, as this requirement only applies to ready-made software solutions.	-
Requirement 8	• The frontend was implemented with React. • The backend was developed with Spring Boot. • The database is based on PostgreSQL. • All the technologies mentioned are modern and widely used frameworks and systems [Sta23].	✓

Requirement 9	• The architecture of Spring Boot promotes the use of the "controller service repository" pattern, which clearly structures the source code, clearly separates responsibilities and significantly improves overall maintainability.	✓
Requirement 10	• The business logic was tested using unit tests. • The test coverage is over 95%.	✓
Requirement 11	• Comprehensive pipelines were developed that support linting, unit tests and automatic building and deployment of the application.	✓
Requirement 12	• Not applicable, as this requirement only applies to ready-made software solutions.	-
Requirement 13	• All data is retrieved directly from the original data sources by the data fetcher and forwarded to the backend.	✓
Requirement 14	• The data fetchers are run at least once a day, but most are run once an hour. This means that the latest data is always displayed on the dashboards.	✓
Requirement 15	• Access to "DevMon" is secured by OAuth2 and HMAC. • An additional reverse proxy is also used to restrict access to predefined IP ranges.	✓

Requirement 16	• The URLs to almost all data are saved, which means that the source of the information can always be traced. There are only a few exceptions.	✓
Requirement 17	• All information is stored with concise titles and detailed descriptions.	✓
Requirement 18	• The data for each metric can also be retrieved via a REST route, with the response being provided in JSON format.	✓
Requirement 19	• All data is stored on the same server.	✓

Table D.2: Tabular representation of the review of "DevMon" with regard to the fulfillment of the non-functional requirements

E Questionnaire results

1. Wie häufig nutzt du den 'DevMon' im Arbeitsalltag? (Falls ihr gemeinsam im Team draufschaut rechnet einfach die Nutzung zusammen)

17 Antworten

Figure E.1: Results of the first question of the questionnaire

2. Falls du "Einmal pro Woche', "Seltener", oder "Nie" angekreuzt hast, was hält dich davon ab, ihn öfter zu nutzen?

4 Antworten

ID ↑	Name	Antworten
1	anonymous	Ist noch nicht in den Workflow übergegangen.
2	anonymous	[redacted]
3	anonymous	Gewohnheit da regelmäßig draufzuschauen
4	anonymous	Wird im Team nicht so häufig benötigt, da aktuell hauptsächlich lokal gearbeitet wird.

Figure E.2: Results of the second question of the questionnaire - Personal data or internal company information has been made unrecognizable

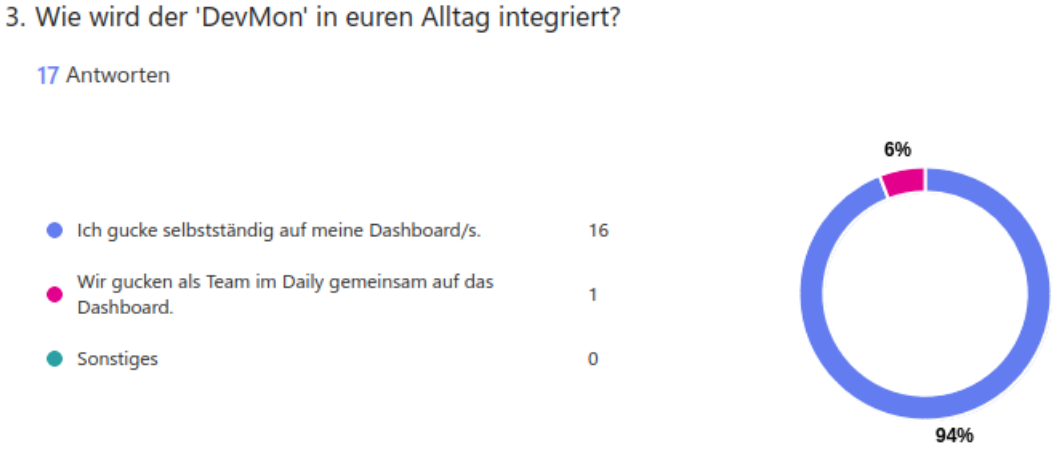

Figure E.3: Results of the third question of the questionnaire

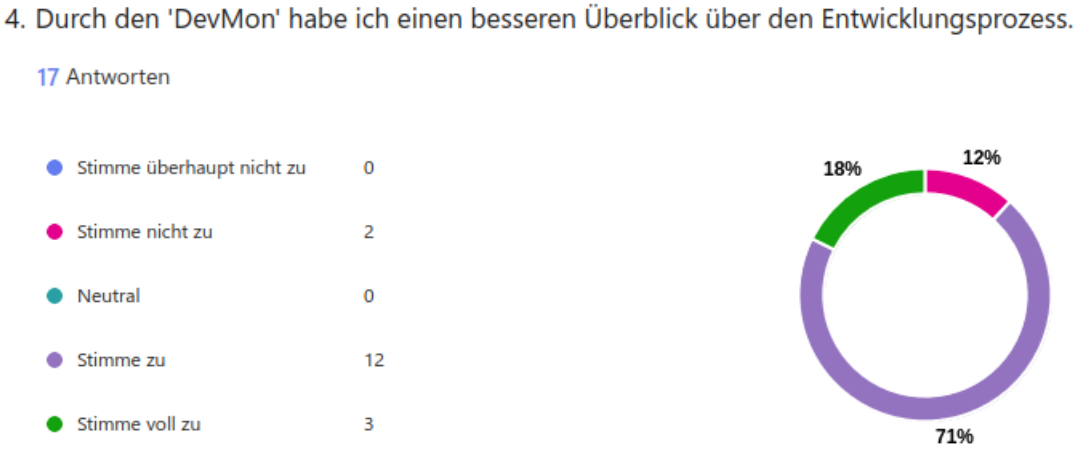

Figure E.4: Results of the fourth question of the questionnaire

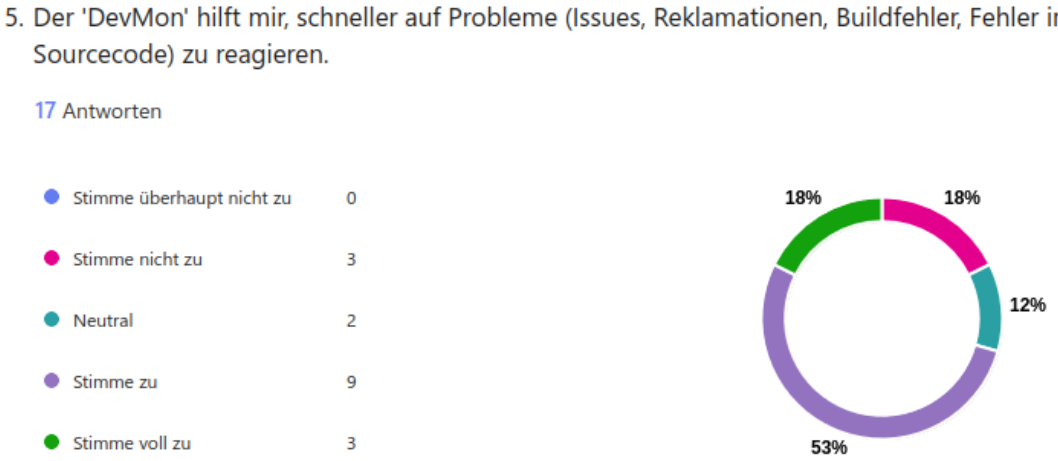

Figure E.5: Results of the fifth question of the questionnaire

6. Die zentrale und zielgerichtete Darstellung der Informationen, Buildjobs und Metriken spart mir Zeit.

17 Antworten

Stimme überhaupt nicht zu	0
Stimme nicht zu	1
Neutral	2
Stimme zu	6
Stimme voll zu	8

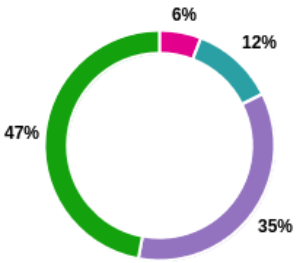

Figure E.6: Results of the sixth question of the questionnaire

7. Der 'DevMon' hat meine Arbeitsweise verbessert.

17 Antworten

Stimme überhaupt nicht zu	0
Stimme nicht zu	2
Neutral	7
Stimme zu	6
Stimme voll zu	2

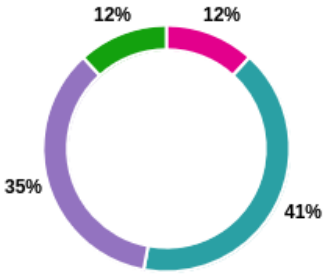

Figure E.7: Results of the seventh question of the questionnaire

8. Wie hat sich deine Effizienz bei der Arbeit seit der Einführung vom 'DevMon' verändert?

17 Antworten

Stark verschlechtert	0
Etwas verschlechtert	0
Unverändert	8
Etwas verbessert	7
Stark verbessert	2

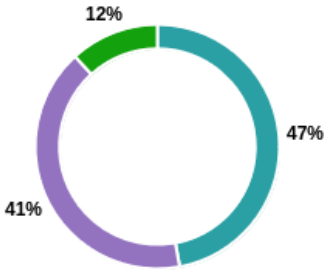

Figure E.8: Results of the eighth question of the questionnaire

9. Haben sich die Kommunikationswege innerhalb deines Teams durch den 'DevMon' verändert?

17 Antworten

Stark verschlechtert	0
Etwas verschlechtert	0
Unverändert	13
Etwas verbessert	4
Stark verbessert	0

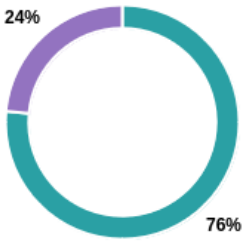

Figure E.9: Results of the ninth question of the questionnaire

10. Haben sich die Kommunikationswege zu anderen Teams durch den 'DevMon' verändert? (z.B. im Bezug zum Monitor-Paten)

17 Antworten

Stark verschlechtert	0
Etwas verschlechtert	0
Unverändert	8
Etwas verbessert	4
Stark verbessert	5

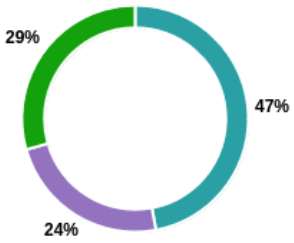

Figure E.10: Results of the tenth question of the questionnaire

11. Betriebszugehörigkeit zu Janitza

17 Antworten

<= 1 Jahr	5
> 1 Jahr und <= 3 Jahre	7
> 3 Jahre	5

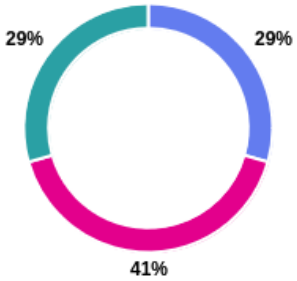

Figure E.11: Results of the eleventh question of the questionnaire

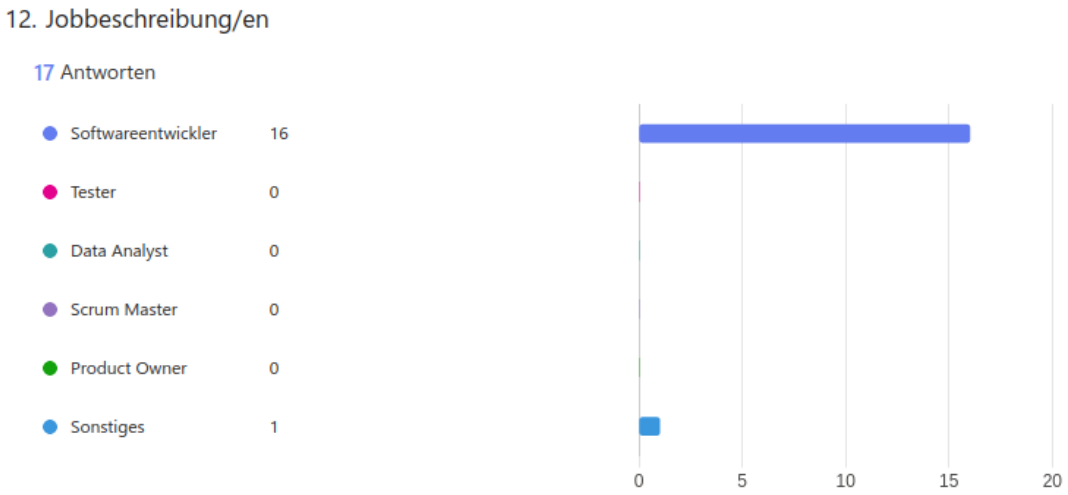

Figure E.12: Results of the twelfth question of the questionnaire

13. Gibt es weitere Anmerkungen oder Kommentare, die du mir mitteilen möchtest?

7 Antworten

ID ↑	Name	Antworten
1	anonymous	Ich würde mir noch die zyklischen Aufgaben als eine extra Matrik wünschen.
2	anonymous	Ich finde, der devmon ist auf einem guten Weg, um eine bessere Übersicht und detaillierte Einsichten zu verschaffen. Gerade die Aufteilung schafft bessere Ordnung
3	anonymous	Der Fragebogen wird sich ziemlich negativ lesen, aber so ist es nicht gemeint. Ich finde den neuen DevMon super! Es wird einerseits einige Zeit dauern, bis er sein Potenzial entfaltet. [illegible]
4	anonymous	Ein gutes Tool, es fehlt einfach nur noch die Gewohnheit da regelmäßig draufzuklicken
5	anonymous	Code-Reviews und Pipeline-Fehler bekomme ich normaler weise über E-Mail-Notifications mit. Der Aufwand wegen/für Monitor-Pate wird durch den DevMon reduziert/verbessert. Auch wenn es keine große Veränderung meines Entwickler-Alltags ist - super Projekt!
6	anonymous	"DevMon" = cooler Name
7	anonymous	Durch das große Wachstum der Software-Entwicklungsabteilung in den letzten Jahren, ist die Aufteilung des Build-Monitors auf die einzelnen Teams ein notwendiger Schritt. Das zeitraubende Suchen der "Verantwortlichen" durch den Monitorpaten entfällt weitestgehend.

Figure E.13: Results of the thirteenth question of the questionnaire - Personal data or internal company information has been made unrecognizable