Tulio Navarro Tutui

Contribuição ao desenvolvimento de software de uma ferramenta de Data Science

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Produção da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Produção (PRO)

Supervisor: Prof. Dr. Marcelo Schneck de Paula Pessoa

São Paulo, SP

2023

Acknowledgements

First and foremost, I would like to thank every person that helped me in any way to get to where I am professionally, and consequentially be selected between many others to pursue my double degree, and the internship in Dataiku.

I would like to express my utmost gratitude to my esteemed schools, both École Centrale Lyon and University of São Paulo, as well as the exceptional professors for providing me with the opportunity to succeed in this step of my career. Throughout my educational experiences, I have been equipped with practical skills, knowledge, and confidence that have played a pivotal role in shaping my professional journey. The support and guidance I received from my professors were truly remarkable, as they not only imparted academic wisdom but also instilled in me a sense of passion and dedication towards my chosen field. Their mentorship and encouragement motivated me to push beyond my limits and achieve excellence in my internship project. I am deeply thankful to my schools for fostering an environment of learning and growth, where I was encouraged to explore, ask questions, and apply classroom concepts to real-world scenarios.

A special thanks is due to Professor Marcelo Pessoa, who oriented me trough this intense process. With his guidance, it was possible to complete this process in time, without a loss in quality or in the whole learning experience. Marcelo's knowledge and previous works were a fundamental piece in constructing this document, so I truly appreciate his support.

I also extend my heartfelt gratitude to Dataiku, the remarkable company where I had the privilege to intern, for providing me with an outstanding and welcoming environment during my internship. From the moment I stepped through the doors, I was warmly greeted by the team and made to feel like an integral part of the organization. The support and encouragement I received from my colleagues, team, and supervisors were instrumental in making my internship a truly transformative experience. Furthermore, the challenges entrusted me, I was allowed to apply the knowledge gained from my academic studies, and I feel better prepared to take on future challenges in the field of software development and data.

It is also important to highlight the presence of my internship manager, Theo Joubert, who was present daily to assist me. He was an essential piece to my learning experience and in helping me deliver the desired results. His approach to following my development and teaching me was very effective, allowing me to over-achieve in the relevant aspects of the job.

Finally, this would have never been possible if not for the help of my friends and

family, who supported and gave me incentive to pursue my degree outside my home country and strive throughout the difficulties I had, being able to in a short, yet fruitful period of time finish this paper with the imposed time constraints.

Abstract

This thesis aims to provide an analysis of the software development processes of Dataiku, a leading company in the field of data science and machine learning platforms. After introducing the company and its product, the study presents the ideal landscape of practices in software development and structure models for its life cycle.

The primary objective is to scrutinize the existing development practices, focusing on organizational structure and adherence to methodologies, with the ultimate goal of verifying if there are areas for improvement. This is done by employing a case study of two features in the platform (DSS), emphasizing aspects such as team organization, communication channels, and the application of agile principles.

The results reveal a well-structured implementation process at Dataiku, characterized by the adoption of agile methodologies, including Scrum and Kanban, and short iterations to guarantee continuous improvement. Teams exhibit strong collaboration, iterative development cycles, and a commitment to delivering high-quality software. However, the analysis also uncovers areas with potential for enhancement, such as communication efficiency, testing structure and preparation for emerging changes in the technology/programming industry.

Keywords: Software development life-cycle, SDLF, IT management, programming management, programming good practices, agile programming.

Resumo

Esta tese tem como objetivo fornecer uma análise dos processos de desenvolvimento de software da Dataiku, uma empresa líder na área de plataformas de ciência de dados e aprendizado de máquina. Após apresentar a empresa e seu produto, o estudo apresenta o panorama ideal de práticas de desenvolvimento de software e modelos de estruturação do seu ciclo de vida.

O objetivo principal é analisar as práticas de desenvolvimento existentes, focando na estrutura organizacional e na adesão a metodologias, com o objetivo final de identificar se existem áreas de melhoria. Para tal, recorre-se a um estudo de caso de duas funcionalidades da plataforma (DSS), enfatizando aspectos como a organização das equipes, os canais de comunicação e a aplicação de princípios ágeis.

Os resultados revelam um processo de implementação bem estruturado na Dataiku, caracterizado pela adoção de metodologias ágeis, incluindo Scrum e Kanban e iterações curtas para garantir melhoria contínua. As equipes demonstram forte colaboração, ciclos de desenvolvimento iterativos e um empenho em fornecer software de alta qualidade. No entanto, a análise também revela áreas com potencial de melhoria, como a eficiência da comunicação, a estrutura de testes e a preparação para as mudanças emergentes no sector de tecnologia/programação.

Palavras-chave: Ciclo de vida do desenvolvimento de software, gerência de TI, gerência da programação, boas práticas de programação, programação ágil.

List of Figures

Contents

1 Introduction

Software development is an ever-evolving discipline that lies at the heart of the digital age, driving innovation and shaping the technological landscape. The process of creating software applications, systems, and tools has seen a remarkable transformation over the decades, marked by constant adaptation to the changing needs of industry, technology, and society.

The evolution of software development methods and frameworks is a fascinating journey that mirrors the rapid advancement of technology itself. In the early days of computing, software development was an arcane art, often characterized by ad-hoc coding and limited documentation. However, as the demand for more sophisticated software grew, developers recognized the need for systematic approaches to streamline the development process.

One of the first important marks of this evolution was the advent of structured programming methodologies in the 1960s. This approach introduced the concept of breaking down complex software tasks into smaller, more manageable modules, allowing for greater organization and maintainability. Simultaneously, the Waterfall model emerged, advocating a linear, step-by-step approach to software development, where each phase had to be completed before the next began.

Later on, in the rapid expansion of the software industry in the following decades, the limitations of rigid, sequential models became apparent. This realization paved the way for agile methodologies, which prioritized collaboration, flexibility, and iterative development. Agile introduced a fundamental shift in the software development landscape, allowing for adaptability to changing requirements and faster time-to-market.

The evolution did not stop there. The rise of object-oriented programming (OOP) ushered in a new paradigm where software was designed around the concept of objects and classes, promoting reusability and modularity. Concurrently, the open-source movement and the proliferation of programming languages and libraries provided developers with a rich tapestry of tools and resources to choose from.

Recently, in the past two decades have witnessed the ascendancy of DevOps, which seeks to bridge the gap between software development and operations, streamlining deployment, and enhancing collaboration between development and IT teams. Additionally, containerization and orchestration technologies, such as Docker and Kubernetes, have revolutionized the way software is packaged and deployed, enabling scalability and efficient resource utilization.

Today, software development is shaped by a diverse ecosystem of methodologies, including Scrum, Kanban, and Extreme Programming, as well as an array of tools and frameworks tailored for various development needs. The integration of machine learning, artificial intelligence, and cloud computing further expands the possibilities for software development, allowing for intelligent, scalable, and data-driven applications.

On the other hand, even tough the evolution of these methods and frameworks has undeniably brought about profound improvements in the efficiency and quality of software development, it has also introduced a unique set of challenges and complexities that demand careful consideration. As we progress in our exploration, it's imperative to pivot our focus towards the main difficulties and issues faced by software developers and the orchestration of software development teams.

One of the central challenges in contemporary software development lies in the complexity of modern systems. As software applications and systems have grown in scale and sophistication, so too have the intricacies of managing their development. Coordinating numerous modules, integrating multiple technologies, and ensuring the compatibility of different components can lead to formidable challenges. The need for efficient communication, seamless collaboration, and meticulous project management becomes paramount in this context.

For that matter, software development teams are composed of diverse individuals with unique skills and backgrounds, and the effective organization of such teams is a critical factor in the success of any project. Communication gaps, differing coding styles, and varying interpretations of project requirements can all lead to misalignments within the team, which can, in turn, result in delays, misunderstandings, and quality issues.

The balance between adhering to established methodologies and embracing flexibility is another challenge. While agile methodologies emphasize adaptability and incremental development, the overarching frameworks and processes in an organization may demand compliance with more traditional, linear methodologies. Striking the right balance between structure and agility is a perpetual challenge that developers and team leads must navigate.

Moreover, the ever-expanding array of technologies, libraries, and frameworks presents developers with the dilemma of choice. Deciding on the most appropriate tools and technologies for a project, along with ensuring they are well-maintained and up to date, can be a complex undertaking. The software development landscape is further complicated by the need to accommodate legacy systems and technical debt, which can weigh down development efforts and compromise the quality of the software.

Security remains a persistent concern, with the ever-present threat of data breaches, cyberattacks, and vulnerabilities. Ensuring that software is developed with robust security measures in place is an ongoing challenge, necessitating constant vigilance and adaptation

in response to evolving threats.

As we delve deeper into this exploration, the author will scrutinize these challenges and issues in greater detail. How they manifest in various phases of the software development life cycle and explore strategies and best practices to address them are also aspects considered. Additionally, we will examine the role of project management methodologies, collaboration tools, and cultural aspects in fostering effective software development teams.

Considering the evolution, the constant changes and improvements brought to this medium over the years, this paper seeks to bring forth the issues presented in this process, through a real case scenario, and the past experiences of the author, demonstrating how a large scale company tries to deal with challenges while evaluating these solutions, comparing them to the history of software development and to other contemporary companies.

By addressing these difficulties and issues, the thesis provides insights that will empower software developers and teams to navigate the complexities of modern software development successfully. This exploration underscores the notion that the evolution of software development methods and frameworks is an ongoing process, driven by the need to overcome these challenges and deliver high-quality, innovative software solutions.

1.1 Motivation

The main motivations for this work are my previous experiences as a software developer and the double degree program that I was privileged to enroll.

For the experiences, I have worked as a freelancer for companies such as Nestlé, creating a social network with an external team, for small NGOs, in a consulting company, where we developed platforms for healthcare services, and in two startups, where I was able to build products from the ground up.

In each of these scenarios, I went through a different team structure, interacted with different people, used different tools and frameworks, but in all of those cases it was possible to achieve or surpass what was required. That said, every organization had its purpose and function, being suited for the demands of the client. Being fully immersed in these diverse contexts, helped me have a better grasp of the process of developing software, and what practices, tools and frameworks deliver the best results for a given situation.

As for the double degree program, I got the chance to enroll in this partnership between the Polytechnic School of the University of São Paulo and École Centrale Lyon (ECL). This program allowed me to follow the production engineering course in São Paulo and a generalist course with a specialization in computer engineering in Lyon, a fact that broadened my vision of the engineering field.

Basically, after studying three and a half years in the University of São Paulo, I

moved to Lyon, where I studied 1 year of generalist engineering and 1 year of specific field of the student's choice. For the students departing in the second half of their fourth year in Brazil, Centrale Lyon provides them with the opportunity to take the courses of its second and third years. The curriculum abroad really emphasizes the ability of an engineer to work in diverse fields and projects, making it a perfect fit for my experiences in different areas. All the classes offered abroad were in French, in the same environment as the French native students, a fact that allowed a profound cultural and language exchange.

That said, as I chose computer engineering as my specialization, in the end of the studies the courses went more deeply into computer science, preparing better for the end of studies internship, which was as a software developer at Dataiku, one of the most significant players in the Data Analysis/Science field. The background that I had, with this experience, were the building blocks for the examples and analysis further presented in the paper.

Because of this internship, the language chosen for this work is English. To finish my studies in ECL, it was mandatory to create a report about the internship, given that a lot of people in the company did not speak French, this report was written in English. As some extracts were used here, and to avoid losing information in the translation, this paper was also written in English

1.2 Dataiku and the internship

1.2.1 Status and size

Dataiku was founded in 2013 by Florian Douetteau, Marc Batty, Thomas Cabrol, and Clément Stenac (CAI, 2020). The company originated in France, and its founders aimed to address the growing need for a collaborative and efficient platform for data science and machine learning. They envisioned a solution that would enable data professionals and business analysts to work together seamlessly, breaking down silos and accelerating the process of turning raw data into valuable insights.

The company has achieved significant growth and had secured several rounds of funding to support its expansion. A steady competition has always been present during the startup's history, some of the more formidable rivals include Databricks, which raised $1.6 billion in August 2021, and DataRobot, which secured $300 at a $6.3 billion valuation in July 2021 according to TechCrunch (WIGGERS, 2022).

Today, it has over 10000 employees and had raised over $246 million in funding, with a valuation of $3.7 billion (DATANAMI, 2022). The company has also established a strong presence in the market, with over 450 customers worldwide, including major companies such as General Electric, Unilever, and BNP Paribas. Dataiku's platform had

been recognized as a leader in the Gartner Magic Quadrant for Data Science and Machine Learning Platforms for four consecutive years, highlighting its industry leadership and innovative solutions. Overall, Dataiku's significant funding, customer base, and industry recognition indicate that it is a well-established and successful player in the data science platform market.

1.2.2 Location and presence

The company has offices across the globe, with headquarters in New York City and offices in Paris, London, Munich, Sydney, Singapore, and Tokyo. In addition to these locations, Dataiku had expanded its presence in North America with a new office in San Francisco. As for its clients, they are present in various industries, including healthcare, finance, retail, and manufacturing. These customers have a strong presence in Europe and North America. Overall, Dataiku had established a strong global presence, with a diverse customer base that spanned various industries and geographies.

The Paris office, where then internship took place, is located at Rue de Bercy 201-203, on the 5th to 7th floor of a commercial building, right in front of Gare de Lyon. The R&D team which I have integrated works at the 6th floor.

As one of the largest offices of the company, it is built to nourish collaboration and productivity. To achieve this feat, it is different from standard office spaces, being mostly open, in a way that everyone sitting or walking through this space can see what one is doing and easily communicate with this person.

In the following photo 1, the space where the HR team work is shown, exemplifying the characteristics just mentioned.

Figure 1 – Paris Office

1.2.3 Mission, vision, and future plans

Dataiku's mission is to enable organizations to transform their businesses through the power of data. The company's platform provides a collaborative and scalable environment for data teams to build and deploy machine learning models and drive innovation across their organizations. Dataiku's goal is to democratize data science and empower users of all skill levels to leverage data to drive business outcomes.

Dataiku's core values include innovation, collaboration, and diversity. The company encourages experimentation and creativity, allowing employees to challenge themselves and push boundaries. Dataiku also values collaboration, recognizing that effective data science requires the integration of multiple perspectives and skill sets. Additionally, the company values diversity and inclusion, recognizing that a diverse workforce leads to better outcomes and a more positive workplace culture.

Looking to the future, Dataiku plans to continue to innovate and expand its platform. The company's vision is to be the go-to data science platform for businesses around the world, enabling them to unlock the full potential of their data. Dataiku plans to invest in product development, including features such as AI-assisted data preparation and automated machine learning, to continue to make data science more accessible to users of all levels. The company also plans to expand its global footprint, with a focus on developing markets, to enable businesses worldwide to drive innovation through the power

of data.

1.2.3.1 Dataiku Stories

Thinking in this direction, in the last few years Dataiku acquired a company called Powerslide, which worked on a similar product to DSS, but focused more on the data storytelling, than on the analysis itself. From this, Dataiku Stories was born, which is a new product under the company's umbrella based on Powerslide.

From this point on, this fact helped the company shape some of its goals so that both products are more aligned in terms of features, making data, charts, and analysis flow seamlessly between them. Moreover, it is also important that the company makes sure that each one of these has its own scope and objective.

1.2.4 Culture

At Dataiku, collaboration is central to the culture. The company fosters teamwork and cross-functional collaboration, bringing together individuals from diverse backgrounds such as data scientists, engineers, designers, and business professionals. This collaborative environment encourages open communication, knowledge sharing, and collective problem-solving, nurturing creativity and innovation. From the beginning of the internship this could be seen in the onboarding, where the activities encouraged interaction between the group.

Another point present in its culture is Innovation. The company encourages employees to push boundaries, explore new ideas, and continually seek innovative solutions. They empower individuals to take ownership of their projects and experiment with cutting-edge technologies. Dataiku also emphasizes continuous learning, providing opportunities for professional growth and staying abreast of the latest advancements in the field.

Diversity and inclusion are also core pillars of Dataiku's culture. Recognizing the importance of diverse perspectives, Dataiku actively promotes a diverse and inclusive work environment. They strive to create a culture that embraces different backgrounds, experiences, and ideas. This commitment to diversity is evident in their recruitment practices, employee resource groups, and inclusive policies, fostering a strong sense of belonging among employees. It is impressive the different backgrounds that I got to know during my internship. Above all else, it was great to see how everyone is respected and valued in their teams and in the company as a whole.

Dataiku places a strong emphasis on work-life balance and employee well-being. They understand the significance of maintaining a healthy work-life balance for long-term productivity and employee satisfaction. Dataiku supports flexible work arrangements, remote work options, and offers a range of employee benefits to support well-being.

They organize team-building activities, wellness initiatives, and provide opportunities for personal growth and development.

Transparency is also present in all aspects of the company, which is of utmost significance in the realm of data-driven decision-making and analytics. By embracing transparency, Dataiku ensures that all stakeholders, including data scientists, analysts, and business users, can collaborate seamlessly and gain a comprehensive understanding of the data they are working with. This accessibility promotes trust, accountability, and reproducibility, making it easier to detect and rectify potential biases or errors in data analysis. This is also very present internally, where we had frequent calls where the C-level employees presents us with the results of the period, future and previous goals (and if they were achieved or not), the main client cases for the period, etc.

In summary, Dataiku's culture is characterized by collaboration, innovation, diversity, and employee well-being. By fostering these values, Dataiku creates a vibrant and inclusive work environment that empowers employees to excel, deliver exceptional results, and make a meaningful impact in the data science industry.

1.2.5 Teams and R&D

As Dataiku currently has more than 1200 employees, to organize all those talented workers, teams had to be structured. There are many teams in the company, including the standard finance, marketing, Human Resources, research and development, cloud, etc.

As a product and data oriented company, technology related teams tend to have a large amount of people, for this reason there are many subteams under the CTO. In the image 2 below, there is an overview of the themes and teams related to engineering, programming, as well as the product itself in our company:

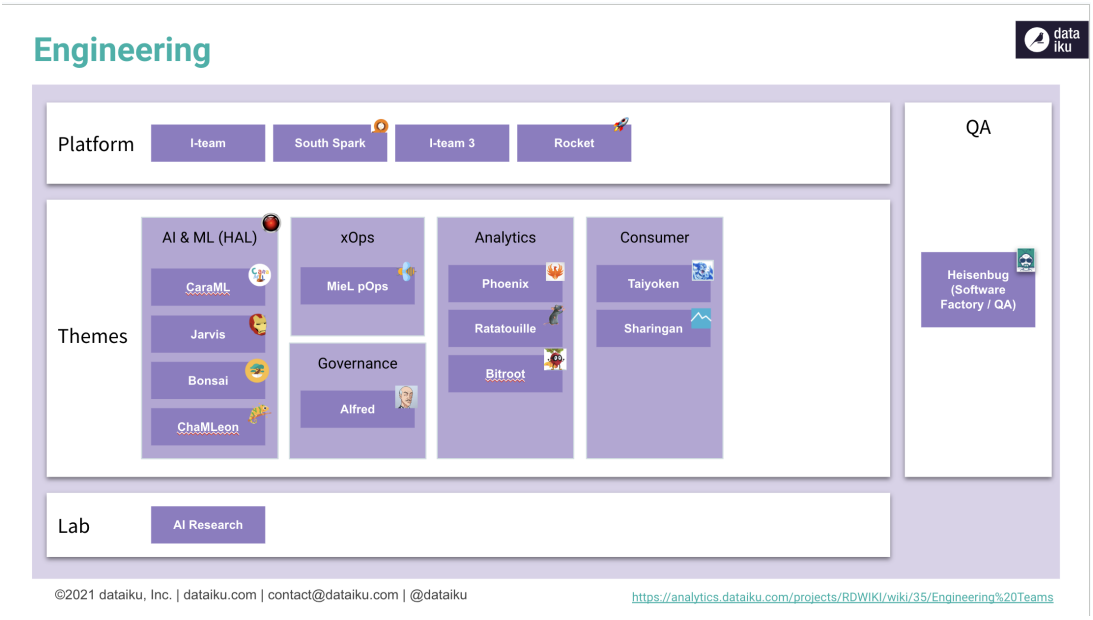


Figure 2 – R&D team

In the case of my team "Tayoken", we are under Consumer theme umbrella, and we focus on charts, dashboards, and workspaces. Taking some information from Workday's (the platform where the work related information of the company is presented) organizational charts, I joined some images to create visualization 3 as follows:

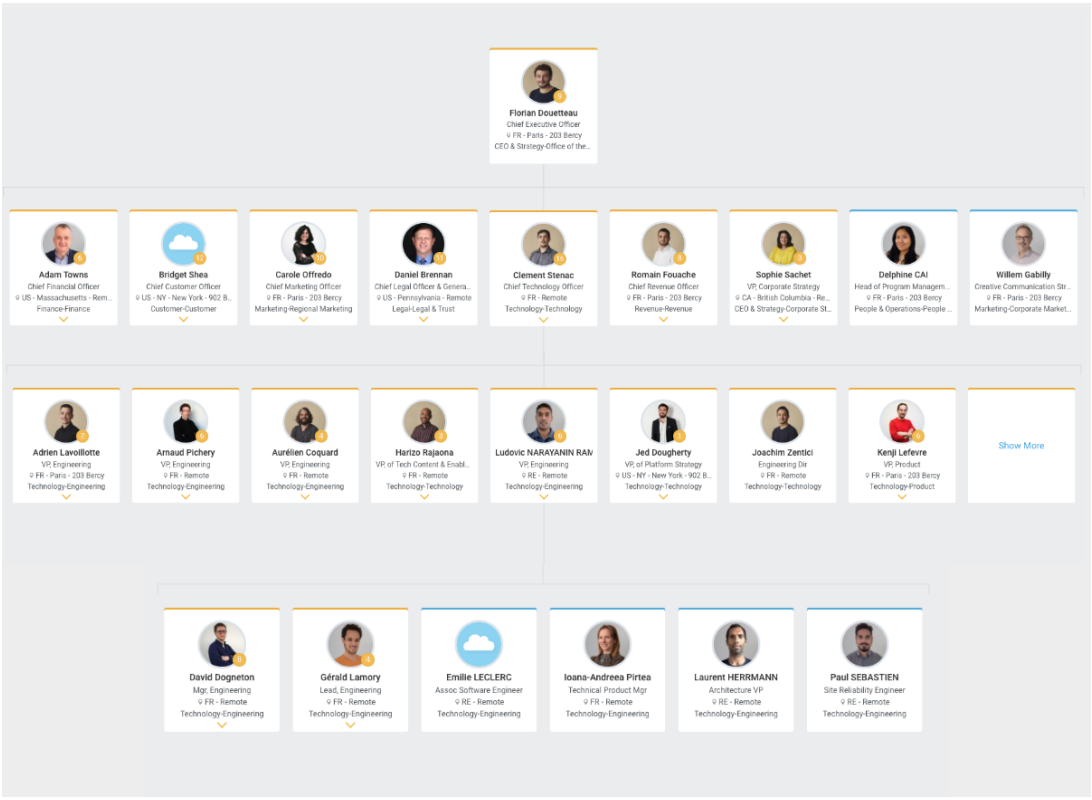


Figure 3 – Dataiku's organization structure under the CTO

Here we can see the people that make this structure possible. In the daily activities

of our team, interactions happen mostly with Ludovic (responsible for the consumer theme: DSS and Stories), Iona (the technical product manager), David (Tayoken's engineering manager) and with other members of the team itself.

1.2.6 Internship mission

To understand the mission of the internship, it is first relevant to point out the various technologies, languages, and frameworks used in each and every feature of the product:

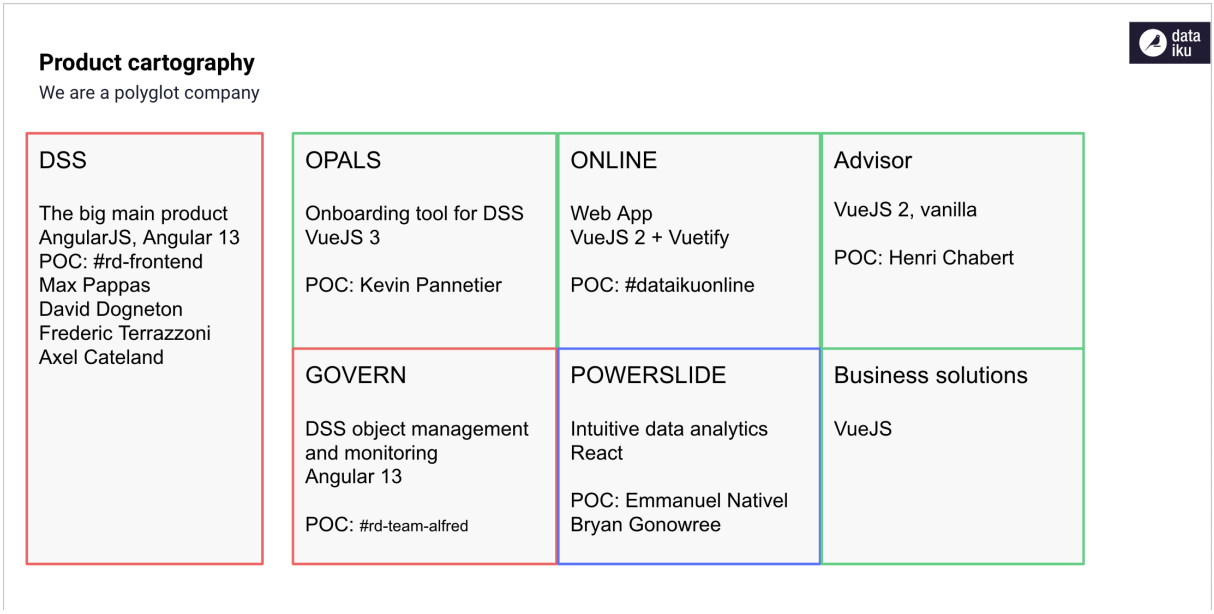


Figure 4 – Dataiku's product cartography

As previously mentioned, the team I was part of worked on DSS, so on the left side of the figure 4 are the technologies that I interacted the most with. From time to time, it was necessary to chance small things and understand how the back-end worked, which meant knowing Java was also important.

That being said, the mission of the internship can be divided in three main steps:

1. Get familiar with Dataiku, its machine learning capabilities, model explanation and responsible AI techniques, the D3.js, ECharts libraries, etc.
2. Propose other possible enhancements and help prioritize them.
3. Implement what was previously proposed.

During my first weeks, I deepen my knowledge of the 6 pillars of the platform:

- Data preparation: Dataiku DSS allows users to clean, transform, and prepare their data for analysis using a range of built-in tools and integrations.

- Data visualization: The platform includes a range of visualization tools that enable users to explore and visualize their data, allowing them to gain insights and identify patterns.

- Machine learning: Dataiku DSS includes a suite of built-in machine learning algorithms, as well as the ability to integrate with external libraries, to enable users to build and deploy machine learning models.

- Collaboration: Dataiku DSS provides collaboration features that enable teams to work together on projects, share insights, and track changes.

- Deployment: The platform includes a range of deployment options, enabling users to integrate their models into their existing IT infrastructure and deploy them at scale.

- Governance and security: Dataiku DSS provides advanced governance and security features, enabling users to control access to their data and ensure compliance with regulations.

With this in mind, I was able to better design and plan the charts and features of the platform, considering that I knew the different user profiles and use cases offered for each of them.

To be more precise about my mission, it was to add a radar chart to DSS. That said, I had no previous experience with Angular, AngularJS, and ECharts (the chosen library to do the chart), so part of my task was to study those technologies and be capable of using them to achieve my final goal.

Even if the objective of my internship is what was stated above, due to the speed I was able to finish this task, other tasks were proposed to me: create the sankey chart and create the gauge chart. Both of these new activities will be further detailed in their own section, with the radar. Briefly, since those new visualizations were not specified by the manager and the TPM beforehand, part of the job was to research, benchmark and describe how the feature will be implemented, which was very interested since it expanded the scope of the internship beyond the development and programming themselves.

Furthermore, during the summer break - around august -, most of the team was on vacations, so there was not a lot of people to verify features that were finished. With that in mind, I was allowed to verify some minor bug fixes, to make sure they fixed the issues were fixed when they were shipped to production, and approve a simple pull request, checking the code for mistakes.

1.3 Objectives

Considering the motivation and the context presented beforehand, this thesis aims to contribute to the field of software development by bringing to light established concepts and foundation to ensure the quality of the product. Then, describe how those were applied in a specific case, where the author realized one of his internships, and how successfully the company was in this implementation. Finally, some aspects of the software's life cycle will be brought up, as well as how this aspect is affected by the software development process.

To achieve this result, smaller milestones were created:

- Present Dataiku, the company used in the case study.
- Describe and explain how DSS, the company's platform, work and its main functions;
- Review the objectives as well as the concepts and fundamentals of software development;
- Explain the rationale for how software is developed in Dataiku;
- Investigate decision-making processes in the business;
- Explain and demonstrate the tools used in the company;
- Go in detail about what was developed in the case;
- Conclude if Dataiku's process is accomplishing and following what it's expected to.

2 Dataiku's main product: DSS

2.1 Functions and purpose

Dataiku main product is DSS (Data Science Studio), a collaborative data science platform that enables data teams to build and deploy predictive models and machine learning applications at scale. With Dataiku DSS, users can easily connect to various data sources, preprocess data, build models using a variety of machine learning techniques, and deploy models into production. The platform also includes features such as automated machine learning, data visualization, data governance, and version control, which help teams work more efficiently and effectively. Overall, Dataiku DSS streamlines the data science workflow and empowers organizations to turn data into insights and actions.

Dataiku DSS is designed to be accessible for both technical and business-oriented clients through its intuitive and user-friendly interface. The platform offers a visual flow-based interface that allows users to drag and drop various components to build data pipelines, perform data analysis, and develop predictive models without the need for programming. Additionally, Dataiku DSS provides built-in machine learning algorithms, automated machine learning, and a data preparation library that enables users to perform data cleaning, feature engineering, and other data preparation tasks without advanced technical skills.

For more technical users, Dataiku DSS offers a code environment that allows users to write and execute their own code in Python, R, or SQL, which will be neatly integrated with the rest of the process created in the native DSS environment. The platform also supports version control for code and models, which enables users to collaborate, track changes, and revert to previous versions.

Furthermore, Dataiku DSS includes collaboration features such as sharing projects, notebooks, and dashboards, enabling team members to work together and share insights easily. Other than that, Dataiku DSS provides a suite of deployment options, allowing businesses to integrate models into their existing IT infrastructure easily. Overall, Dataiku DSS offers a flexible and accessible platform that caters to the needs of both technical and business-oriented clients.

2.2 Homepage

Now, diving deeper into some of the main pages that a user will face when using DSS, starting with the first page encountered after logging in, the homepage, shown in

figure 5.

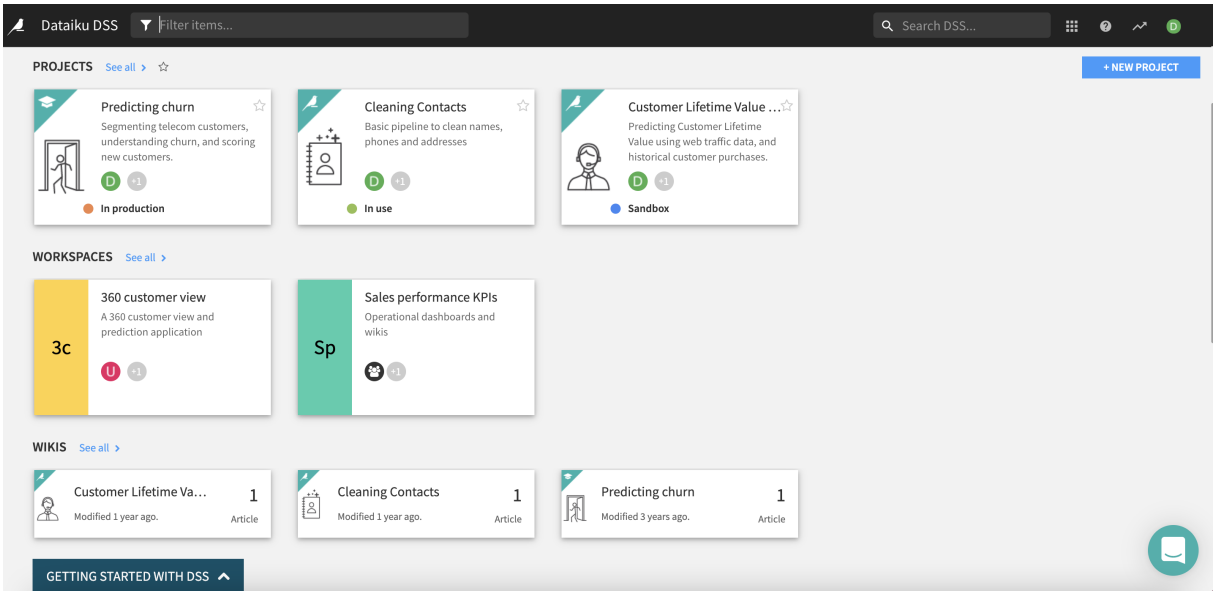


Figure 5 – DSS's homepage (DATAIKU, 2023e)

There is not a lot to explain in this page, as it is basically a list of things that can be accessed or configured by the given user:

- My Items: This section provides quick access to your personal items, such as projects, folders, workspaces, and wikis.
- Projects and Project Folders View: As a data scientist or designer, you'll see the projects you're currently working on. This section allows you to navigate directly to the projects and objects within them.
- Workspaces: If you're a data consumer without project access, the homepage takes you to the Workspaces page. Workspaces provide easy access to shared assets, allowing users to explore other user's work without having edit access, and even analyze and use their data in their work (if that is allowed by the owner).
- Getting Started With DSS Panel: This panel offers essential information for new users, including guidelines, procedures, organization details and access to the software's documentation.
- Changing the Order of Sections on the Homepage: You can customize the layout of your homepage by rearranging these sections according to your preferences.

The search bars on the top are also an important part of DSS's homepage. They allow the user to search for projects, workspaces, wikis and even for information that are contained inside those items. Once a user has been invited to many projects, and has a

lot of his own, this can come in handy. The search bar on the right is also functional to help users learn more about DSS, because it allows them to search for topics in DSS's documentation.

2.3 Project page

When the desired screenshot has been selected, the following Screenshot 6 will be presented. It is an example of the project page in the developer instance of the platform, running locally. In the picture, most of the features of the platform can already be identified: Under "Flow", the "DATASET" tab, the "RECIPES" tab - where one can treat and filter data, and the "MODELS" tab. Under "Lab", some technical features are present, "NOTEBOOKS" has python and R code notebooks and "ANALYSES" contains data and machine learning model comparisons and analyses.

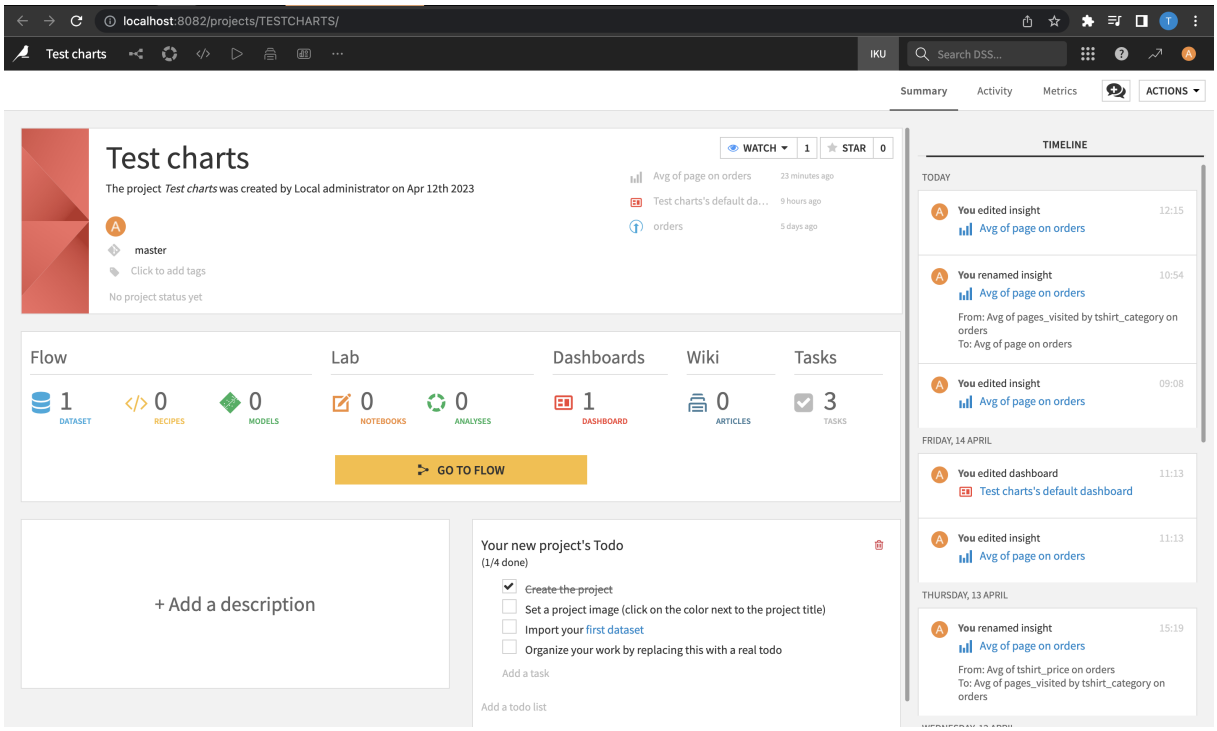


Figure 6 – Developer instance of DSS

Some other features worth pointing out in the screenshot are the "Wiki" articles, where the organization can document and explain everything that is done in the project. This can become essential for huge projects where there are Gigabytes of data, a lot of code and metrics that are not clear for all users. Next to the wiki there are tasks, which are initially things the user should do as a tutorial to start a project and understand its structure, but later can be an asset to organize what to do, considering a team of multiple people can be working on the same project. Right below this item some To-Do items are displayed, as it is important to show them to the user right when they open a project,

clarifying what they should work on next. These are derived from the tasks, and they can easily be edited without the need to enter the tasks section.

Other than that, there is the description section (currently with the "+Add a description" placeholder. It serves as a cornerstone for effective data management and collaboration. It functions as a concise and informative narrative that sheds light on the purpose, nature, and intricacies of various data objects within the Dataiku ecosystem. Another way this section has been used is as a description of the project as a whole, highlighting its objectives, progress, people involved, etc., helping people that are not directly working on it understand where it stands and know how to get more information about it.

Finally, the timeline on the right shows every modification and addition in the project. In this specific example there are not many changes, but in this example project it is already clear how this can be useful when a project becomes bigger and more complex, as more users work on it, in various features.

2.4 Flow

From the project page 6, it is possible to click on the "GO TO FLOW" button and be redirected to the flow.

In the image 7 extracted from DSS's documentation(DATAIKU, 2023a), many of what is done in the flow can be seen. In the section, some of those functionalities will be further explained and exemplified.

The flow is one of if not the most relevant element in DSS. In it, the flow of data is represented, showing all sources, how it was transformed and prepared, the visualizations performed, the plugins used, and the final database produces after everything is executed.

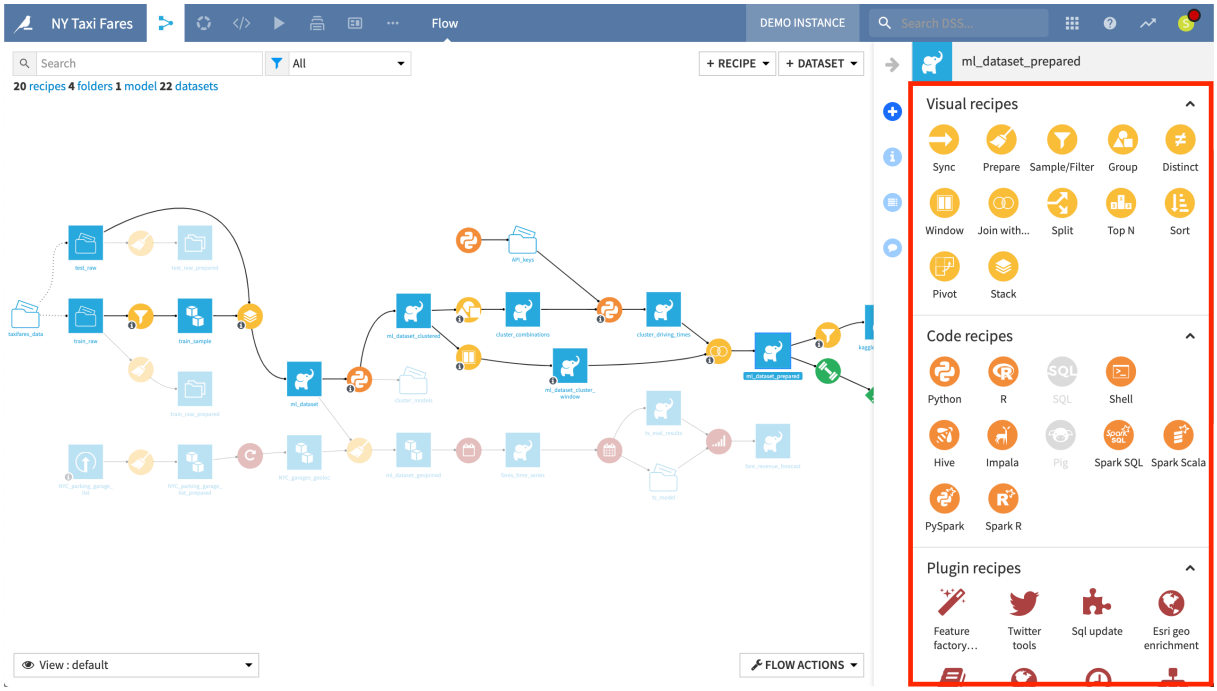


Figure 7 – Example of a flow in DSS (DATAIKU, 2023a)

2.4.1 Data sourcing

First of all, it is imperative to address how the data is imported into the flow. There are a plethora of manners to get data into the flow, including some major player in data storage in the B2B space, all the most used databases (relational and non-relational) and different file types.

It has a direct integration with Snowflake, a cloud-based data warehousing platform. Users can easily connect to Snowflake databases, import tables, and perform various data preparation and analysis tasks within DSS. The integration allows for both reading and writing data to and from Snowflake, enabling a smooth flow of information between the two platforms. Dataiku leverages Snowflake's scalability and performance, enabling data scientists and analysts to work with large datasets efficiently. Users can build data pipelines, clean and transform data, and even deploy machine learning models directly on Snowflake for seamless integration into business processes.

A robust integration with Databricks, a cloud-based big data analytics platform, is also offered. With it, users can connect to Databricks clusters and leverage the power of Apache Spark for distributed data processing. Dataiku supports the creation of Databricks-based code recipes, allowing users to execute Spark code directly within their projects. This integration facilitates scalable data processing and analysis, especially beneficial for handling large datasets and complex computations. It is possible to pull data from Databricks tables into DSS, perform data preparation tasks, build machine learning models, and then seamlessly push the results back to Databricks for further analysis or production deployment.

In terms of supported databases:

- Relational Databases: DSS can connect to and query data from popular relational databases such as MySQL, PostgreSQL, Microsoft SQL Server, and Oracle.
- NoSQL Databases: It supports NoSQL databases like MongoDB and Cassandra.
- Big Data Platforms: Integration with big data platforms such as Apache Hadoop, Spark, and Hive is also possible.

Lastly, for file types:

- CSV and Excel Files: DSS can easily import data from CSV and Excel files, which are common formats for storing structured data.
- Parquet and Avro: These are columnar storage formats often used in big data environments. they are both supported for efficient data storage and processing.
- JSON and XML: For semi-structured or unstructured data, importing and processing data in JSON and XML formats is allowed.

2.4.2 Data processing and preparation

Then, there is the data processing and preparation step, considered perhaps one of the longest for data professionals. Taking again the flow example displayed in the figure 7 above, the elements on the right tab (in the red box) are all tools to achieve a clean and clear dataset. These are divided in 3 big categories: Visual recipes, Code recipes and Plugin recipes.

Following the company's vision to make Data Science and Analysis accessible to everyone, all visual recipes use a known structure to display the data to users, as seen in the screenshot 7 from Dataiku's tutorial YouTube. It shows the column names on the first line, the type of the column under it (which sometimes is detected automatically) and a bar that represents how much of the column is correctly formatted according to the assigned data type, if it is completely green it means that all the data of the column corresponds to the data type. This format of data visualization, using a table with column names in bold on top, is also commonly used in other BI and data tools such as Power BI, Tableau and Excel.

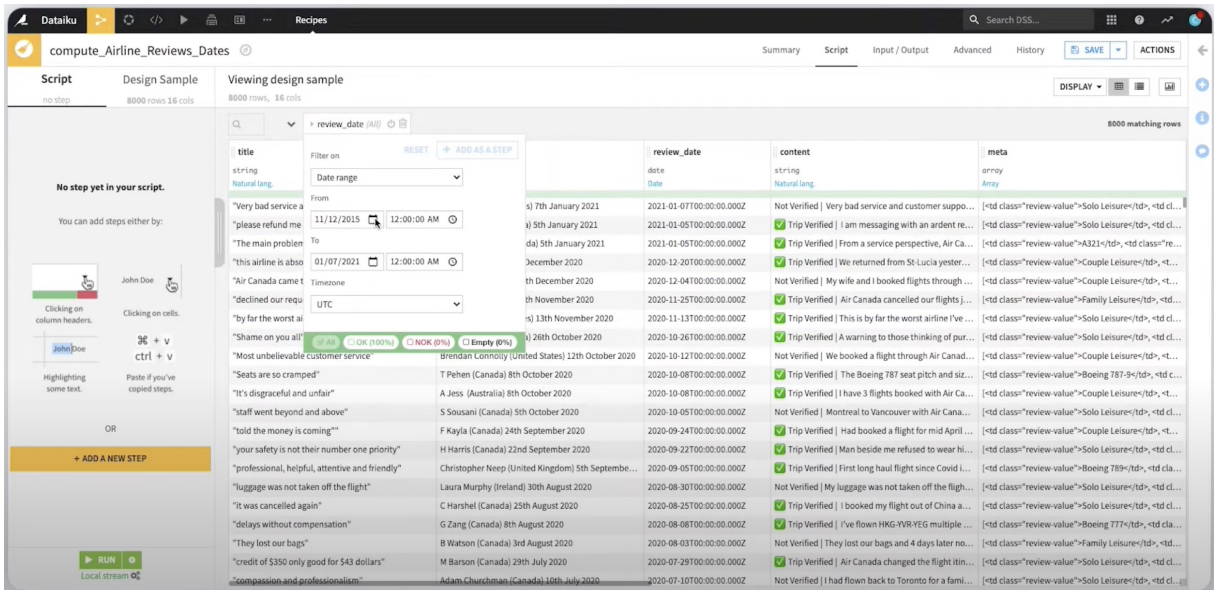


Figure 8 – Example of a date filter in DSS (DATAIKU, 2021)

Speaking about the feature of the screenshot 7 itself, a prepare visual recipe is presented. In this kind of recipe, a user can apply any number of different operations to the data, whilst seeing exactly the results of the transformations in the data, just like in Excel, but much more powerful in terms of transformations available.

The operation that is exemplified in the picture 8, is a date filter. For this data, a dialog box is rendered allowing filters of date intervals, using different time zones. Once the filter is applied, the data will automatically change, showing the result of the filter.

In addition to this feature, the DSS prepare visual recipe also includes a variety of pre-built transformations that you can apply to your data. These include:

- Cleansing: Transformations for removing errors and inconsistencies from your data.
- Normalization: Transformations for converting your data to a consistent format.
- Enrichment: Transformations for adding new information to your data.
- Grouping: Transformations for aggregating your data into groups.
- Window: Transformations for performing calculations on subsets of your data.
- Distinct: Transformations for removing duplicate rows from your data.
- Join: Transformations for combining two or more datasets.
- Fuzzy join: Transformations for combining two datasets based on fuzzy matching.
- Geo join: Transformations for combining two datasets based on geospatial features.
- Splitting datasets: Transformations for splitting a dataset into multiple datasets.

- Top N: Transformations for retrieving the first N rows of a dataset.
- Stacking datasets: Transformations for combining two or more datasets into a single dataset.
- Sampling datasets: Transformations for sampling a dataset.
- Sort: Transformations for sorting a dataset.

When all the desired operations are applied to the data, they will show up in the left column, under the script menu. This space is designed to facilitate the tracking of all the transformations on the data, ordering them and detailing what has been done. It also allows for copying and pasting steps between recipes, configuring the run environment (that can be either local or on the cloud) and, in the case where there are many operations, run all of them after they are created, avoiding wasting processing power.

On the same location, there is the sampling tab, that allows one to work quickly and efficiently with large datasets. When a dataset is sampled, a smaller subset of the original dataset is created, that is ideally statistically representative of the whole. This allows analyses to be performed on the smaller dataset, which can be much faster than performing analyses on the entire dataset.

To achieve this ideal representation of the whole with only a part of the dataset, some sampling options are provided, each for its fitting condition: First Records, which selects the first N records from the dataset, providing a quick glimpse into the initial portion of the data; Random Sampling, which randomly selects N records from the dataset, ensuring that the sample represents the overall distribution of the data; Class Rebalancing, which over-samples the minority class in a dataset to ensure it is adequately represented in the sample, overcoming class imbalance issues; etc.

On a more technical side of recipes, there are Code Recipes, designed for the more experienced users that are more comfortable using code to manipulate the data, or they need a very specific solution that is not included in DSS at the moment, that being in terms of how the data is manipulated itself, or in terms of performance, using methods of parallel computing such as spark and hive.

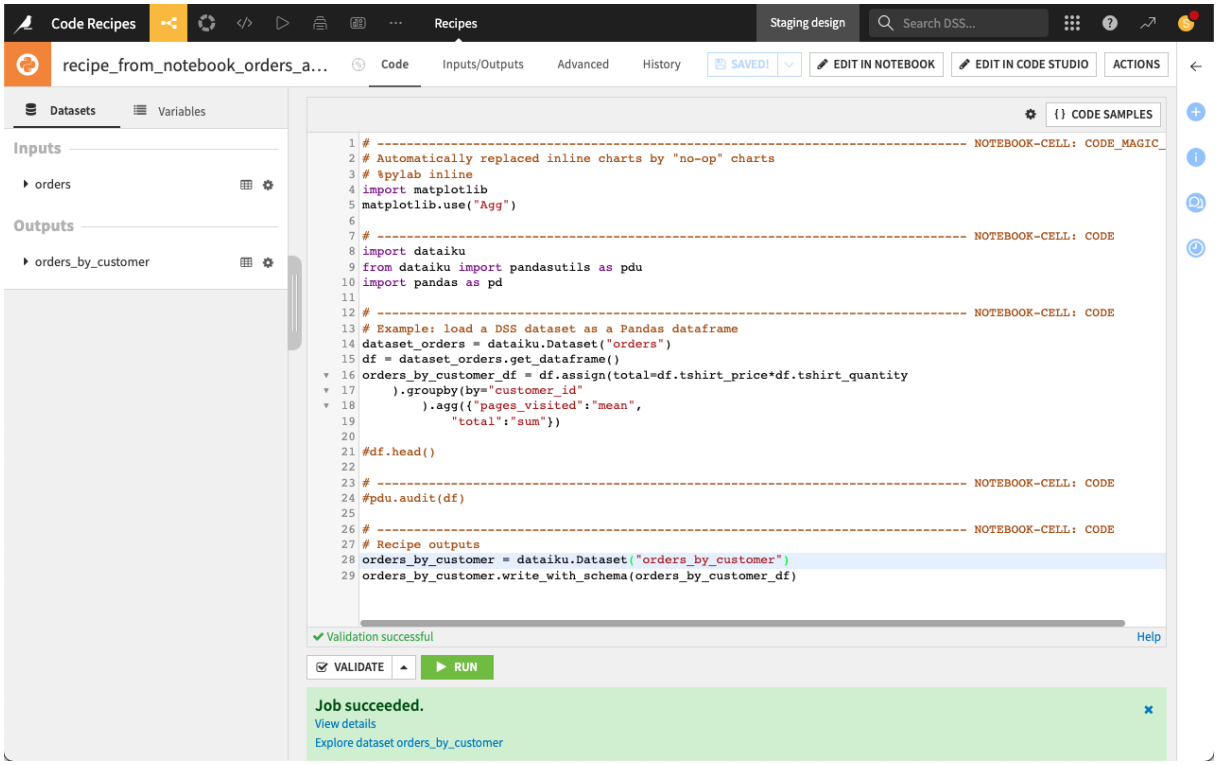


Figure 9 – Example of a flow in DSS (DATAIKU, 2021)

It can be seen in figure 9, an example of a python code recipe, that DSS provides with a code editor like interface for the user, and it automatically imports the required libraries to manipulate the data, organizing the code into cells to facilitate running it and avoid wasting resources.

The capability to use programming languages such as Python is particularly beneficial when dealing with intricate data transformations, specialized feature engineering, or any operation that requires a level of customization beyond what the visual tools offer. Code recipes provide a bridge between the visual and the code-centric approaches, allowing users to seamlessly transition from the simplicity of point-and-click actions to the intricacies of coding when necessary.

There are other scenarios where using code recipes is necessary, for example:

- Integrate external data sources and APIs: Code recipes can be used to connect to external data sources, such as APIs or databases, and retrieve relevant data for analysis. This allows for the integration of diverse data sources into a unified data pipeline, mainly when Dataiku does not have official support for a specific API yet.

- Execute machine learning algorithms: Code recipes enable the direct implementation of machine learning algorithms within the prepare recipe. This facilitates the exploration and evaluation of different algorithms without the need for separate modeling recipes, that will be explored in the next subsection.

The last type of recipes are the plugins, which have a similar objective to code recipes: extend DSS's native capabilities, allowing expert users to have a tailored experience to suit their specific needs. They are a powerful extension mechanism, allowing users to integrate custom functionality and connect with external services seamlessly.

Plugin recipes empower users to bring in custom data processing logic, which is particularly useful for unique data transformation or cleaning tasks. Moreover, they allow for the seamless integration of DSS with external tools, databases, cloud services, or APIs, enabling a smooth flow of data between DSS and external systems. Data scientists can leverage plugin recipes to implement and deploy specialized algorithms or models, extending the machine learning capabilities of the platform. This flexibility also extends to the automation of specific workflows or tasks within a DSS project, contributing to overall project efficiency.

Currently, DSS has a plugin store with more than a hundred options available, enabling users to look for and develop their own plugins, enhancing the community experience, once plugins in the store can be developed by other users as well as by non developer teams in Dataiku. They can also be installed using external tools, either by a ZIP file or using git (DATAIKU, 2023b).

Taking the Twitter (X) plugin as an example, it is designed to retrieve all attributes of your Twitter followers and store them in a Dataiku dataset. This functionality is particularly useful for generating comprehensive reports on your follower population based on various attributes such as location, language, and more. For more advanced analysis, the plugin seamlessly integrates with Dataiku's clustering algorithms, enabling users to group followers based on specific criteria. Regularly extracting follower data also allows for tracking changes in your follower base over time (BETHKE, 2016).

Furthermore, for users interested in specific topics, the Keyword Search Plugin identifies Twitter users who have particular keywords in their profiles or descriptions. This functionality is versatile, allowing you to search for relevant accounts by specifying keywords. For instance, you can identify CTOs or CIOs on Twitter based on their profiles. When combined with the Follow Recipe, this plugin enables you to follow a list of specified Twitter accounts related to your chosen keywords (BETHKE, 2016).

2.4.3 Data Analysis and Machine Learning

Lastly, after the data is imported, properly managed, integrated and prepared inside the Flow, a user can start analyzing the data and creating machine learning models to get conclusions and establish prediction from what was gathered. The interface and functions for data visualization will be explained in a later section, once it was the subject of the internship used as a case for this paper.

The platform is designed to handle a wide range of tasks related to machine learning and artificial intelligence (AI), spanning from data preparation to model deployment and monitoring. The platform offers a multitude of features and capabilities, and in Dataiku's documentation (DATAIKU, 2023c) some of those are mentioned:

The user is able to build and evaluate advanced machine learning models using AutoML and the latest AI techniques. DSS provides access to a variety of algorithms for tasks such as prediction, clustering, time series forecasting, causal ML, and computer vision. Additionally, it supports custom models developed in languages like Python, R, Scala, Julia, Pyspark, and allows for importing models from repositories such as MLFlow or Cloud ML.

It facilitates feature engineering through its feature store, allowing users to discover and reuse reference feature sets. Automatic feature generation expedites the process, and Dataiku DSS applies handling strategies for feature selection, reduction, missing values, variable encoding, and rescaling based on data type.

Integration with generative AI services, such as OpenAI's ChatGPT, is also supported. Data Science Studio provides tools for building and operationalizing Natural Language Processing (NLP) models, contributing to large language models (LLM) - augmented projects with prompt engineering via Prompt Studios.

To validate and evaluate models, k-fold cross tests, automatic diagnostics, model assertions, prediction overrides, interactive performance and interpretation reports, fairness analysis, what-if analysis, and stress tests are various of the options provided. The core principle relied upon is explaining results and delivering reliable, accurate models.

For deployment, monitoring, and maintenance of models, Dataiku operates within a unified platform, supporting MLOps and application development. It aids in scaling AI with oversight and allows users to prioritize data projects and models for maximum value.

Notably, Dataiku DSS aims to meet the diverse needs of data scientists, data engineers, business analysts, and AI consumers. Its comprehensive and user-friendly solution makes it a versatile platform for machine learning and AI workflows.

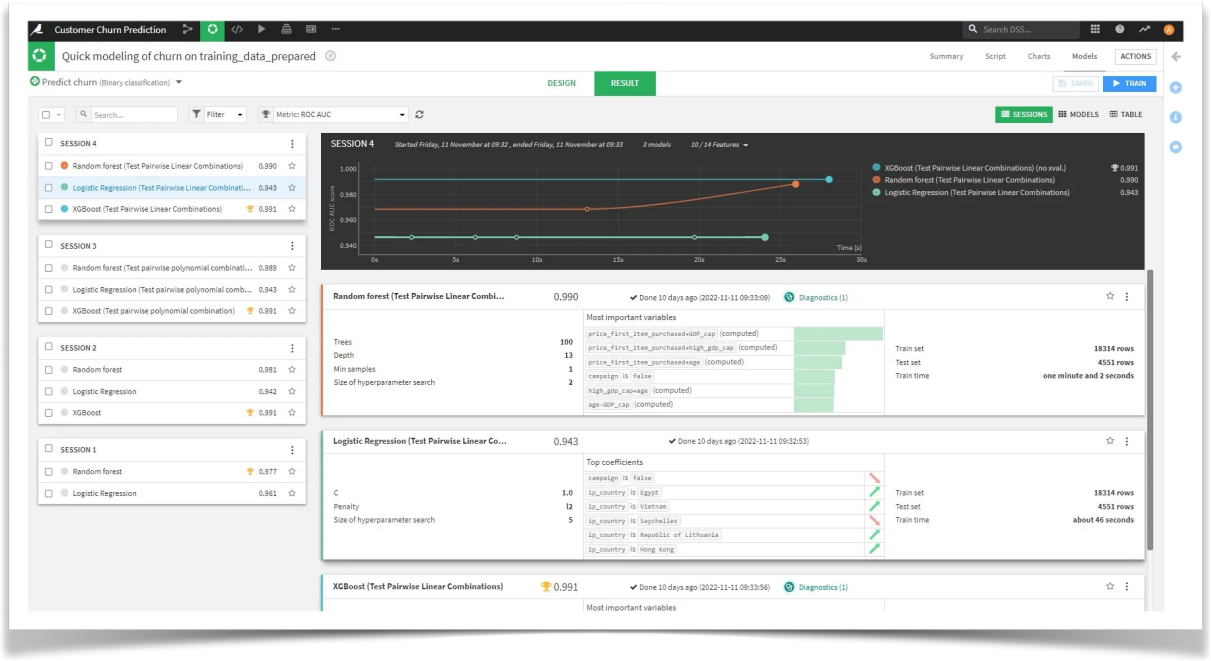


Figure 10 – Auto ML used in DSS (DENG, 2023)

Taking the figure 10 from a tutorial (DENG, 2023) we have a clear demonstration of what DSS can do and how easy it is to interpret the results.

Starting by the chart and the left column, displaying the progress of all models during the training sessions. From this, it is easy to identify if the models improved, if there is overfitting and which model performed the best in each scenario. In the last session (4), XGBoost had the highest accuracy (0.991), receiving a trophy indicating this fact.

Other details for each session are also available. In this case, the parameters of each model are shown, in the case of random forest the number of trees and the depth of them is clear to the user. For the same model, there is also an option to run further diagnosis, as well as inspect the importance of each variable when determining the result, shown by the plot "Most important variables" in the middle of the Random forest row.

3 The software development process

Now, before going into the specific case that will be unraveled in this paper, it is necessary to start by going through a generic overview of what the intricate process of software development must look like to get a robust, efficient, and reliable system, what principles should be followed, what are the steps necessary to achieve this goal, what should be executed in each of these steps, and so on and so forth. As the demand for sophisticated software solutions continues to surge, understanding the nuanced stages and practices within the software development life cycle becomes imperative.

The exploration commences with an in-depth analysis of the Software Development Life Cycle (SDLC), elucidating the strengths and weaknesses of prominent methodologies such as Waterfall, Agile, Iterative, and Spiral. From the embryonic stages of requirements analysis to the culmination of deployment and maintenance, each facet of the development process plays a pivotal role in shaping the final product. This paper delves into the multi-faceted dimensions of requirements gathering, design considerations, and the meticulous implementation of code, shedding light on best practices and emerging trends that drive the field forward.

Quality assurance stands as a sentinel against software defects and imperfections. Investigating the critical role of Software Quality Assurance (SQA) processes is a must, encompassing a spectrum of testing methodologies and tools that validate the integrity and functionality of software systems. Additionally, the significance of configuration management, project management methodologies, and documentation practices is expounded upon, emphasizing their role in fostering collaboration, traceability, and project success.

Moreover, ethical and legal considerations in software development are explored, recognizing the importance of aligning technological progress with ethical standards and legal frameworks. As the realms of deployment and maintenance is traversed, the interactions between development and operations is examined through the lens of DevOps and continuous integration/continuous deployment (CI/CD).

This comprehensive analysis through the software development process aims to equip readers with a holistic understanding of the work performed by developers, testers, and project managers alike. From this section, the fundamentals for the rest of the paper are established, and will be brought again as a base for comparison.

3.1 Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a methodical and structured approach to developing, testing, deploying, and maintaining software systems. It provides a comprehensive roadmap for the entire software development process, with a focus on efficiency, customer satisfaction, and quality.

The idea of having a structured process to develop software started in the 1960s (ELLIOTT, 2004), when the notion that it is necessary to proper steps that should be followed began to arise from the community. However, during the next two to three decades as this actually began to take shape, principles were created to guide what actually should be done in the SDLC.

It was known that a systematic approach was fundamental to SDLC, breaking down the entire process into manageable phases with specific activities and deliverables. In all of them, customer-centricity is a key principle, emphasizing the importance of understanding and incorporating customer needs, so their satisfaction is guaranteed (GITHUB, 2023).

Risk management is also a core idea in SDLC, focusing on identifying potential issues early in the development process to proactively address challenges. Quality assurance activities are integrated into each phase to ensure the software meets or exceeds expectations. Comprehensive documentation is crucial, serving as a reference for the development team and providing a foundation for future maintenance.

Flexibility is another key idea, allowing SDLC to accommodate changes prompted by evolving customer needs or technological advancements. This had mainly been added recently, with the rise of agile methodologies. Even tough it aims to be flexible, just by existing it already imposes some rules and structures that need to be followed, resulting in a trade-off between organization and flexibility that should be determined beforehand.

In terms of goals and tracking, SDLC aims to make the software development process more predictable by defining clear milestones, deliverables, and timelines. Here, the project management is key to supervise the whole cycle, helping manage resources and meeting project deadlines. The efficiency of the development process is also improved by optimizing resource utilization, minimizing redundancies, and streamlining workflows. This all happens while enhancing the trackability and visibility of what is being done to the rest of the team as well as to the whole company.

Lastly, cost control is a significant objective, achieved through effective resource management, careful planning, monitoring, and risk mitigation.

It is interesting to point out that, according to (TAYLOR, 2004), the SDLC and the project management/life cycle can and should exist and run in parallel, complementing each other. The author states that "the project life cycle encompasses all the activities of

the project", on the other hand the software development's life cycle in mainly targeting to have the desired product requirements.

As of today, after all the advances in technology and in the SDLC, it is composed of 7 main steps that, in more complex scenarios, are executed each by a different team:

1. Planning:

The planning stage sets the foundation for the entire development process. It involves defining the project scope, objectives, timelines, and resource requirements. Key activities include requirement analysis, feasibility studies, and project planning. The goal is to establish a clear roadmap for the development team.

2. Requirements Gathering:

In this stage, the focus is on understanding and documenting the software requirements. This involves communication with stakeholders, end-users, and domain experts to identify functional and non-functional requirements. The result is a comprehensive document that serves as a reference for the development team throughout the project.

3. Design:

The design stage transforms the gathered requirements into a blueprint for the software. It includes both high-level architectural design and detailed design of individual components. Design decisions address factors such as system architecture, database design, user interface, and integration points. The aim is to create a robust and scalable design that aligns with the project requirements.

4. Implementation (Coding):

The implementation stage involves translating the design into actual code. Developers write, test, and debug the code according to the specifications outlined in the design phase. This stage focuses on coding standards, best practices, and collaboration among team members to ensure the efficient creation of software components.

5. Testing:

Testing is a critical phase where the software undergoes rigorous testing to identify and fix defects. Various testing methods, including unit testing, integration testing, system testing, and user acceptance testing, are employed to verify that the software meets the specified requirements and functions as intended. The goal is to ensure the reliability, security, and performance of the software.

6. Deployment:

Deployment involves releasing the software to the production environment or end-users. This phase includes activities such as installation, configuration, data migration, and user training. The deployment process must be carefully managed to minimize disruptions

and ensure a smooth transition from development to production.

7. Maintenance and Support:

After deployment, the software enters the maintenance and support phase. This stage involves addressing issues discovered in the production environment, making updates to meet changing requirements, and providing ongoing support. Maintenance may include bug fixes, security updates, and feature enhancements to extend the lifespan of the software.

While these stages are all equally important and complex, some of them are broken down into minor steps to ease the process or to better distribute tasks within and between teams. The clearest one is testing, which in a company with a huge project can be divided in different types of technical tests (that can be written by various teams), and tests with users (mostly executed by a product team).

Although the modern and most used version of this procedure is as shown above, there are other versions that may have different stages. In the case of the infographic 11, created by the Department of Justice of the United States (JUSTICE, 2003), there are 10 steps, but on a closer look it is evident that the principles and core ideas are maintained no matter which one is used. Here, the main differences are the "Installation" and "System Concept Development", before "Planning" and the fact that "Deployment" and "Maintenance and Support" are broken into 3 steps

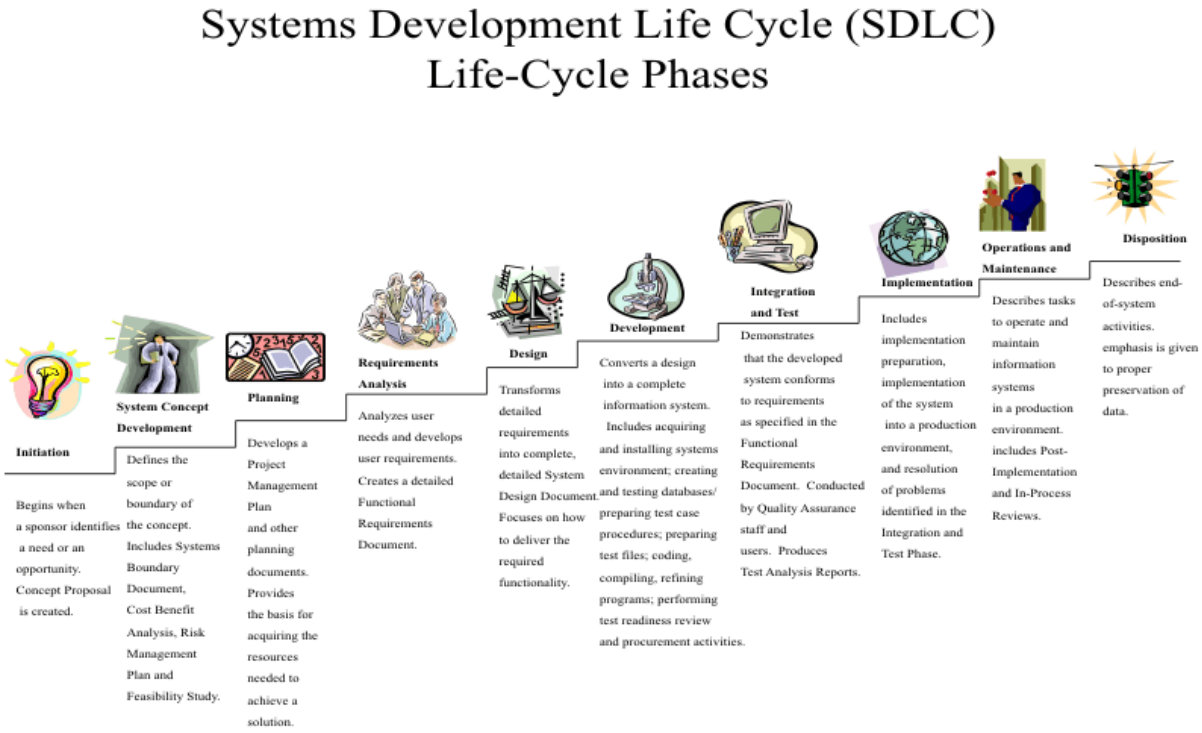


Figure 11 – DOJ SDLC steps (JUSTICE, 2003)

While various models such as Waterfall, Agile, and Spiral have been designed to implement SDLC, the core principles, ideas, objectives, and steps are consistent across

methodologies.

The next sections will be focused on giving some insight into what is the scope of the most crucial steps among the 7 mentioned, detailing what goes into these, what should be achieved in them and what are the key things to keep in mind to measure if they were successfully executed or not.

3.2 Software requirements

This section includes the second step in the Software Development Life Cycle (SDLC), Requirements Gathering, a pivotal phase that lays the groundwork for the entire development process. This stage focuses on understanding, documenting, and clarifying the needs and expectations of stakeholders, end-users, and the system itself.

The term "requirements" itself began to be used in the early 1990s (PESSÔA, 2009), and it is relevant to this day. The main two categories of them are functional and non-functional requirements (EDINBURGH, 2003), which are specifications that the software must fulfill. Functional requirements define the specific features and capabilities the software must deliver, addressing questions such as "What should the system do?" and "How should it respond to user inputs?". On the other hand, non-functional requirements, on the other hand, outline qualities like performance, security, and usability, basically they encompass the experience of the user, and tend to be more subjective.

There are also Domain requirements, that refer to specific needs and constraints related to a particular application area or industry, they can be either functional or non-functional. These requirements capture the unique characteristics and functionalities relevant to that domain. The most common examples of domain requirements are constraints of libraries that have to be used for the software to run according to what was planned (EDINBURGH, 2003).

The importance of having a structure process to gather requirements, an alignment with stakeholders, as well as clear requirements that can be easily followed by the development team have known relevance. For this reason, they are also employed in other models such as the BABoK (PESSÔA, 2009), that greatly emphasizes the need for a communication and accord to define the scope of what should be done.

For the requirements gathering process of the 7 stages SDLC, it begins with collaborative engagement with stakeholders, including end-users, clients, product owners, and domain experts. Various elicitation techniques, such as interviews, surveys, workshops, prototyping and careful observation (GITHUB, 2023), are employed to extract relevant information and ensure a comprehensive understanding of user needs.

The gathered requirements are then meticulously documented to create a clear

and concise reference for the development team. This documentation includes functional requirements specifying features and interactions, non-functional requirements defining performance and security criteria, and any constraints that need consideration. They can be documented using models such as user stories, software requirements specifications, use case documents, etc.

Once documented, the requirements undergo analysis and prioritization. Conflicting requirements are evaluated, dependencies are identified, and critical features for the initial release are determined. This prioritization ensures that development efforts are focused on the most valuable and impactful aspects of the software.

Validation and verification processes follow, ensuring the accuracy, completeness, and consistency of the gathered requirements. Stakeholders often participate in reviews and sign-off processes to confirm that the documented requirements align with their expectations and discuss feasibility, negotiating a reasonable solution.

Nonetheless, establishing traceability is essential, linking each requirement back to its source and forward to the corresponding design and testing activities. This traceability ensures that every aspect of the software is developed in alignment with the specified requirements, and is linked to specific aspects of the plan, so when it is developed, the person or the people responsible for that part of the plan also is/are responsible for satisfying the requirement.

Recognizing that requirements may change due to evolving business needs, technological advancements, or shifts in user preferences, a robust change management process is established. This process evaluates, documents, and implements changes while minimizing the impact on the development timeline, even tough it was not considered in the first versions of the SDLC and in software engineering in general.

In conclusion, effective requirements gathering is foundational to successful software development. It mitigates the risk of misunderstandings, reduces the likelihood of costly rework, and provides a roadmap for subsequent phases of the SDLC. A thorough and well-documented understanding of software requirements serves as the compass that guides the development team toward delivering a product that meets or exceeds the expectations of stakeholders and end-users.

3.3 Design

The Design phase, the third step in the Software Development Life Cycle (SDLC), is a critical juncture where the conceptualized ideas and gathered requirements take shape as a detailed blueprint for the software system. It involves translating functional specifications into a comprehensive design that guides the development team throughout

the implementation (coding) process.

To start off, it is essential to tackle the prevailing misconception that design is solely confined to the visual aspects — the aesthetics that captivate the eye. However, this notion falls short of capturing the true essence of design in this dynamic field. Design, far from being a superficial layer, encompasses a spectrum of critical elements that extend well beyond the visual façade.

At its core, design in software development is not merely about creating visually appealing interfaces, this is only a fraction of what it is, called conceptual design (ALAM, 2019). This subject is a comprehensive discipline that weaves together various threads, including but not limited to creating User Experience (UX), system architecture, and functional aesthetics. Each of these components plays a crucial role in shaping the user's interaction with the software and, consequently, the success of the application.

Above all, for this phase, the requirements (characteristics) of the system must be already precisely defined. The Design step is only concerned of making a blueprint out of the requirements, not the underlines of the problem.

On the same line, the difference of high level design (HLD) and low level design (LLD) should be stated.

High-level design focuses on providing an abstract view of the entire system. It outlines the overall structure and components of the system without delving into the finer details. HLD is concerned with defining the system architecture, identifying major modules or subsystems, and establishing the relationships and interfaces between them, for this reason it is also technology-agnostic.

In contrast, the Low-level design follows the high-level design and provides a detailed blueprint of each module or subsystem outlined in the HLD. LLD is concerned with specifying how each module will function, the data flow within it, and the algorithms and data structures to be employed. It is a more granular and specific design phase compared to HLD.

On a more foundational level, the architectural design of a software system is an integral aspect of the overall design process. It involves structuring the software in a way that promotes scalability, maintainability, and efficiency. A well-thought-out architecture lays the groundwork for a robust and adaptable software solution.

Delving into how it should be done, a crucial aspect is the detailed design, which involves specifying the functionality of each module or component in detail. This includes decisions on data structures, algorithms, database schemas, and the intricate interactions between different components. Comprehensive interfaces between modules, a focus on user interface (UI) design for an intuitive user experience, and considerations for accessibility ensure that the design accommodates the needs of end-users, and will be easily maintainable

for new and old members of the team.

On a more technical aspect, in the architectural design phase, the high-level structure of the system is conceptualized. This involves identifying key components, their interactions, and determining the system's overall architecture, whether it follows a monolithic, microservices, or another architectural pattern. It is important to keep in mind the system's decomposition, breaking it down into manageable components or modules, promoting modularity and facilitating a more organized and clean development. This is a key component of the component based software design (PESSÔA, 2009), that is very popular in modern software development.

Scalability and performance design decisions can address factors like load balancing, distributed architectures, and overall system efficiency to ensure the system can handle increased demand. These can be consequences of the previous architectural decisions, but should also structure the future steps in the software's life, mainly how it will be tested - catching issues as quickly as possible by defining test cases and routines, so in the future bugs are avoided without much effort -, and how it will be monitored and analyzed.

Last but not least, security considerations are integrated into the design phase with tools like threat modeling, identifying potential threats and vulnerabilities. In more complex systems, having detailed schemas on user privileges, combined with cryptography and robust authentication systems, are essential to avoid information leaks.

These last 2 elements (scalability and security) are an example of how this phase includes non-functional requirements into the modeling and design of the software, simulating and predicting how the software will perform (GITHUB, 2023).

In the next schema 12, some aspects explained are ordered and mentioned. It is worth noticing that the back and forth between steps is very common in modern design, mainly by stakeholder's feedback.

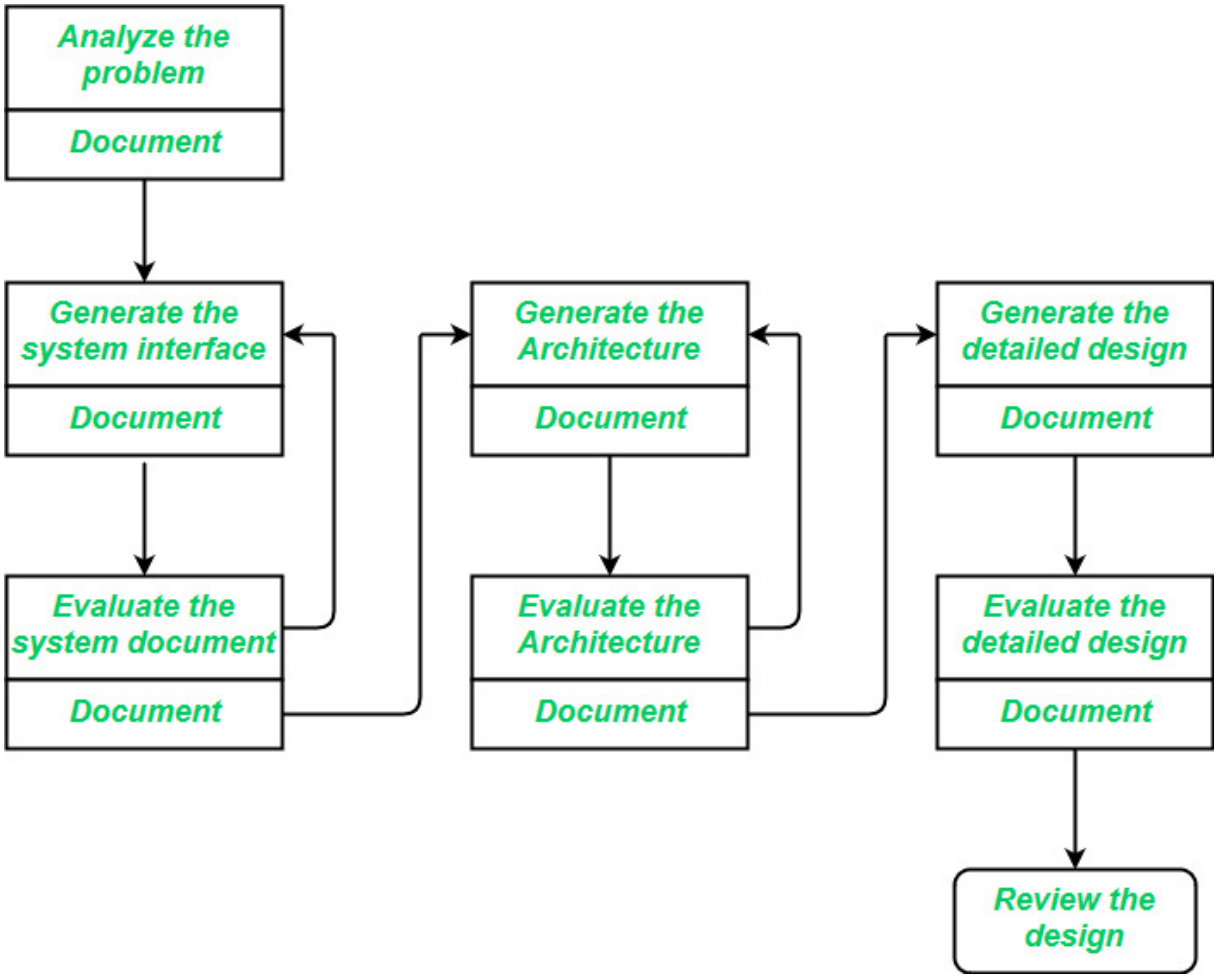


Figure 12 – software design elements (ANONYMOUS007, 2023)

3.4 Implementation and testing

3.4.1 Implementation

The development phase is where coding itself happens. Known to be one of, if not the most, time-consuming phases in the SDLC (COURSERA, 2023), it is the stage where the carefully planned and designed system takes tangible form through the creation of software components. Implementation is a hands-on process where developers bring the system to life based on the specifications outlined during the previous stages, aiming to translate the design into functional and efficient code.

Developers work on creating individual modules or components based on the detailed design. The coding phase involves writing, some testing, and refining the code to ensure it meets the specified requirements and design. Lately, version control systems are often employed to manage changes to the codebase, enabling effective collaboration and tracking modifications.

This collaboration among team members is crucial during implementation. They should work together to integrate the code seamlessly, addressing any dependencies or

conflicts that may arise. To ensure this goal is achieved, regular code reviews are an efficient way to assess the quality of the code and its adherence to coding standards, promoting better quality and maintainability.

There are 3 key aspects that will be further explored in the step:

Coding Standards and Best Practices:

Coding standards are a set of guidelines that developers adhere to when writing code. They ensure consistency, readability, and maintainability across the project. Best practices, on the other hand, are proven techniques and methods that have demonstrated effectiveness in producing high-quality software.

Adhering to coding standards and best practices is vital for several reasons. First and foremost, it promotes collaboration by making the code more accessible to other developers. It also enhances code maintainability, as standardized code is easier to understand, update, and debug. Furthermore, it contributes to the overall quality of the software, reducing the likelihood of bugs and errors.

By implementing and enforcing coding standards and best practices, companies, and development teams create a cohesive and efficient coding environment, fostering a culture of excellence that pays dividends throughout the entire SDLC. As a consequence, internal and external understanding of the code is improved, facilitating the design of new architectures and documentation.

Version Control Systems:

Version control systems (VCS) are the guardians of code history, tracking changes made by developers over time. These systems enable collaboration, provide a safety net against accidental code loss, and facilitate the management of different project versions.

The importance of version control in the implementation phase cannot be overstated. It allows developers to work concurrently on the same codebase without conflicts, roll back to previous versions in case of issues, and branch code for new features or bug fixes. This level of control ensures that the software development process remains organized and predictable, reducing the risk of errors and enabling seamless collaboration among team members.

Code Reviews:

Code reviews represent a critical checkpoint in the implementation phase. They involve team members systematically reviewing code changes made by their peers. This collaborative process offers several imperative benefits for the resulting product.

Firstly, code reviews serve as a quality control mechanism, helping to identify and rectify issues early in the development process. They promote knowledge sharing and continuous learning among team members, as developers gain insights into different

approaches and techniques. Code reviews also play a crucial role in maintaining coding standards, ensuring that the codebase remains consistent and adheres to established guidelines.

Moreover, code reviews contribute to team cohesion by fostering communication and collaboration. By providing constructive feedback, team members can collectively improve the overall quality of the codebase, resulting in a more robust and reliable final software.

Reviews are necessary to end the implementation step. Only when the code and the resulting software is thoroughly reviewed and is satisfying all requirements, it should be declared as ready (UNIVERSITY OF CONNECTICUT, Unknown)

To sum up, implementation is where the theoretical design transforms into a tangible product, allowing stakeholders to interact with the envisioned functionalities. The phase involves a feedback loop, with collaboration among developers, testers, and stakeholders to refine the system iteratively. Rigorous testing during and after implementation ensures the quality and reliability of the software, contributing to the creation of a robust and stable system.

The Implementation phase offers adaptability, allowing developers to make adjustments based on real-world implementation experiences and respond to unforeseen challenges or changes in requirements. It is a dynamic and collaborative effort demanding attention to detail, adherence to coding standards, and a commitment to delivering a functional, reliable, and high-quality software.

3.4.2 Testing

Even when using this structure to maintain consistency and alignment, tests should be created. Unit tests, for example, are performed to validate the functionality of individual units or modules of the code, identifying and fixing bugs at an early stage. Another type of tests are integration tests, that verify the interactions between different modules, ensuring a smoothly functioning integrated system.

Taking other models into account, there is the ISO 12207 (STANDARDIZATION, 2023), that has multiple steps for what the SDLC calls testing. The focus in this standard is in 3 different processes: verification, transition and validation, present in the image 13 as 6.4.9, 6.4.10 and 6.4.11 respectively, where the processes are mostly manual. Transition will not be covered in this document, because it did not have a clear presence in the case.

Verification is defined as: confirmation, through the provision of objective evidence, that specified requirements have been fulfilled (STANDARDIZATION, 2023). As for validation: confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled (STANDARDIZATION, 2023).

ISO/IEC/IEEE 12207:2017(E)

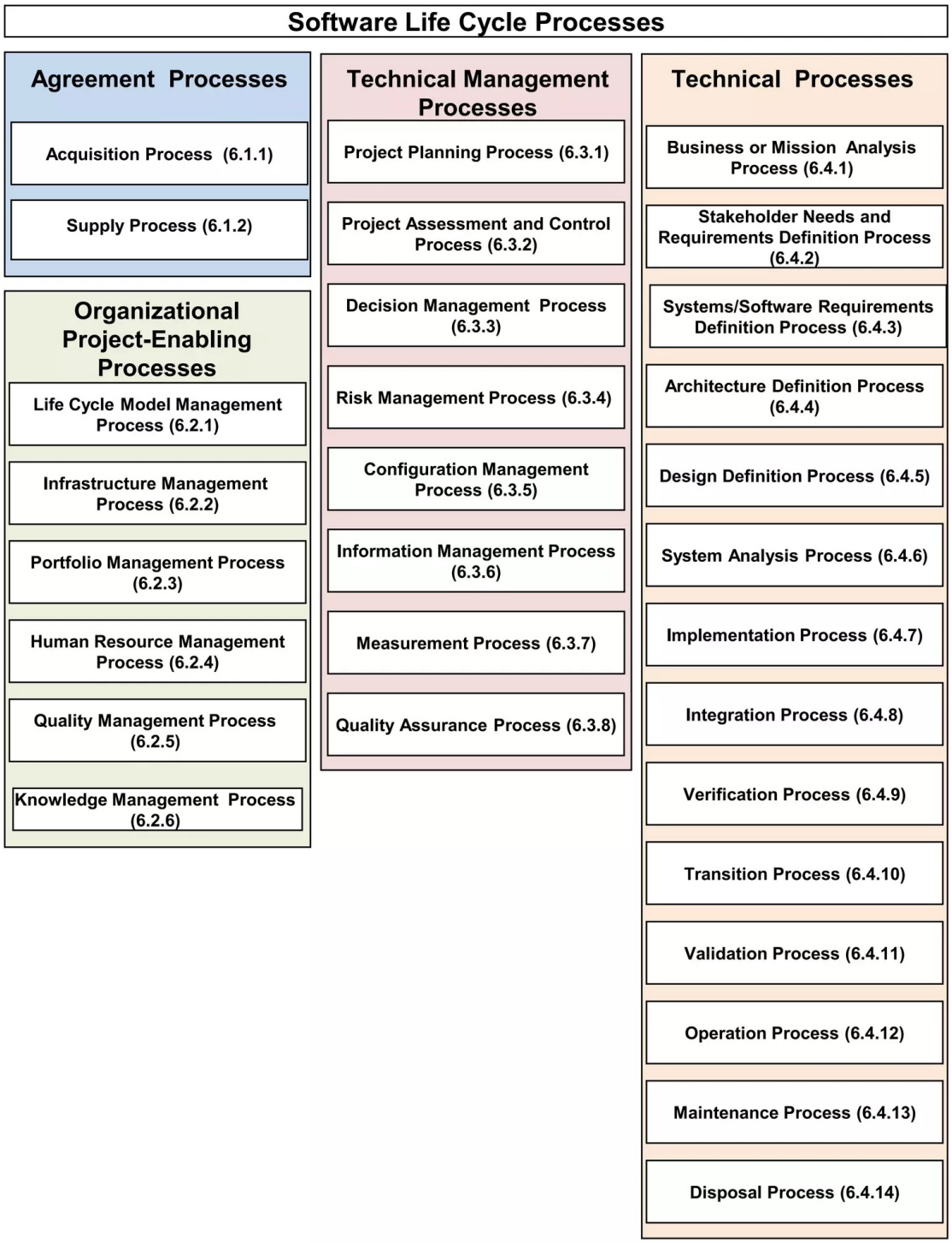


Figure 13 – Processes in ISO 12207 (STANDARDIZATION, 2023)

They seem similar, but the main difference relies on the fact that validation is actually executed in the conditions that the software will actually be used, by stakeholders. On a more operational sense, verification is a process to find anomalies, and gather information about them, while validation aims to check if the software is fulfilling its mission, eventually identifying anomalies and gathering information about them.

Once problems are identifies, by automatic means or manually, they need to be dealt with. This is done by developers, that engage in debugging to identify and resolve issues within the code, contributing to the reliability and correctness of the software. Usually, an order of actions should be followed to deal with the identified issue.

The first step is to reproduce the problem, as understanding the conditions under which the bug occurs is crucial. Developers use a combination of tools, such as debuggers, logging, and monitoring, to gather information about the program's execution.

Once the bug is reproducible, the next step is to locate the root cause. This involves examining the code, understanding the algorithms, and inspecting data structures to identify the point at which the program deviates from the expected behavior. Debuggers are invaluable tools during this phase, allowing developers to step through the code, inspect variables, and track the program's execution flow.

Understanding the context in which the bug occurs is essential. This includes considering the environment, inputs, and any external factors that may influence the program's behavior. Thoroughly reviewing the code and documentation can also provide insights into the intended functionality and reveal potential areas of concern.

While documentation is emphasized in earlier phases, it remains important during implementation. Developers document their code to provide insights for maintenance, facilitating a better understanding of the codebase for them and the rest of the team. This should also consider aspects of the previous phases, including the requirements and design of the software, to give a complete understanding to anyone that will be in contact with it.

It is important to note that the automated tests themselves are not created in the implementation phase according to most SDLC models, usually they follow this step. However, there is an overlap of phases happening in the same time, and the implementation and test phases are temporally simultaneous (SLDC FORMS, Unknown), which can help satisfy the established design and requirements. Moreover, when working in a large product, tests are executed in the coding phase, identifying problems before the end of this specific process.

In the image 14, there is an example of a sdlc, the v model. In this case, the above mentioned tests follow right after the implementation (coding). The reason for this order is what those tests are created for: both of them are, on a more fundamental level, verifying if a part of the system is working, and are usually directly related to the code that was changed, whereas the system testing and acceptance testing, as their names imply, are executed on a more global level.

To be more specific, system testing is directly linked to the system design phase. These tests thoroughly assess the overall functionality of the entire system and its compatibility with external systems. They align with the fundamental requirements of the

project, encompassing the expected actions of the software, its interactions with other software, and so on. The execution of these system tests is instrumental in uncovering most issues related to product and hardware compatibility (GURAV, 2022).

The last test, acceptance testing, can be divided in 2 main groups:

- User Acceptance Testing (UAT): involves end-users or representatives of the end-users validating the system's functionality. It typically occurs in a real-world environment, allowing users to interact with the software and ensure it meets their needs.
- Business Acceptance Testing (BAT): focuses on verifying that the software aligns with the business objectives and requirements. It ensures that the software meets the broader goals and expectations set by the business.

By focusing on end-user satisfaction and business objectives, acceptance testing serves as a critical checkpoint before software is released. It helps bridge the gap between technical development and real-world application, ensuring that the final product not only functions correctly but also meets the expectations and needs of its intended users and stakeholders.

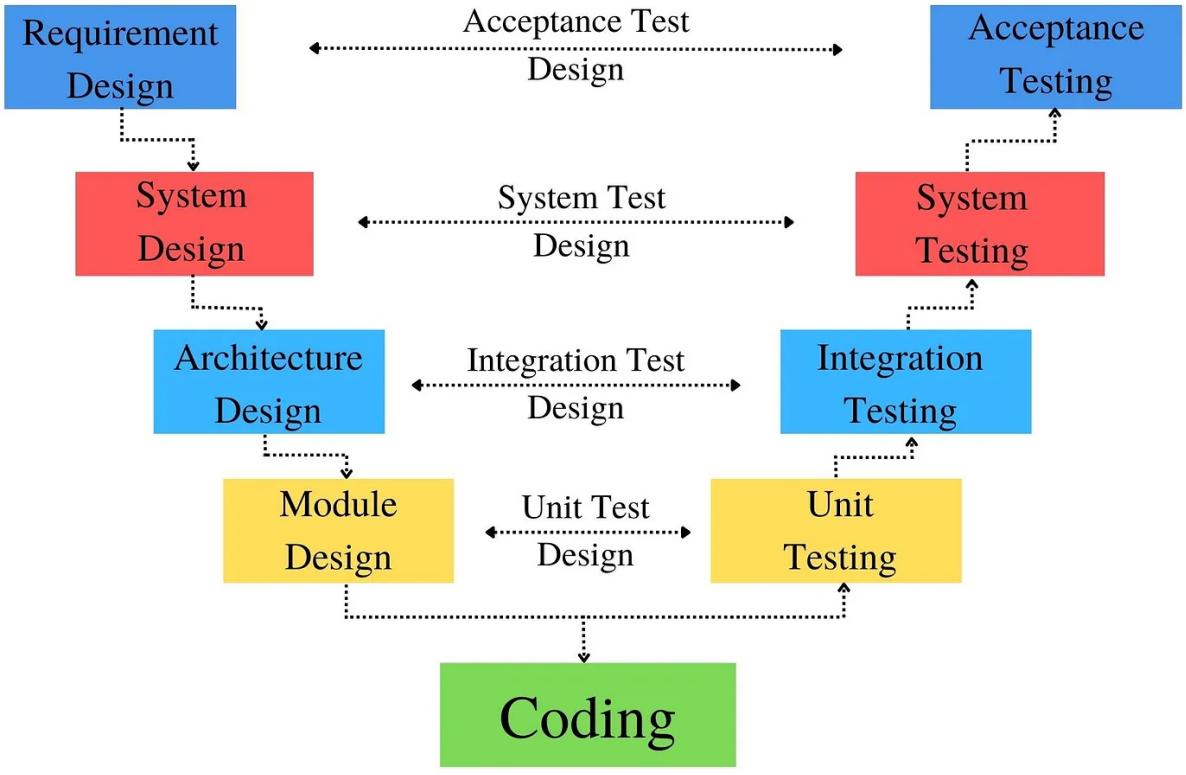


Figure 14 – SDLC V model (GURAV, 2022)

Debugging, an integral part of the testing phase, involves systematically reproducing and identifying the root cause of issues. Developers employ tools such as debuggers and logging to scrutinize the code, algorithms, and data structures, enhancing the software's

reliability and correctness. Throughout this phase, documentation remains essential, providing insights for maintenance and facilitating a comprehensive understanding of the codebase.

In conclusion, the testing phase in the Software Development Life Cycle (SDLC) is a crucial step that plays a pivotal role in ensuring the reliability, functionality, and alignment of the software with both technical specifications and business objectives. The meticulous process involves various types of tests, including unit tests for individual code units, integration tests for module interactions, and ultimately system and acceptance tests to evaluate the software on a broader scale. It not only verifies the correct functioning of the software but also validates its alignment with user expectations and business requirements. This meticulous testing process contributes significantly to the overall success and quality of the software product before its release.

3.5 Maintenance and Support

Even after all the time spent in developing and testing the software, the journey is not concluded with the deployment of a solution. Instead, it pivots into a new phase known as software maintenance and support, a critical facet that ensures the vitality and effectiveness of digital applications throughout their lifecycle.

After all the time spent structuring how the software should be developed and executing the development itself, this step may be the longest in terms of length, but the implementation and tests themselves consists of time spent. The true measure of success of a software lies not just in the initial release but in the sustained performance and adaptability of the software over time. The maintenance phase of SDLC becomes instrumental in achieving this sustained excellence.

Taking the scope and definition of this step, according to the ISO 14764:2022 (STANDARDIZATION, 2022):

> During operation of the software, problems may be detected that were not detected during verification, validation, and acceptance. Therefore, a maintenance effort is needed to cope with these problems. This maintenance effort also covers software improvements needed to meet new or modified user requirements. Software maintenance is commonly needed when upgrading system components, such as operating systems and databases, as well as when changes are made to external software and systems' interfaces. Software maintenance is typically a significant portion of life cycle costs, even when a part of the system under maintenance includes COTS software.

Other authors (Soleimani Neysiani; BABAMIR; ARITSUGI, 2020) also point out that, "finding and handling the bugs and managing the changes are the most critical tasks (of the maintenance phase)", indicating the efforts to automate bug detection, categorize them and properly document their information.

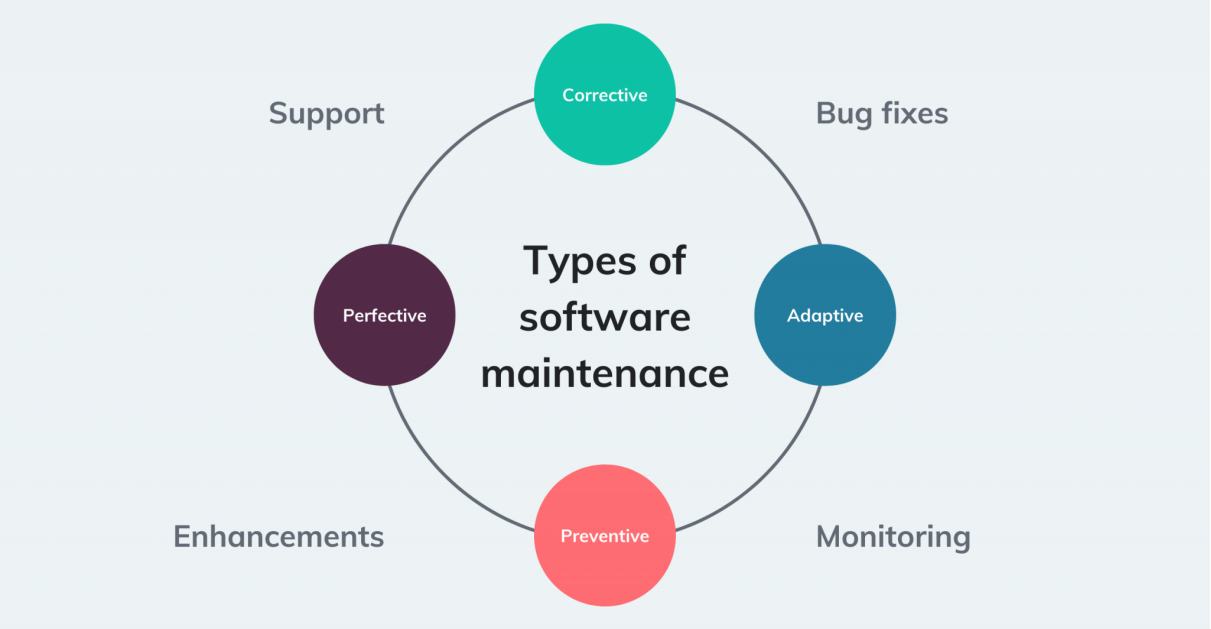


Figure 15 – Types of maintenance (PAVLOU, 2023)

Once the definition of this step is stated, some classification and precision of the tasks is due. There are four primary types of software maintenance, illustrated by the image 15, each addressing distinct aspects of a system's lifecycle, and having their own importance.

Corrective Maintenance:

Corrective maintenance is reactive in nature, focusing on identifying and rectifying defects or issues that surface after the software has been deployed. This type of maintenance ensures that the software operates as intended, meeting user expectations and resolving any unforeseen problems promptly. The goal is to correct errors and restore the system to its desired state.

Adaptive Maintenance:

In the ever-evolving landscape of technology and business environments, software must adapt to changes. Adaptive maintenance involves modifying the software to accommodate alterations in external factors such as technology upgrades, regulatory requirements, or shifts in business processes. The aim is to ensure the continued relevance and compatibility of the software with its surrounding ecosystem.

Perfective Maintenance:

Perfective maintenance is geared towards enhancing the software's functionality,

performance, and user experience. Unlike corrective maintenance, which addresses defects, perfective maintenance focuses on proactive improvements. This may involve optimizing code, refining user interfaces, or adding new features to meet evolving user needs. The goal is to elevate the overall quality and efficiency of the software, without addressing specific issues.

Preventive Maintenance:

Preventive maintenance is a proactive strategy aimed at minimizing the occurrence of future issues. It involves activities such as code refactoring, implementing security updates, and optimizing system performance to prevent potential problems. By taking preventive measures, development teams can reduce the likelihood of defects, enhance system reliability, and fortify the software against emerging challenges.

If the ISO 14764:2022 standards are taken into consideration (STANDARDIZATION, 2022), the categories are somewhat different, but the main ideas are the same, emphasizing the importance of all of these aspects in maintenance.

An aspect that has not been directly mentioned yet is support. This concept usually is linked with a knowledge base available to the end-user, and to some kind of assistance given to the user when they are in need (GARTNER, 2023). Although that may be true, the information from this interaction is actually the most valuable part of the support: user feedback.

This feedback is essential to identify bugs, business trends and general points of improvement in a product, bridging the gap between the developers - and company in general -, with their stakeholders/clients. With that in mind, a feedback loop should be established, leading to a continuous improvement of the software.

To help the team gather as much feedback as possible, it should also have automated means to collect this information. For an effective automated reporting pipeline, some key items should be considered:

- Swift Issue Identification: Users should be able to report issues promptly and clearly. This agility is crucial for identifying and addressing issues in their early stages, preventing potential escalations and dissatisfaction.

- Effective Communication: Comprehensive and concise bug reports empower developers with the information needed to understand, replicate, and ultimately resolve reported issues effectively.

- Prioritization and Resource Allocation: This, enables development teams to allocate resources efficiently, focusing on critical issues that might significantly impact software functionality or user experience.

- Iterative Improvement: Patterns in reported issues can guide development teams in refining their processes, enhancing code quality, and proactively addressing recurring problems.

By implementing user-friendly reporting mechanisms, utilizing automated tools, and integrating the pipeline into the broader SDLC, organizations can enhance their ability to identify, address, and learn from reported bugs, ultimately contributing to the delivery of high-quality software products.

All in all, software maintenance and support are continuous, collaborative processes demanding agility and commitment to excellence. Embracing these principles ensures not only the extension of digital solutions' lifespan but also their continued value in an ever-evolving technological landscape.

4 The development process in Dataiku

In Dataiku, agile development methods are a constant in the software development team. The structure of SCRUM, using sprints, points to tasks, reviewing what was done at the end and having a Product manager (in this case a technical product manager), for example, are all present in this case. Some other principles that will be highlighted are:

- Iterative Development: Dataiku employs an iterative approach, breaking down tasks into manageable cycles or sprints for quick adaptations to changing requirements;
- Collaboration and Communication: The company facilitates continuous collaboration through features like version control, commenting, and project sharing among cross-functional teams.
- Customer Feedback: Dataiku encourages constant user feedback on model outputs, visualizations, and platform usability to enhance the user experience.
- Flexibility and Adaptability: Agile methods enable Dataiku to support flexibility in modifying data pipelines, models, and analytics workflows during development cycles.
- Continuous Improvement: The platform likely supports a culture of continuous improvement, offering monitoring and analytics features to assess and enhance the efficiency of data science processes.

4.1 Onboarding

Starting the journey at Dataiku, it was already visible the efforts all teams do to integrate and contextualize the new employees by dedicating a whole week so that they can meet their perspectives teams and familiarize themselves with Dataiku's main product (DSS) and other internal processes and tools.

The onboarding program of the first week was also prepared with the intention to integrate new employees, presenting the different areas of the product and the company. More so, session emphasizing the values of the company, its plans for the future and how to deal with different situations at work were also present, as seen in the google calendar image 16.

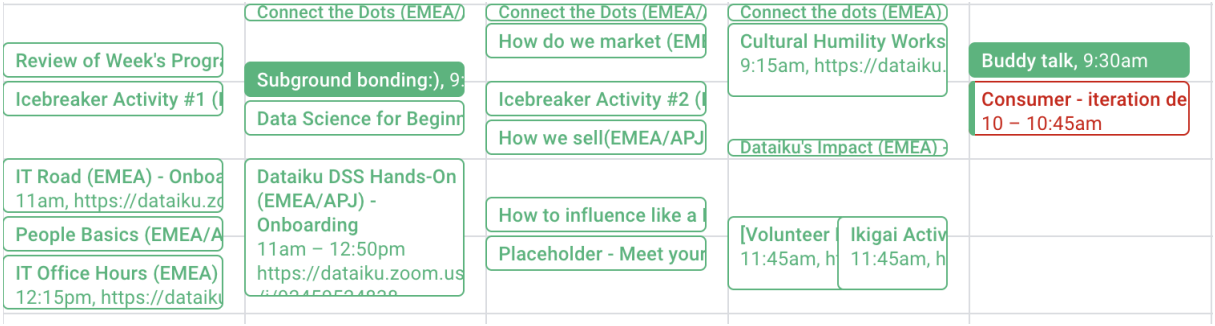


Figure 16 – Google calendar with activities of the first week

During this period, the interns were also instructed to schedule one-on-one meetings with everyone on their teams, easing the integration with the corresponding teams in work and not work related situations, while learning everyone's roles in the company.

Furthermore, some online courses were also proposed, focusing on compliance, reviewing the content that was presented during the sections and going more in depth about specific functions and tools present in the platform.

For these learning methods, the platform used most of the time was the newly created Mindtickle, shown in the screenshot 17, that has internal courses and information about the company. As it is owned and developed by Dataiku, everything available in it is created by other employees.

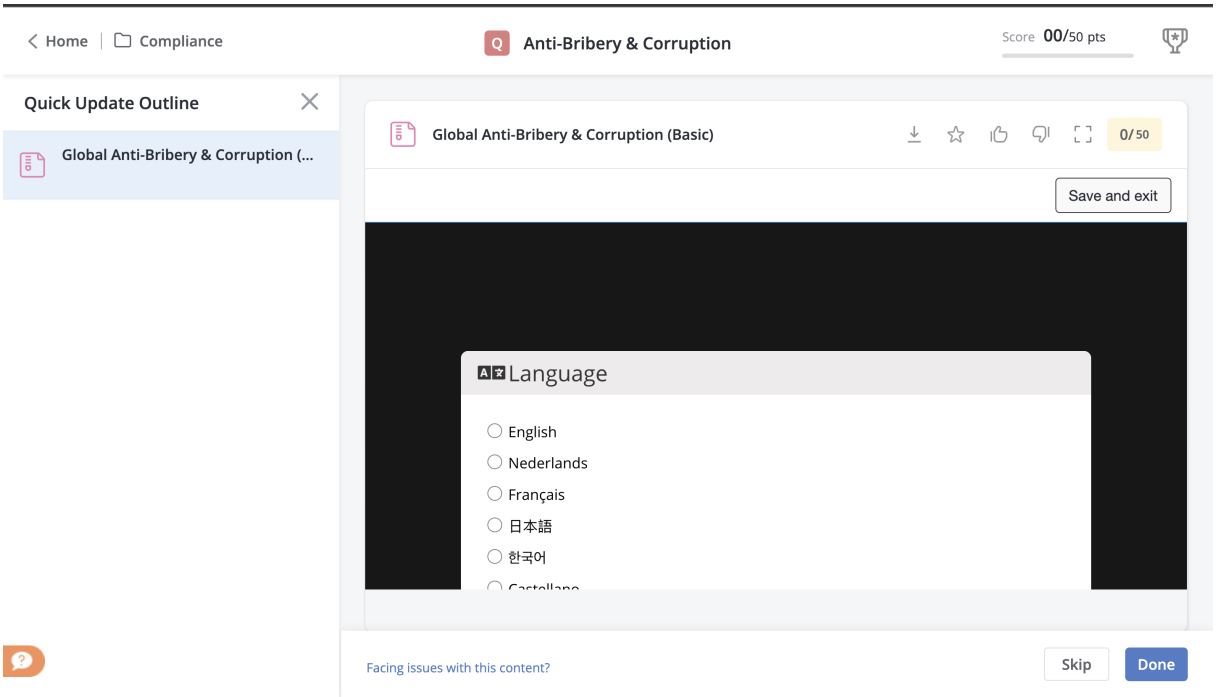


Figure 17 – Mindtickle

4.2 Company communication

For our more direct communication, the software we used was slack. In it there are channels for each team and each feature where we discussed details of the implementation, bugs and asked for feedback. For more particular matters, private messages with other users were also possible.

In slack, there were also channels dedicated for nonwork related purposes, such as the #office-paris where general communication about events, problems, and maintenance were communicated to everyone. Furthermore, channels for after work events like random-volley and random-soccer also existed, which shows how co-workers can integrate better and get to know each other thanks to the amicable environment of the company.

Meetings were also a frequent occurrence in the internship. Following the transparency principle stated, there are quarterly updates about the company status, progress and new projects that are available for everyone to join. There were also meeting that brought people from all over the company to discuss or teach a topic, mainly on the "Dataiku University" meeting on Fridays.

Last but not least, our team had weekly and bi-weekly meeting to track the progress of everyone, share what each person has been working on and plan what should we do for the next iteration/period. Other than participating in these meetings, I also had one-on-ones with my manager and dedicated ones to decide details about what I was working on. With this, the core of fast iteration and listening to users was executed, with constant changes made and suggested during this period

All the reunions were neatly organized in google calendar and executed in zoom (when there were numerous people) or sometimes on slack (in some one-on-ones). They were always scheduled in advance to make time for planning and organization, avoiding waste of time. To the same end, most of the time they were limited to 30 minutes.

4.3 Progress tracking

During the development of the task, the mission was broke in smaller sub-tasks to make the progress tracking easier and have a better grasp of the scope of each part of the development. The tool of choice for this kind of organization is shortcut. In it, there are epics, the project itself, which in this case corresponded to the radar, and stories, the tasks of a given epic. In the image below 18, it is possible to see some of the tasks that belonged to the radar epic.

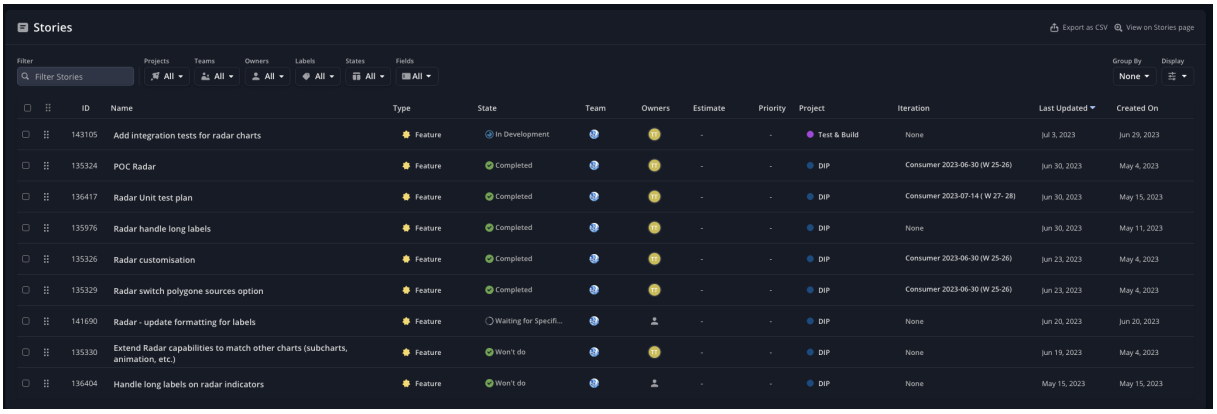


Figure 18 – Stories of the radar epic in shortcut

Shortcut also offers a variety of options to help the organization of the team, such as integrated docs, issue tracking, sprint planning, roadmaps, etc. It supports Agile project management concepts in Scrum, Kanban, and everything in between, which helps teams follow a cycle of planning, executing, and evaluating its performance.

A major feature of the platform is its categorization and classification. Although most of the time done manually, it is possible to assign specific teams and people to tasks, attribute it to an iteration (sprint), and have its type (feature, bug, improvement, etc.), facilitating the development process.

It also offers tightly integration Docs, Issue Tracking, Sprint Planning, and Roadmap features that unite planning and development in a single experience. Even though these capabilities are all useful, the most important feature for Tayoken team was the integration with github.

This integration is done by following the Pull Request template, where a Shortcut link is added, and the PR is linked to the shortcut task. Once that is done, it is possible to easily access the task from the PR in github, and see its main information directly from Shortcut, such as name, status (draft, merged, etc.), repository, and number of lines changed, as exemplified in 19.

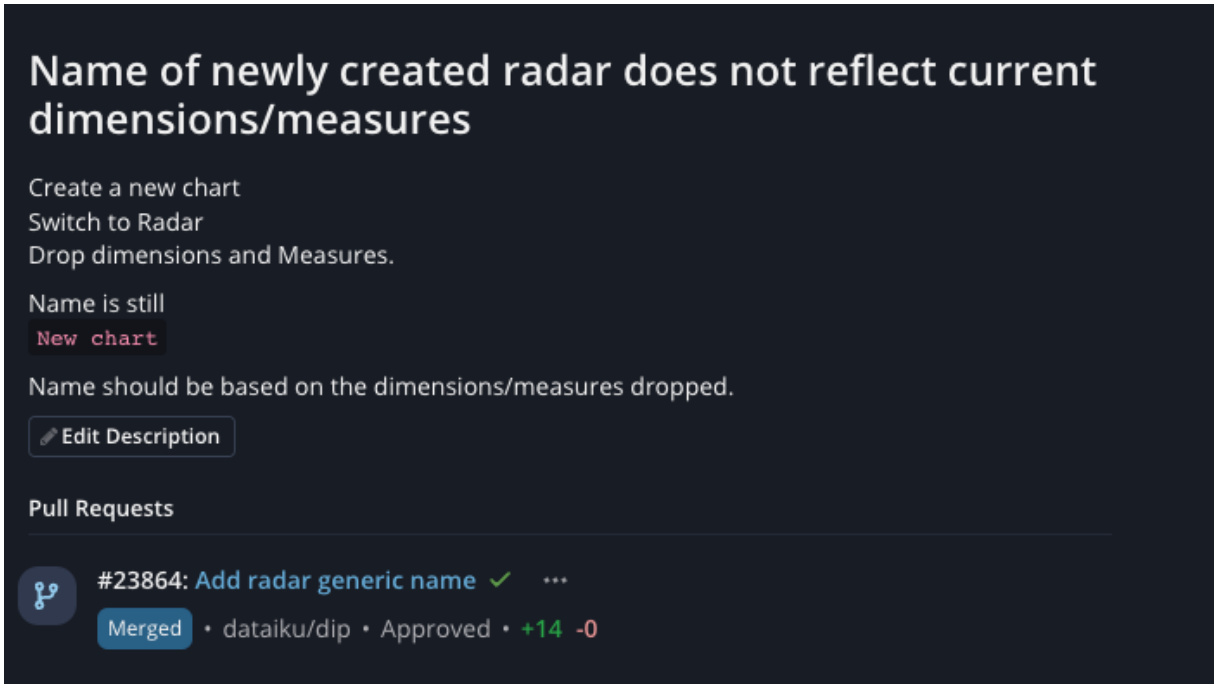


Figure 19 – Shortcut task with PR integrated

It is important to highlight the state of each story, which can be “in development”, “in review”, “waiting for specification”, “waiting for review”, “completed”, “new”, etc., which is based on a Kanban and reflects the sdlc model. These columns are always kept up to date so that anyone in the company can follow the progress of any epic and story in development.

The same strategy and tools were used in the second project of the internship: the sankey chart, but this time there were no previous specifications to follow, the process was a bit longer, and included more of the planning and design phases. However, this time the knowledge of the code base and best practices was already acquired, resulting in a smaller development window.

Even tough shortcut was very useful, for a more granular and daily view of the tasks progress’ github was used, so shortcut ended up being used more by managers and non-technical teammates, whereas the direct supervisor of the projects took a closer look into github.

In this platform one can follow individual commits, exact modifications done in the code, comment and review those changes, and do a lot more. In the example figure 20 below, a comment and some commits are shown.

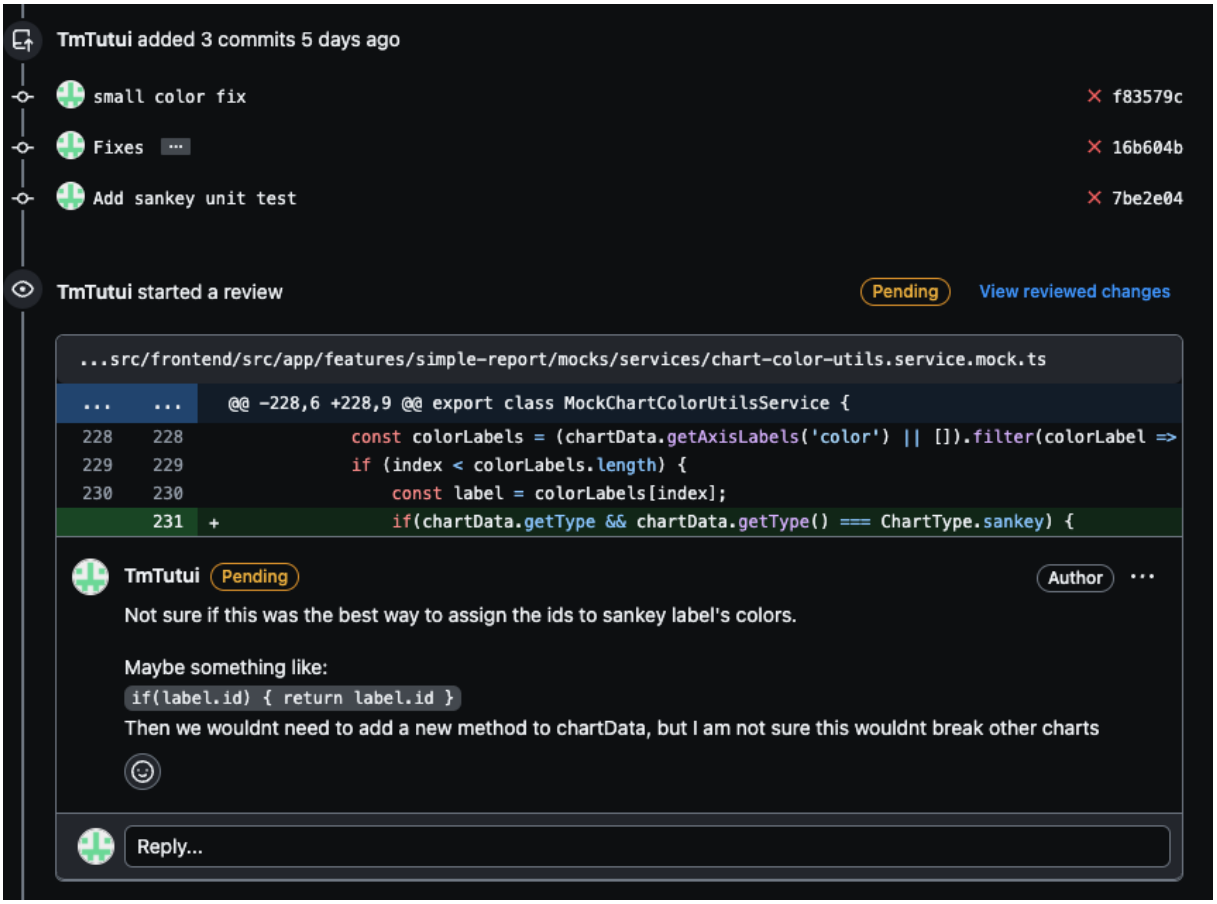


Figure 20 – PR history in github

4.4 Codebase, angular and Echarts

After the welcome week, a couple of meetings with my direct manager and the colleagues I will be directly working with happened to establish how I will familiarize myself with the code and both Angular and AngularJS's framework.

I received some direct instruction from my peers, as well as some video lectures previously recorded by other members of the company explaining specific aspect of the code or of the framework. Other than that, some specific onboarding for the research and development team was set in place. During its sessions, we were able to have some specific explanation on how and why the codebase was structured the way it is, and ask our question about it. This material hinted at the knowledge base that was built and organized in the company over the years.

With these resources, I was able to get an overview of everything DSS related. For the infrastructure, the java backend has its activity minimized by delegating jobs to JEK and FEK processes, because if the backend itself crashes, the whole application stops. The front-end runs on nginx, which is known for being an open source, high performance, efficient, and versatile (F5, INC, 2023). There is also a jupyter instance to run python code. All these processes are started by supervisord, which is a python instance responsible

for monitoring and logging everything that happens with what it started.

Another interesting thing about the java backend is that, due to the huge amount of interactions between it and the front, a utility called "j2ts". This application is used to convert java classes and convert it to typescript interfaces. With that, all API response types are available in the front end, which can speed up a lot of the development process by avoiding mistakes. As running this utility every time a change is made in the codebase would take too long, there is a specific command that triggers it, so if a change is made in the java classes or the developer has to change to a branch too far ahead or being their current branch, this command should be executed.

For the front-end, which is the part that I interacted the most during my day-to-day activities, some of the code that is still used is written in the old angularJS (GOOGLE, 2021), which is no longer supported, but the new code is written in angular (GOOGLE, 2023), which is what I ended up using the most. To make these 2 frameworks interact seamlessly, the team implemented a translation layer that downgrades angular components, services, and constants to their angularJS counterparts, so I had to keep in mind that depending on what I create in angular I will have to include it in the migration service. The opposite may also happen, so there is an upgrade service that transforms angularJS code into angular.

It is important to note that most of the new code also uses Typescript instead of Javascript. Doing this, we can save a lot of time when debugging or when doing the code itself by specifying the type of each variable. When dealing with nested objects it becomes a bit more complicated because sometimes it is necessary to create types (or interfaces depending on the usage (COURTER, 2022)) for these objects, but once this is done it will be easy to deal with it as anywhere it is used it is possible to know its keys, key types and value types.

That said, I often dealt with the standard structure of an angular component, as shown in the figure 21 below. It contains the component itself in a ts (typescript) file - *values-display-form.component.ts* - and the template in a html file - *values-display-form.component.html*. In this specific case there is no style file associated with the component (which could be a css, sass, etc.), and there are some other files such as the *.spec* used for tests.

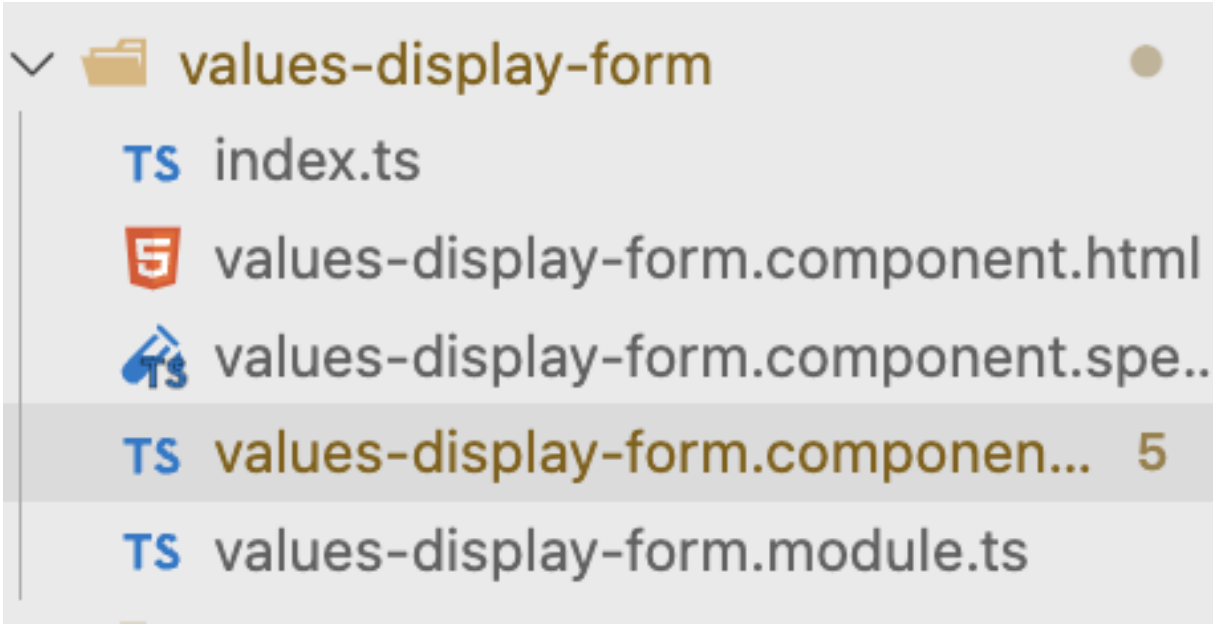


Figure 21 – Standard angular component folder structure

Finally, the documentation has been and will be of great support to understand both the framework and the library needed to succeed in the task. It is very clear and organized, with examples and related topics, as seen by their website 22. Even if Angular documentation's design is better, they are both informative, and great for their purpose.

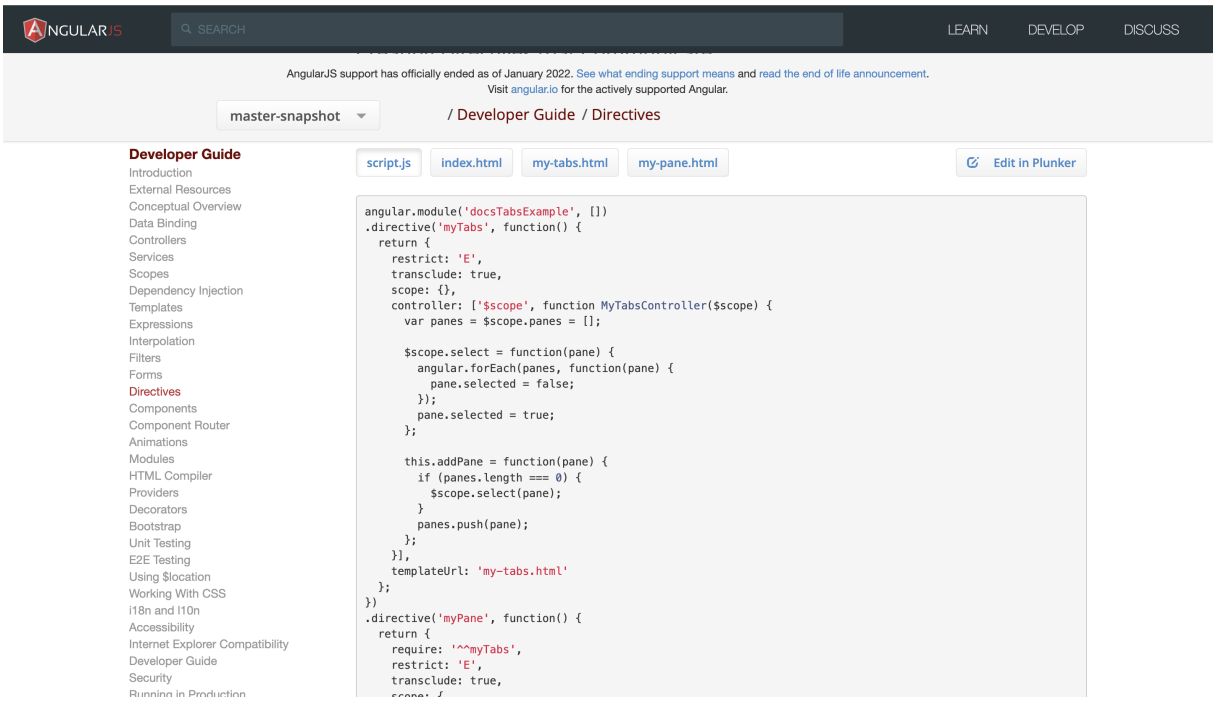


Figure 22 – AngularJS's documentation

4.4.1 Building charts

Choosing a suitable charting library for front-end development is a pivotal decision that developers often struggle with due to the array of options available. Two prominent contenders in this realm are Echarts(THE APACHE SOFTWARE FOUNDATION, 2023a) and D3.js, each with distinct strengths and considerations. The choice between them hinges on striking a balance between customization and development speed, catering to the specific needs of a project.

Echarts, renowned for its impressive out-of-the-box visualizations and user-friendly API, prioritizes development speed without sacrificing customization entirely. It offers a wide range of pre-designed charts that can be easily integrated into projects, reducing the time and effort required to create complex visualizations. This is particularly advantageous when aiming to swiftly develop functional dashboards or data-driven applications. Echarts' API enables developers to quickly configure and customize charts to align with the project's aesthetic and functional requirements. However, Echarts' customization might be constrained to some extent when compared to a more low-level library like D3.js.

On the other hand, D3.js, although considered more complex, empowers developers with unparalleled customization possibilities. It operates at a lower level, allowing developers to have fine-grained control over every aspect of a visualization. This approach is ideal when dealing with unique data visualization challenges or when the project's design demands a high degree of specificity. D3.js enables the creation of innovative and intricate visualizations. However, this depth of customization comes at a cost of increased development time and a steeper learning curve. Developers using D3.js must often build visualizations from the ground up, which can be time-consuming, particularly for projects with tight deadlines.

In essence, the decision between Echarts and D3.js boils down to a trade-off between customization and development speed. Echarts suits scenarios where rapid development and good-looking visualizations are paramount, but with a potential limitation on extreme customization. On the other hand, D3.js thrives in projects that demand unparalleled customization and are willing to invest more time in development. A thoughtful evaluation of project requirements, deadlines, and development team expertise is crucial when making this choice, as it ultimately shapes the success and efficiency of the front-end development process. As it is choice between technologies, it should be made in the end of the design phase, in the LLD, when the base information of the project is already defined.

Those two libraries are used in DSS, but considering the amount of plots available, the company decided to migrate some old charts to Echarts and develop the new ones in this library, acquiring more speed to develop its features. That is the reason during my internship I will be using and focusing only on Echarts, to satisfy the end-user needs of

new charts and catch up with the competition.

In terms of the code itself, there are two objects that are widely used that need some highlight: chartDef and chartData. The first one is derived from a back-end class (so j2ts converts it to a typescript interface), and is mostly used to store chart configurations and declare chart options. For example, an *enum* "LegendPlacement" is declared with the possible positions for a chart legend (inner top left, inner top right, outer top, etc.) and a property of the class called "legendPlacement" is declared with type "LegendPlacement" and default value "LegendPlacement.OUTER_RIGHT".

The other object, chartData, as the name implies, hold most of the data of the chart itself, contrary to chartDef that has more layout and settings. An example of the data one could find in this object is the request with the data points or the maximum and minimum values present in the chart.

Other than that, it is crucial to define measures and dimension, those 2 concepts are vital for the understanding of the entirety of the following explanations. Measures are columns in the database that will be aggregated following an aggregation function (sum, average, count unique, etc.) according to the dimension. Which means, dimensions are the columns used to aggregate the measures by.

4.4.2 Tests

The tests' codebase was located in the same github repository as the rest of the DSS code, "dip". The quality assurance (QA) team was responsible by the test infrastructure and the creation of the tests themselves, even though most of the time the team that developed the feature was also involved in the planning of tests scenarios and the development.

In terms of test planning, templates and a script, basically the instructions were to run a test to check if the basic functionally of the charts is working as expected. Depending on the feature other tests were executed, but for the explored team, that focused on creating the chart, this was the base case, and since every chart has different settings, tests could be created to check if these are working. The basic test is done either by a code output in unit tests or by comparing the generated to a reference image in integration tests.

These are the 2 main types of automated tests in DSS that will be mentioned, other tests like the acceptance test do not involve the developer team directly and may happen later when the feature is released, so they are not discussed in detail:

Unit tests: These are subdivided in 2, angular and angularJS tests, which makes sense since the frameworks are different and there is a significant amount of code using both. They are written in their respective languages (typescript and javascript), and they aim to check if the parameters and outputs of the code are correct.

Integration tests: Integration tests, on the other hand, focus on testing the user interface. For this reason are written in python using the selenium library, and located in a completely separate folder (*/integration/python*). These tests basically drag and drop components, check if a chart is being displayed, and compare the resulting chart to a reference screenshot. The QA is mostly responsible for these tests, creating the infrastructure for them to work, the screenshot database and interface, as well as updating the tests when something changes.

After these tests, manual tests are executed: validation and verification. The validation step is actually more code related, it is when another member of the team that is not directly working on the project looks at the code to evaluate if there are points to be improved, mostly looking for optimization and improving reusability. On the other side of the spectrum, verification is focused more on the feature, and is executed by a different person from the previous test, in this case it is not necessarily someone from the same team. It is actually in these two steps that most of the bugs explored in 4.5.6 and 4.6.6 were detected.

4.4.3 Environment configuration

As previously stated, following standardized and good code practices is essential to producing high quality code, and as a consequence, a high quality product. For this specific practice, Dataiku has a couple of scripts that are run in the work device (in this case, it was a 13 inches m2 macbook pro with 16 GB of ram) to set up this environment. The scripts were mostly automated, but depending on the architecture of the processor specific steps had to be followed, which was made easy by a tutorial on the internal wiki.

The scripts handled most of the set-up (specific python version and libraries for test, a version of JDK and its dependencies for the back-end, etc.), but some things had to be installed manually.

The first one was docker. This software is known to have some emulation issues when running an image of the x86 processor architecture on an ARM architecture processor, but the tutorials had specific workarounds for these issues. Once installed, the function of docker was to emulate a version of chrome to speed up the automated integration tests, that ran slowly natively.

The other manual installation was specific visual studio code extensions for linting and formatting code, as well as their configuration files. Even tough there was a github pipeline to verify if the code did was following all establish patterns, this was imperative to make everyone in the team use the same linting and formatting standards before deployment, gaining time and saving resources.

4.5 Radar chart

The first project implementation in this study is the radar chart. To start working on it, first it was important to break the goal down in a more concrete and simple task that I would be able to achieve in a smaller period of time, as well as get some more practical experience with the codebase.

Keeping this information in mind, my manager and I decided that my task for the first days would be to display a static radar chart with hard coded data, following the PDCA (Plan, Do, Check, Act) principle of "learning-by-doing" (LEAN ENTERPRISE INSTITUTE INC., 2023). Doing that, I would be able to understand the interactions between the old and the new code, where functions and classes are exported, how templates mold the charts, etc. Then, after this is finished, other concepts such as how requests are made, how to deal with these requests' data, and where the user inputs are stored would be tackled.

As mentioned before, the specifications for the radar were already set, so before I started to code, I read through the radar wiki and added what I thought it was necessary.

The first notable information from the wiki was the radar terminology, which will be used from now on, demonstrated in this schema 23:

Other than that, there were 3 cases that the chart was supposed to satisfy:

1 **Dimension values as axis**, figure 24:

The axis would be the unique values of the dimension, and each polygon would correspond to an aggregated measure (according to the dimension value).

2 **Dimension values as polygons**, figure 25:

Each polygon corresponding to a value in the dimension column, and each axis would be a measure.

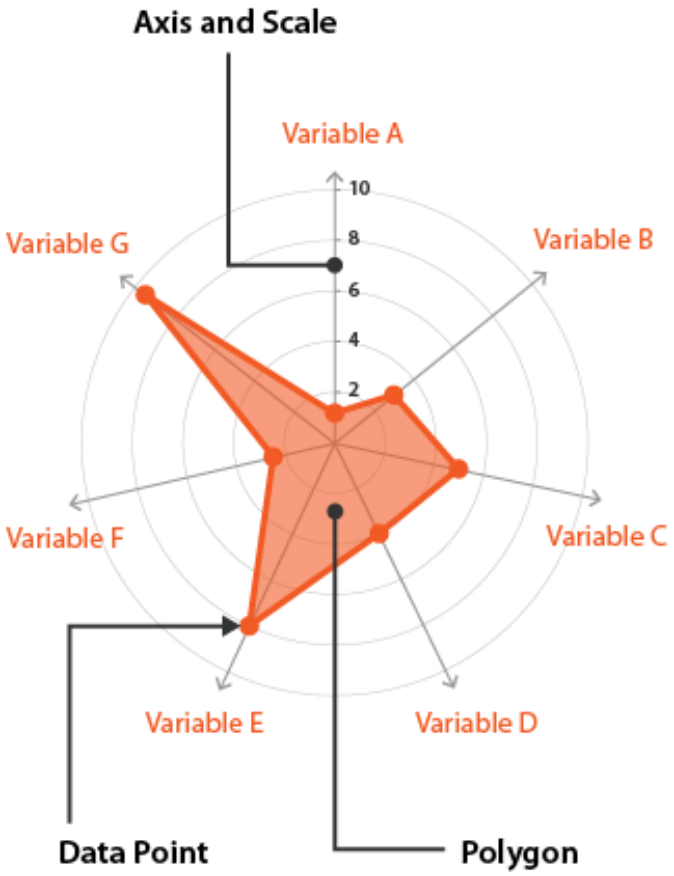


Figure 23 – Radar terminology (THE DATA VISUALISATION CATALOGUE, 2023)

3 **No dimensions**, figure 26:

   Measure would be an axis, and the polygon would be created the aggregation of all values of these axes. This setting was later discarded because it would not have much use by itself.

This process corresponded to understanding what was already in place for the planning and design phase of the SDLC, and showed how the development process at Dataiku was already consolidated and effective. It is relevant to highlight that the radar chart derived from user requests, which are of great importance to the company, following the support and maintenance principles.

That said, the next step was the development of the chart. As it should, the company uses a version control software, in this case, github. How this is organized will be explained in a further section (4.5.3), but to start off, it was necessary to create a new branch from master. First of all, the structure of the chart was added: the name of the chart in the chart list, its icons and its variants (in this case there is only the normal radar). Then I analyzed how other charts that used echarts were created: there is a service that generates the echarts options for the specific chart.

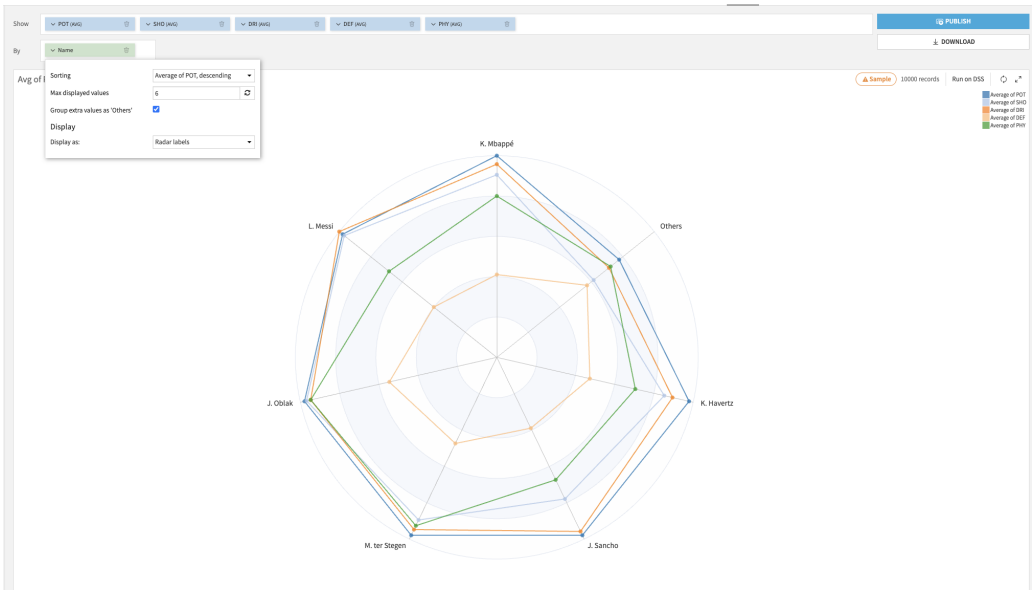


Figure 24 – Radar dimension as axis

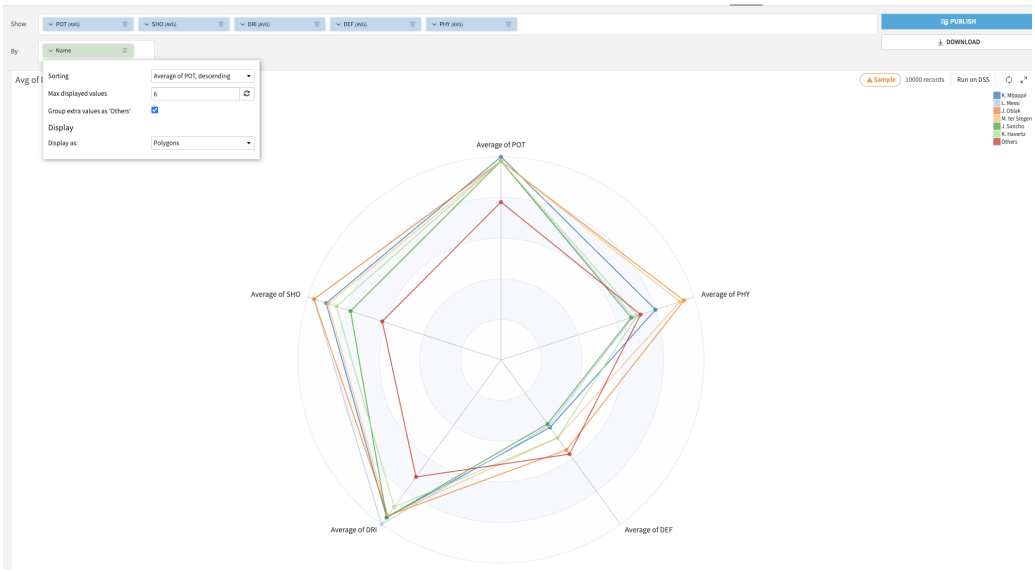


Figure 25 – Radar dimension as polygons

Once that was done, it was necessary to create **RadarEChartDefService** which would be the service responsible for creating the radar. The services used for this type of charts have 3 main methods:

getSeries: Used to format the chart points according to the corresponding echarts series. Depending on the chart type, it also assigns formatting functions to the chart values, colors, etc. For the radar specifically, each series has 'data' key, which is a list with the polygons displayed, as in figure 27.

getOptions: Same idea as in getSeries, but for echarts options, which contains the series. Usually contains information about the grid, axis (names, formatting, color), etc.

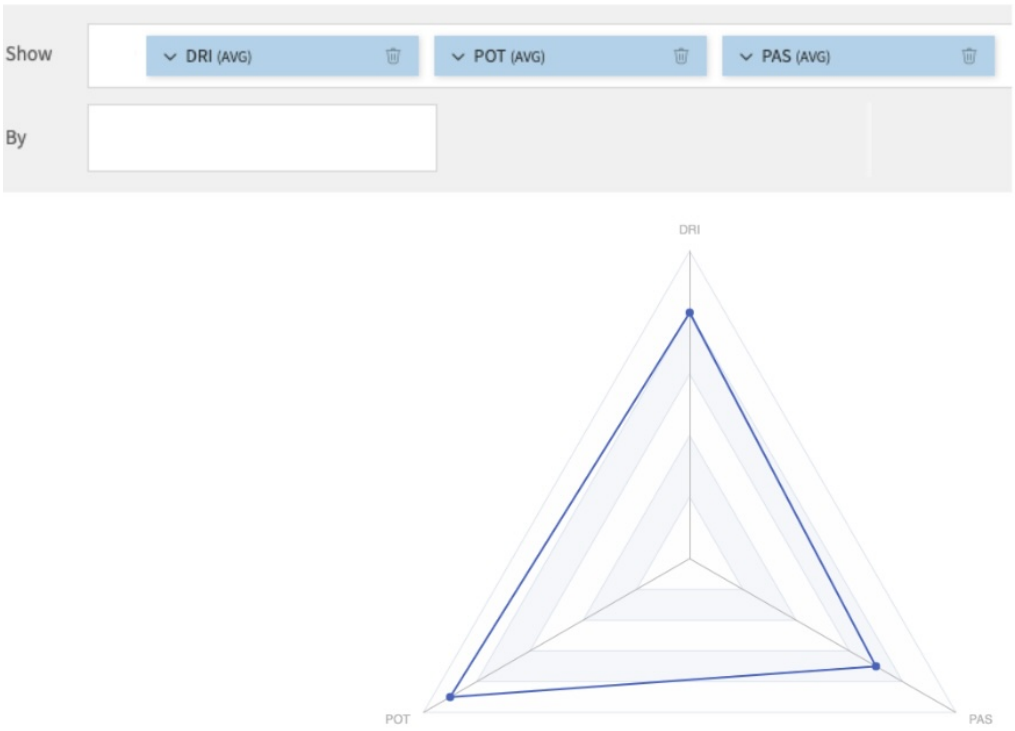


Figure 26 – Radar with no dimensions

draw: Method that calls both getOptions and getSeries as well as organize the parameters for these methods. It also calls *prepareData*, that formats the data for getSeries with a coordinate of each chart point.

There are some other methods such as *getThumbnail* that are not directly related to the creation of the chart itself.

It is import to note that the class I created extends $NonCartesian2DEChartDef$, which also extends $EChartDef$, and both serve as a base for what was created, providing already existing code (abstract and implemented methods) that assist and guide the creation of new visualizations.

After the main structuring was done, and a radar with hard coded data was being displayed, my manager showed me the new chart guide: in the research and development wiki, there is a guide to create new charts that show what files and functions should be modified or created to include a new chart. Using this wiki page, I was able to double-check if there was nothing I missed when creating my chart and if I had a baseline understanding of what was happening.

The reason this was not shown to me earlier is that, from experience, my manager knows that tinkering with the code and having to search where functions and methods are called by yourself will give the engineer a better grasp and ownership of the code in the long term. Following the tutorial is more like manual work, where the first contact with

```
series: [
  {
    name: 'Budget vs spending',
    type: 'radar',
    data: [
      {
        value: [4200, 3000, 20000, 35000, 50000, 18000],
        name: 'Allocated Budget'
      },
      {
        value: [5000, 14000, 28000, 26000, 42000, 21000],
        name: 'Actual Spending'
      }
    ]
  }
]
```

Figure 27 – Basic example of radar series (THE APACHE SOFTWARE FOUNDATION, 2023b)

the codebase must have a lot of thinking and testing.

This fact reinforced that the company has good documentation processes, and a lot of information gathered about everything that was and is being developed.

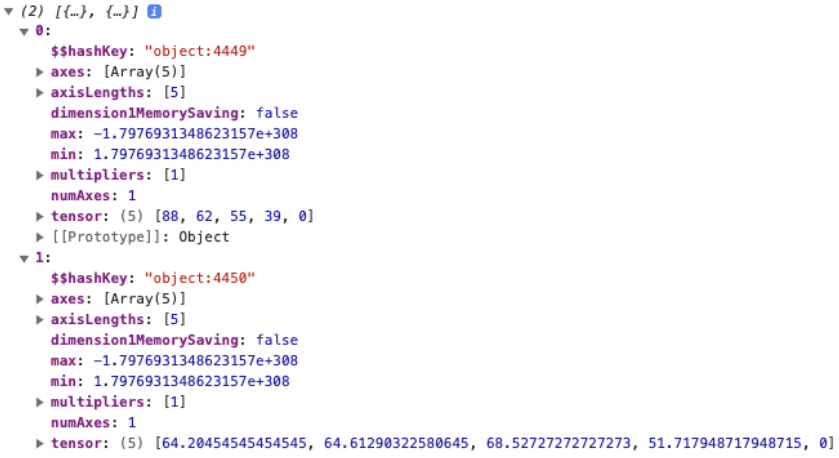
4.5.1 Structuring data

The biggest challenge without a doubt in this step was to structure the data for the radar, while also being able to restructure this data when the polygon source was changed (from measures to dimensions or vice versa).

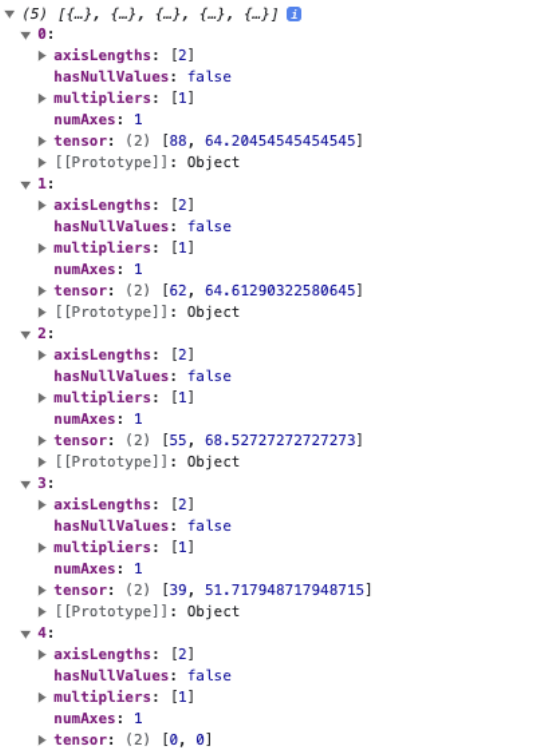
From early stages of development, I was made aware that it would be ideal to have this logic inside the "chartData" object, derived from $ChartTensorDataWrapper$, which was the same class used for other Echarts charts. However, the parameters of this class did not store the information about whether the polygon source was measures or dimension, so the solution was to create another class, specific for the radar, that holds this information and extends $ChartTensorDataWrapper$, so the methods used across the code would be present for it too.

This new class was called $ChartRadarDataWrapper$, and it has specific logic to deal with the polygon source change. It starts by creating a new aggregation in the constructor, whereas $ChartTensorDataWrapper$ used the aggregation from the request, in which the aggregation is kept as is for when polygons are drawn from measures, or the matrix is transposed for when polygons are drawn from dimensions. This operation is based on the "tensor" attribute of each object, which contains the values used to plot the

images. In the aforementioned comparison, it is possible to compare the structure of the original (measures as polygons), in figure 28a, and the transposed list of objects (measures as dimensions), in figure 28b:



(a) Default list



(b) Transposed list

Figure 28 – Default and transposed aggregations

Another implementation that is worth to highlight, is the use of "chartPoints" and *prepareData*. The latter mentioned is a method used to organize the data into chartPoints, where each chartPoint is a value that has a unique object for the key "coord". The main two components of this object in this chart were "measure" and "colors": each one had its own index, and combined they were unique for each polygon point. For the base case, each polygon would have a color and a corresponding color index, whereas the measure index would vary depending on the axis of the point.

Other "coord" components such as facets are used when multiple charts are displayed at the same time, and animations for animation purposes, both not relevant for the radar proof of concept.

For the same reason as the transposition of the aggregations, the chartpoints have to be rearranged for when polygons are drawn from dimensions. Basically, the color index which was used to define which polygon belong to, will be used as the measure index, to determine which axis the point will be in.

Other than those two main aspects, not many changes were made: methods to get max and min of all data were added, some changes to methods that used the aggregations from data to use the newly created aggregations, and a method to check if the polygon source was measures or dimensions.

Being able to reuse functions and code, was thanks to the company following good code practices, creating modular clear code that can be reused in new tools and contexts.

4.5.2 Implementation decisions

At the point where most of the structure and base logic was developed, we had to decide how to implement some details that were postponed to the point when we could better interact and see the resulting chart.

Aside from some minor/clear naming, the first decision was: where to put the button to change the polygon source. There were tree places where most of the settings were, the left bar, the measure settings dropdown and the dimension settings dropdown. As there was no sub-menu dedicated for polygons on the left bar, it was decided to put it in the dimension dropdown, rephrasing the button name to “display as”, with the options being “Polygons” and “Radar labels”.

As for the settings such as if the polygons are filled or not, the labels’ font size and color, the colors of the polygons, etc. it was known that these belong to the left bar, to maintain the organization of other charts. Even tough that was true, some of those did not have existing sections to contain them, for this reason, some new sections were created: “Polygons”, with “Stroke width”, “Line style” and “Fill polygons”, and “Axis”, with the font settings for the axis labels.

That decision ended up bringing back the discussion about where to put the button to change the polygon source. Now that there was a polygons section in the left bar, it would be easier for the users to find the settings there. However, the dropdown down remained in its original position because it seemed more consistent. The only change that happened was renaming the "Radar labels" option to "Axis" to keep the same name as the sidebar.

To reach this point, we had a couple of different versions of the settings. In the image 29 below, it is shown the software used for the iterations and one of the latest versions of the design:

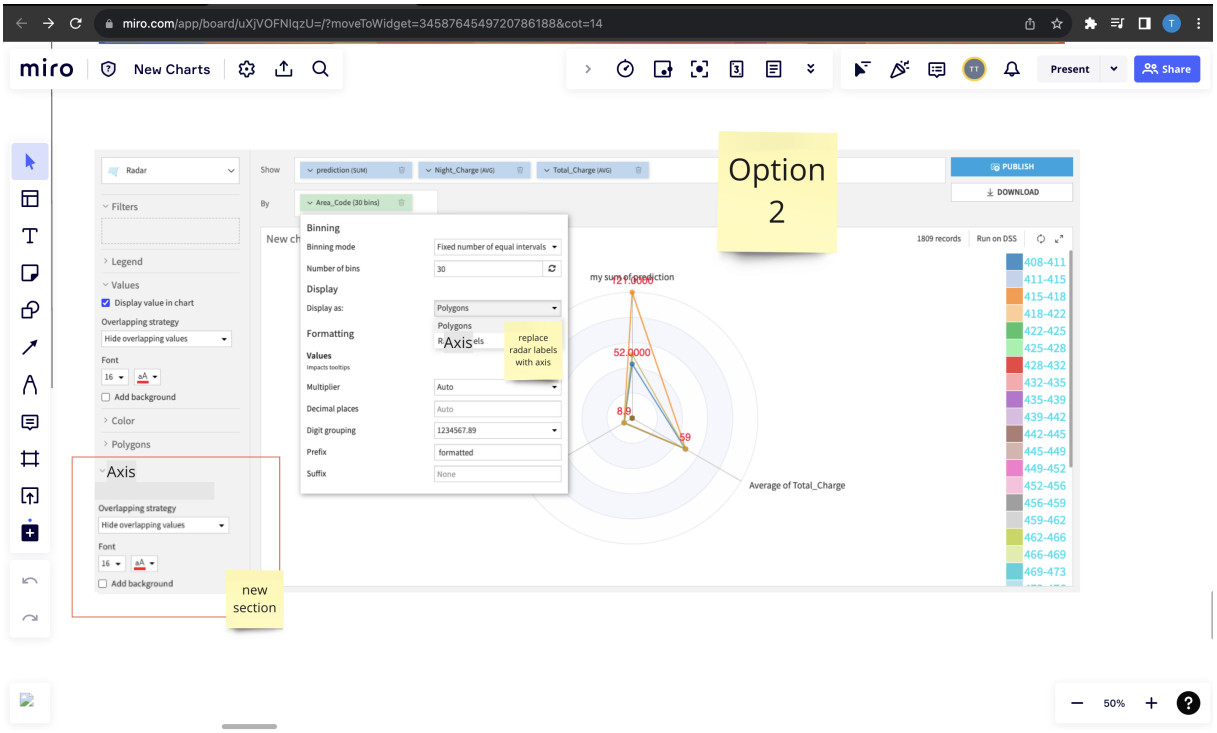


Figure 29 – Miro board with radar design

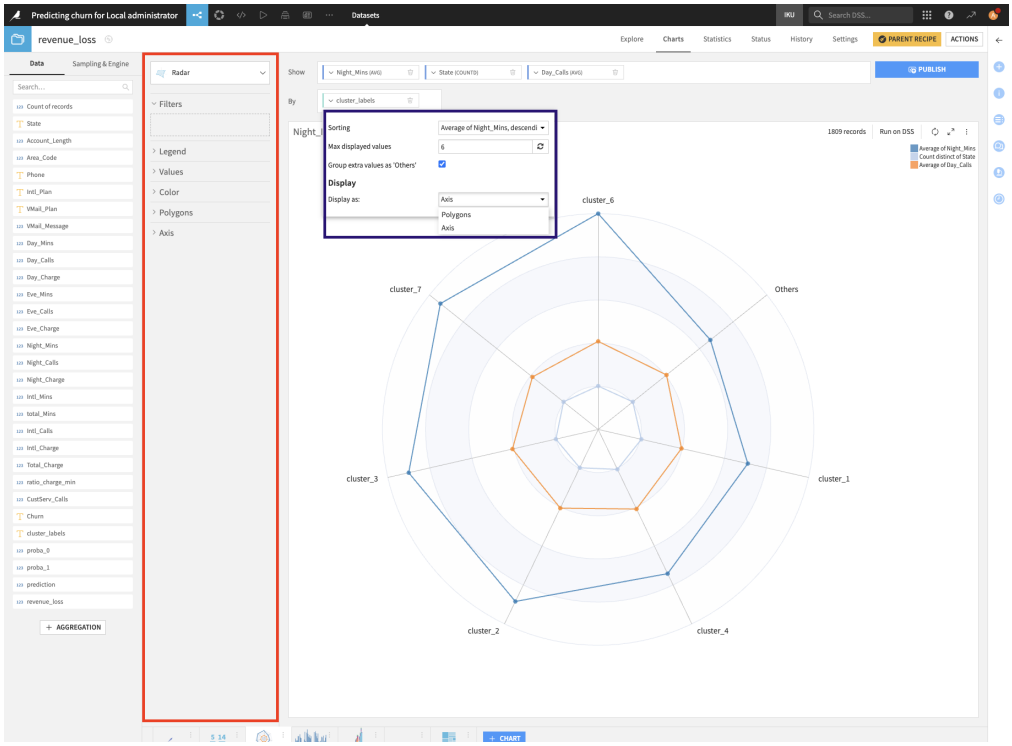


Figure 30 – Radar final settings

The final version, in the image 30 above, of the feature is as follows. Here, the left bar highlighted in red and the dimension dropdown is expanded and highlighted in dark blue.

Other than decisions related to settings, some important points on how to configure

the axes of the chart were set. Initially, we used the same min and max for all axis to simplify the coding process, but we stuck with this decision in the long term. This would provide to the user a clearer visualization when comparing values across axes. The downside is: if axes have different scales, it would be difficult to compare them, and it is possible that axes with different ranges will be poorly represented.

On the same topic, the minimum for all axes would be either zero or the minimum number of all data if it was lower than zero, and the maximum is the maximum of all data. This makes for an easier understanding of the ranges, since we also opted to hide the ticks and values of axes.

For this version of the radar, those options cannot be changed by the user, but once their harms were noticed, a story to add settings to change these was added to a future release. The fact the team was constantly discussing and changing what was defined in previous steps of the SDLC shows a crucial aspect of the agile methodology, the fast iteration and non-linearity of the development.

4.5.3 Working with git

After a few weeks, I had a better grasp of the code and my managers were already able to better plan a release version for the radar chart. So, considering the period of developing tests, verifying the feature and finishing the development, the planned version for this chart was DSS version 12.2, scheduled for the end of august. Since I made my branch from “master”, and this release would be based on the branch “release/12”, I had to find a way to move only my code to a branch based from “release/12”, since those 2 were very different.

To do that, I started by creating a new branch from “release/12”. Then I had to use the git command cherry-pick (CHACON, 2023) in every individual commit I did in the original radar branch to this new branch and solve some of the conflicts that existed. The hardest part was that some of the functions that I used were not available in “release/12”, so I had to adapt some of the code to what was available.

Working with numerous branches and pull requests (PRs) in a GitHub repository can become complex, especially when there are interdependencies between branches and changes need to be propagated across multiple branches. At some point, the main branch of the radar feature had 3 derived branches from it, and consequently 3 PRs targeting this base branch.

The reason for this happening is that the chart itself was basically ready, but some other features would be desirable, such as options to format the axis (font size and color). This kind of feature is ideally developed in a new branch, so that the base branch can still be reviewed and tested without new code being added, given that this code can change or

break the main feature.

As a result of this many branches, one change in the “release/12” that affected the radar chart made me spend some time merging “release/12” into the base radar branch, then merge this branch into all the feature branches. Other than that, it was necessary to merge the base radar branch into the features other features once one of the features was merged into the base chart branch. The diagram 31 below is a good representation of this situation, where “release/12” is a release branch, the radar base branch is a develop branch and the feature branches are the features developed for the radar.

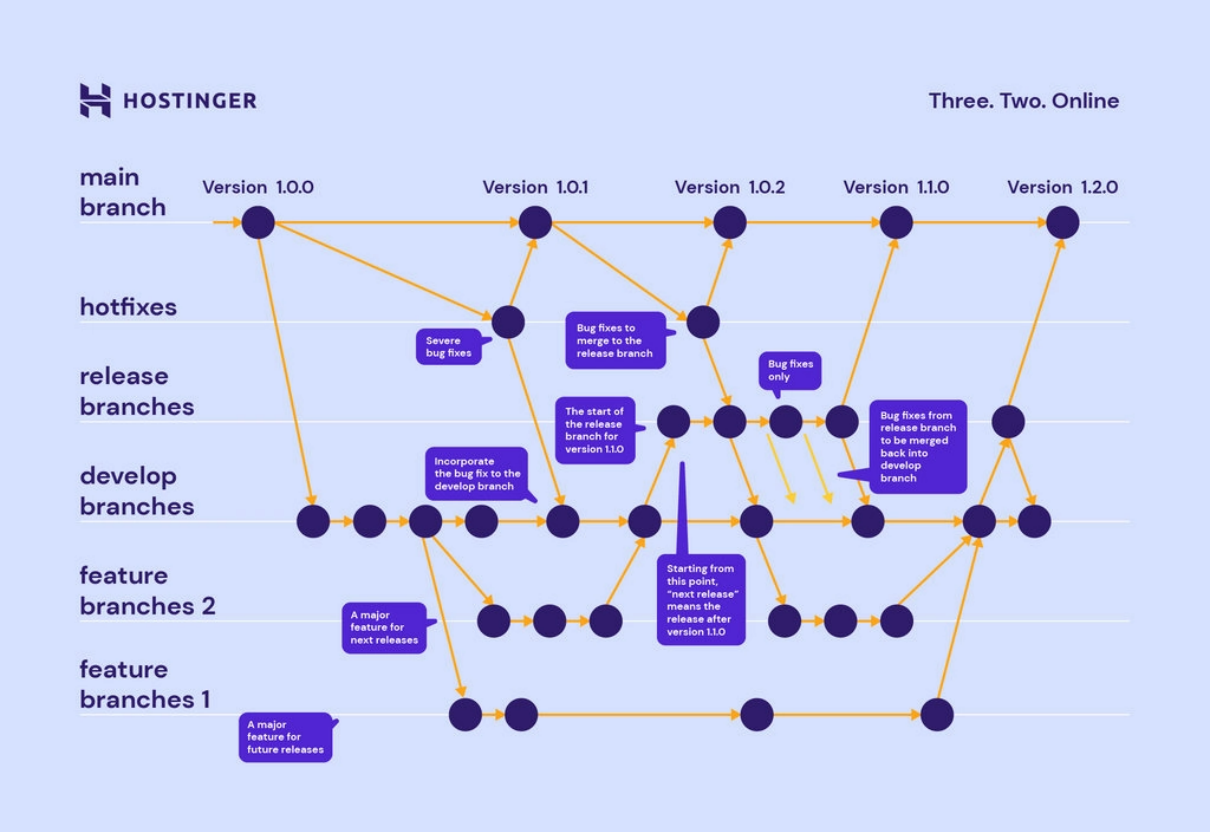


Figure 31 – Diagram of git branches (S., 2023)

4.5.4 Echarts problems

One limitation that became evident while using echarts is that it is much more prone to bugs than d3, due to its complex charts and customization options. This problem was noted after the core part of the radar POC was already finished, and we were performing tests with different datasets and columns. Theo noticed a bug in the implementation where with long axis labels, the text displayed was not truncating, so it would just have a part of it missing (as shown in the figure below). This should be an easy fix, just adding a ‘truncate: true‘ parameter should fix the problem.

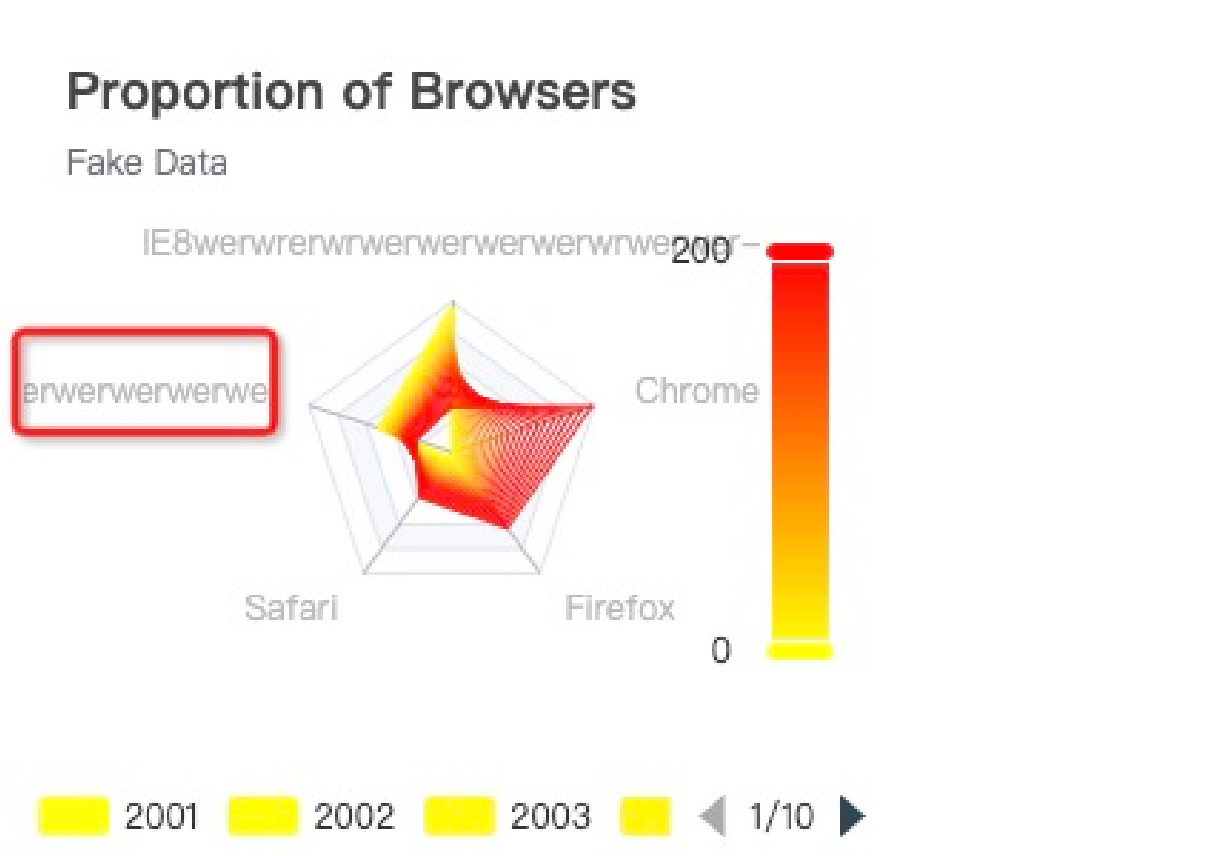


Figure 32 – Label not truncating for radar (JIANYF-JIANYF, 2022)

However, this option was not working in echarts, as seen in 32, and it has not been for a while. According to this github issue (JIANYF-JIANYF, 2022), the bug seems to exist for at least a year and a half, and it has not been solved yet.

Since Stories already had the radar chart, a similar solution was developed, in this case a *truncateAxisNames* function. The logic is quite simple, get the largest space a label can occupy right where the chart would be cut in half (because this is the smallest space a label can have). To achieve this result, basically the function will calculate how much space each character is taking in pixels, according to its font size and font family, calculate how much space is available o insert the label, based on the total canvas size, the percentage occupied by the radar, the grid margin and the legend size, and then truncate the label according to it.

```
this.truncateAxisNames = (
    label: string,
    formattingOptions: {
        fontFamily: string,
        fontSize: number
    }
) => {
    // Same truncate logic as in stories
    const padding = 6;
    const maxLabelSpacing = (
        (
            chartWidth -
            Math.min(chartWidth, chartHeight) * ratioOfRadarRadius
        )/2 -
        (gridOptions.left || 0) -
        (gridOptions.right || 0)
    ) - padding;

    const labelWidth = this.stringUtils.getTextWidth(
        label,
        `${formattingOptions.fontSize}px ${formattingOptions.fontFamily}`
    );

    const truncatedLabelLength = (
        label.length *
        maxLabelSpacing /
        labelWidth
    );

    if (labelWidth <= maxLabelSpacing) {
        return label;
    }

    return `${label.slice(0, truncatedLabelLength)}...`;
};
```

It is noteworthy that for this piece of code, and for all other code extracts in this article, the formatting may have been altered to fit the code in this format, but strict formatting rules were always enforced during the implementation.

Nevertheless, even after putting this idea in place, sometimes the labels would still overflow a little bit, they were truncated, but apparently not truncated enough. After talking to the team, it was found out that the calculation of the legend size suffers from a race condition, where sometimes $chartWidth$ and $chartHeight$ do not include the legend before the chart is drawn, which leads to this issue. As this was a general problem that had to be dealt with, it was out of the scope of the radar chart, so it was decided that this truncate function was sufficient for the moment.

Other than that, there was an expected downside of this solution: labels of axis that were not in the position with the lowest amount of space available would be truncated anyway, even tough they have more space than the others, like in the radar chart 33 below, where this happens for the top and bottom labels.

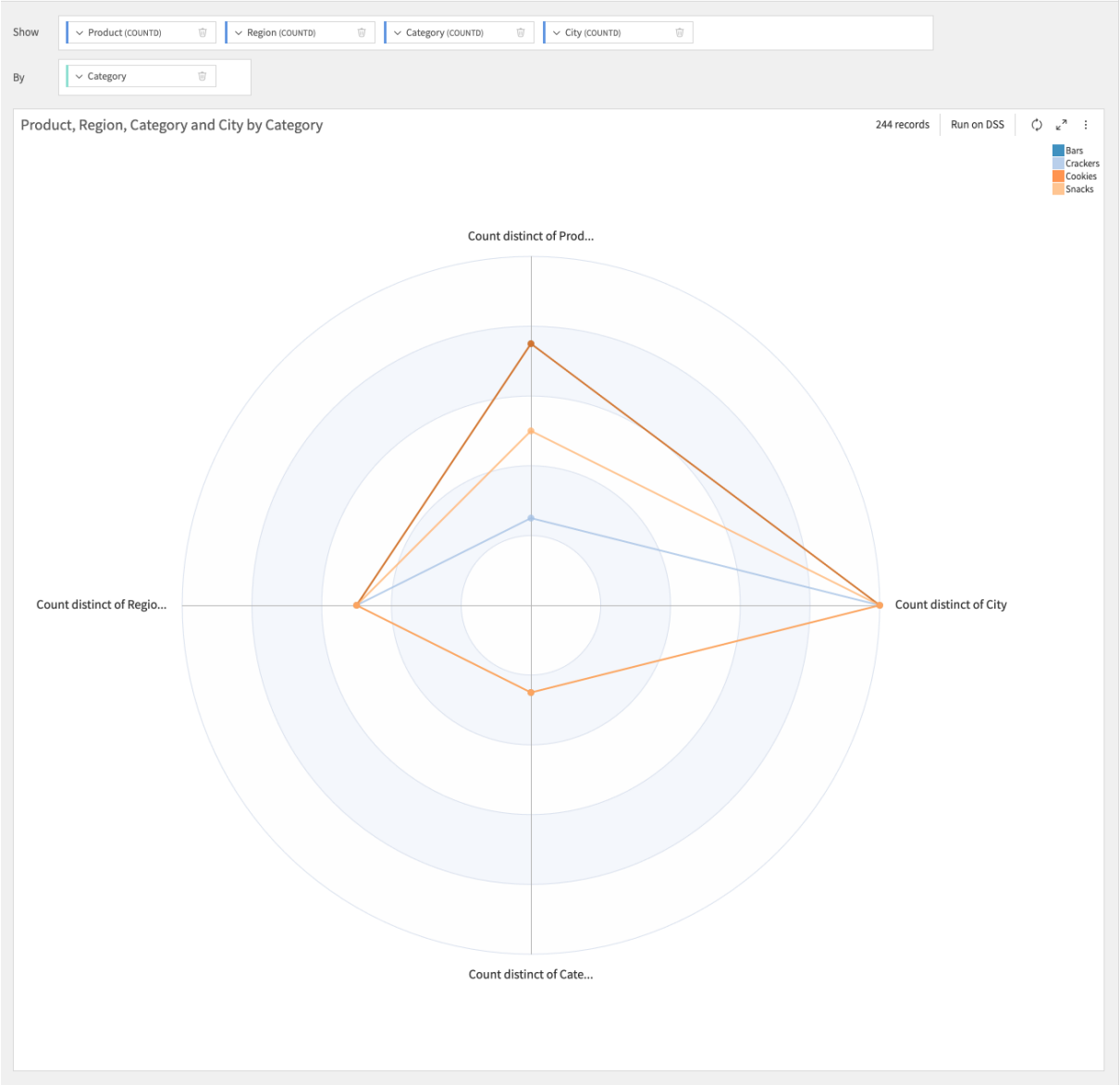


Figure 33 – Truncated labels for radar

4.5.5 Tests

As mentioned previously, there are two types of tests, they are created separately and have different purposes.

There was a bit of set up needed to run the tests, but nothing as complicated as setting up the DSS developer instance to run. Like the aforementioned process, this setup included special steps for ARM macs, which included using specific versions of docker images due to compatibility issues. Overall, the process included using the Intel version of the terminal (x86) to set up a 3.6.15 virtual python environment, downloading the dependencies in this environment and running a chrome, a selenium (python library for automation) and a PostgreSQL (open source relational database) docker containers. As my machine had some compatibility issues with the docker containers (probably because I got one of the first m2 machines of the company), I had to run integration tests with the flag ––driver chrome, running it on my native chrome, which made the tests much slower. This could be a problem if I had to run tests very frequently, but since I only had to create tests for the charts I finished, or when finishing PRs, this did not seem to be a issue worth spending a lot of time solving.

It is important to highlight the importance of running tests before commits. To make sure a change made in the code for a feature did not break something else, programmers are incentivized to run a test every time before a commit is pushed. Furthermore, a github actions pipeline was implemented to, every time a PR has a new commit pushed, tests are run in a server to assure that everything is working as expected. This is also important on the verification process of stories (tasks) in shortcut, the person responsible for verifying a story must run all tests before marking the story as verified.

4.5.5.1 Unit tests

The idea of these tests is to compare the resulting options of $RadarEChartDefService$ to a set of options that were known to draw the desired chart correctly.

Basically, to code these tests, we get the request from the server with the chart data (from the endpoint *get-pivot-response*) in the network tab of chrome's developer tools, get the raw data from $chartDef$ and call $RadarEChartDefService$ with this data. Then a function is created to compare the resulting echarts options with a correct data that was previously defined, and verify if the formatting options were correctly set.

As some tests are older than others, the radar test was created based on the structure of pie's test, which is a rather recent test. This lead to a question if we should test the resulting options by themselves and then test the value formatting, as it was done for the pie, or do both in one test. It was decided that it would be more practical and simple to test both of the in the *validateOptions* function.

One complication that exists for the radar is that it is basically two charts in one, because it is possible to change the polygon source. To account for that, there are 2 tests for each scenario, one for measures as polygons and one for dimensions as polygons. Other than that, the functions created for the tests also had to be adapted for this, for example the function that compares the formatting of the values should compare them by polygons they are derived from measures, and by axes when they are derived from measures.

This newly created tests concerned the corresponding chart, so they were all in Angular. For this reason, I was mainly running the karma start ––browsers Chrome command in the *dip/src/main/platypus* folder for these tests, which runs the Angular tests. However, when finishing a PR or committing, I also ran npm test in *dip/server/src/frontend*.

4.5.5.2 Integration tests

As these were my first tests, I spoke with my manager, got a test example and immediately started to replicate the test that was done for my chart. Using selenium, a python library to interact with user interfaces, the idea was to find the element one was supposed to interact, such as the columns that would be used, the area that they were supposed to be dropped, the dropdowns and their respective desired options, etc., interact with them, wait for the resulting chart to be displayed and compare it to a screenshot.

To achieve this kind of interaction, selenium would have to locate the elements based on a propriety of the html tag, such as class, id, tag name, etc. For the example I took, class names were used for this purpose, always following this pattern: "qa_charts_component name", like shown in the screenshot 34.

```
<h2 class="foldable-title qa_charts_polygons-submenu-header" ng-click="optionsFolds.polygons = !optionsFolds.polygons">
    <i class="{{optionsFolds.polygons ? 'dku-icon-chevron-down-16' : 'dku-icon-chevron-right-16'}}"></i>Polygons
</h2>
```

Figure 34 – Class name used to identify html tag

After this was done, the tests were committed to the feature branch, and the QA team started to review them. It was pointed out that the test used as an example was actually an old version, and the new tests were actually structured in a different way.

For the standard test, where the objective is basically to check is the chart is working, it was simply necessary to add the chart type to the list of tested charts and choose what database and columns will be used to create it, so there was no need to select where and what elements in the HTML are going be used, because it was all standardized. The resulting code is as follows:

```
SeleniumRadarChart: {
    "chart_class": RadarChart,
    "dataset": "wine_quality",
    "mult_show": [
        Measure("free sulfur dioxide", "MIN"),
        Measure("alcohol", "AVG"),
        Measure("fixed acidity", "AVG"),
    ],
    "by": Dimension(
        "quality",
        numParams = {
            "mode": "FIXED_NB",
            "nbBins": 6,
        },
    ),
}
```

However, for more specific tests, such as changing the polygon source or the polygon filling, with features that are specific to one chart, the same process stated in the beginning o the section should be followed, but in a different file structure. On the same note, the way html tags were identified also changed, now attributes were used instead of class names, resulting in something like the example in the subsequent image 35:

```
<div class="options-zone" ng-show="optionsFolds.polygons">
    <form class="dkuform-chart-options" data-qa-polygons-settings>

        <div class="control-group">
            <div class="control-label">Stroke width</div>
            <div class="controls">
```

Figure 35 – Attribute used to identify html tag

After these tests were finished, the only thing remaining was to update the reference screenshots for each scenario. This is done in a tool created by the QA team, that compares the old and new screenshots, and once the user selects the correct ones, they are added to a database. In the screenshot 36, it is possible to see the tool, and how easy and user-friendly it was to follow this process.

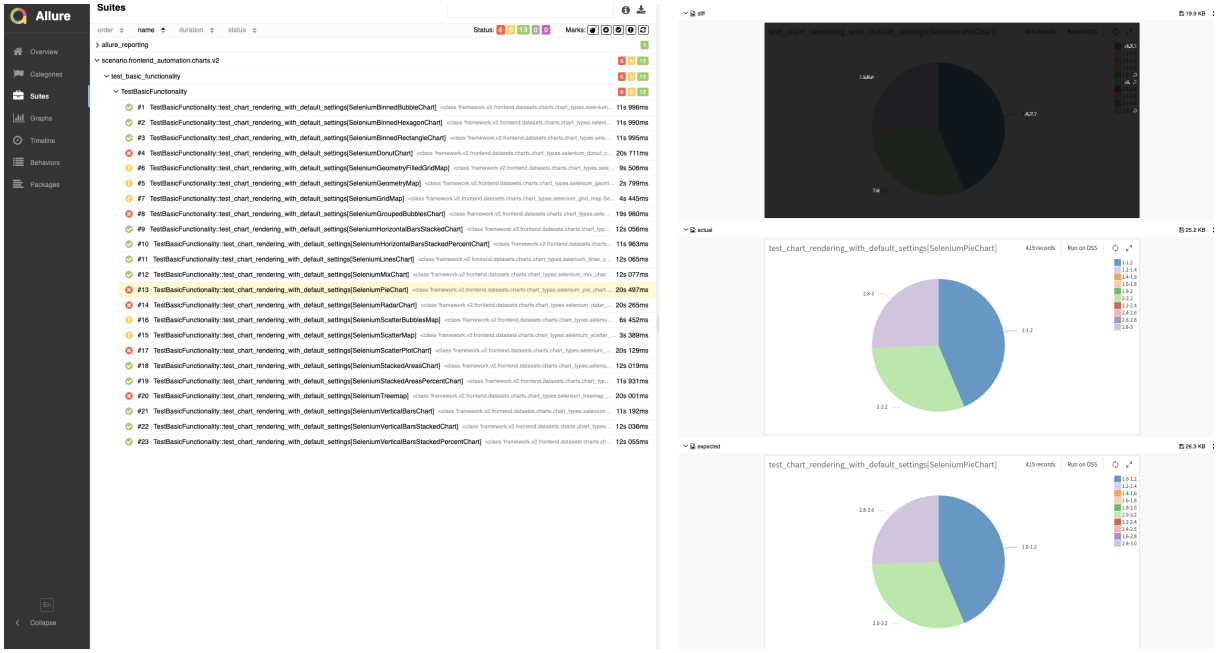


Figure 36 – Screenshot update tool

In the end, I was made aware of the process to create tests, that should start by a couple of meetings including the QA team and the feature developers to align what scenarios should be included in the test specification and how they should be approached.

4.5.6 Bugs

In our continuous pursuit of enhancing the radar chart functionality, our development process uncovered a few bugs that required prompt attention. Following a robust approach, akin to the PDCA cycle (LEAN ENTERPRISE INSTITUTE INC., 2023), we diligently identified and rectified these issues.

One of the details that I ended up overlooking during the development of the radar was the title of the chart. Thankfully, during the verification process, another engineer noticed that the new radar charts ended up with “New chart” as their title, and opened a story in shortcut, starting the "plan" phase.

Following this phase, there is an analysis of other charts, to check if this problem has been faced before, and if an already existing solution could also deal with the problem.

In this case, the fix was relatively easy, new charts have their names defined by the **computeAutoName** function in *chart-type-change-handler.service.js*, so the "do" step was simply adding a condition for the radar in this function fixed the issue. The result is the following 37:

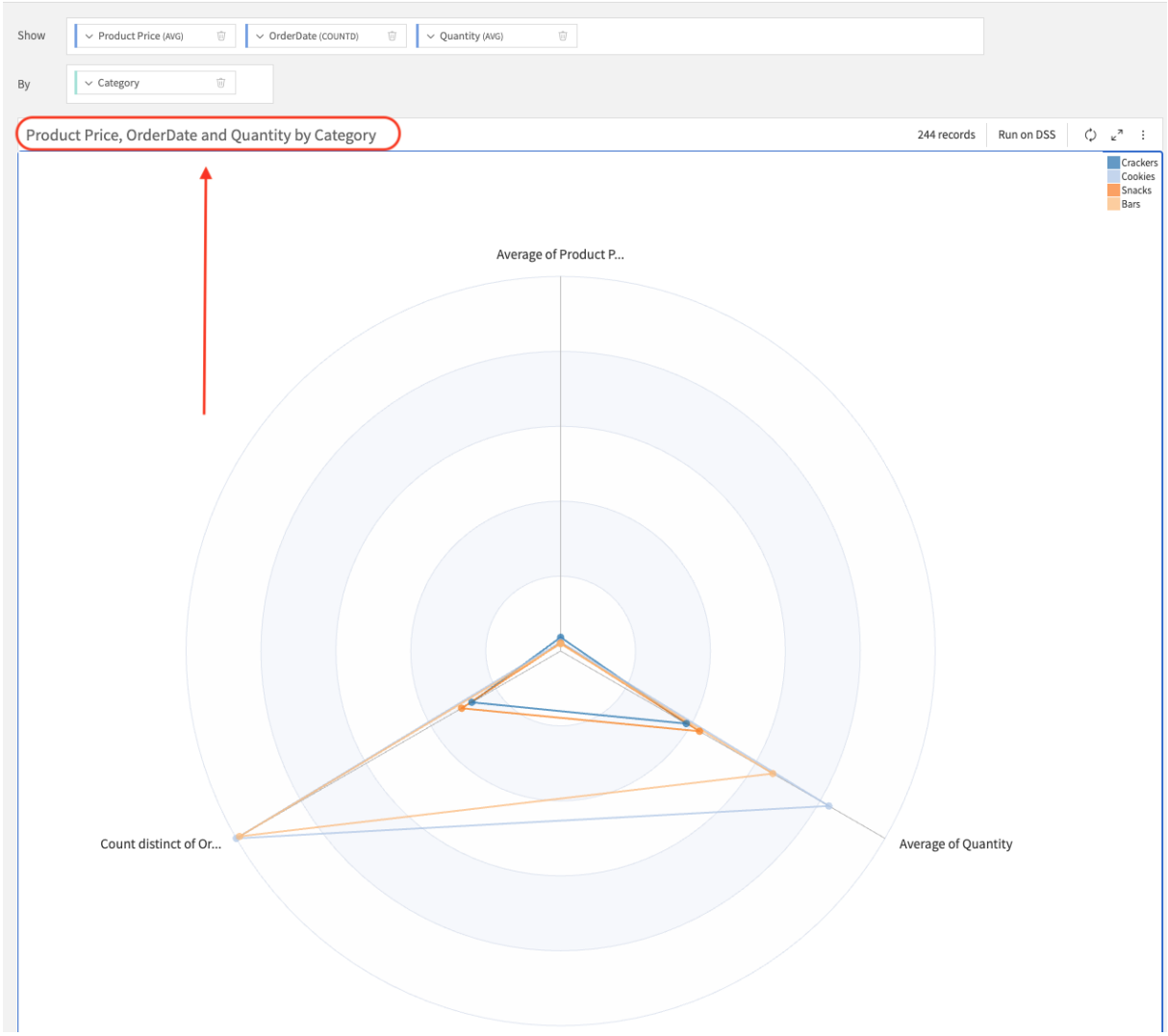


Figure 37 – Radar with generic new title

Then for the checking, me and another engineer responsible for the PR would test if the bug was really fixed, in different conditions, which was rapidly done.

Finally, the last part, act, is more of consolidating this knowledge, documenting where the bug was, why it was caused so it does not happen again, elevating the quality standard.

Other than that, there were 2 notable bugs that were found by colleagues: filtering using the tooltip when the aggregation function was count (either distinct or non-null) would give an error, and in the dashboards there was a “show x-axis” and a “show y-axis” options, which did not make sense for the radar, since it does not have either of those axes.

The planning phase was kept the same, so, in terms of the code necessary to fix them, both had simple solutions: hide the filter icon for when the aggregation function is count, and add the radar to the list of charts that do not have x/y-axis. However, after further inspection, it was noticed that the filter in the radar tooltips were for measures,

where for other charts the tooltip filters were always for dimension.

This fact lead to a discussion where the conclusion was that it is not that useful to filter dimensions (no matter if they are axis or polygons) for this specific plot. As a result, to streamline the interface and coherence for the platform, it was decided to remove the tooltip filter for the radar.

4.5.7 Lessons learned

Taking into account the last section, it is clear the importance of the code, review and verification workflow put in place trough shortcut. With a product this size, it is almost impossible to take into account every single use case and interaction of a feature that is being developed. In the specific case mentioned in this report, even with the welcome introduction to DSS, and the courses in Mindtickle, I did not know of the existence of dashboards until the middle of the development of the chart, a fact that contributed to the overlook of the dashboard chart settings. Without the solid quality assurance framework and the dedication and attention of all the parts involved, the issues mentioned would go unnoticed.

Equally important are the tests, that are more valuable for mistakes made in the code than for interface and usability. Again, for a codebase of this size, it is very hard to predict what effects some changes in the code will result in, so tests make sure that a change made to suit the radar chart for example did not break other charts, since the idea is to reuse components and create code that can be shared between different parts of DSS. It is not realistic to test absolutely everything every time a change is made, so they help the developers to assure that everything else remained unchanged. In my case, there was a moment that I mistyped a condition and made tooltips filters disappear for all charts, not only for the radar, and when I ran the tests locally they pointed out that there was a mistake in the tooltips of all charts.

My knowledge and mastery of git and github increased, and now I can appreciate its uses in large organizations not only as a tool that helps store and manage code, but also as a management and organization asset. Some functions like cherry-pick and squashing commits are not that often brought up when studying it or when doing a small project, but they can be essential when working in a company context. Subsequently, the importance of PR to organize and verify code of different branches was clear, but now I can see that it is also pivotal to follow what one is doing, by creating a PR with a proper description while working on the head branch (source of the changes).

On the same note, the importance of a clear communication and guidelines to be followed stood out to me during this period. The existence of the wiki was very important in the beginning of this journey, and the ability to search through messages in slack was

also helpful. In terms of the communication, the previously mentioned PRs were very important to receive helpful and updated feedback on what I was doing, as well as to keep conversations relevant between me and my manager. On the other hand, if I had followed the right path to create tests, by meeting with the QA team from the start, a lot of time would have been saved.

Other structures that contributed significantly were the quick and dynamic development cycles using agile and PDCA, allowing me to learn quick, fail quick and be able to get feedback and consolidate a new strategy to tackle the problem. This was essential for a deep comprehension of the system set in place in the company, and all the dependencies used during the programming process.

Finally, at first I had a lot of difficulty navigating on the codebase, which lead me to better understand the tools and shortcuts available in my code editor, Vscode. Another aspect that helped me gave a better grasp on the links and organization of the repository was the wiki, for the folder structure and uses of individual functions/pieces of code, and slack, where I could search some of the issues I faced were already been solved, and if not, I could ask the whole team for help, since someone has definitely already worked on something similar.

4.6 Sankey chart

Differently from the task of developing the radar chart, this time there was no scope, no specification and no wiki to base the sankey chart on. For this reason, I started with the planning and high level design phases of the sdlc, by doing research on how is this chart used, how are its components called, what kind of customization would a user want to have, how is it implemented in other data analysis tools, etc.

The first step was to define what a sankey chart is: "A sankey diagram is a visualization used to depict a flow from one set of values to another." (GOOGLE FOR DEVELOPERS, 2023). Then define what are the names of its components. The clearest resource that I found explaining clarifying this information was chartExpo (CHARTEXPO, 2023), that pointed out the main parts of the chart: nodes, levels and links. Even though the explanation was clear, there was now good visual representation of what was stated, so based on an image from google developers (GOOGLE FOR DEVELOPERS, 2023) a brand new visual representation 38 was created to indicate these components:

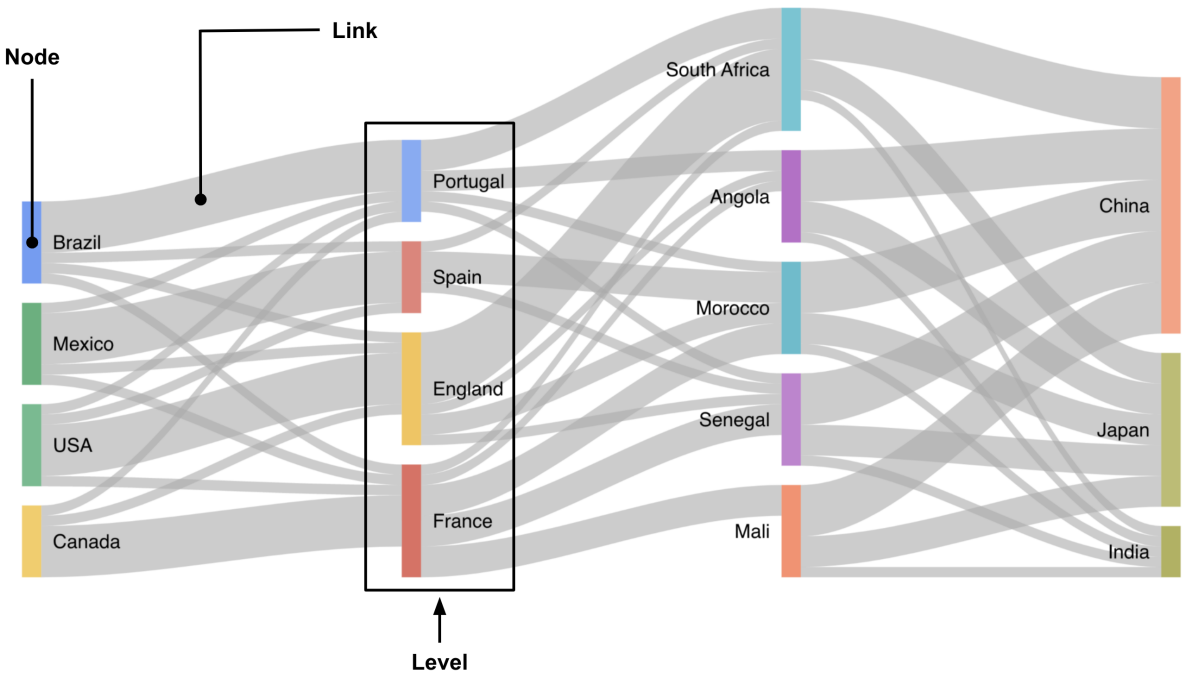


Figure 38 – Sankey terminology

Looking deeply into the same google developers website (GOOGLE FOR DEVELOPERS, 2023), I was reassured to check that the names used were the same. Furthermore, this page has an API for the sankey with its main customization parameters, which could also be mostly seen in the sankey documentation of echarts (THE APACHE SOFTWARE FOUNDATION, 2023d). That also helped to straighten the scope of the feature.

After gathering this information, I proceeded to write the wiki (the documentation) for this feature. The template used was the same as for the radar, which included: the

objective, the terminology, the scope (possible configuration of measures and dimensions), the features/settings, some links and a "where to get more information" section, naming the slack channel and the people working on the feature.

Initially, for the chart configuration scope, I thought about doing 2 possible settings:

- 0 Measures + n dimensions ($n \geq 2$)

- 1 Measure + n dimensions ($n \geq 2$)

For the first case, as there was no aggregation function and no weight column, the weight would simply be the count of records, how many times the links appeared. As for the second one, the thickness of the links would be determined by the measure and the connecting nodes.

Once the wiki page was set up, an alignment meeting was scheduled with the engineering manager, the engineering lead, the TPM and my direct manager to mainly discuss the scope and features brought up by me.

There was a lot of discussion around the fact that some aggregations functions such as average, min, and max ended up not being as coherent as others like sum and count non-null because the sum of the incoming and out coming links would not be the same as the value present in the node. Momentarily, it was decided to leave all aggregations functions available for the user, and in a second moment we may remove them if confusion happen. On the same topic, it was pointed out that, for the same reason just stated, it would be ideal to set the default aggregation function for sum in this case, but it would be incoherent with other charts, so the default function was maintained as average.

Another discussed topic was the scope. Differently from before, we came to the conclusion that the setting with 0 measures would be confusing to the user, considering that sometimes we would use a dimension, sometimes we wouldn't. That said, it is still possible to do something very similar to what this case would do, using the column used as a dimension also as a measure, with the "COUNT" aggregation function.

4.6.1 The sankey plugin

Before I could start, it was important to test what was already in place for this chart. There was already a plugin that was supposed to draw the sankey charts, with two variations: static (a standard sankey) and browsable (where it is possible to click in a node and get a new visualization where only the links connected to this node are displayed).

A little contextualization about plugins: they are very similar to chrome extension, they can be programmed by anyone that has access to DSS and are available in a specific store to be downloaded. They are meant to increase the capabilities of the platform

(DATAIKU, 2023d), mainly when there is a niche problem that has not yet been addressed by the research and development team.

That said, some plugins are developed internally aiming to solve a punctual problem. When this happens, the team that takes care of the issue usually is business development, as they have a deep knowledge of the desired feature, of the situation that the client is facing and people with that developed plugins and access to the source code.

The sankey plugin was developed a while ago, most likely in the scenario described above. However, in the limited testing that I did, the static version of the plugin had some critical problems, resulting in it not working most of the time, or only rendering floating labels, as demonstrated in the image 39:

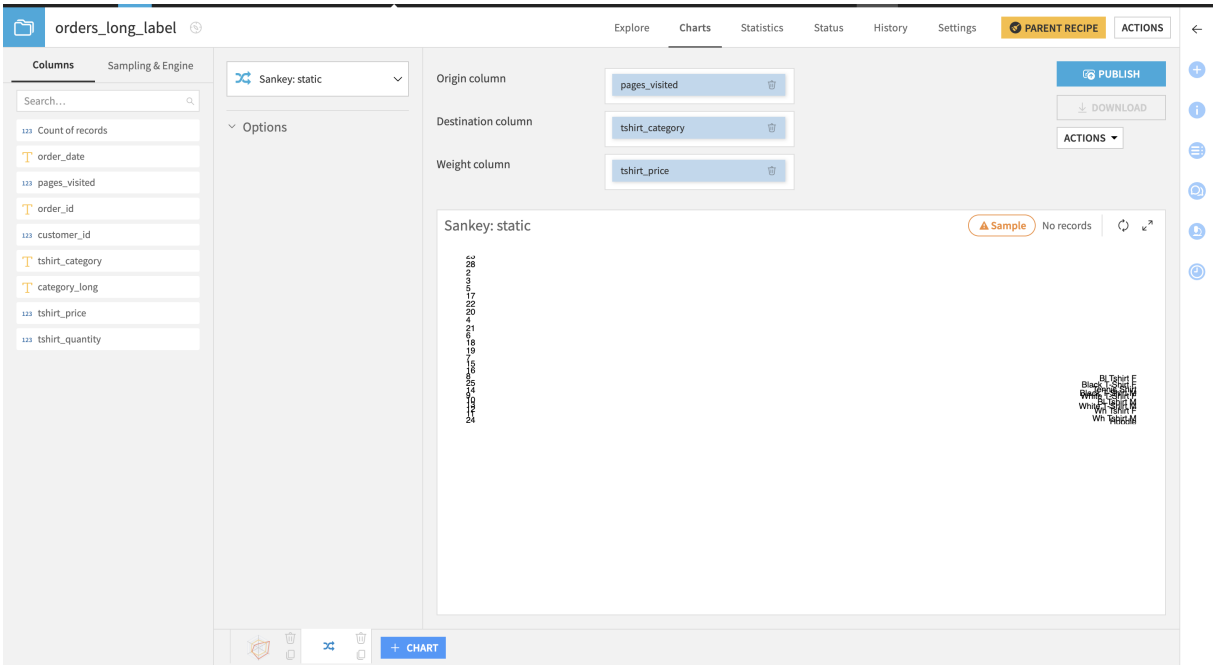


Figure 39 – Static sankey plugin example

The browsable sankey worked a little better, but it still had some bugs where it would not display the plot sometimes. In the following example 40, it is possible to observe the browsable sankey focusing on the "East" node.

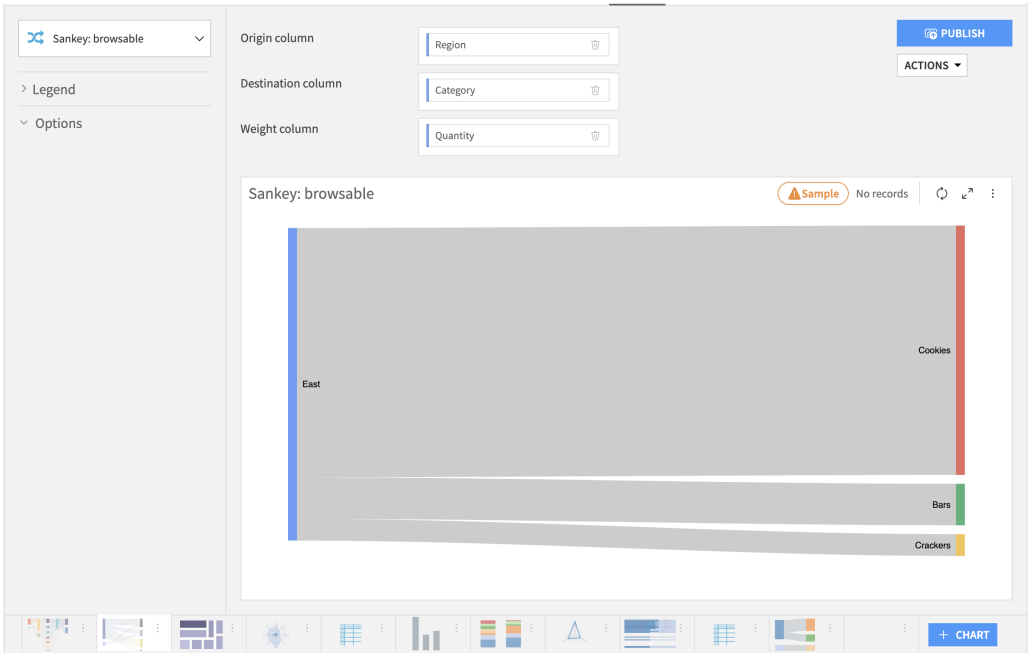


Figure 40 – Browsable sankey plugin example

As a point of comparison, a chart without the focus on the "East" region was created, with the final version on the sankey chart. In this new visualization 41, the "West" node, and the nodes that are linked to "West" but not linked to "East" are also present.

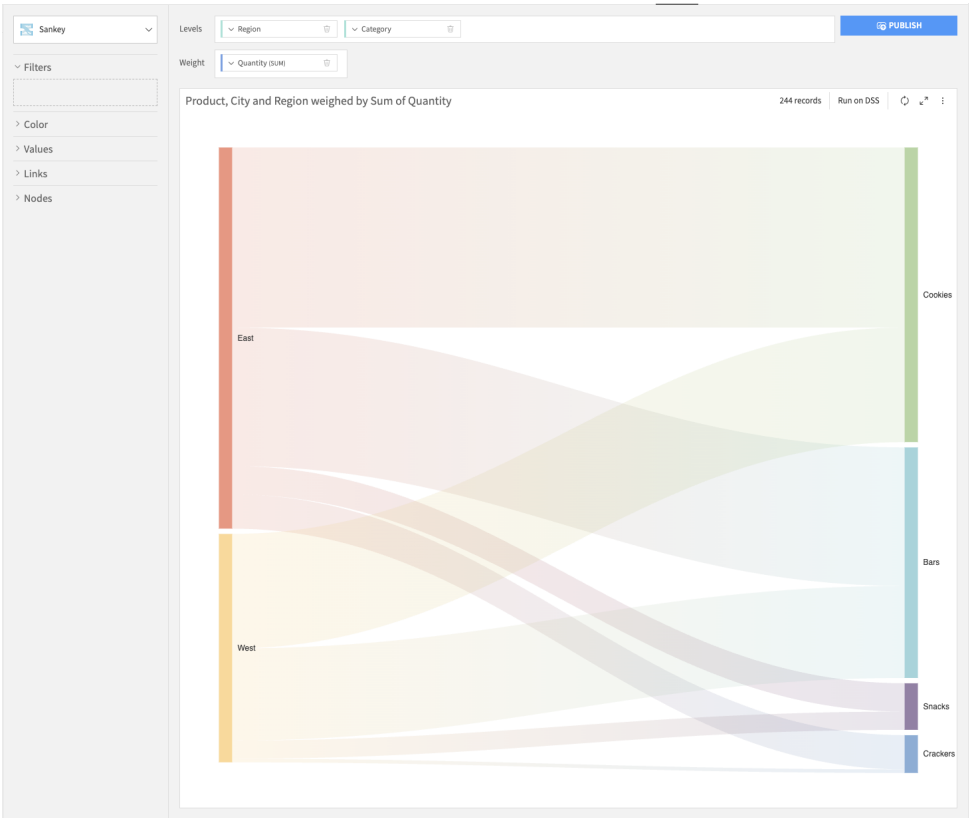


Figure 41 – Complete sankey

With the capabilities of the plugin in mind, the scope of the sankey feature could

be better defined. The main issue that would be tackled by the new chart, other than the basic functionality which did not seem to be 100% working with the plugin, would be being able to have multiple levels in the sankey, as well as have the standard settings that most of the other DSS charts have, for example: color configuration and filters.

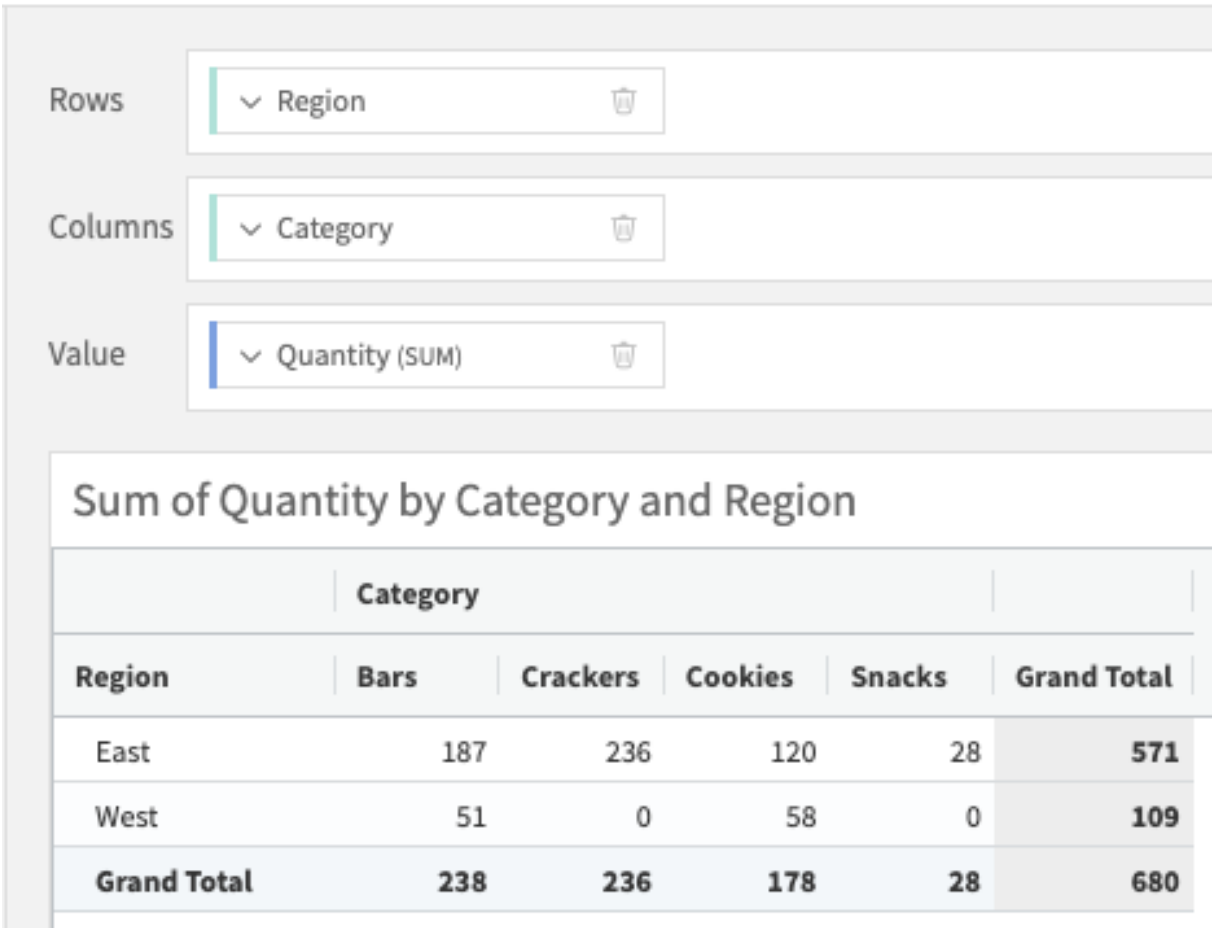
4.6.2 Chart data

The last item needed to be verified before the development itself, was to create a backend endpoint with the data needed for the chart or find what existing request was suited to fill the chart (considering that some manipulation on the frontend was possible).

A priori, the hypothesis was that no other endpoint was outputting all the information needed, and I would look for the endpoint that was doing something close to what was needed, then create a new one adapting it to suit sankey's options. There was also the chance that the plugin would have some valuable insight into how the sankey data should be requested and structured, but this was not the case.

The beginning of this task was to filter off some charts that had nothing to do with sankey's data. So, any chart that included geographic data was instantly excluded, radar, kpi, donut, and pie also did not present any visible similarity to the sankey. Finally, area, lift, line, and bar charts were excluded in this step of the research.

Eventually, the remaining charts were: scatter plot, treemap, binned charts (bubbles, hexagons, etc.) and pivot table. Overall, most of them were not very interesting for the objective in hand, however, the latter chart 42 seems to be very promising. In fact, taking a deep look into the information provided by this table, it is almost equivalent to what the sankey aims to display.



Sum of Quantity by Category and Region

	Category				
Region	Bars	Crackers	Cookies	Snacks	Grand Total
East	187	236	120	28	**571**
West	51	0	58	0	**109**
Grand Total	**238**	**236**	**178**	**28**	**680**

Figure 42 – Example of pivot table

For instance, in the pivot table above, we have the intersection of categories and regions, whereas in the sankey each of those would correspond to a level. Similarly, each region corresponds to a row, and each category to a column, and in the sankey they would all be nodes in their respective levels, so the total of a row/column would be the value of a node.

The example analyzed beforehand is a very simple and basic one, created specifically to show the similarities between the displayed information on the two charts. Some time was taken for this link to be made, because when more columns are being used, it is harder to create clearly see this connection.

As a result, it was clear that the request used for the pivot table would fit the necessities of the sankey. The requests type was "AGGREGATED_ND", and it was the same type used for treemaps, which would be useful later considering that treemaps were also coded recently, using echarts and the newer version of angular, as a result some of the code could be used as a reference for sankey's development and data manipulation.

4.6.3 Creating the chart

Now that this process had already been done one for the radar, it was just a matter of repeating basically the same logic for the sankey, adapting what would be necessary, to get the hard coded visualization. There was also the fact that the new charts' wiki was available, which gave reassurance in the steps that had to be followed.

The main difference to achieve the hard coded chart in DSS was that for the radar, some constants, and icons were already included. In other words, I had to ask in the newly created slack channel '#feat-sankey' for the icons, so that they could be included and assigned to the chart. At first, it was not a problem to use other icons just to avoid error when developing the feature. Here is an example of this situation, where the text is located on the left side of the image, after the "c" node.

4.6.3.1 Data structuring

As previously mentioned, the type of the request used will be "AGGREGATED_ND", which results in the following response 43:


```
▼ result: {pivotResponse: {,…},…}
  ▼ pivotResponse: {,…}
      afterFilterRecords: 10
    ▼ aggregations: [{numAxes: 2, axisLengths: [7, 5],…}]
      ▶ 0: {numAxes: 2, axisLengths: [7, 5],…}
    ▼ axisDefs: [{column: "Category", type: "ALPHANUM",…}, {column: "Region", type: "ALPHANUM",…}]
      ▶ 0: {column: "Category", type: "ALPHANUM",…}
      ▶ 1: {column: "Region", type: "ALPHANUM",…}
    ▼ axisLabels: [[{label: "___dku_total_value___"}, {label: "Bars"}, {label: "Crackers"}, {label: "Cookies"},…],…]
      ▼ 0: [{label: "___dku_total_value___"}, {label: "Bars"}, {label: "Crackers"}, {label: "Cookies"},…]
        ▶ 0: {label: "___dku_total_value___"}
        ▶ 1: {label: "Bars"}
        ▶ 2: {label: "Crackers"}
        ▶ 3: {label: "Cookies"}
        ▶ 4: {label: "Snacks"}
      ▼ 1: [{label: "___dku_total_value___"}, {label: "East"}, {label: "West"}]
        ▶ 0: {label: "___dku_total_value___"}
        ▶ 1: {label: "East"}
        ▶ 2: {label: "West"}
      beforeFilterRecords: 10
    ▼ counts: {numAxes: 2, axisLengths: [7, 5],…}
      ▶ axes: [[20, 8, 4, 6, 2, 0, 0], [20, 16, 4, 0, 0]]
      ▶ axisLengths: [7, 5]
      ▶ multipliers: [1, 7]
        numAxes: 2
      ▶ tensor: [10, 4, 2, 3, 1, 0, 0, 8, 3, 2, 2, 1, 0, 0, 2, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      engine: "LINO"
```


Figure 43 – Pivot request response for "AGGREGATED_ND"

Regarding this object, the list "axisLabels" has lists, where each of them has all the names of the nodes in their respective level, here the first item of axisLabels has "Bars", "Crackers", "Cookies" and "Snacks" and the second one has "East" and "West". It is important to note the "____dku_total_value____" represents the total values.

It took some time, but looking into the request and taking into the account the source data, its structure became clear: the "tensor" list contains the values of

the permutations of axisLabels, in the order they are presented. Referencing again the response in the figure, 10 is the combination of "_____dku_total_value_____" with "_____dku_total_value_____", which is the sum of all "Quantity", 8 is the combination of "East" with "_____dku_total_value_____", which is the total for "East", and so on and so forth.

That said, the object was to manipulate the data in a way to get to the options needed to create a sankey. To summarize, a list of all the nodes was needed, with a list of links with the source node, the destination node and a value that correspond to the weight column. The following image 44 is the most basic example of this structure from echarts' website:

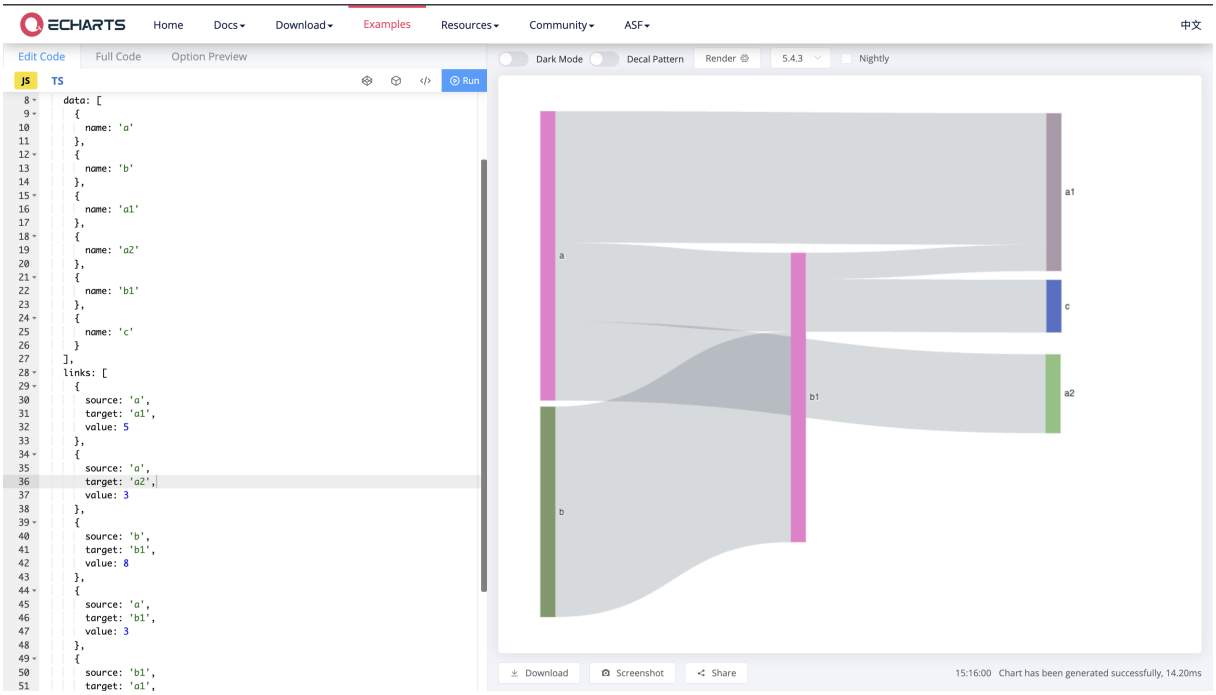


Figure 44 – Basic sankey example (THE APACHE SOFTWARE FOUNDATION, 2023c)

The idea was to just do a simple algorithm to structure the data and improve as we developed the feature if some information was lacking, giving more importance to it in case it would bottleneck the loading time of the chart.

To begin with, to get all nodes, the axisLabels array would be flattened, and all labels that were not null and "_____dku_total_value_____" would be added to a list. For the links, the *getCombinations* function was created, which was inspired by the sliding window algorithm (JAIN, 2023). The idea is to get sequential pairs of axisLabels, and store the indexes of "_____dku_total_value_____" of other axisLabels elements, allowing it to get links from one level to the next, without storing and calculating unnecessary data.

As good as the idea mentioned beforehand was, there were some issues that came with its implementation. First of all, as all axisLabels were added to the node list, even

those with no links (meaning their values on the tensor as 0), the resulting image 45 had some text with these node labels rendering in a corner of the chart.

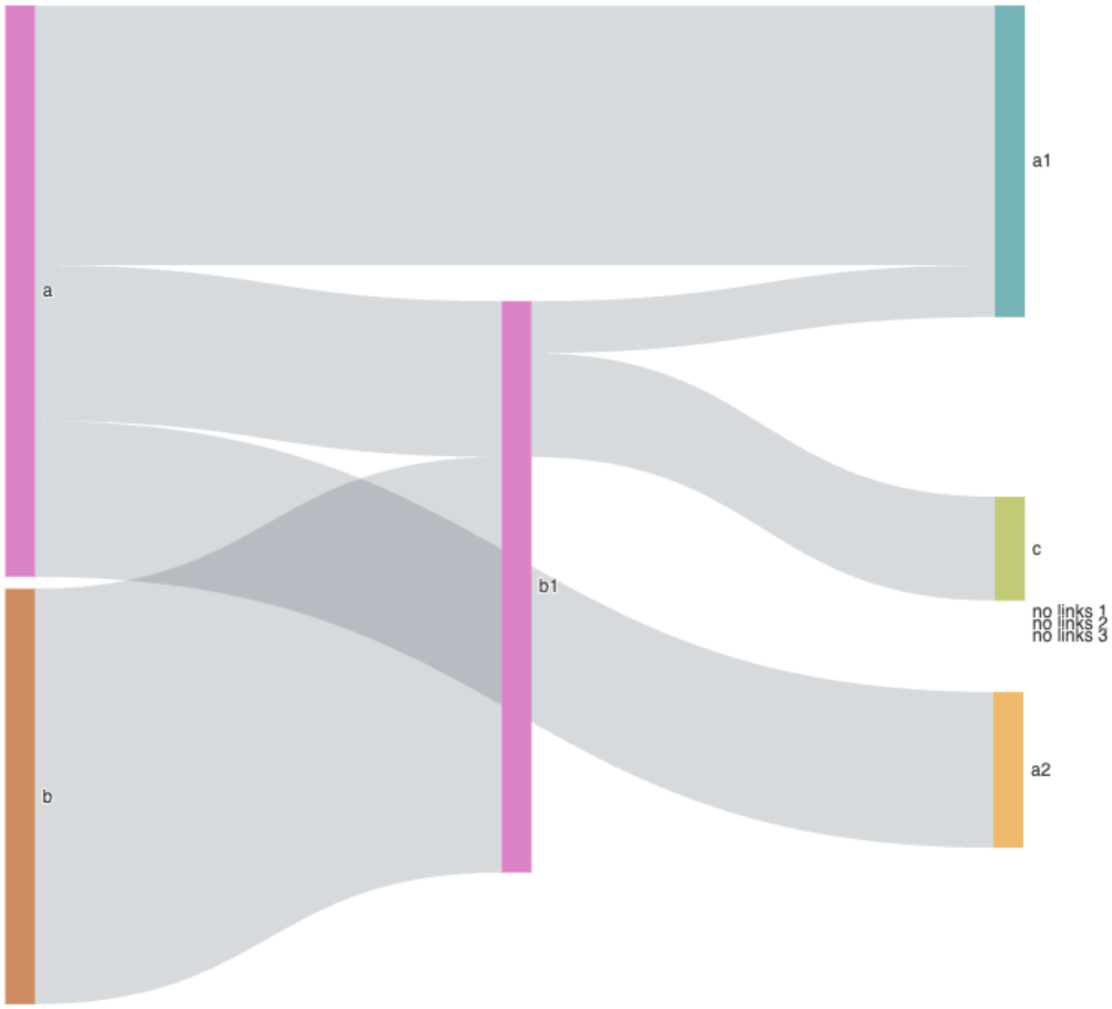


Figure 45 – Sankey with nodes with no links

To solve this problem, an element was added to the node list at the moment a link was created, guaranteeing that each node would be present in at least one link. Even tough this fixed the node with no links' situation, the solution created another problem: the user defined sorting did not work. In brief, taking the image above as an example, if the user set the last level in ascending order (which is "c", "a2" and "a1"), the actual order would be "a1", "a2" and "c", because the order of nodes ended up depending on the order of the creation of links, which is ordered by level. In this case, the first links created would be "a" to "a1", "a" to "b1" and "a" to "a2", which would result in those nodes being added in this order in the node list, meaning they would appear in this order and the node "c" would appear last. Finally, a *nodeIndex* object was created based on the order of *axisLabels* and each of its objects, and used after the nodes were added to sort all them in the user defined order.

Meanwhile, some other improvements in the overall function were made, but they

were just code/readability related, not to solve a specific bug or layout issue. Above all, some new information was added in the nodes and links to better configure the rendered chart and suit the user's need. In the case of the algorithm's loop, instead of shifting the current dimensions from one list to another, a for loop was used, and the current 2 dimensions would be identified from the loop's index.

First version where the dimension data was shifted between lists:

```
const previousDimensions: {
    dimensionId: string; subtotalLabelIndex: number
}[] = [];

// current pair of dimensions
const currDimensions = dimensionData.slice(0, 2);
const nextDimensions = dimensionData.slice(2, dimensionData.length);

while(nextDimensions.length >= 0) {
    const yCoordDict: { [id: string]: number } = {};
    previousDimensions.forEach(
        ({dimensionId, subtotalLabelIndex}) => {
        yCoordDict[dimensionId] = subtotalLabelIndex;
    });

    nextDimensions.forEach(({dimensionId, subtotalLabelIndex}) => {
        yCoordDict[dimensionId] = subtotalLabelIndex;
    });
   ...
   code to add links
   ...

    if(nextDimensions.length == 0) {
        break; // every pair has been added
    }

    currDimensions.push(nextDimensions[0]);
    nextDimensions.shift();

    previousDimensions.push(currDimensions[0]);
    currDimensions.shift();
}
```

Final version using only indexes:

```
for (
    let currentLevel = 0;
    currentLevel < dimensionData.length − 1;
    currentLevel++
) {
    const yCoordDict: { [id: string]: number } = {};
    const firstDimension = dimensionData[currentLevel];
    const secondDimension = dimensionData[currentLevel + 1];

    // Add subtotalLabelIndex of previous dimensions
    dimensionData.slice(0, currentLevel).forEach(
        ({ dimensionId, subtotalLabelIndex }) => {
        yCoordDict[dimensionId] = subtotalLabelIndex;
    });

    // Add subtotalLabelIndex of next dimensions
    dimensionData.slice(
        currentLevel + 2,
        dimensionData.length
    ).forEach(({ dimensionId, subtotalLabelIndex }) => {
        yCoordDict[dimensionId] = subtotalLabelIndex;
    });

    ...
    code to add links and nodes
    ...
}
```

With the different version of the code, another evidence is shown of the ever present improvement, from discussions with peers, new iterations and re-planning/re-designing what was previously defined.

4.6.4 Settings

As for the settings of this chart, it was mostly following what was already used for other charts and using already existing components, such as the "display value in chart" check box and its font + color settings, which were used to display or not link values and configure the text itself. The main difference here were the names used for the submenu sections to show those options.

Apart from these, were 3 main settings that had a specific logic or implementation that are worth highlighting:

The first one is a slider added to control how curved the links are. The logic was simple, using an existing component and setting the limits of the slider from 0 to 1. Nevertheless, some once the test instance was online, it was pointed out that the title used for the slider "curveness", was actually not a word. This was initially used because it is the name of the parameter used by ECharts. Then, after consulting with a native english speaker, the title "curvature" was set.

The second one was the possibility to manually change nodes' position, which was enabled by default. This could be really important for users that want to have a very specific arrangement of nodes, allowing them to move nodes from one lever to another and sort them as they please. Even tough this could come in handy, ECharts had no option to set this manual setting, resulting in no way to save the changes made. For this reason, it was decided to disable this possibility, because once the user refreshes the screen, those changes would be lost, so there was no efficient way to store and share what was done.

The last and most troublesome setting was the ability to change the color of nodes individually. To sum up, the problem was that the color settings were created from the legend, and since the sankey did not have a legend, they were not created at all. Eventually, after investigating deeply, the solution for the problem was to compute the legend, getting all node names, ids and colors into a *legend* property in the scope, and use the "ng-show" parameter to hide the legend itself from the canvas.

Here, in the element 46, we can see the color options with all nodes present:

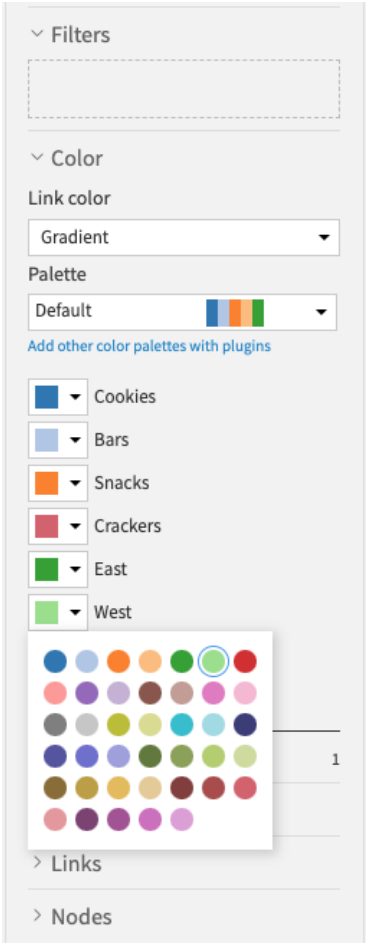


Figure 46 – Node color options in the left menu

There was also a dropdown to set the link color, with "gradient", "source", and "target" as options. As its implementation and uses were simple and intuitive, it was not worth it discussing it further in this section.

4.6.5 Tests

The tests themselves were relatively simple. There were no tests for sankey specific settings, so the process for integration tests was basically choosing an appropriate dataset, the columns for the levels, the column for the weight and create the respective *Selenium-SankeyChart*. It was a similar situation for unit tests, doing the same process as before, but this process was a little longer, due to the many scenarios that are taken into account.

At some point, some changes were made to improve readability and the look of the chart, and the unit tests failed right after them. The reason was fairly simple, the padding values were not updated on the test. This shows again the importance of the tests, that when properly integrated, will identify if some unwanted change has been done.

When the code was first revised, only some small corrections were pointed out, mainly to enclose some code into functions to make their objective more clear, and some

linting fixes. That was quickly taken care of with no apparent issues.

Since what was indicated was mostly code organization, errors were not expected to arise. However, one test kept failing on the github verification flow: “chart features treemap has group, size and color options”. Because of what was mentioned, I thought that it was an issue caused by something else, so I decided to merge the base release branch “release/12.2” into the feature branch.

Immediately after that was done, my manager noticed that I made a mistake, I should have merged “release/12” not “release/12.2”, because the feature was not scheduled to release in release 12.2, and its branch had different commits from release 12s. To solve this problem, the strategy proposed was to cherry-pick all the individual commits pushed in the branch into a fresh branch from “release/12”. To get the commit hashes from all the sankey commits, we tried 2 commands: git log −−author="tulio" −−pretty=format:"%h − %an,%ar : %s" - which gets every commit created by me with its hash and message, and git log −−no−merges release/12 ^perso/tulio/sankey which gets all commits other than merges that the sankey branch is ahead of the release 12 branch.

After all this, still, the same error would happen on the test, for this reason a deeper look into it was necessary. The error was on expectIdToBeVisible('qa_charts_show −dimension−labels−in−chart−input');, right after \$('.measure−wrapper')[0].click();, indicating that this element should be visible after a click on the first measure box.

With that in mind, I figured it out that one of the functions that I created to replace a condition was implemented in the wrong class, and since this was a very specific setting hidden behind a click, I did not notice it at first. When it was implemented in the correct class, I could check in DSS that the element was present for this specific dropdown, for both the treemap and the sankey.

After re-running the test, this specific test was still outputting an error. Even after looking deeply into it, I was not able to find where the issue was. After speaking with Theo, and explaining the details of my line of thought, he pointed to me that, since this was a angularJS test, the function that I created was not downgraded for the tests and I should mock it in a separated file for the test to run properly.

4.6.6 Bugs

When the process above was finished, the chart PR was ready to be reviewed, in the meantime, the sankey was still being manually tested for errors. These ended up missing a major problem that was later identifies in the review: the thumbnails were broken, and when changing a sankey setting, errors would appear in the console.

Thumbnails were not mentioned yet due to their simple implementation, but they are images that represent the chart in the tabs on the bottom section of the page. For the

radar thumbnail, for example, axisLabels and values in the chart (if any) are removed (see image 47 below).

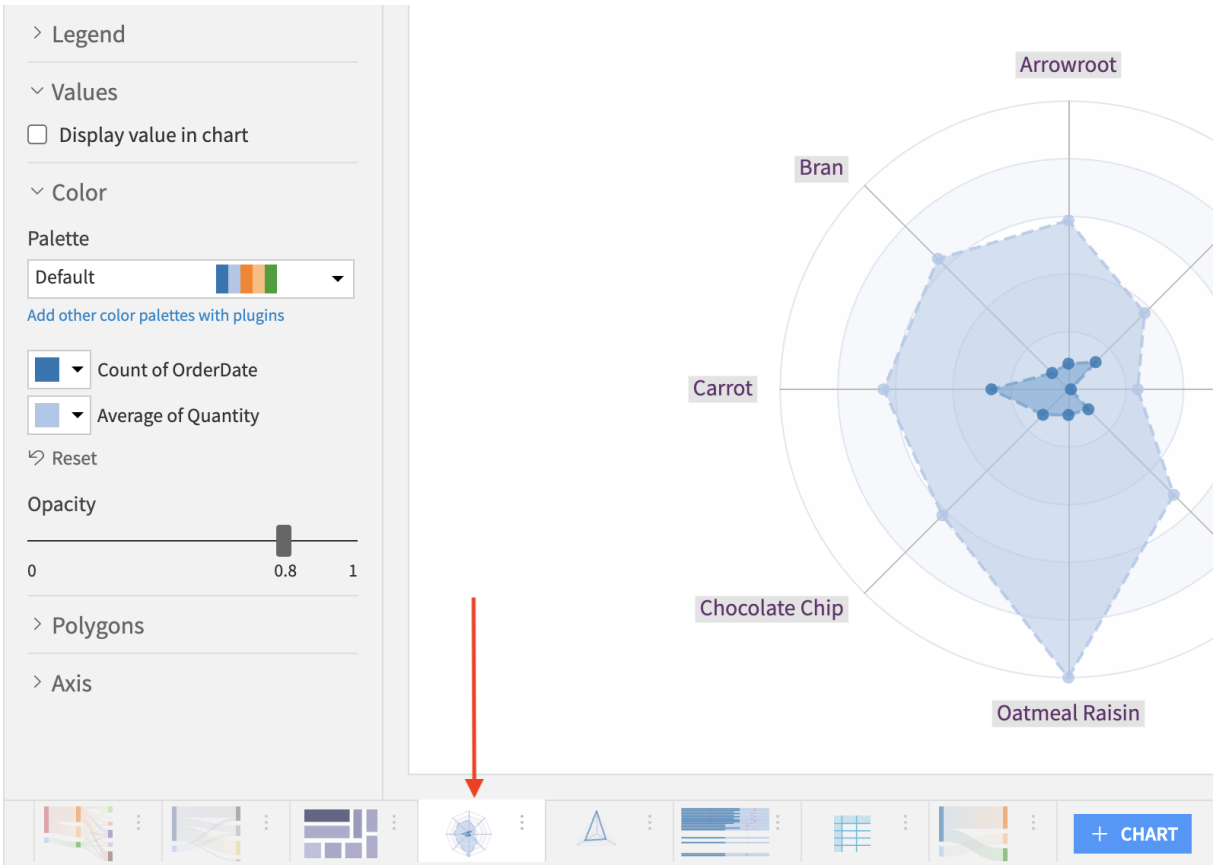


Figure 47 – Thumbnail tab and radar thumbnail example

Code wise, they are also not complicated, in summary, the idea is to keep the basic data of the chart and reduce the size of some elements to make them fit better in the small space available for the thumbnail.

Considering the information that is used to create the thumbnail, it has to be computed almost every time a setting in the chart is changed, that was the reason the error showed when the chart settings were modified. Eventually, during the check, it was discovered that the cherry-picks ended up rolling back the code created for the sankey thumbnail, in a way that the data to create the chart would not be present when trying to create it, throwing an error on the console.

The second issue we had was related to the levels of the sankey. In case there were nodes that were not connected to the previous level, they would end up falling in the first level. This could happen, for example, if filters were applied or when the number of elements in a measure is decreased. In the end, the solution was quite simple, checking the documentation, it was discovered that the argument used to identify the level of each node was not called "level", but "depth", so simply changing the name of the argument made each node stay on their assigned level.

The final bug that was found, was that the fact that the button to download the chart was not working. The error displayed was "The image argument is a canvas element with a width or height of 0". In short, this was caused by the legend that, as previously stated, was computed but was not displayed, so the function to export the chart would try to find the corresponding html element, even tough it was not displayed. To solve this problem, it was simply necessary to check if the chart displayed a legend before trying to include it in the exported image.

4.6.7 Lessons learned

In conclusion, the development of the Sankey chart within our data analysis platform has been a pivotal learning experience for me as a software engineer. This journey has allowed me to solidify what I have learned and expand my knowledge in several key areas.

First and foremost, this project has reaffirmed the fundamental importance of thorough testing in the software development process. Through countless iterations and troubleshooting sessions, I have come to appreciate the value of robust testing methodologies. Testing not only ensured the reliability of our Sankey chart implementation but also served as a valuable tool for rapid iteration and debugging.

Next, the ability to test what I learned with the previous experience in the company and consequently fix quickly discrepancies between the tests and the result has been instrumental in achieving the high-quality results we aimed for. The possibility to apply the process to create a new chart in the platform, whilst being able to dig deeper into aspects that were not fundamental or not taking into account for the radar, kept the experience interesting and fruitful.

Furthermore, this chart provided an excellent opportunity to test my proficiency in Git and command-line tools. As mentioned multiple times during this section, git, and version management in general, were pivotal to ensure the quality of the code and rollback errors when they happened. This experience has deepened my understanding of the role of git in modern software engineering, and allowed the quality standard for version control knowledge to be set higher than before, given that the PDCA cycle of the first project already had implemented actions in the work flow to improve quality.

Moreover, the development of the Sankey chart has given me a broader perspective on the overall process of creating new chart types within our data analysis platform. From scoping and specification to implementation and testing, I have gained valuable insights into the intricacies of chart development. Understanding how each stage of the process contributes to the final product's success has been enlightening.

In summary, beyond only technical skills, I was also able to hone my ability to approach complex software development projects systematically. Solidifying my knowledge,

testing diligently, mastering Git and the command line, and gaining a comprehensive view of the chart development process have been invaluable takeaways from this experience. As I move forward in my career, these lessons will undoubtedly serve as a solid foundation for tackling future challenges in the field of software engineering.

5 Conclusion

Firstly, throughout the text, Dataiku's, and specially the Tayoken team's, adherence to general good practices of software development, and SDLC, is unquestionable. Even if some of the aspects of them are happening automatically, under the hood, where the programmer cannot even notice, they are present and make for an ideal development environment, as a consequence, the software produced is also of great quality.

The case study presented, mostly based on the development of new charts for DSS, actually falls under the category of perfective and adaptive maintenance, given the situation where the software already exists, and new features are added and bugs are fixed, following market trends and business logic (as the implemented charts were user's suggestions, and present in competing platforms).

Even so, due to the magnitude of the software and the size of the operation, this maintenance is actually a whole sdlc in of itself. This can be clearly seen by the steps presented in the previous section, with clear aspects of planning, design, implementation, tests, and maintenance (even if future) phases.

In all of these steps, shortcut is a great tool present to organize, support and document progress. It helps greatly with categorizing and administrating tasks, giving an overview of the teams' productivity, organizing sprint and retros, creating bug reports, etc. There was not a specific point in the case where this was necessary but, it is great to be able to look into shortcut and see when and why specific decisions were made. This is specially useful for with the github integration, given that github is not very user-friendly for these types of people.

Speaking of git and GitHub, they have proven to be not just a good tool, but a must-have in modern development. A version control is necessary, but these together provide a tight integration with task management programs and code editors. Moreover, some advanced commands (like cherry-pick) have immense value when working in a complex environment with many branches, saving time and preventing losses. Lastly, the website's interface provides the necessary information to compare and track code changes over time and in different branches, aiding team cooperation.

In that sense, Meetings and interaction were also pivotal to achieve team-wide fluid interactions. Following some ideas from the agile methodology, weeklies helped the team understand what is being done and more frequent meeting to discuss each project were similarly relevant. Other company meetings had a different goal: make the employees feel they belong to the company and are part of its success.

For the support, Dataiku has lots of tutorials and guides for their users, in formats like video, texts (some cited in 2) and direct human support. This last aspect is not directly mentioned, but is hinted by the fact that the features developed came from user requests, indicating that the company keeps a close channel of communication with their users.

One interesting aspect worth discussing is the Continuous improvement and non-linear cycle of development that derive from the agile methodologies and the startup culture of development. This enables a very dynamic team, where definitions, requirements, and design are constantly changing. Another positive aspect of this approach is the possibility of developing features that interact simultaneously, because there is no need to wait another feature to complete its cycle - as demonstrated by the formatting options functions being updated during the chart development -, a fundamental aspect of a massive software like DSS.

This end up resulting in a not so clear definition of each step of the traditional SLDC. Due to new changes in the code being added during the development of other features, it may be necessary to go back and plan how the current feature will interact with this new code. Furthermore, the iterative nature of the agile team, implies that feedback and suggestion may arrive at any time, which may force a redesign of what was initially thought. Finally, the hands on, fail quickly programming culture culminated in an overlap between the LLD and the implementation, where the functions and interactions are design and tested on the go by the employee.

Overall, the team and the company do work in conformity with what was presented as best practices for quality and efficiency. Following their own version of the SDLC with agile methods and quick PDCA cycles, they are able to achieve a fast-paced environment, attend to user needs, deliver a great product and implements improvements during each step of development.

That said, there is always room for improvement. For example, there are more effective methods of categorizing and doing the triage of bugs, most of them without the need of constant human feedback/interaction. The already mentioned paper (Soleimani Neysiani; BABAMIR; ARITSUGI, 2020), goes even beyond, studying machine learning models on how to evaluate if these systems are effective in detecting duplicate bug reports, based on feature extraction. This and many other technologies could be a great addition to Dataiku software development environment, increasing even more the already existing efficiency.

Nonetheless, there are still points that need a lot of improvement. The main one is the test phase of the sdlc, in particular the communication and documentation. As shown, a lot of time was spent in refactoring and redoing tests, as a cause of lack of organization and clear processes. In addition, there were points where time was spent just waiting for the QA team to review the tests, wasting development time. This may have been caused

by bad timing, because the team was re-building the test codebase nevertheless, this could be handled better in the future.

There is still the issue with maintainability over time that cannot be prevented or detected by those tests. The fact that 2 different Javascript frameworks (Angular and AngularJS) are still used is a testament to this fact: as new technologies emerge, the support for old languages and frameworks may end. This ranges from security updates to performance improvements, both resulting in an obsolete product from the client's point of view.

Considering these changes, the company has to structure better processes to deal with modification in the software development market. An effort to migrate Javascript frameworks exists, but it should be better structure and adapted to all technologies of the company in case of need. In this case, the compliance aspect may be relevant, given that AngularJS is not officially updated anymore.

On a more personal level, the whole experience of doing the double degree program and this internship was very enriching. During this period, I was able to interact with people from all over the globe, and learn about their culture and habits, a great lesson for culture humility. Furthermore, talking to these people every day made me improve my communication skills massively, not only in French, but in English, given that some of my peers were not fluent in French.

In terms of hard skills, the work performed in the graduating was completely different to what I had done in my home university, due to the different emphasis, while also applying different learning methodologies. For the internship, being able to work in a company with thousands of employees and an enormous code base was a first, and allowed me to have a completely new look into how codebases and teams should be structured and organized

Bibliography

ALAM, A. *Importance of Software Design*. [S.l.], 2019. Disponível em: <https://medium.com/swlh/importance-of-software-design-7ffea48ede17>. Cited on page 43.

ANONYMOUS007. *Software Engineering | Software Design Process*. [S.l.], 2023. Disponível em: <https://www.geeksforgeeks.org/software-engineering-software-design-process/>. Cited 2 times on pages 9 and 45.

BETHKE, U. *5 Ways to Manage Your Twitter Account With Dataiku*. [S.l.], 2016. Disponível em: <https://blog.dataiku.com/2016/07/28/twitter-plugin-with-dataiku-dss-from-sonra>. Cited on page 34.

CAI, K. *Data Science Startup Dataiku Raises $100 Million To Keep Growing It AI Enablement Software*. [S.l.], 2020. Disponível em: <https://www.forbes.com/sites/kenrickcai/2020/08/24/dataiku-ai-enablement-for-enterprises-startup-raises-100-million-series-d/?sh=40be94095d70>. Cited on page 16.

CHACON, S. *Git cherry pick documentation*. [S.l.], 2023. Disponível em: <https://git-scm.com/docs/git-cherry-pick>. Cited on page 75.

CHARTEXPO. *ChartExpo sankey diagram*. [S.l.], 2023. Disponível em: <https://chartexpo.com/charts/sankey-diagram>. Cited on page 87.

COURSERA. *What Is the Software Development Life Cycle? SDLC Explained*. [S.l.], 2023. Disponível em: <https://www.coursera.org/articles/software-development-life-cycle>. Cited on page 45.

COURTER, A. *TypeScript: What is the difference between type and interface?* [S.l.], 2022. Disponível em: <https://levelup.gitconnected.com/typescript-what-is-the-difference-between-type-and-interface-9085b88ee531>. Cited on page 61.

DATAIKU. *Dataiku DSS Feature: Interactive Date Filters*. [S.l.], 2021. Disponível em: <https://www.youtube.com/watch?app=desktop&v=py8SMUNQask>. Cited 3 times on pages 9, 31, and 33.

DATAIKU. *Concept | Recipes in Dataiku*. [S.l.], 2023. Disponível em: <https://knowledge.dataiku.com/latest/data-preparation/visual-recipes/concept-recipes.html>. Cited 3 times on pages 9, 28, and 29.

DATAIKU. *Installing plugins*. [S.l.], 2023. Disponível em: <https://doc.dataiku.com/dss/latest/plugins/installing.html#installing-from-the-store>. Cited on page 34.

DATAIKU. *Machine Learning*. [S.l.], 2023. Disponível em: <https://doc.dataiku.com/dss/latest/machine-learning/index.html>. Cited on page 35.

DATAIKU. *Plugins and Connectors*. [S.l.], 2023. Disponível em: <https://www.dataiku.com/product/plugins/>. Cited on page 89.

DATAIKU. *Projects, Folders, Workspaces, Wikis Views*. [S.l.], 2023. Disponível em: <https://doc.dataiku.com/dss/latest/concepts/homepage/projects-folders-dashboards-wikis.html>. Cited 2 times on pages 9 and 26.

DATANAMI. *Dataiku Announces $200M Series F Funding*. [S.l.], 2022. Disponível em: <https://www.datanami.com/this-just-in/dataiku-announces-200m-series-f-funding/>. Cited on page 16.

DENG, D. *Machine Learning*. [S.l.], 2023. Disponível em: <https://medium.com/@deborah.deng/automatic-machine-learning-with-dataiku-dss-14a85c02f9bb>. Cited 2 times on pages 9 and 36.

EDINBURGH, T. U. of. Software requirements. *The University of Edinburgh*, v. 1, n. 1, p. 9, 2003. Cited on page 41.

ELLIOTT, G. *Global Business Information Technology: An Integrated Systems Approach*. Illustrated. [S.l.]: Pearson Addison Wesley, 2004. Acesso em: 15 nov 2023. Cited on page 38.

F5, INC. *Faster Web & Mobile Performance*. [S.l.], 2023. Disponível em: <https://www.nginx.com/solutions/web-mobile-acceleration/>. Cited on page 60.

GARTNER. *Software Support Services*. [S.l.], 2023. Disponível em: <https://www.gartner.com/en/information-technology/glossary/software-support-services>. Cited on page 53.

GITHUB. *Understanding the SDLC: Software Development Lifecycle Explained*. [S.l.], 2023. Disponível em: <https://resources.github.com/software-development/what-is-sdlc/>. Cited 3 times on pages 38, 41, and 44.

GOOGLE. *AngularJS landing page*. [S.l.], 2021. Disponível em: <https://angularjs.org/>. Cited on page 61.

GOOGLE. *Angular landing page*. [S.l.], 2023. Disponível em: <https://angular.io/>. Cited on page 61.

GOOGLE FOR DEVELOPERS. *Google sankey*. [S.l.], 2023. Disponível em: <https://developers.google.com/chart/interactive/docs/gallery/sankey>. Cited on page 87.

GURAV, P. *Introduction of Software Development Life Cycle V-Model*. [S.l.], 2022. Disponível em: <https://cryptextechnologies.medium.com/introduction-of-software-development-life-cycle-v-model-ae49da7eab25>. Cited 2 times on pages 9 and 50.

JAIN, A. *Sliding Window Algorithm*. [S.l.], 2023. Disponível em: <https://www.scaler.com/topics/sliding-window-algorithm/>. Cited on page 94.

JIANYF-JIANYF. *[Bug] Radar chart setting the overflow property of the indicator text has no effect*. [S.l.], 2022. Disponível em: <https://github.com/apache/echarts/issues/16391>. Cited 2 times on pages 9 and 77.

JUSTICE, U. D. of. *INFORMATION RESOURCES MANAGEMENT*. [S.l.], 2003. Disponível em: <https://www.justice.gov/archive/jmd/irm/lifecycle/ch1.htm>. Cited 2 times on pages 9 and 40.

LEAN ENTERPRISE INSTITUTE INC. *Plan, Do, Check, Act (PDCA)*. [S.l.], 2023. Disponível em: <https://www.lean.org/lexicon-terms/pdca/>. Cited 2 times on pages 66 and 83.

PAVLOU, S. *Types of software maintenance your business needs and why*. [S.l.], 2023. Disponível em: <https://www.lightflows.co.uk/blog/types-of-software-maintenance-your-business-needs-and-why/>. Cited 2 times on pages 9 and 52.

PESSÔA, M. *Processos e projetos em uma fábrica de software eLab-TI*. Tese (Doutorado) — Escola Politécnica, Universidade de São Paulo, São Paulo, 2009. Disponível em: <https://doi.org/10.11606/T.3.2011.tde-01042011-115726>. Acesso em: 2023-11-03. Cited 2 times on pages 41 and 44.

S., E. *How to Use a Git Branch*. [S.l.], 2023. Disponível em: <https://www.hostinger.com/tutorials/how-to-use-git-branches/>. Cited 2 times on pages 9 and 76.

SLDC FORMS. *SDLC Definition - Testing and Acceptance Phase*. [S.l.], Unknown. Disponível em: <https://www.sdlcforms.com/PopupPhase-Testing.html>. Cited on page 49.

Soleimani Neysiani, B.; BABAMIR, S. M.; ARITSUGI, M. Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems. *Information and Software Technology*, v. 126, p. 106344, 2020. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584920301117>. Cited 2 times on pages 52 and 106.

STANDARDIZATION, I. O. for. *ISO/IEC/IEEE 14764:2022*. [S.l.], 2022. Disponível em: <https://www.iso.org/standard/80710.html>. Cited 2 times on pages 51 and 53.

STANDARDIZATION, I. O. for. *ISO/IEC/IEEE 12207:2017*. [S.l.], 2023. Disponível em: <https://www.iso.org/standard/63712.html>. Cited 3 times on pages 9, 47, and 48.

TAYLOR, J. *Managing Information Technology Projects: Applying Project Management Strategies to Software, Hardware, and Integration Initiatives*. 1. ed. [S.l.]: PHI Learning, 2004. Acesso em: 15 nov 2023. Cited on page 38.

THE APACHE SOFTWARE FOUNDATION. *Echarts home page*. [S.l.], 2023. Disponível em: <https://echarts.apache.org/en/index.html>. Cited on page 63.

THE APACHE SOFTWARE FOUNDATION. *Echarts radar basic example*. [S.l.], 2023. Disponível em: <https://echarts.apache.org/examples/en/editor.html?c=radar>. Cited 2 times on pages 9 and 70.

THE APACHE SOFTWARE FOUNDATION. *Echarts sankey basic example*. [S.l.], 2023. Disponível em: <https://echarts.apache.org/examples/en/editor.html?c=sankey-simple>. Cited 2 times on pages 10 and 94.

THE APACHE SOFTWARE FOUNDATION. *Echarts sankey documentation*. [S.l.], 2023. Disponível em: <https://echarts.apache.org/en/option.html#series-sankey.type>. Cited on page 87.

THE DATA VISUALISATION CATALOGUE. *Radar chart*. [S.l.], 2023. Disponível em: <https://datavizcatalogue.com/methods/radar_chart.html>. Cited 2 times on pages 9 and 67.

UNIVERSITY OF CONNECTICUT. *Activity 8 – Implementation*. [S.l.], Unknown. Disponível em: <https://sdlc.uconn.edu/activity-8-implementation/>. Cited on page 47.

WIGGERS, K. *AI and analytics platform Dataiku raises $200M at a reduced valuation*. [S.l.], 2022. Disponível em: <https://techcrunch.com/2022/12/13/ai-and-analytics-platform-dataiku-raises-200m-at-a-reduced-valuation/>. Cited on page 16.