

Final Report

Customer maturity analysis improvement for TOPdesk

by Krzysztof Baran, Cees Jol, Rover van der Noort and Wander Siemers



Preface

In this project report, we outline and evaluate the project we have completed for the Bachelor End Project course as part of the Bachelor of Science in Computer Science and Engineering at the Delft University of Technology. We conducted this project at TOPdesk¹, an international company providing service management software.

Over the span of ten weeks, we researched, designed, implemented and validated a software product to optimize a part of TOPdesk's business, namely its consulting department. To do this, we first extensively automated a previously manual process called the Mini Health Check. We then used these results to build a tool to compare clients based on Mini Health Check results, a process called *benchmarking*. We found that these tools significantly improve the process by making it easier, cheaper and faster to perform a Mini Health Check and that the benchmarking tool provides valuable context to its results.

This report serves as the main result of this project. It answers the question of whether the project solves the problem it set out to. It also encapsulates the knowledge gained and the work performed during the project to be used as a future reference to expand upon our work.

Krzysztof Baran
Cees Jol
Rover van der Noort
Wander Siemers

Delft, January 2021

¹<https://topdesk.com/en>

Summary

TOPdesk is a service management software provider in a wide variety of domains and industries. TOPdesk also offers consultancy to their customers that aims to continuously assess and improve the customer's experience and service efficiency. TOPdesk offers a *Mini Health Check* (MHC) to their customers in which a consultant analyzes how efficiently the customer uses their software based on six Key Performance Indicators (KPI). However, the process of creating an MHC report is very time-consuming as it requires performing a lot of manual steps. Also, the norms used for the KPIs provide little meaning as they are arbitrarily chosen and not specific to the customer's industry.

This report aims to improve the current process of performing an MHC. Research has been done on how the MHC is performed, identifying the suitable technologies and learning the currently existing infrastructure that helped us pave the way to create our product.

During our project we managed to create a product that automates the MHC. Through user testing we found that this process now takes about two minutes, where the manual process took about two hours. To create more meaningful norms for the KPIs, we also implemented a benchmarking feature. This allows a company to compare the results of their MHC to other TOPdesk customers in the same sector, country or of similar size.

We have some recommendations for TOPdesk for the further development of our product. The MHC process could be streamlined in a few ways, most importantly with respect to the process for getting access to customer data. Benchmarking could become even more useful if data can be more easily gathered from more TOPdesk customers.

Acknowledgements

We are grateful for the valuable guidance and feedback we have received from everybody that was involved with this project. In particular, we would like to thank Frank Mulder, our Delft University of Technology supervisor, for his valuable guidance for our work. Jan-Fabian Humann and Corina Stratan supervised us within TOPdesk and were available for all of our questions. We would not have been able to conduct our work without them. Sylvie Bruys was our product owner and was invaluable when defining our project and for stakeholder management during the project. We would like to thank our stakeholders, Anne van Beusekom, Job van Grieken, André Kevenaar, and Sophie Steinmeijer, for their valuable feedback on our product. Many other people at TOPdesk helped us with technical questions and while there are too many to list them all here, we are particularly indebted to Joep Weijers, Jesse van Assen and Jirí Horák for their technical assistance.

Contents

Preface	ii
Summary	iii
Acknowledgements	iv
List of Figures	vii
List of Tables	viii
Abbreviations	ix
1 Introduction	1
1.1 Maturity Model	1
1.2 Mini Health Check	1
1.2.1 Execution	2
1.3 Success Criteria	2
1.4 Document Structure	2
2 Research	3
2.1 Problem Definition and Analysis	3
2.1.1 Problem Overview	3
2.1.2 Labor-Intensiveness	3
2.1.3 Norms	3
2.1.4 Problem Domain	4
2.1.5 Research Question	4
2.1.6 Requirements	4
2.2 Solutions	6
2.2.1 Mini Health Check Automation	6
2.2.2 Benchmarking	8
2.2.3 Verification and Validation	9
2.3 Technologies	9
2.3.1 Progress Tracking and Version Control	10
2.3.2 Hosting	10
2.3.3 Testing	10
3 Design	12
3.1 Goals	12
3.1.1 Persona	12
3.1.2 TOPdesk Integration	12
3.2 Structure	13
3.2.1 Vue Front End	13
3.2.2 Firkin Back End	14
3.2.3 Django Back End	15
4 Implementation	17
4.1 Agile Workflow	17
4.1.1 Scrum and Research	17
4.1.2 Flexibility	17
4.2 Automating the MHC	18
4.2.1 Front End	18
4.2.2 Back End	19
4.3 Benchmarking	20
4.3.1 Visualization	21

4.4	Deployment	21
5	Testing and Quality	22
5.1	Testing	22
5.1.1	Front-End Testing	22
5.1.2	Back-end Testing	22
5.1.3	Integration testing	22
5.1.4	End User testing	23
5.1.5	Test-Driven Development	23
5.2	Quality	23
5.2.1	Technologies	23
5.2.2	SIG Evaluation 1	23
5.2.3	SIG Evaluation 2	25
5.2.4	Pipeline	26
5.2.5	Code review	26
5.2.6	Performance testing	26
6	Product Evaluation	27
6.1	Evaluation of Requirements	27
6.1.1	Functional requirements	27
6.1.2	Non-functional requirements	28
6.2	Requirements evolution	29
6.3	User Experience Evaluation	29
6.4	Key Findings	30
7	Conclusion	31
8	Discussion	32
8.1	Process evaluation	32
8.2	Limitations	32
8.2.1	Stakeholders	32
8.2.2	Technologies	33
8.3	Recommendations	33
8.3.1	Automating the MHC	33
8.3.2	Benchmarking	33
8.3.3	Customer Data Permissions	34
8.3.4	Back-End Architecture	34
8.4	Ethical Implications	34
	Bibliography	36
	Appendices	39
A	Info Sheet	40
B	Supporting Material	41
B.1	GDPR Table	41
B.2	Example of KPI assement	42
B.3	MHC Manual Steps to export KPIs	43
B.4	Benchmarking Plots	44
B.5	SonarQube results	44
C	Original Project Description	47
D	Manual	48

List of Figures

1.1	TOPdesk's Maturity Model	1
2.1	Flow of the MHC automation process.	6
2.2	Process to classify custom incident types into their respective "best practices" types.	8
3.1	CSM Persona	13
3.2	General system design.	14
3.3	Vue framework pages & components.	14
3.4	Firkin Service Endpoints	14
3.5	Benchmark database schema.	16
4.1	Implementation phase timeline	18
4.2	Benchmark database schema	20
5.1	First SIG submission results	24
5.2	Second SIG submission results	25
5.3	A passing GitLab pipeline	26
B.1	Self-service portal usage assessment guide	42
B.2	Overview of instruction set to export Key Performance Indicators (KPIs) for the MHC.	43
B.3	Example of Self-Service usage KPI with mocked data with a Box Plot	44
B.4	Example of Self-Service usage KPI with mocked data with a Violin Plot	44
B.5	Vue SonarQube quality results.	45
B.6	Django SonarQube quality results.	45
B.7	Firkin SonarQube quality results.	46

List of Tables

1.1	List of KPIs used in MHC.	2
2.1	Functional requirements.	5
2.2	Non-functional requirements.	6
3.1	Paths and description of Django Application Program Interface (API) suite.	15
6.1	Evaluation of functional requirements.	28
6.2	Evaluation of non-functional requirements.	29
B.1	GDPR compliance explanation table	42

Abbreviations

API Application Program Interface. viii, 7, 10, 13–15, 18–20, 22, 34, 40

CI/CD Continuous Integration/Continuous Delivery. 10, 29

CSM Customer Success Manager. 5, 7, 9, 12, 15, 18, 23, 27–34

DUT Delft University of Technology. 17

EU European Union. 4

FHC Full Health Check. 1, 2, 15

GDPR General Data Protection Regulation. 4, 5, 28

GUI Graphical User Interface. 6, 8

HTTP Hypertext Transfer Protocol. 15, 19

KPI Key Performance Indicator. vii, viii, 2–9, 15, 18–22, 28, 31, 33, 40, 43, 44

MHC Mini Health Check. vi–viii, 1–9, 12, 13, 15, 17–20, 23, 27–31, 33, 34, 41, 43

ML Machine Learning. 5, 8, 19, 20, 27, 33

MS SQL Microsoft SQL Server. 9, 33

MySQL My Structured Query Language. 20, 33

NDA Non-Disclosure Agreement. 41

NLP Natural Language Processing. 8, 19

REST Representational State Transfer. 7, 19

SaaS Software as a Service. 4, 7, 9, 10, 42

SIG Software Improvement Group. vi, vii, 23–25

URL Uniform Resource Locator. 19

Introduction

TOPdesk is a service management software provider in a wide variety of domains and industries. TOPdesk also offers consultancy to their customers that aims to continuously assess and improve the customer's experience and service efficiency. For this purpose, TOPdesk created standardized objective metrics to express a customer's service level, categorizing its service efficiency.

1.1. Maturity Model

The maturity model in Figure 1.1 displays the classification TOPdesk assesses their customers based on their business focus and mindset. The model's level is determined by quantifiable tests measuring their TOPdesk utilization and interviews with the customer's TOPdesk operators. Most companies strive to achieve a higher level in the maturity model to provide better service to their users.



Figure 1.1: TOPdesk's Maturity Model

TOPdesk offers so-called *maturity health checks*¹ in which a consultant analyzes the customer's TOPdesk usage. TOPdesk offers two variants of their health checks, namely the MHC and the Full Health Check (FHC). The MHC is a free service, which analyzes three of the maturity dimensions: *Customer*, *Process* and *Supplier*. To fully assess all the company's maturity levels, including *People & Culture* and *Integration & Automation*, an FHC can be requested, which is a paid upgrade on the MHC and includes more qualitative data analysis, such as interviews with operators. These dimensions can give a sense of the efficiency of a customer's service level, described in this maturity model.

1.2. Mini Health Check

To quantify the level of service excellence of their customers, TOPdesk consultants do a thorough analysis of their customers' TOPdesk usage, which customers use to provide better service to their users. This section will show the current process for determining this maturity level by analyzing the MHC.

¹<https://page.topdesk.com/maturity-model>

The FHC will be kept out of this project’s scope, because it focuses more on the social aspects of the customer and it is less quantifiable. The free MHC, which is usually executed on a yearly basis by a consultant, analyzes data exported from a customer’s TOPdesk application based on six KPIs, which are shown in Table 1.1.

	KPI	
1	Self-service usage	Percentage of incident calls reported in the self-service portal
2	Knowledge management	Percentage of information/user request incident calls that are linked to a knowledge item
3	Match best-practices incident management	Percentage of incident calls that are not in the best-practices call types
4	Efficiency incident management	Percentage of incident calls that are resolved and achieved on time
5	Standardization incident management	Percentage of service request incident calls that are linked to a standard solution
6	Supplier incident management	Percentage of incident calls with known suppliers

Table 1.1: List of KPIs used in MHC.

These KPIs are chosen because they quantitatively measure the service level of a company. They measure the efficiency of the client’s usage of TOPdesk by looking at the percentage of incident calls that were handled in the predetermined most efficient way, which relates to the efficiency of the overall service of the company. These “best-practices” are to use the self-service portal, knowledge items, standard incident types, standard solutions, supplier management tools, while resolving the incidents within the set time, as much as possible. With these results consultants can determine the company’s before-mentioned three maturity dimensions, which determine the level in the maturity model.

1.2.1. Execution

When performing an MHC, consultants run a process described in a 17-page manual [30] explaining the steps to create the needed data exports for analysis. The results of these steps then need to be transferred to a report template document and a consultant puts these results inside a presentation, which is presented to the customer for a discussion about their maturity level. In Figure B.2 the export instruction steps that the consultant follows are visualized, including the norms that determine the level of the maturity dimensions based on the KPIs.

1.3. Success Criteria

Due to time constraints, our primary goal is to implement all of the Must Have requirements. We will also implement other requirements if we have time for it. Further, the product will serve as a proof of concept for further TOPdesk development. We consider our project a success if we implement all Must Have requirements and create a valuable product for TOPdesk. This will be assessed using user experience evaluation and stakeholder feedback sessions.

1.4. Document Structure

This report starts with a research report in chapter 2. In section 2.1 we will identify some of the problems consultants are currently facing with regards to determining the level in the maturity model. We recommend technical solutions based on this problem analysis in section 2.2. After this initial research, we give a final outline of our design in chapter 3 and explain implementation details of our product in chapter 4. Then, we explain how the software quality of our product is assessed in chapter 5. In chapter 6, we evaluate our requirements and discuss our key findings. Finally, in chapter 7, we conclude whether our product solves the problems outlined in our research, followed by a discussion in chapter 8.

2

Research

To understand the project, we first researched at the beginning of the project, which is described in this chapter. We refer to our implementation here in the future tense, since this research report was finished before the implementation phase.

2.1. Problem Definition and Analysis

2.1.1. Problem Overview

From the overview given in From the overview given in chapter 1, three main problems can be identified. First, we define and analyze the problem of the labour-intensiveness of the current process in subsection 2.1.2. Second, we look at the norms of the KPIs in subsection 2.1.3, in which we identify two problems; the lack of domain-specific norms and the norms are arbitrary. For the latter, we will discuss a subproblem, namely, a privacy issue when data is used to create less arbitrary norms.

2.1.2. Labor-Intensiveness

A main TOPdesk goal is to help their customers achieve service excellence [31]. The MHC is an integral part of the process to determine their maturity level.

The current process for performing an MHC is labour-intensive and manual, as shown in section 1.2. It is estimated to take about two hours, making it hard to scale up to many customers, and is error-prone. The process is labor-intensive, since the consultant needs to perform a lot of manual steps in the TOPdesk reporting interface and report document. This includes taking screenshots of the reports, saving those to files with particular filenames, and manually assembling the MHC report. It also involves manual classification of custom incident types, because the calculation of the MHC requires default incident types to exist.

It is clear that there is potential for time-savings: given that the average frequency of an MHC is about once a year, TOPdesk has thousands of customers, and an MHC takes about two hours on average, there are potentially thousands of hours per year to be saved. Since the MHC is a free service provided by TOPdesk, there is a possibility for large cost reductions by automating it. TOPdesk has set out to increase revenue per employee in the coming years [32], and automating the MHC can therefore contribute to their goal.

2.1.3. Norms

2.1.3.1. Lack of Domain-Specific Norms

A second issue is the lack of domain-specific norms in the MHC. In the current report, the results for the six KPIs are presented and compared to their standard norms. These values are the same for every company, independent of company size, country and industry. We measure company size by employee count in this context. There are three general quality grades: *Good*, *Satisfactory* and *Needs Improvement* which are represented by traffic light colours, as indicated in Figure B.1.

The problem with this standardized approach is that there are domain-dependent differences in a companies' usage of TOPdesk. For example, in a hotel, guests expect to be able to call the reception with their issues and do not want to use an app on their phone to make a request. Therefore, hotels having a low score on the KPI for knowledge base use is expected. However, university students are often much more comfortable using a knowledge base article to resolve issues. It would be interesting for companies to be compared to a standard based on the average for their industry. This comparison could improve the assessment of their service level.

2.1.3.2. Arbitrary Norms

A third issue is that the norms used for these KPIs on the bottom of Figure B.2 are not based on real average data, but are based on the work experience of an individual consultant. However, it would be interesting for both the customer and TOPdesk to base this on more substantial data. A benchmark currently does not exist in TOPdesk, but it is part of their road map to 2022, and this project should act as a proof of concept for future TOPdesk development.

It has become apparent from speaking with TOPdesk consultants that customers are interested in comparing their result to less arbitrary norms than those used now. As Motwani et al. [20] states about benchmarking: “It is the process of identifying, understanding, and adapting outstanding practices from organizations anywhere in the world to help an organization improve its performance.” and as [1] said “It is an activity that looks outward to find best practice and high performance and then measures actual business operations against those goals.” By witnessing on which KPIs customers lag behind, they can discuss potential service level improvements with a consultant. Currently, customers can only compare their service level resulting from the MHC to their previous results. This takes time because multiple MHCs have to be performed. The customers are requesting a method to have real data supporting the determination of their maturity level compared to others in the same domain or country.

2.1.3.3. Privacy

A requirement for benchmarking the data of a customer against other customers in a certain domain is that data of other customers are needed. Since this project deals with customer data, it is essential to consider how it is handled. The data determining the KPIs of most customers are stored on TOPdesk servers as they are part of the Software as a Service (SaaS) product. It requires permission from the customer to access it, which should be granted when consultants want to perform the MHC.

We consulted the legal team to ensure our product adheres to privacy regulations. As this project will use data of customers that are solely located in the Netherlands and the service will be hosted by TOPdesk, a Dutch company, our product has to follow the General Data Protection Regulation (GDPR) rules which are made available by the European Union (EU) [9]. The MHC is already compliant with GDPR regulations and automating it does not change this. For benchmarking, however, customer data needs to be shared and stored, which is not the case for the MHC. Therefore, for this use of data, we need to ensure GDPR compliance. We address the full GDPR compliance in the checklist in Table B.1.

2.1.4. Problem Domain

TOPdesk has two types of customers: on-premises and SaaS [29]. Ideally, we would create a solution for both types of customers. However, the process for getting access to on-premises data varies between clients because their set-ups are not standardized. On-premises clients may, for example, implement their firewall in front of their TOPdesk environment, which adds customer-specific complexity and that does not fit within our project scope due to time constraints.

For SaaS customers, however, installations are hosted by TOPdesk in a standardized way. This means automating the MHC for SaaS customers is easier to implement and much more scalable. On top of that, according to our product owner, around 89% of the current TOPdesk customers use the SaaS platform. After discussing with our product owner, we have chosen to only provide our solution for SaaS customers.

TOPdesk has both Dutch and international clients, but most clients are located in the Netherlands. To simplify our problem domain, we have chosen only to support Dutch clients. This decision has been made in coordination with our product owner. It means we do not have to implement features like internationalization and localization, leaving more time to implement core functionalities.

2.1.5. Research Question

This report will analyze possible technical solutions for these problems and choose the solutions best fitting for our prototype. We support these solutions with the relevant literature and our discussions with the TOPdesk stakeholders. Generally, we will try to answer the question: *How can we improve the process of determining the level of Service Excellence on the Maturity Model?* For this project, a prototype will be developed to assert our findings in this research and formalize a final answer.

2.1.6. Requirements

To give a concrete view on how the product should function, requirements have been set up shown in Table 6.1. They were drawn up based on the problem definition and analysis in section 2.1 and are categorized

under a respective milestone. The milestones are the division of the automation part and the benchmark tool of our solution. Meetings with stakeholders and the product owner allowed us to set up and prioritize our requirements.

We identified the design goals of our project, which are: efficiency, usability, maintainability, correctness and reliability. Because of our agile workflow, we discussed these goals during the implementation and used them to justify our design choices. In chapter 3 we elaborate on why we picked these design goals and how they relate to the design choices. Our prototype also has a list of non-functional requirements. We put the related design goals in Table 2.2 to make the justifications for the design that should satisfy these requirements [6, 10]. In the next chapters we elaborate on the proposed solutions. We will frequently refer to which requirements the solution satisfies.

2.1.6.1. Functional Requirements

Name	Requirement	Milestone
Must Have 1	The consultant can generate the MHC report automatically, apart from having to classify custom call types.	Automating the MHC
Must Have 2	The customer can compare the results of their MHC to other companies within a certain domain so they can see how they are doing compared to their competitors and see how they can improve.	Creating Benchmark
Must Have 3	The consultant can choose which call types should be grouped together when executing the automated MHC.	Automating the MHC
Should Have 1	The product can assist the user in classifying categories, using technology such as Machine Learning (ML), that have the same best practices definition but with a different call type.	Automating the MHC
Could Have 1	The Customer Success Manager (CSM) can generate a presentation based on the report automatically.	Automating the MHC
Could Have 2	A potential new customer can see the performance of TOPdesk specifically in their domain before they purchase it.	Creating Benchmark
Could Have 3	The customer can see the generated report and presentation automatically through the TOPdesk software after they have been presented by the CSM.	Automating the MHC
Could Have 4	The customer can press a button within the TOPdesk software to request an MHC, and their benchmark status, at any time.	Automating the MHC & Creating Benchmark
Could Have 5	The product will use some new KPIs that give a better view of the maturity level than the current KPIs.	Creating Benchmark
Won't Have 1	The customer can generate the MHC report themselves at any time.	Automating the MHC

Table 2.1: Functional requirements.

2.1.6.2. Non-Functional Requirements

Name	Requirement	Goal
Must Have 4	The product produces a report in less than half of the amount of time that it takes for a consultant to produce it.	Efficiency
Must Have 5	The benchmark tool should show correct statistics for the selected comparisons.	Correctness
Must Have 6	Only active authenticated users who are authorized to use the software can access the software.	Correctness, Reliability
Must Have 7	Data usage must be GDPR compliant.	Reliability
Should Have 2	Performing the automated health check should not negatively affect the performance of the TOPdesk environment.	Reliability, Usability
Should Have 3	The final application should be usable in current versions of Google Chrome (87.0) and Microsoft Edge (87.0).	Usability, Maintainability
Should Have 4	The final software product must be well-documented using code comments and a manual.	Maintainability, Usability
Continued on next page		

Table 2.2 – continued from previous page

Name	Requirement	Goal
Should Have 5	There should always be a working version of the software running after each sprint.	Reliability, Maintainability
Could Have 6	The user interface should be in the corporate identity of TOPdesk software.	Efficiency, Usability
Could Have 7	The final application should be usable in current versions of Mozilla Firefox (83.0).	Usability, Maintainability

Table 2.2: Non-functional requirements.

2.2. Solutions

In this section, we present solutions for the problems outlined in section 2.1. More specifically, in subsection 2.2.1 we present a solution for the labor-intensiveness of the current MHC. Subsequently, in subsection 2.2.2 we present a solution for the norm problems with regards to their service level, namely customers not having a reference point, and the norms being arbitrary. We explain how we came with our decisions, including the technologies that will help solve that related solution.

In section subsection 2.2.3, we also look at methods for validating¹ and verifying² these solutions based on the requirements, as recommended by Adrion et al. [2].

2.2.1. Mini Health Check Automation

To decrease the execution time of the MHC, automation of the check is recommended as a solution. As shown in the overview of the current system in Figure B.2, the MHC is a series of simple exports inside the TOPdesk application and the extraction of required data from a table. This data is copied and pasted into a template report and presentation, which is a process suitable for automation.

2.2.1.1. Program Flow

Figure 2.1 shows the expected flow of the automation process. The consultant can have straightforward procedures and a smaller number of steps to execute to achieve an MHC report and presentation, which should satisfy “Must Have 1”. It should only need to select a client and provide its credentials. Secondly, it would need to verify the classified incident types under its best practices, and lastly, the consultant automatically downloads the filled report and presentation for their client.

The program itself also consists of exact steps that need to be developed. It needs to maintain a front end web page for the consultant to provide its input data. It can log in on the back end part of the application of the client’s TOPdesk environment. This part extracts all the necessary data and sends it back to the main program. Our program classifies the incident types as best as it can and presents it to the consultant. After the consultants’ approval, it can calculate the KPIs and put them into a report and presentation, ready for a download.

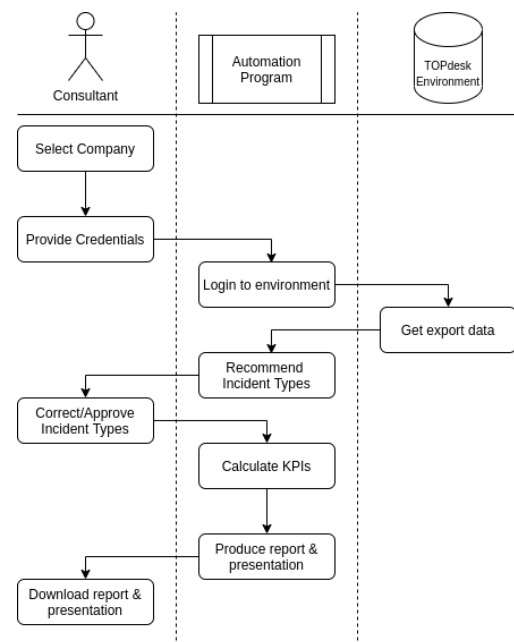


Figure 2.1: Flow of the MHC automation process.

2.2.1.2. Application

Consultants are already familiar with working in a Graphical User Interface (GUI) for the current procedure of executing the MHC. Additionally, they use desktop PCs to perform their work, so our program should run on those devices. Therefore, we decided that we will create a web application with a simple and intuitive GUI, such that consultants can easily learn to work with it. In the following subsections, we will explain some more

¹Validation is focused on the external view of the product: “are we building the right product?” [4]

²Verification is focused on meeting the product specifications: “are we building the product right?” [4]

specific technologies required for this web application.

Front End

Our group has experience with the three common front-end frameworks: Angular, React and Vue. Vue³ is a standard at TOPdesk and will have the most internal support available. According to Wohlgethan [35], compared to the other two frameworks “Vue has a shallow learning curve”. This choice leaves the decision to implement it with either TypeScript or JavaScript. However, we think that typing benefits do not outweigh the drawbacks of added complexity by using TypeScript because it requires more work and learning [35]. Therefore we decided to follow the TOPdesk standard and go with Vue in JavaScript.

For the structure and styling of webpages, we plan to use Bulma⁴ or Buefy⁵, and CSSgrid⁶, as they have pre-made components and grids. This choice eliminates the task of having to recreate existing well-designed UI components. They are low-level and integrate easily with Vue. TOPdesk already maintains a Bulma theme so we can easily stay within TOPdesk style.

Alternatively, Bootstrap⁷ could serve a similar purpose. However, after further research, we found out that Bootstrap has a verbose styling that is difficult to change [27]. Also, it uses jQuery⁸ which might interfere with our Vue.js app according to a TOPdesk front-end expert at TOPdesk. Therefore, we will probably not use Bootstrap.

Back End

TOPdesk has an in-house framework for back-end Java services, called Firkin. This framework provides built-in authentication, authorization, and direct coupling with the already existing TOPdesk Representational State Transfer (REST) API suite from which we can request the data. This offers a lot of pre-existing functionalities for our data extraction service and is therefore preferred to use. This service should run on the SaaS environments of the customer, where the other Firkin services are located, and sends the exported data on request back to our main service.

We decided on a web framework and language that is more focused on data analysis and processing for the main service. Django⁹ is an option very suitable to our needs since we have experience with Python and it provides a better full-stack option than for instance Flask [11]. It has plenty of helpful libraries, which we can use to handle the KPI data and aggregate this data for our benchmark [18]. The integration with the Vue front end, communication with the Firkin framework via REST APIs and filling of file templates fulfil all of our needs [3].

Another option is the Express¹⁰ framework on NodeJS¹¹. This framework has high performance and many libraries but only runs in a single thread [5]. This limitation removes the possible scalability options to process parallel, which the benchmark tool discussed in a later section might require larger amounts of data.

2.2.1.3. User experience evaluation

To verify if the system improves on the current process and fits the requirements set out in subsection 2.1.6, we will do user experience evaluation.

We know from discussions with stakeholders that it takes a consultant or CSM around two hours to fully export the MHC data into a report, which they then use to create a presentation. It should be noted here that since both CSMs and consultants are the target users for our application and we refer to them interchangeably. To measure the time it takes consultants to execute the same tasks using our product, we will ask them to do these tasks in user testing sessions which the team will set up.

2.2.1.4. Call Types

As stated in subsection 2.1.2, customers can create their call types, which hinders the automation. It should be noted here that we will refer to call types and incident types interchangeably. The MHC assumes that certain default incident types exist, based on TOPdesk’s “best practices”, but some customers choose to deviate

³JavaScript framework, <https://vuejs.org>

⁴CSS framework, <https://bulma.io/>

⁵UI components for Vue.js based on Bulma, <https://buefy.org/>

⁶CSS grid: table layout CSS <https://learncssgrid.com/>

⁷Website building tool, <https://getbootstrap.com/>

⁸Javascript library, <https://jquery.com/>

⁹Python web framework: <https://www.djangoproject.com>

¹⁰Minimalist web framework: <https://expressjs.com>

¹¹Asynchronous event-driven JavaScript runtime: <https://nodejs.org/en/about/>

from this format. Therefore, the consultant usually uses their judgment to categorize these custom incident types into the default incident types. One option would be to let the consultant manually classify these requests in a GUI, but we could still improve performance here. Our solution, therefore, should also categorize these custom incident types based on its knowledge of the language and previous incident types.

In Figure 2.2, a model is shown of a possible process that could assist with the required decisions for this classification. A ML algorithm that is being trained online, based on the consultants' recommendations, classifies the encountered incident types of a customer. This classification is displayed to the consultant performing the MHC. The consultant has the opportunity through a front-end menu to make necessary changes or to accept the recommended output, satisfying requirements "Must Have 3" and "Should Have 1". The consultant's changes serve as the learning model's error margin, so the model continuously improves based on the consultant's implicit feedback. After accepting the classification or making changes to it, the rest can be executed without any user input.

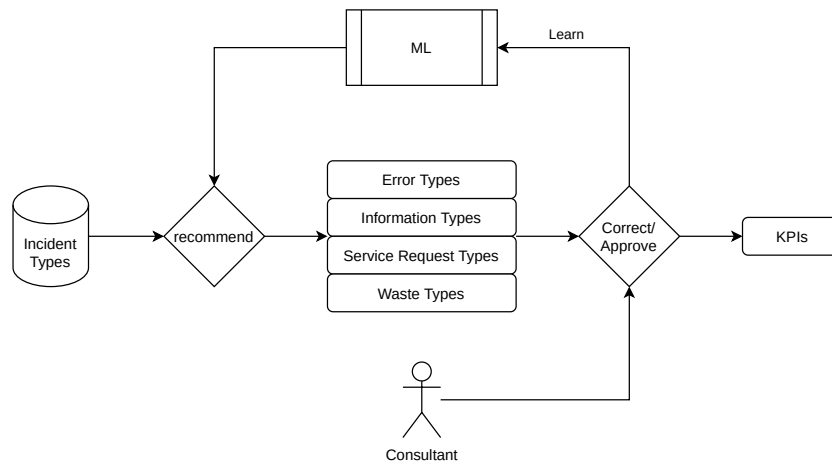


Figure 2.2: Process to classify custom incident types into their respective "best practices" types.

Machine Learning

There are multiple ML libraries available for Python. We need a Natural Language Processing (NLP) algorithm such that incident types that have a similar meaning can be classified under the same best practice type. To do this, we can utilize a semantic similarity approach where we map words to numerical vectors inside an unsupervised learning algorithm [26]. This algorithm then classifies these vectors based on their similarity.

We need to include text sanitation, language and data aggregation. We could use a pre-trained program like Word2Vec [19], GloVe [22] or FastText [14], which already have basic knowledge about word relations.

FastText seems to have the most significant advantage since it also looks at the context inside the word itself by breaking it up [14]. Word2Vec on which FastText is based only looks at the text context of the word, but our problem does not provide this context, making it harder to classify [14, 19] correctly. However, looking inside the word could create important context, so that the custom incident type "faulty printer", which contains the subword "fault", is classified as "error". We need to test if we can make this approach increase the efficiency of the consultants' workflow.

2.2.2. Benchmarking

As mentioned in section 2.1, the current MHC does not provide a point of reference to the performance of the competitors. This chapter describes a solution to tackle the lack of reference point for determining the maturity level.

2.2.2.1. Method

To solve the problem of lack of reference point, we could create a benchmark tool to determine a more accurate set of norms for each domain. We will display the benchmark in a clear and concise manner such that consultants can explain the results to their clients.

Our goal is to create a report that contains the same MHC and extends it with benchmark data. Hence, for the benchmark tool, we decided to use the same six KPIs the MHC uses to be consistent. However, finding better KPIs has been added as a could-have requirement.

Benchmarking will provide a better indication of the norms of the specific domains. It will be added to the generated report if customers have given permission to use their data for benchmarking. Once a sufficient number of customers from a domain have been acquired, the tool provides feedback to the customers about their service level compared to their domain and provides TOPdesk insight into domain standards across different domains. We will then also be able to generate new KPI averages for certain domains and suggest new more accurate norms.

2.2.2.2. Database

To compare benchmarks, the six KPIs need to be stored in a database. Currently, for each SaaS customer, the data required for the KPIs is stored on TOPdesk's servers. In a database separate from TOPdesk's database, we will store each company's anonymized KPI values. By storing all the values and generating the average when necessary, the relevant benchmarks are always available when a customer requests the result of their MHC.

We plan on using Microsoft SQL Server (MS SQL)¹² as a database, since TOPdesk's IT Operations only supports MS SQL, and we want to run our back end on a TOPdesk cluster. Thus we are required to use MS SQL. MS SQL is a traditional relational database, which are familiar to our group and satisfies our need for a simple database schema to store the benchmark data.

2.2.2.3. Visualization

For visualization, we had three options to consider: displaying the average, box plot and violin plot. We could take the simple approach and use the average to visualize how well a company is performing, but the average will be skewed by anomalies.

The box plot method shows the distribution that is resistant towards anomalies in data [17], unlike the average, and is useful when comparing different distributions across groups [34]. The box plot can be divided into three sections, as seen in Figure B.3, which can represent the quality grades: Minimum to First Quartile (needs improvement), First Quartile to Third Quartile - The Interquartile Range (satisfactory) and Third Quartile to Maximum (good). It is a very simple method to present data's distribution in comparison to the violin plot which incorporates the box plot and a Kernel Density Plot. This provides a more detailed summary of the density of the data yet preserves the box plot properties [13] as seen in Figure B.4.

The violin plot is much more detailed than the box plot. Either a box plot or a violin plot could be used to help visualize the results of the benchmark. A drawback of the violin plot is that it is very detailed and the representation is more difficult to understand compared to the box plot. We have decided that we will implement both and ask the consultants for feedback once we have the data and are able to draw these graphs.

2.2.2.4. Privacy

To protect the privacy of the customers, the data will be anonymized and averaged so that we do not store data that could link back to the customers.

2.2.3. Verification and Validation

The solutions we have proposed will be validated in interviews with the relevant stakeholders. We will ask feedback from consultants and CSMs, who are the target user of our product, and we will ensure they are representative of the wide variety of industries that use TOPdesk. Note that in this report, we refer to industry and sector interchangeably. These interviews will ensure we meet requirement "Must Have 1" and "Must Have 2".

Verification will be done as much as possible using automated testing tools. We explain the testing approach in more detail in subsection 2.3.3.

2.3. Technologies

In this chapter we will look at the remaining technical tools that we can use for our prototype. We have decided on the best choice by considering the TOPdesk development stack, our personal experience, and advantages and disadvantages of certain technologies. We break down our development stack into multiple components, which we discuss separately.

It is important to note that while we feel that all technology choices are appropriate based on our current knowledge, they are not set in stone at the research stage and may change as more information about the problem becomes available during implementation. We will discuss these deviations in a discussion in the final report.

¹²Microsoft SQL Server database management system, <https://www.microsoft.com/en-us/sql-server>

2.3.1. Progress Tracking and Version Control

The progress of process and the code is going to be documented through Git as version control system, since it is the industry standard. We also use the GitLab repository hosting service, which is provided to us by TOPdesk, because TOPdesk requires its self-hosted GitLab for security reasons.

We also use GitLab for feature tracking instead of Jira¹³, which is also commonly used in TOPdesk, because the latter is closely coupled to the product pipeline for the application. Our program is a stand-alone application and we therefore decided to not use Jira and centralize our development process on GitLab.

2.3.2. Hosting

To make our product run on multiple devices, we will containerize it. To this end we will use Docker¹⁴, which is standard use in TOPdesk. We want to host this container on the internal TOPdesk server provided to us. For development we will run software on our local machine using localhost. Every week we want to deploy a working prototype of our developed product, so that our stakeholders and product owner can track our progress. We will create a Continuous Integration/Continuous Delivery (CI/CD) pipeline to automate this process. To deploy the service on the SaaS environment we need to add our Docker container to TOPdesk's Kubernetes¹⁵ cluster, which deploys it to a customer. Because it is a stand-alone service we can automate the roll-out and do not have to wait for the TOPdesk application to update, which does not happen regularly. We will need to pass a pre-configured CI/CD pipeline after which our updates are deployed. For development purposes we can also use a so-called sneak environment which is TOPdesk's development playground that resets every night.

2.3.3. Testing

Because we will maintain a simple web application, our front-end testing will not be extensive, but should cover all of the basic functionalities such as component checking and API results.

The back end of the product consists of two parts: a data-extraction tool written in Java, running inside the customer's TOPdesk environment, and a data-processing and presentation tool, running outside of the customer's environment, written in Python. These parts each require their own testing infrastructure. We will use TOPdesk's internal CI/CD tools to ensure that we always have a working prototype.

Performance-critical code will also be timed, relating to requirement "Must have 3". Which code is performance-critical will be determined during implementation.

2.3.3.1. Front-End Testing

To test the front end, we will use Jest¹⁶. As we have no previous experience with JavaScript testing, we asked a TOPdesk front-end expert for a recommendation. He recommended Jest because it is modern, well-supported, and used by TOPdesk internally. With Jest we can test all of our required front end functionalities.

2.3.3.2. Back-End Testing

Individual components of the back-end will be tested using unit testing. The Java code on the back end will be tested using JUnit¹⁷, since we are all familiar with it, it is still popular in 2020 [15], and it is simple and lightweight [16]. As we see fit, we might need more advanced assertion or mocking tools. This will be determined during implementation on an as-needed basis.

For integration testing between our service and the API, Firkin offers a setup that can bypass authentication using a predefined mock deserializer such that it can be tested coupled loose from the actual program.

Python code on the back end will probably be tested using Pytest¹⁸ since some of us are familiar with it. It is modern and actively-developed [21], unlike other frameworks such as Nose¹⁹. However, we might reconsider as our testing needs might change during implementation.

¹³Agile Development Tool, <https://www.atlassian.com/software/jira>

¹⁴Docker containerization software, <https://www.docker.com/>

¹⁵Kubernetes container orchestrator: <https://kubernetes.io>

¹⁶Javascript testing framework, <http://jestjs.io/>

¹⁷Java testing framework, <https://junit.org/junit5/>

¹⁸Python testing framework, <https://docs.pytest.org/en/stable/>

¹⁹Python testing framework, <https://nose.readthedocs.io/en/latest/>

2.3.3.3. Code Coverage and Static Analysis

Code quality and test coverage will be assessed using automatic tools. We prefer an analytics infrastructure already used by TOPdesk, such as SonarQube²⁰, to benefit from better integration and quicker setup, but will also consider other tools if necessary. This will become clear during implementation.

²⁰Open source code quality suite, <https://www.sonarqube.org/>

3

Design

In the previous chapter, the initial research of the problem and solutions was described. In this chapter, we analyze the explicit design goals and the high-level design of the overall system and individual components, which are required to accommodate these goals. The implementation details of this design will be discussed in chapter 4 and chapter 5. Analyzing design goals can help provide guidance during design and implementation because we know which high-level goals the design and implementation should support [28].

We first explain the choices of our design goals and how they relate to the requirements. After this, we will explain the individual system design of the solutions. To help us justify the design choices for these systems, which we do in section 3.2, we created design schematics to aid in discussing design goals with stakeholders. Because these choices are justified by the design goals and they are linked to the requirements, we have traceability between our design and the requirements, such that the implementation is correctly prioritized and executed to fulfill the stakeholder needs [37].

3.1. Goals

In subsection 2.1.6, we explain that we have identified a list of design goals relevant to our project. We derived them from the implicit stakeholders' needs based on the requirements analysis. We verified these goals with our product owner to ensure that the design and implementation of our application satisfy the requirements. In the following list, we state each design goal we have, followed by a justification of why we chose it.

1. We mainly focus on **efficiency** because we are creating an automation program for the MHC report, which should improve the efficiency of the process of creating a MHC report.
2. We need **usability**, because the user of the application should have access to and understand how to execute the program, otherwise our application will not improve efficiency.
3. Our project lasts for only ten weeks, but we hope it will prove valuable enough to warrant further development after our time at TOPdesk ends. Therefore, **maintainability** is a goal of the design. We want to ensure that our code is readable and extendable such that a new team could further develop our application to fit future needs.
4. The application is required to deliver correct results, otherwise, the other goals are no longer relevant: the report would then still need to be created manually. Thus, **correctness** is one of our design goals.
5. Lastly, it is important to have a **reliable** application such that it is secure and available to use with real customer data, without breaking the TOPdesk environment or resulting in an error.

3.1.1. Persona

The process is as automated as possible but still requires a few manual steps. We created a persona of the user of the program to identify what the motivation and background is of the user we design for. This links back to our usability design goal: the user should understand how to use the application.

To give a picture of this user, Figure 3.1 displays a persona of a possible user, who is a CSM. This persona shows us that the users have a high priority for customer contact and already have to use many online tools to do their job. Time-consuming tasks such as executing the MHC hold them back from executing their actual responsibilities. Because the user desires to have a fully functional application, all our aforementioned design goals are relevant for a stakeholder to increase their job performance.

3.1.2. TOPdesk Integration

Because of the team's unfamiliarity with the internal structure, processes and technology stack of TOPdesk, the integration of our project within TOPdesk required a few weeks of our time. Although designing software



Figure 3.1: CSM Persona

inside a new working environment can be expected to come with difficulties [33], we consulted TOPdesk developers to design our product in a suitable way within our time constraint and to fulfill the stakeholder needs.

Our prototype is designed as a stand-alone application to reduce implementation time as much as possible. However, it is also designed in a maintainable way to be possibly transferred into the TOPdesk application by using a similar technology stack as TOPdesk. Another possibility would have been to fully integrate it inside the TOPdesk web application. However, running within TOPdesk would require even more stringent quality control since any bad version could potentially impact reliability of an entire TOPdesk environment. It would also cost more time, as we would need more time to understand the TOPdesk architecture to implement an integrated product. Therefore we decided not to integrate the product into the TOPdesk web application.

3.2. Structure

In this section a deeper look is given into the structural architecture of the system and its complexity and quality are described. In Figure 3.2 the general design of the application is shown with the communication flow between the individual systems. We will give a high-level overview of the architectural design and in chapter 4 we provide low-level implementation details.

3.2.1. Vue Front End

Because the user of the program described in the persona is used to working with the TOPdesk interface, our design has to use the TOPdesk style theme to adhere to their corporate identity. Because of the design goal of efficiency, this should support fast user interaction with our web application. Since the user is already familiar with applications with a similar design, it should be easier and thus faster for them to use the product. The components style designs can be seen in the figures in the manual that is in Appendix D.

We designed the application to be a single web page with few components, as shown in Figure 3.3, such that the main functionalities are usable and intuitive to the user. We think that making multiple pages with only a single component would increase routing complexity. We have little functionality and state on the front end, so a single page application makes sense for our use case.

Each component handles a single process step in the MHC process and calls the Django APIs when the

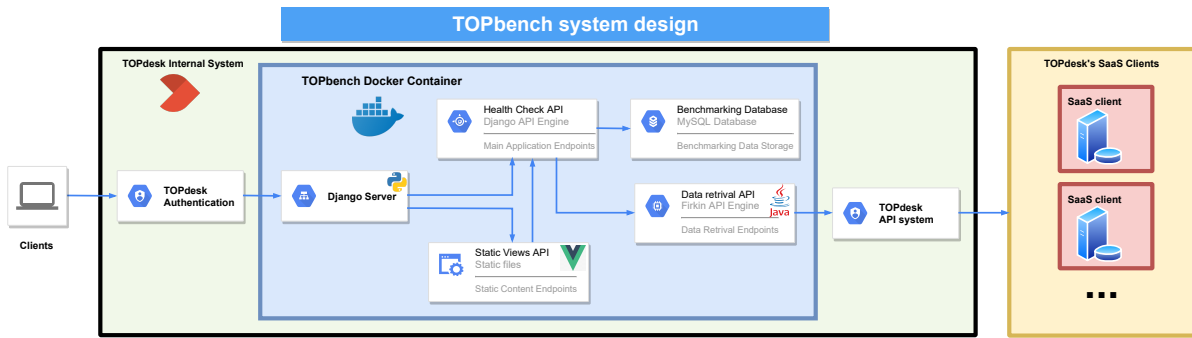


Figure 3.2: General system design.

user interacts with the system, such as a dropdown selection choice or a button click. Every component is placed on a single page and the components are hidden and shown based on the application's state, such as whether the user is logged in.

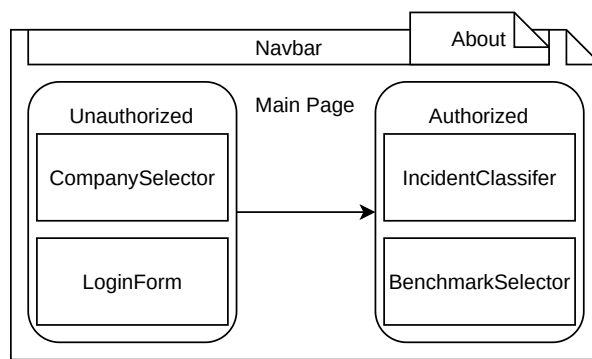


Figure 3.3: Vue framework pages & components.

3.2.2. Firkin Back End

On the back end, we created the necessary endpoints to the TOPdesk environment APIs using a Firkin service, shown in Figure 3.4, such that the interaction with the TOPdesk API is decoupled from our main application. This service is made for direct communication with the TOPdesk environment. Every data request uses a query to filter only the necessary meta-data from these incidents, such that the privacy of the incident content is protected and there is no unnecessary data transmission.

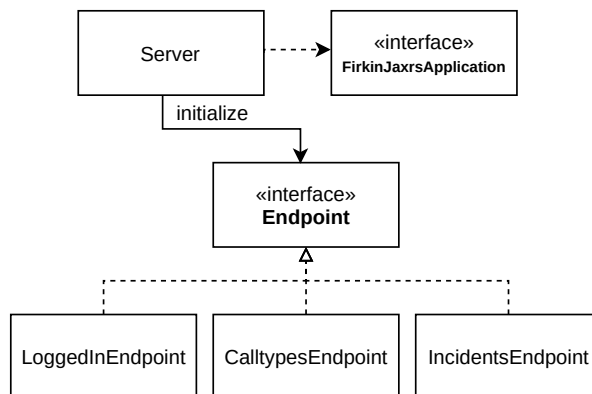


Figure 3.4: Firkin Service Endpoints

3.2.3. Django Back End

The Django service is the main part of the program. It hosts the API suite and provides a routing interface, which serves the front-end static bundles that are created by the program and routes the user to our main page. It calculates the necessary KPI values to put in the MHC report document and stores the values in a database. These values are then plotted and these plots are stored in the report.

3.2.3.1. Application Program Interface (API)

Aside from serving the main page, the router also forwards Hypertext Transfer Protocol (HTTP) requests from the client to the API suite of the program. The APIs used by the Django service are outlined in Table 3.1. These APIs execute different parts of the program as explained in the description, which are modularized into separate Python scripts for readability and so that they can be extended more easily.

Type	Path	Params	Description
GET	/api/companies/	None	Retrieve list of companies for selection.
GET	/api/company/	None	Retrieve current company data.
GET	/api/calltypes/	None	Retrieve call types of company.
GET	/api/report/	None	Retrieve the report document.
PUT	/api/calculate/	None	Calculate the KPI values.
POST	/api/credentials/	username, password	Store and authenticate credentials of company.
POST	/api/data/	calltypes, benchmarking	Store MHC and benchmark data.
POST	/api/company_selection/	company	Store company selection data.

Table 3.1: Paths and description of Django API suite.

3.2.3.2. Report & Presentation

TOPdesk already maintained a Microsoft Word and PowerPoint template that the consultants and CSMs use, which provide a user-friendly interface for both the CSM and the customer. Reports are also often created using PDF files, however, these cannot be edited after creation, while Microsoft Word files can be. Using Microsoft Word for the report helps with our usability goal since CSMs expect to be able to edit the report after creating it. Therefore we did not deviate from this format.

Using document template libraries, the calculated data such as the KPI values can be inserted into both a report and a presentation. These generated files can then be sent back to the front end. Using a readable template makes the generated fields to be easily maintained, since the template is made in Microsoft Word, which is easy to change if necessary.

3.2.3.3. Benchmark Database

To store the benchmark data we designed a database model which contains the MHC results of the companies. Initially a model was created which was extendable to more than six KPIs, but due to low priority to change the number of KPIs, we decided to simplify this schema such that there is no unnecessary complexity added, which should improve the maintainability of the database interaction.

TOPdesk is thinking about changing the KPIs used for the MHC and the FHC, but not in the near future. Hence, we limit our schema to have only six already existing KPIs in the MHC table as it can be seen from the new schema diagram Figure 3.5.

To anonymize the company's name such that it is not directly traceable, the name of the company is hashed before being stored. In practice, hashing does not provide effective anonymized data [7], however, the hashed names are never retrieved for the benchmark procedure and malicious users would need to enter the secured database in order to view the relevant data. Only developers and maintainers can access the database. The primary use of hashing is to make sure that they cannot easily identify companies when looking through the database. We could have created a more secure solution, but since we had a time constraint, we have chosen to keep it simple. Therefore we believe to have provided sufficient security to protect the privacy of the customers for this specific case.

We could also have not stored the company name at all. A problem that then might arise is that a company performs multiple MHCs within a time frame. The incidents and thus the resulting KPIs will then overlap in the database, creating inaccurate averages for the KPIs as some incidents are counted multiple times. Thus, the storage of the name of the company is required along with the time frame to ensure no duplicated data is stored.

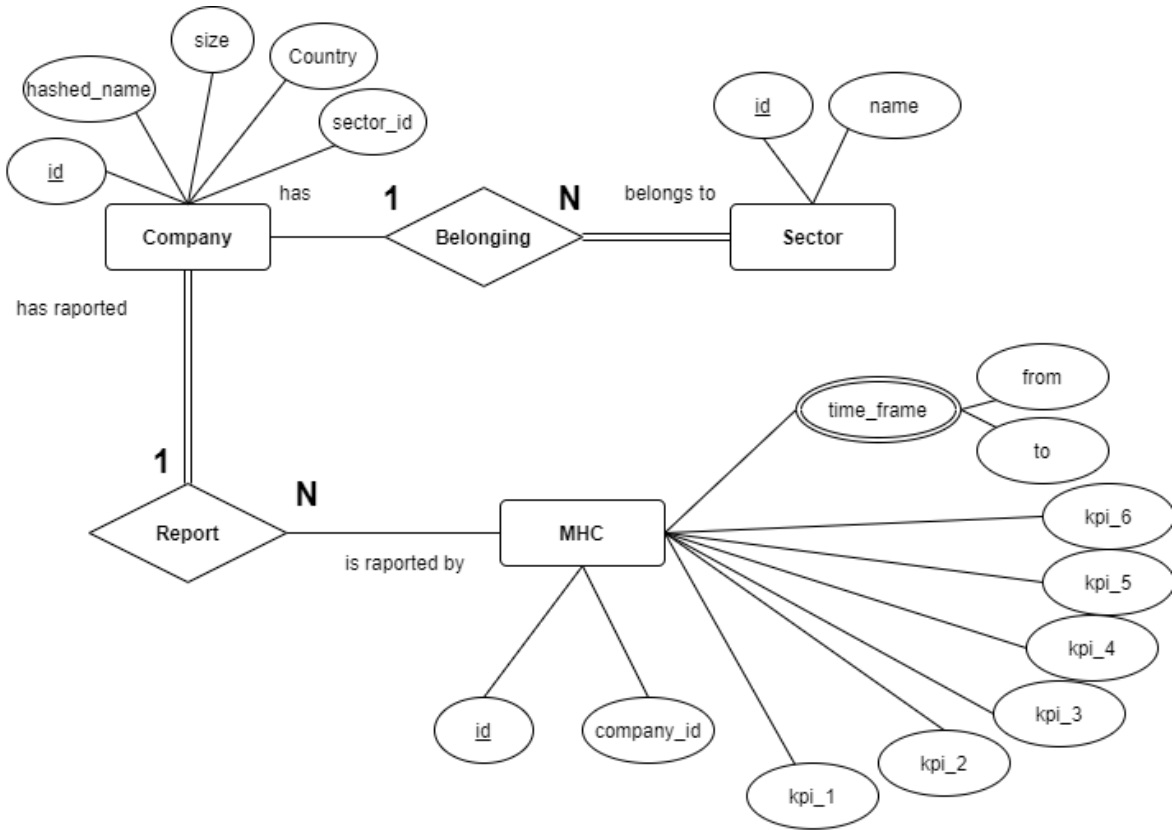


Figure 3.5: Benchmark database schema.

4

Implementation

In this chapter, we look at how we built the product. In section 4.1, we consider the agile workflow that we used and look at how it changed compared to how we envisioned it in the project plan. We discuss how we implemented the automation of the MHC and benchmarking in section 4.2 and 4.3, respectively. Finally, we explain how we deployed our application in section 4.4.

4.1. Agile Workflow

After the preliminary design and technology choices were made in the research report, we started working on the implementation. We decided in the Project Plan to develop the product using the Scrum methodology since we all had experience using the Scrum methodology.

4.1.1. Scrum and Research

Early during the research phase, we felt our use of an up-front research phase was not according to the Scrum methodology. We spoke to Joris Slob, a Scrum Master and Agile Coach at TOPdesk, to clarify these issues. He explained that software projects developed using Scrum usually do not have a research phase. How could we do our research in an agile way, still reaping agile benefits? A Scrum approach would focus much more on delivering an early product and gathering feedback, researching the problem, and designing the solution as parts are being implemented.

Of course, we still needed to do the research part of the project and produce a report on it because the course is set up in that way. We therefore, would not do Scrum 'by the books' according to Mr. Slob. He suggested we use elements of the Kanban method for our research. He said that Scrum is optimized for flexibility and continuous learning, whereas Kanban is less flexible but focuses on transparency about the project's state. He suggested we use Kanban techniques, such as using an issue board, to keep track of our research phase, still using elements of this agile methodology without fully implementing Scrum.

This discussion with Mr. Slob prompted us to start thinking about how we could build a basic but functional piece of software to check our assumptions about the product. Therefore we focused on developing this so-called "walking skeleton" as early as possible. The time limit of ten weeks meant that the implementation process should be carefully planned and executed. Every sprint, we reviewed our backlog, cleaning out or updating old issues, and determined which issues we wanted to work on for that week. We played planning poker [12] to determine the prioritization and the estimated time each issue was going to take. Planning poker helped us to discuss the time estimation and prioritization of issues such that the work could be finished within the time frame of one week. The general overview of the implementation phase is visualized in Figure 4.1, in which the specific implementation detail that was implemented in every Scrum cycle is outlined.

4.1.2. Flexibility

In our plan, we were very strict about how we were going to approach the project. We felt like we needed to be specific and concrete in our proposed approach. However, in discussions with Mr. Slob, it also became clear that this rigidity conflicted with the focus on flexibility of the Scrum process. We also discussed this with Frank Mulder, our Delft University of Technology (DUT) supervisor. In the end, we decided that we will be allowed to make changes to our project methodology if necessary but we should document and motivate these changes well, which we discuss in section 8.1.

We were happy about this course correction towards flexibility because then we could reap more of the benefits of the Scrum method, such as validating the product often, in our case by using the walking skeleton,

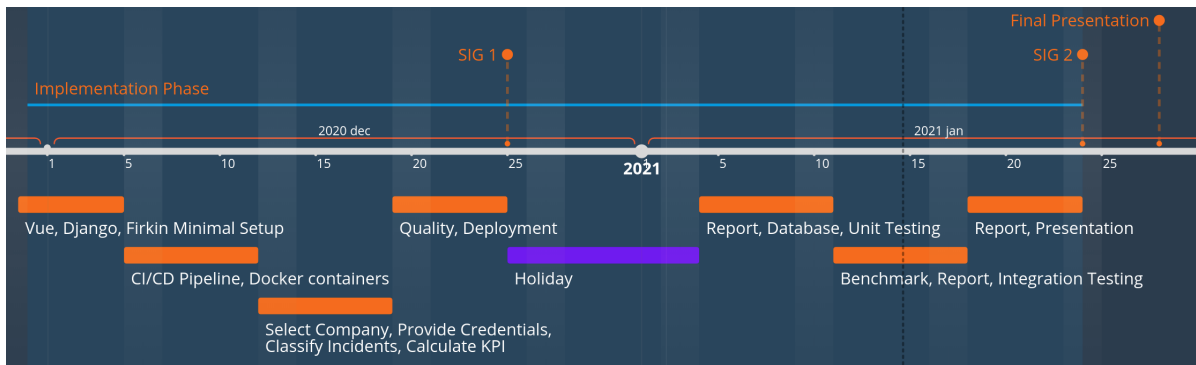


Figure 4.1: Implementation phase timeline

and learning more quickly, for example by using pair programming. We initially did not plan to try pair programming, but we ended up doing a fair amount throughout the project. Since our project was done during the COVID-19 outbreak, pair programming helped share our knowledge to handle certain issues more efficiently.

4.2. Automating the MHC

The automation tool was planned to be created first. The first few sprints of the implementation phase were dedicated to setting up the walking skeleton and gradually, the full process was implemented. In this section, we discuss the implementation details of the automation tool that generates the MHC.

4.2.1. Front End

On the front end, the consultant takes four steps for the process of creating a MHC: selecting a company, logging in, classifying incident types, and downloading a MHC report. A visualization and explanation of the program are explained in a manual in Appendix D.

To select a company to perform the MHC on, the home screen provides a drop-down menu which lists all the companies that have given permission to perform an automated MHC. The companies are sorted alphabetically to make them easier to find. However, looking for the right company name by scrolling through a large drop-down list is inconvenient as it takes much time to find the right company. TOPdesk has many customers and many of them might want the MHC to be automated. Therefore, a search box is implemented at the top of the drop-down menu. This solution scales better for a high number of companies.

Once a company has been selected, the user can provide credentials for this company in a login form. To provide security, the user can log in with an application password generated by the CSM. An application password is a password for a specific application and needs to be created once for each company before they can create a MHC. A standard consultant or CSM account does not have permissions to use the TOPdesk API, which our application uses, therefore a separately generated application password is required. This application password is used by our application to authenticate against the company environment. When the user logs in, the login form will either give feedback that the credentials were wrong, or it will bring the user to the next step.

The next step is classifying the incident types in the best-practices categories: *error*, *information*, *service requests* and unclassified incident types. The latter category is referred to by TOPdesk and our application as *waste*. This is implemented using a draggable list library so the user can drag each incident type to the desired category. We implemented it as a draggable list because we considered this a convenient and familiar way to categorize for the user.

Finally, when the user has classified the incident types, they will get an option to enable benchmarking. If this option is disabled, the MHC report will be requested without benchmark data and use the original TOPdesk norms to compare their KPI results. If this option is enabled, three options are presented: benchmark by industry, country or company size. A request for the appropriate report, with or without benchmarking, is made to the back end, which in turn returns a report.

4.2.2. Back End

The back end consists of two parts: a Django service and a Firkin service. The front end makes API calls to the Django service based on the steps that the user needs to take. The Django service then makes an API request to the Firkin service.

4.2.2.1. Firkin Service

The purpose of the Firkin service is to connect with the TOPdesk environment: it provides authentication, specifies the required fields, and paginates the results from the TOPdesk API. It consists of three endpoints: one endpoint for authentication, one which requests the call types of the customer, and one that requests the recent incidents of the customer.

The Firkin service was implemented to decouple the API calls from the connection with the TOPdesk environment. During implementation we found out that this did not add much value however, since little code is required to communicate with the TOPdesk API.

We also thought we would need the service to run inside of a client's TOPdesk environment to be able to access their data. During the project we found out that this assumption was incorrect and we could retrieve the data using the environment's REST API accessed by HTTP requests from services other than a Firkin service, such as a Django service. Having a separate service for this became unnecessary. This led to the point that the Firkin service was a standalone server inside our own Docker container, not adding the expected value we thought it would have based on our research. Due to time constraints, the Firkin service is currently still part of the application. We will give recommendations for different solutions to having a separate service in section 8.3.

4.2.2.2. Django Service

First, the front end is pre-built by the Django service into static script and style files by webpack¹ to integrate with the Django back end. These files are served on request of the base Uniform Resource Locator (URL).

After this, the front end should receive the company selection from the Django service. When a company is selected, the selection is put into a session cookie so it can be used in further requests. This cookie contains the company's name, environment URL, industry, size and country.

When the Django service makes a request to the TOPdesk API to check the credentials of the consultant, it makes a request to the Firkin service to check whether they work on the environment using the URL of the TOPdesk environment of the company. When authorized, the Django service stores the credentials in a session cookie. It returns an HTTP status code; code 200 (OK) means valid credentials and 401 (UNAUTHORIZED) means invalid credentials.

If the user has provided valid credentials, the Django service makes a request to the Firkin service which retrieves the incident types and returns them for classification to the front end.

When the user has approved the classification of the incident types, the Django service will request the incidents data from the Firkin service. It uses those incidents to calculate the six KPIs by filtering the incidents and dividing them by the total amount of incidents to get a percentage value, according to the process explained in the MHC manual [30]. These calculated values are also stored in session cookies.

Classifying Incident types

Automatically classifying call types would save the consultants time, since they need less time to classify these incident types themselves: they would only need to verify the results. The first step to create an automatic classifier with a ML algorithm was to aggregate data such that we could train an initial NLP model with the specific task of classifying call types.

Unfortunately, no such data set existed yet, which made it hard to create a system that could actually improve the efficiency of automating the MHC, since we would need to create this data ourselves. We weighed this potential efficiency increase against the amount of work it required to create this data and train the model. We did not think that this improved efficiency was worth it for the amount of work. Implementing it would be time consuming and we had a time constraint. Additionally, according our product owner, classifying incident types is usually not a difficult or time consuming task. Therefore we decided to not provide ML assistance when classifying incident types.

We do see the potential of time saving in the future for TOPdesk by using this feature, however, as we will explain in section 8.3. Therefore we did set up a system which collects this data from the users that execute an MHC with our application. We expanded our database schema to include the storage of the incident types'

¹Static module bundler, <https://webpack.js.org/>

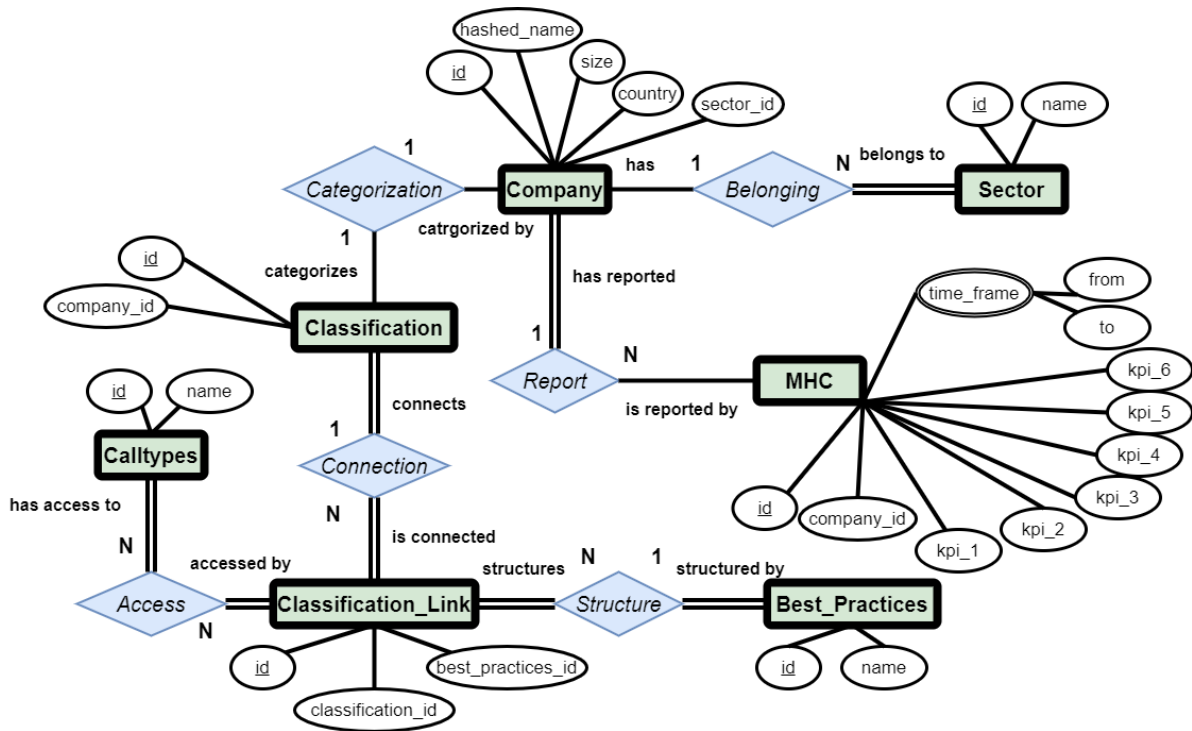


Figure 4.2: Benchmark database schema

classification as shown in Figure 4.2. This means any MHC performed using our tool adds to the data set that is required to possibly include an ML-assisted classification system in a future version.

Document Generation

When the users presses the download button, the report is created and downloaded, using the docxtpl library², a library for modifying Microsoft Word files. The program places the calculated values inside a template report. For each KPI, an explanatory texts is added based on the value of the KPI. These texts explain, based on this value, how well the customer performs on this KPI, so the reader can understand what the value means for them.

4.3. Benchmarking

Aside from handling client interaction and report creation, the Django service is also connected to the My Structured Query Language (MySQL) database where it stores metadata of the company and the KPI values of their MHC results. It contains a list of pre-defined sectors, which we link to the company. Before adding the MHC results, we check if the company name already exists in the database and has performed a MHC in the past year. If so, the results overwrite the previous results of their MHC. Otherwise we add the results as a new entry to the database.

We hash the company name, as explained in subsection 3.2.3.3. We decided to use the BLAKE2s³ algorithm to hash the name of the company. We chose this algorithm as it is fast enough for our use case and results in no direct traceability between a company's name and its KPI results.

We use the Django database object model API because it is high-level, which aids maintainability. It has the functionalities we need, such as built-in input sanitation⁴, test databases⁵ and static data loading⁶.

²Python docx template filler library, <https://docxtpl.readthedocs.io/en/latest/>

³Cryptographic hash function, <https://www.blake2.net/>

⁴Overview Django's Security Features, <https://docs.djangoproject.com/en/3.1/topics/security/>

⁵Django tests, <https://docs.djangoproject.com/en/3.1/topics/testing/overview/>

⁶Fixtures, <https://docs.djangoproject.com/en/3.1/howto/initial-data/>

4.3.1. Visualization

For visualization, we considered both the box plot and the violin plot, as explained in subsection 2.2.2.3. The box plot is simpler and thus easier to understand than the violin plot, but lacks detail. The violin plot is more detailed, but might be harder to understand. The stakeholders liked the violin plot the best, because it looks more attractive and conveys information well to the client. Therefore we have implemented a violin plot. The data is inserted in the data visualization library `seaborn`⁷, which creates a violin plot of the benchmark.

The results are put inside the report document with the relevant text. We use a different template for benchmarking, because this version includes more data entry points. The relevant norms for KPIs for this company can be deduced from the created visual by comparing it against its own result. For example, a company scores a 20% on KPI six, which would be categorised as average by the previous norms. If 50% of the other companies in the same industry have a score between 25% – 45%, this would not be accurate and should actually result in a low score based on the benchmark.

4.4. Deployment

Because our program consists of many separate systems, we required a way to execute and test our program automatically. This makes the development process more efficient, as we do not need to start and test every system separately each time we run our code. With help from Joep Weijers, a TOPdesk developer, we containerized our application with Docker and startup scripts. We developed the application running our code and a TOPdesk environment locally. Once a week we pushed our development branch to the master, which was then automatically deployed by pushing the containers to the internal registry and pulling it onto a virtual machine running the web hosting service. After the container is pulled, it is started and listens on the exposed ports for requests. We deploy on an internal TOPdesk server which is secured by TOPdesk multi-factor authentication, so only TOPdesk employees can access our application.

⁷Statistical data visualization, <https://seaborn.pydata.org/>

5

Testing and Quality

5.1. Testing

TOPdesk might continue with the development of our application. Therefore we want to have high maintainability so that TOPdesk can continue working on this project. To this end, we have written front-end and back-end tests, that aim to verify the our code's correctness. In this section, we discuss how we tested our code.

5.1.1. Front-End Testing

We used the Jest testing library together with `test-utils` offered by Vue. Jest was used for unit-level testing of functionalities like making API calls while `test-utils` was used for component rendering. We also mocked the network requests to the Django API using Jest's mock library to verify that the front-end methods that make calls to API work as expected. Using mocking allows us to control the behavior of the APIs. This way, the tests can focus on the logic inside the function.

5.1.2. Back-end Testing

The main functionalities are located in the back end. So testing it was important, as the rest of the components rely on it to work correctly. Debugging through the front end or logs would be difficult and inefficient. Unit testing properly helps us identify bugs more easily.

5.1.2.1. Django (Pytest)

This part of the application was the most difficult to test as it was the middleware for our application. It was directly connected to the front end, the benchmarking database and the Firkin service. The Django back end interacts with all these different system components. To check whether these components are all called, we mocked using pytest's `Mock` class, such as the requests to the Firkin service. The modularized scripts that calculate the KPIs and provide other utility functions are also unit tested.

For the benchmarking database, the model library from Django is convenient because it only needs a database connection when testing the interaction with the database. The data is added to the database and then cleaned up after each test run. This allows us to quickly and efficiently test the correct interactions with less code.

5.1.2.2. Firkin (JUnit)

The Firkin service is tested using JUnit which we use to unit test our methods. We also use the Mockito mocking library to simulate connections to the TOPdesk environment API. Since the Firkin service is a thin wrapper around the TOPdesk API, we decided not to unit test it apart from checking status codes as a smoke test. Furthermore, we used manual testing for verification.

5.1.3. Integration testing

Because we are creating a multi-component application, it is important to test the integration between the systems [8]. However, setting up more testing requires more time and the complexity of integration testing also increases [23]. The tests could possibly be “flaky”, which means the results of the tests are not consistent, as the tests would depend on many systems and their states. Due to these concerns and the limited time available during this project, we decided that integration tests such as end-to-end testing would not be implemented for our application.

5.1.4. End User testing

To test the fully integrated software, we requested CSMs to perform an MHC through our application with real customer data. We analyzed the amount of time the user takes to navigate through the full process as we described in Appendix D and asked for their feedback about the usability of the interface and the functionalities. We will not include creating the user account and application password in our test because we are not allowed to view this data. Therefore we will test the time that is required to create the MHC from selecting a company until the report document is downloaded.

We also contacted many customers to ask for their permission to test our application on their TOPdesk environment. While we are not allowed to log in to their environments, consultants and CSMs are, which allowed them to test our application for us. The test reflects how efficient the application is for the user working with a real environment. In chapter 6 we elaborate on the results of these tests.

5.1.5. Test-Driven Development

For the development of certain functionalities, it was not known how it would be implemented in advance, but we knew the exact result it should output. Therefore, we first wrote the tests and then later on adjusted the code in order to make the tests pass.

We used this method in particular to test the model library of Django which interacts with the database. We first devised a test to get every object from the database and then wrote the functionality. After that, we wrote a test to make sure a filtering function works. This process was repeated until all functionalities and their test cases were devised and that all the tests passed.

5.2. Quality

To make sure that our code has as few code smells as possible, we decided to pick tools that would assess the quality of our code through static analysis. Since we used three different programming languages (JavaScript, Python, and Java), we wanted to find tools that would analyse our code and provide us with quick feedback in the local console as well as on the pipeline.

5.2.1. Technologies

To improve the code quality we used SonarQube, which is a static analysis tool that analyzes potential bugs, vulnerabilities and security issues in the code. In section B.5 we show the final quality scores for the application. It also shows technical debt which gives you tips on how to make your code more maintainable. It imports the coverage results of the tests and shows the uncovered lines and it also alerts us about code duplication.

Apart from that, we had other tools for each of the languages. For Python we used Pylint¹ which is a linting tool that makes sure you follow the PEP8 style created by the founder of Python [25]. For JavaScript we chose ESLint² for standards. Finally, for Java we used Checkstyle³ which is Maven's standard utility for checking the style of code.

5.2.2. SIG Evaluation 1

To evaluate our source code, we were required to let our source code be reviewed by SIG. In this section, we discuss the results of the first submission to the SIG. We will discuss the most critical feedback we got on our submission, how we improved our code, and argue about each feedback item's relevance.

5.2.2.1. Results

The Figure 5.1 shows a summary of the results of the feedback. It should be noted that the grades range from one to five stars, where more stars represent higher quality. The first submission got a test code ratio of 31.6% and a maintainability score of 4.6 stars. Some components even got 5.5 stars, which we suspect is a glitch. They show no feedback inside them, so we assume it should have been five stars.

We consider the test code ratio of 31.6% to be lower than how much we wanted to test. We have aimed to improve our code coverage after we received the SIG report. According to SonarQube, we currently have 69.8% coverage. However, we will only know the results from the second submission after this report is finished, so we will not know if the SIG results will actually improve. However, since we have focused more on

¹Source-code, bug and quality checker for the Python programming language, <https://www.pylint.org/>

²Open-source JavaScript linting utility, <https://eslint.org/>

³Apache Maven Checkstyle Plugin, <https://maven.apache.org/plugins/maven-checkstyle-plugin/>

testing, and have reached higher coverage in SonarQube, we expect to reach a similar coverage of what we reached in SonarQube.

We consider the score of 4.6 stars reflects the quality of our software reasonably well. We got more issues from SonarQube than from the SIG review. We consider that the SonarQube results are in line with the results from the SIG, since SonarQube gave more issues, however each issue was a bit smaller and thus less significant. While the SIG review covered high-level topics, such as duplicated code blocks and unit size of files, SonarQube pointed out issues closer to a single line of code, such as duplicated strings and variables that could be constants.

System fact sheet	
Name	Topbenchwebapp
Division	2020 December
Suppliers	TOPbenchers
Size	1 PM
Test code ratio	31.6%
Maintainability	★★★★★ (4.6)
Volume	★★★★★ (5.5)
Duplication	★★★★☆ (4.1)
Unit size	★★★☆☆ (2.8)
Unit complexity	★★★★★ (4.7)
Unit interfacing	★★★★★ (4.8)
Module coupling	★★★★★ (5.5)
Component balance	★★★★★ (4.6)
Component independence	★★★★★ (5.5)

Figure 5.1: First SIG submission results

5.2.2.2. Unit Size

The SIG manual states the following about unit size: “Software products where more source code resides in large units are deemed to be harder to maintain. To maximize the rating of a product for the unit size property, the software producer should avoid large units” [24].

The lowest score for maintainability was for unit size, scoring 2.8 stars. The worst issue here was the number of lines in the configuration file `settings.py`, with 99 lines of code. However, since this is a file that aggregates the settings for our Django service, we do not consider this an issue. The most obvious way to solve this issue would be to split the settings up into multiple files, and then import them all in `settings.py`. We think this would not improve the maintainability and therefore decided not to implement this feedback.

The second worst issue was for `LoginForm.vue`, with 31 lines of code. All of this code is part of the Vue component and is already split in multiple units of code like methods and variables. Therefore we decided not to implement this feedback either.

While we did not implement the feedback, after we got our feedback report we aimed to ensure short units of code that have a single responsibility to achieve the highest possible code quality.

5.2.2.3. Duplication

It is good practice to avoid duplication in our code, because if a change is made, it might not be updated in all the duplicated code blocks, leading to inconsistent results and potential bugs.

We got four feedback points from SIG on duplication, three of which were about Vue. We were duplicating login input fields and the incident classifier columns. We used *v-for* to loop over data and remove duplication.

The final feedback point was about `kpis.py`, where we duplicated a method. We solved this issue by refactoring this file into a class which allowed us to extract the duplicated method.

Other indicators, such as volume and unit complexity, were rated at 4.6 stars or higher. Due to the high score and our time constraints, we did not prioritize implementing this feedback.

5.2.3. SIG Evaluation 2

At the end of week 9, we resubmitted our code to SIG for the second evaluation. The results are summarized in Figure 5.2.

System fact sheet		
Name	Topbenchwebapp	
Division	2020 December	
Suppliers	TOPbenchers	
Size	3 PM (+1.65 PM)	
Test code ratio	122.8% (+91.2)	
Code touched	1,358 LOC	
Code removed	173 LOC	
Maintainability	★★★★☆ (4.3)	▼ 0.32
<u>Volume</u>	★★★★★ (5.5)	= 0.00
<u>Duplication</u>	★★★★★ (5.0)	▲ 0.88
<u>Unit size</u>	★★★☆☆ (2.7)	▼ 0.07
<u>Unit complexity</u>	★★★☆☆ (3.2)	▼ 1.42
<u>Unit interfacing</u>	★★★★☆ (4.0)	▼ 0.81
<u>Module coupling</u>	★★★★★ (5.1)	▼ 0.37
<u>Component balance</u>	★★★★☆ (3.8)	▼ 0.87
<u>Component independence</u>	★★★★★ (5.5)	= 0.00

Figure 5.2: Second SIG submission results

5.2.3.1. Results

Starting at the top of the figure, it's clearly visible that the size and test code ratio of our code base increase significantly. We increased the amount of test code faster than we increased the amount of code in our components. We are happy with these results and they clearly show our focus on testing.

Maintainability decreased overall between the first and the second SIG evaluation, whereas *code duplication* decreased significantly and *code volume*, *module coupling* and *component independence* remained good. The overall decrease in maintainability is mostly due to a decrease in our performance on the *unit complexity*, *unit interfacing* and *component balance* categories. We will go into these results below.

Unit Complexity

Unit complexity decreased significantly. Taking a closer look, the decrease is caused mostly by an increase in the complexity of the report generation, database filtering, and TOPdesk environment communication parts of our program. As the functionality of these parts increased, their complexity also increased. These parts, specifically, have a lot of preconditions and checks to ensure correctness. In the future these could be moved to separate functions to decrease complexity.

Unit Interfacing

Unit interfacing decreased from 4.8 to 4.0, mostly because the number of parameters to some functions increased. However, the maximum number of parameters across our code base is 4. We feel that this number of parameters does not negatively affect maintainability, but we understand that the size of these interfaces should not be increased further. When adding future functionality, these functions should not be extended but new functions should be added.

Component Balance

Component balance refers to how the size and complexity of our program is divided between its different components. Most of the size and complexity of our program is contained in the Django back end. We are aware of this fact and we think that, for future extensions, it should be carefully considered whether to extend the Django back end further or to build new functionality in a new separate component.

5.2.4. Pipeline

We used GitLab CI pipelines, which are automated processes that runs stages that validate the correctness of the current version of the software. Pipelines are convenient because they are able to automatically detect mistakes in our code, such as failing tests or styling errors. The results are shown in a clear interface so the issues are easy to identify. We set up the pipelines in such a way that code could only be merged when all pipelines succeeded. This forces us to fix the pipeline before merging a merge request. We decided to use the built-in GitLab Runner which runs stages using Docker with a pre-defined order shown in Figure 5.3.

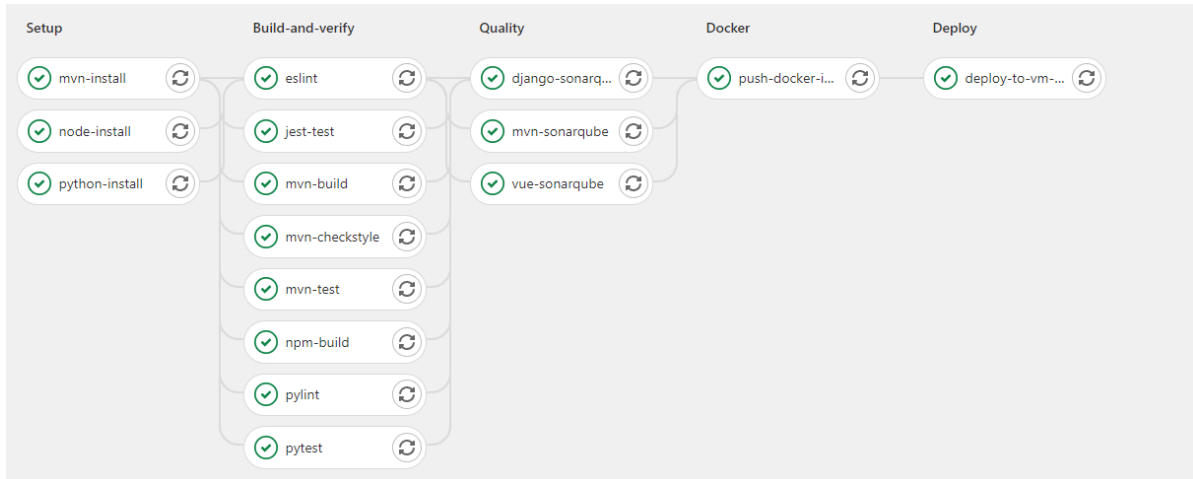


Figure 5.3: A passing GitLab pipeline

1. **Setup:** Install dependencies for npm, pip and mvn.
2. **Build-and-Verify:** Run tests, linting and builds the mvn target and the npm static files.
3. **Quality:** run SonarQube analysis and upload.
4. **Docker:** Builder Docker container and push to internal TOPdesk Docker registry.
5. **Deploy:** Run pull to Virtual Machine and start Docker containers.
6. **Migrate (optional):** Run database migration script if changes were made to the database schema.

5.2.5. Code review

We conducted a code review on each of the merge requests that we created to ensure code quality. The pipelines ensured that we could focus our code review efforts on asking questions about implementation details and suggesting possible improvements. We also manually tested merge requests to verify that they worked as expected before merging them.

5.2.6. Performance testing

One of the requests from TOPdesk for our automation program is that it can handle enormous amounts of data. To this end, we performance-tested our program using large mock databases that have 100,000 and 500,000 incidents, which is typical of the very largest TOPdesk customers. The program handled the large demo databases well, generating the report in less than a minute. We elaborate on how to improve section 8.3 these tests represent production environments more.

6

Product Evaluation

6.1. Evaluation of Requirements

This chapter examines whether the product we have developed solves the problem we set out to solve. To this end, we first look at the requirements listed in subsection 2.1.6 again. First, we present a table in which we systematically evaluate our product for each requirement. Secondly, we evaluate feedback from our stakeholders to assess whether they are satisfied with the product. Finally, we state the key findings from evaluating our requirements in section 6.4.

6.1.1. Functional requirements

Name	Requirement	Satisfied	Elaboration
Must Have 1	The consultant can generate the MHC report automatically, apart from having to classify custom call types.	Yes	Using our program, after the classification of incident types, the report can be generated automatically.
Must Have 2	The customer can compare the results of their MHC to other companies within a certain domain so they can see how they are doing compared to their competitors and see how they can improve.	Yes	We have implemented a benchmark tool as explained in section 4.3. The tool allows the user to choose a subset of TOPdesk clients to compare themselves to.
Must Have 3	The consultant can choose which call types should be grouped together when executing the automated MHC.	Yes	The user can do this using our call type classification interface.
Should Have 1	The product can assist the user in classifying categories, using technology such as ML, that have the same best practices definition but a different call type.	No	Without a sufficient data set this was practically impossible. We did implement a system for storing this classification using the call types classification interface to generate a data set for future use.
Could Have 1	The CSM can generate a presentation based on the report automatically.	No	Although requested by stakeholders, this was not implemented because of time constraints.
Could Have 2	A potential new customer can see the performance of TOPdesk specifically in their domain before they purchase it.	No	In coordination with our stakeholders, we have determined that this feature might be undesirable in certain industries, therefore we decided not to implement it.

Continued on next page

Table 6.1 – continued from previous page

Name	Requirement	Satisfied	Elaboration
Could Have 3	The customer can see the generated report and presentation automatically through the TOPdesk software after they have been presented by the CSM.	No	We decided to leave this out of scope because of limited time and legal concerns with regards to storing the MHC by default.
Could Have 4	The customer can press a button within the TOPdesk software to request an MHC, and their benchmark status, at any time.	No	Stakeholders indicated that they want to retain control over the timing and presentation of the MHC.
Could Have 5	The product will use some new KPIs that give a better view of the maturity level than the current KPIs.	No	Various new KPIs might be of value, but we decided in coordination with our product owner not to change the KPIs since this must be coordinated across the entire company.
Won't Have 1	The customer can generate the MHC report themselves at any time.	No	Stakeholders indicated that they want to retain control over the timing and presentation of the MHC.

Table 6.1: Evaluation of functional requirements.

6.1.2. Non-functional requirements

Name	Requirement	Satisfied	Elaboration
Must Have 4	The product produces a report in less than half of the amount of time that it takes for a consultant to produce it.	Yes	Our measurements during user testing has shown that using our tool, a MHC report can be generated in an average of two minutes.
Must Have 5	The benchmark tool should show correct statistics for the selected comparisons.	Yes	This is tested using the approach set out in chapter 5.
Must Have 6	Only active authenticated users who are authorized to use the software can access the software.	Yes	This is tested using the approach set out in chapter 5.
Must Have 7	Data usage must be GDPR compliant.	Yes	Our program only works for Dutch clients who have given us permission to use their data. We address further GDPR concerns in Table B.1.
Should Have 2	Performing the automated health check should not negatively affect the performance of the TOPdesk environment.	Yes	As explained in chapter 4, we have discussed potential performance impacts with relevant TOPdesk engineer and they are satisfied with the impact of our program. Furthermore, we have run stress tests using large mock databases, as explained in chapter 5.
Should Have 3	The final application should be usable in current versions of Google Chrome (87.0) and Microsoft Edge (87.0).	Yes	This has been tested manually and with user tests.

Continued on next page

Table 6.2 – continued from previous page

Name	Requirement	Satisfied	Elaboration
Should Have 4	The final software product must be well-documented using code comments and a manual.	Yes	The final product has been checked for documentation coverage by automatic tools, for style errors using linting tools, and for code commenting using manual inspection. A manual has been created and is attached as Appendix D.
Should Have 5	There should always be a working version of the software running after each sprint.	Yes	This was ensured by using CI/CD and manual inspection of the deployed product after each sprint.
Could Have 6	The user interface should be in the corporate identity of TOPdesk software.	Yes	We consulted the TOPdesk design guide and used internal TOPdesk layout and design tools as much as possible to ensure consistency in its design.
Could Have 7	The final application should be usable in current versions of Mozilla Firefox (83.0).	Yes	This has been tested manually and with user tests.

Table 6.2: Evaluation of non-functional requirements.

6.2. Requirements evolution

As is visible in the lists of requirements above, some initial requirements were changed or even not implemented at all. As we had envisioned the possibility of this happening during our planning phase, we adopted an agile methodology to make sure that we could deal with changing requirements. For example, we decided not to implement machine-learning assisted incident type classification. Because we worked in an agile fashion, after realizing the effort it would require to create the necessary data set, we put this feature on low priority to focus on other functionalities.

Other requirements were changed because they became irrelevant or undesirable, such as Could Have 2, in which the benchmarking data would be available to prospective customers. Some stakeholders suggested this feature to us early on. However, when researching it in more detail, multiple stakeholders indicated they would not want this feature for various reasons, such as confidentiality for their clients or higher priority for other features.

6.3. User Experience Evaluation

To evaluate the user experience, we set up online sessions in which target users of our product, namely consultants and CSMs, could test our application. We set up a few assignments based on what we expect will be the most common use cases of our product. We first asked them to do three assignments: generate a report with benchmarking based on industry, generate a report with benchmarking based on country, and generate a report without any benchmarking. During these assignments, they had to use all aspects of our application:

1. Select company
2. Log in
3. Classify incident types
4. Enabling and disabling benchmarking depending on the assignment
5. Download report
6. Enter consultants name in report
7. Check report for mistakes

We timed these assignments to see how much our tool reduced the execution time of the MHC. On average, the assignment took the users two minutes.

Afterwards, we interviewed the users to get as much feedback as possible. We asked them, among other items, about the purpose, speed, navigation, usability, and attractiveness of our application. The purpose of our product and its navigation were clear to the users. The users were delighted with the speed of our application. They were generally happy with the layout and usability of our product. They had some remarks

about the drag-and-drop incident classifier, however. This element was not perceived as user-friendly, and the dragging of an incident type sometimes took them multiple attempts.

It was particularly interesting to hear the users describe what they were doing and thinking about as they used our application. We have seen and used our software so often during development that we risk becoming blind to some issues that others might notice easily. This helped us improve the layout and some unclear wording. It was really valuable to see users figure out our application in real time. We did not show them the manual (see Appendix D) beforehand, meaning they had to figure out how the application worked.

In general, most of the feedback we received from the users was about not knowing exactly what they were supposed to do next. We took this feedback seriously and updated the user interface to be more user-friendly.

All users were able to complete the tasks we asked them to do. Any information on how to complete tasks that we gave them is also listed in the manual, so we are confident that our target users will be able to use our application without our help in the future. All of the users said that they would be interesting in using our application for future MHC executions.

6.4. Key Findings

Based on our evaluation, we are able to present two key findings:

1. After executing the end-user tests as described in chapter 5, the average execution time of the MHC was reduced from two hours to two minutes.
2. The benchmark tool gives a domain-specific comparison against similar companies for the results of the MHC. By showing the companies performance compared to the benchmark, we create new norms based on substantial, real-world data and specific to the company's domain. This allows the CSM to provide more insight to the customer about their specific service level.

7

Conclusion

TOPdesk wanted the process of making their MHC automated, as the current process is labor-intensive and manual. Further, since the norms of the KPIs are arbitrary and not domain-specific, they wanted companies to be able to benchmark themselves against other companies in their domain.

We managed to automate the process of making a MHC report. After performing our user experience evaluation, it became clear that performing this MHC now takes significantly less time, namely two minutes instead of two hours and requires significantly fewer manual steps. We also implemented a benchmark feature that allows a company that has given permission for it to compare themselves to companies in the same sector or country or of similar size. The result of this feature is visualized using a violin plot. This provides a visual aid for CSMs to convey to a client how well they are performing.

Our product serves as a proof of concept that automating the MHC has the potential to save many hours per year for CSMs. Also, the benchmark feature allows customers to get an idea of how well they are performing by using new norms based on substantial data. With this proof of concept, we can answer our initial research question and conclude that our proposed solutions can improve the process of determining the level of Service Excellence on the Maturity Model.

We demonstrated the feasibility and value of automating the MHC, as well as adding benchmarking for more relevant KPI norms. Since we fulfilled all of our Must Have requirements and created a valuable product for TOPdesk, we consider the project a success.

8

Discussion

As we mentioned in chapter 7, we consider this project a success. However, there are still some discussion points we have about our work. In this chapter, we reflect on the process and limitations of our project in section 8.1 and 8.2, respectively. We make recommendations for further development in section 8.3. Finally, in section 8.4, we consider the ethical implications of our product.

8.1. Process evaluation

As we have stated before, we used the Scrum process during implementation. Each sprint lasted one week, which we found to be a fair amount of time given the project's relatively short duration. Each week we had a retrospective meeting, which reflected on the past week, which allowed us to give each other feedback and improved our teamwork quickly. The Scrum Master role cycled between all members every two or three weeks, which allowed us, as a team, to continuously try to improve the Scrum meetings by experimenting with new formats. Overall we enjoyed this approach, as this gave everyone a chance to experience this position and get feedback from other team members.

By recommendation of Mr. Slob, we decided to use more free and flexible types of retrospective meetings to keep it exciting and engaging for everybody. Eventually, we also decided to change the day of this retrospective meeting from Monday of the next week to Friday of the same week to get all feedback from the previous week because we noticed that we sometimes forgot important feedback over the weekend.

We could also have changed the days we start and end our sprints from Monday and Friday to Wednesday and Thursday. This could possibly improve the productivity of the sprint review and planning. This is because we experienced the so-called “Monday Fog”, where we had to get back into the working mindset after the weekend, and ‘Friday Fatigue’, where motivation and tiredness occur after a long work week [36].

We would have done a few things differently in hindsight, given the knowledge we obtained during the project. Our time estimation could have been less optimistic. We were quite ambitious with possible features that would improve our product's experience, such as machine learning to classify incident types. Our research phase and getting familiar with the TOPdesk environment took longer than expected, which left us with less time to implement than we estimated. Emailing clients for permission was something we expected CSMs to be responsible for and cost much valuable time. In the end, we had to prioritize which features we wanted to implement. We followed the requirements, focused on satisfying must-haves and leaving out could haves and other nice-to-have features.

8.2. Limitations

Due to COVID-19, we had to work from home for the entire duration of the project. This meant we could not directly talk to any TOPdesk developers when we had problems or to stakeholders for feedback. Instead, we had to plan an online meeting with them and use screen sharing for technical assistance. We also required video calls for all of our meetings and to perform pair programming. We were excited to work in a company but missed getting coffee and having drinks together.

8.2.1. Stakeholders

We aimed to deploy our working application weekly after each sprint, which required time and assistance from TOPdesk colleagues. At first, the incentive to deploy the web application was to have it available for the stakeholders to provide feedback. Due to limited availability, most feedback was provided during designated stakeholder meetings, where our progress was presented. Other issues were resolved with our product owner

in separate meetings.

At the start of our product, our supervisors and product owner assumed we could access clients' login credentials to test using their data. However, due to the unforeseen necessity of a VOG¹, which we could not acquire due to time constraints, we needed to fall back on our stakeholders to help us test with real customers. Luckily, we already had a deployed version running that was accessible for the consultant to test with customer data.

In the individual stakeholder meetings, it became clear that some stakeholders have different ideas about the project than others. Some issues arose because of this situation, such as the availability of this tool to the customer instead of only to the CSMs and what elements should be in the template MHC document. We discussed and prioritized these issues in collaboration with our product owner.

8.2.2. Technologies

During our work, we have also faced a few problems related to the technologies we used. Some of our team members ran into problems with the laptops that TOPdesk gave us to work on, for which we had to make an appointment and visit the company separately. On top of that, some laptops sometimes had too little storage to work with. As a result, some members had to clean up their main storage drive to keep developing.

We also faced problems on the software side, namely with Django, Docker, and incompatibility with dependencies. First, we faced some compatibility issues between our Django service and the MS SQL database, the latter of which TOPdesk mainly uses. We found out that MS SQL is not officially supported by Django, which makes integration with the model difficult because we would need to use possibly unreliable third party wrappers. Another popular relational database supported by Django which allows us to use the Django models is MySQL. We decided to switch to this database. Other problems were related to Docker breaking for our product, too little RAM to run Docker and npm or maven dependencies breaking due to incompatibility. Most of these problems were due to our inexperience. Luckily we got the necessary help we needed from TOPdesk employees.

8.3. Recommendations

During our project, we created an application that will serve as a proof of concept for TOPdesk. Considering the time-saving potential of automating the MHC, it seems worthwhile to implement an extended version of our application into TOPdesk. We think that the time saved by CSMs no longer having to perform this check manually should be a significant financial motivation for TOPdesk to automate the MHC soon.

8.3.1. Automating the MHC

To make the MHC more automated than described in this report, we give a few recommendations. First, as explained in Figure 2.2.1.4, an ML algorithm could be implemented to classify incident types, which reduces the work required from classifying manually to verifying the result from the algorithm. We already store the data required to be able to train this algorithm in the future.

Secondly, the automation program could produce a presentation alongside the report. The presentation that is used by the CSMs to present the outcome of the MHC to the customer still has to be manually filled in, which could be automated based on the KPIs that are already calculated by the automation program. We kept it out of scope for this project, however, due to time constraints.

Different sectors sometimes use varying templates, and they could benefit from a standardized solution of the report. We recommend doing this to ensure uniformity to the client and completeness of the MHC report, including the optional benchmark plots.

If the MHC can be executed more frequently because of our application, it could be beneficial to customers to have a MHC executed multiple times a year. Therefore, we recommend looking into a better data-driven service to the customer by developing a way to increase the frequency of the MHC.

8.3.2. Benchmarking

For benchmarking, the current solution allows only one filter option. If later it is desired to combine these filter options, this could be added. The current project does not benefit from this since we only have a small set of data, and adding extra filter options would only reduce the available data points for benchmarking. This could mean that the results are not accurate because of the sparsity in data.

¹Verklaring Omtrent Gedrag (Declaration about Behaviour): A document that declares that someone has not been convicted for any crime relevant for the person or institution requesting the VOG

These benchmark plots can be used to create non-arbitrary norms based on the deviation of the data per selected domain. These new norms could replace the already existing norms to change the feedback provided based on the MHC result. However, without enough benchmark data, the norms might not be very accurate. It should then be implemented when enough data is available to increase the accuracy of the norms.

8.3.3. Customer Data Permissions

We think it would be valuable to extend our tool in the future to support automated health checks for every TOPdesk customer, not just companies that have given permission. To this end, our application would need to be connected with internal TOPdesk systems to retrieve all customers and get authorization. Legal issues around this data access should also be resolved. TOPdesk should develop a way for our tool to easily ask the customer to approve the execution of a health check. Currently, users of our application need to create an application password to get access to the customer environment. This process takes a few minutes. However, an application password is by default valid for one year, so any future MHCs performed within a year will not require the creation of a new password.

More generally, we recommend that TOPdesk updates their terms of conditions to allow TOPdesk projects to have access to customer data for product improvement purposes. This would significantly reduce the effort required to build new data analysis projects.

8.3.4. Back-End Architecture

To speed up the product and reduce its complexity, we recommend TOPdesk to combine the Django and Firkin service to either a Django or a Firkin service. Currently the data has to flow through both services which makes understanding and debugging the application harder. Combining them into a Django service would be convenient, as the Django service is much larger than the Firkin service. And combining them into Firkin means that the entire back-end can be a service fully integrated with the TOPdesk environment instead of a stand-alone service, but would require more work.

We have spoken to TOPdesk about their improvements of the incidents API. We have implemented the recent improvements already, but more improvements might come that would allow the automation to go even faster than it currently does. If the API gets made faster we recommend looking at the improvements and implementing them into the product.

In the current architecture, all the data is sent one package from the Firkin service to the Django service. This creates quite a large response object, which might be an issue when using large databases. We suggest to stream the data from the Firkin service back to the Django service.

In general we recommend testing the application with large databases to see how it performs. We tested with large local mock databases that contained 100,000 and 500,000 incidents. According to our product owner, big companies have a similar amount of incidents. So the test results are representative of big customers in terms of size. However, we are not sure how the product will perform in production on large companies with many incidents. Large amounts of incidents could lead to problems such as timeouts or running out of space, which would prohibit the user from generating a MHC with our tool.

8.4. Ethical Implications

To assess the ethical implications of our project, we look at the *ACM Code of Ethics* as a guide. While we will not exhaustively treat each principle in this report, we want to highlight a few important principles for our project.

Principle 1.1: Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing.

We think our product is a contribution to society and well-being because we expect it increases labor productivity for TOPdesk employees. This is a desirable effect for society, because it frees CSMs from having to perform a repetitive task that can be automated instead. This leaves them with more time to work on other things.

Principle 1.2: Avoid Harm

This product can only be used for a very specific purpose by a very specific set of people, namely TOPdesk employees. No other data than what the CSMs already had access to is processed using our tool. The main

concern with data sharing is privacy infringement, which we consider in the next subsection. Therefore, we consider our product to be unlikely to cause harm.

Principle 1.6 Respect Privacy

As outlined in subsection 2.1.3.3, we took care to ensure that our tool handles customer data responsibly and according to relevant legislation. We only use anonymized data after asking explicit permission to do so. Therefore we are confident that our tool respects privacy.

Bibliography

- [1] Measuring up a report on education standards and assessments for oregon, 2000. URL <http://www.achieve.org/files/Oregon-Benchmarking3-2000.pdf>.
- [2] W. Richards Adrion, Martha A. Branstad, and John C. Cherniavsky. Validation, verification, and testing of computer software. *ACM Computing Surveys (CSUR)*, 14(2):159–192, 1982.
- [3] J G Arévalo, L Viecco, and L Arévalo. Methodology to define an integration process between frameworks SCRUM, django REST framework y vue.js, implemented for software development, from quality management approach and agility. *IOP Conference Series: Materials Science and Engineering*, 844(2020): 012022, 2020.
- [4] Barry W Boehm. *Software engineering economics*. Prentice-Hall, Englewood Cliffs, N.J., 1985.
- [5] Lakshmi Prasanna Chitra and Ravikanth Satapathy. Performance comparison and evaluation of node.js and traditional web server (iis). In *International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies*, ICAMMAET, pages 1–4, Chennai, India, 2017. IEEE.
- [6] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On non-functional requirements in software engineering. In Alexander T. Borgida, Vinay K. Chaudhri, Paolo Giorgini, and Eric S. Yu, editors, *Conceptual modeling: Foundations and applications*, pages 363–379. Springer, Berlin, Heidelberg, 2009.
- [7] L. Demir, A. Kumar, M. Cunche, and C. Lauradoux. The pitfalls of hashing for privacy. *IEEE Communications Surveys Tutorials*, 20(1):551–565, 2018.
- [8] G. A. Di Lucca, A. R. Fasolino, F. Faralli, and U. De Carlini. Testing web applications. In *International Conference on Software Maintenance, 2002. Proceedings.*, pages 310–319, 2002.
- [9] Horizon Framework EU. GDPR compliance checklist. <https://gdpr.eu/checklist/>, 2020. Accessed on 19-11-2020.
- [10] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Aspects of Computing*, 24(2):163–186, 2012.
- [11] Devndra Ghimire. Comparative study on python web frameworks: Flask and django. <https://www.theseus.fi/handle/10024/339796>, 2020. Accessed on 25-11-2020.
- [12] James Grenning. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, 3:22–23, 2002.
- [13] Jerry L Hintze and Ray D Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 1998.
- [14] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Herve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. <https://arxiv.org/abs/1612.03651>, 2016. Accessed on 24-11-2020.
- [15] Billy Korando. Five java frameworks for improving your automated testing. <https://developer.ibm.com/devpractices/continuous-delivery/blogs/five-frameworks-for-improving-your-java-automated-tests/>, 2020. Accessed on 23-11-2020.
- [16] P. Louridas. Junit: unit testing and coiling in tandem. *IEEE Software*, 22(4):12–15, 2005.
- [17] Desire Luc Massart, A J Smeyers-verbeke, et al. Practical data handling visual presentation of data by means of box plots. *LC GC Europe*, 18(4):215–218, 2005.
- [18] Wes McKinney. *Python for Data Analytics*. O’Reilly, Sebastopol, CA, 2013.

- [19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. <https://arxiv.org/abs/1301.3781>, 2013. Accessed on 24-11-2020.
- [20] Jaideep G Motwani, Victor E Sower, Ashok Kumar, Jiju Antony, and Tej S Dhakar. Integrating quality function deployment and benchmarking to achieve greater profitability. *Benchmarking: An International Journal*, 13(3):290–310, 2006.
- [21] Ashwin Pajankar. pytest. In Celestin Suresh John, editor, *Python Unit Test Automation*, pages 87–100. Apress, Berkeley, CA, 2017.
- [22] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing, EMNLP*, pages 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics.
- [23] M. Pezzè, K. Rubinov, and J. Wuttke. Generating effective integration test cases from unit ones. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 11–20, 2013.
- [24] dr. Haiyun Xu Reinier Vis MSc, Dennis Bijlsma MSc. Evaluation criteria trusted product maintainability: Guidance for producers. URL <https://www.softwareimprovementgroup.com/wp-content/uploads/2019/11/20190919-SIG-TUViT-Evaluation-Criteria-Trusted-Product-Maintainability-Guidance-for-producers.pdf>.
- [25] Guido van Rossum. Pep 8 – style guide for python code. URL <https://www.python.org/dev/peps/pep-0008/>.
- [26] Frane Šarić, Goran Glavaš, Mladen Karan, Jan Šnajder, and Bojana Dalbelo Bašić. TakeLab: Systems for measuring semantic text similarity. In *The First Joint Conference on Lexical and Computational Semantics - Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation, SEM*, pages 441–448, Montréal, Canada, 2012. Association for Computational Linguistics.
- [27] StackShare. Bootstrap vs bootstrap vue: What are the differences? <https://stackshare.io/stackups/bootstrap-vs-bootstrap-vue>, 2020. Accessed on 24-11-2020.
- [28] R. N. Taylor and A. van der Hoek. Software design and architecture the once and future focus of software engineering. In *Future of Software Engineering (FOSE '07)*, pages 226–243, 2007.
- [29] TOPdesk. Saas information: Topdesk software and consultancy. URL <https://page.topdesk.com/saas-information>.
- [30] TOPdesk. Handleiding mini health check. Private TOPdesk communication, 2020. Accessed on 11-11-2020.
- [31] TOPdesk. Guides to service excellence. Private TOPdesk communication, 2020. Accessed on 11-11-2020.
- [32] TOPdesk. Change team scalability and acceleration – R2D2. Private TOPdesk communication, 2020. Accessed on 11-11-2020.
- [33] Diane B. Walz, Joyce J. Elam, and Bill Curtis. Inside a software design team: Knowledge acquisition, sharing, and integration. *Commun. ACM*, 36(10):63–77, 1993.
- [34] Hadley Wickham and Lisa Stryjewski. 40 years of boxplots. *Am. Statistician*, 2011.
- [35] Eric Wohlgethan. Supporting web development decisions by comparing three major javascript frameworks: Angular, react and vue.js. <https://reposit.haw-hamburg.de/handle/20.500.12738/8417>, 2018. Accessed on 23-11-2020.
- [36] Jim York. Which days of the week to start and end your sprint, Nov 2017. URL <https://www.foxhedge ltd.com/blog/2017/11/1/which-days-of-the-week-to-start-and-end-your-sprint#:~:text=Many%20teams%20new%20to%20Scrum,may%20be%20a%20bad%20idea.>

-
- [37] Yijun Yu, Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, and Julio C. S. P. Leite. From goals to high-variability software design. In Aijun An, Stan Matwin, Zbigniew W. Raś, and Dominik Ślęzak, editors, *Foundations of Intelligent Systems*, pages 1–16, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

Appendices



Info Sheet

General information

Title: Customer maturity analysis improvement for TOPdesk

Client: TOPdesk

Final presentation: January 28, 2021, 13:00

Description

TOPdesk is a company that provides service management software. A Mini Health Check (MHC) report which is a process performed by consultants at TOPdesk that measures their efficiency of customer service. The core challenge of the project was creating an automation tool that connects to the TOPdesk environment. We researched the proper tools to use and the architecture of the TOPdesk environment. By developing the product in an agile way, issues could be prioritized on a weekly basis. The final product has two main features. First, it allows the user to automatically generate and download a MHC report. Secondly, benchmarking allows customers to compare their customer service performance to other customers in their sector, country, or to a similar company size. We have made recommendations to improve the program's speed and maintainability. The product we made is a prototype, but TOPdesk is investigating if they can use our product.

Project members

Krzysztof Baran: Interested in software engineering, full-stack development and artificial intelligence. Krzysztof contributed to environment setup, backend testing, pipeline, database (MySQL) modelling, Django application and code linting.

Cees Jol: Interests go to web development, artificial intelligence, marketing, psychology, and rowing. Cees contributed to the front-end design and logistics, KPI calculation, the Firkin application, and code quality.

Rover van der Noort: Interest lie in back-end development, system architecture, Scrum and testing. Rover contributed to the environment setup, containerization, quality, pipeline, Django APIs and customer contact.

Wander Siemers: Areas of interest are software engineering, machine learning, and mobile. He likes weight training and whisky, not necessarily simultaneously. Wander contributed to the creation of the Mini Health Check Report, Firkin application, API performance and stakeholder management.

All members contributed to the research report, the final report, the final presentation, code review, documentation and testing

Client & Coach

Client: Jan-Fabian Humann and Corina Stratan, TOPdesk

Coach: Frank Mulder, Software Technology, Computer Science and Engineering Teaching Team

The final report for this project can be found at: <http://repository.tudelft.nl>

B

Supporting Material

B.1. GDPR Table

Group	Regulation	Our application
Lawful basis and Transparency	Conduct an information audit to determine what information you process and who has access to it.	We keep all data within TOPdesk environment and do not use external storage systems.
	Have a legal justification for your data processing activities.	Consultants get permission to use data from companies for MHCs and our team cooperates this with the legal team of TOPdesk.
	Provide clear information about your data processing and legal justification in your privacy policy.	For the MHC, a standard description is explained and benchmark will require an explicit agreement with the customer.
Data Security	Take data protection into account at all times, from the moment you begin developing a product to each time you process data.	The data will be stored only in the internal TOPdesk systems. We will minimize file sharing as much as possible.
	Encrypt, pseudonymize, or anonymize personal data wherever possible.	Whenever a benchmark will be performed, statistics will have no associative attributes.
	Create an internal security policy for your team members, and build awareness about data protection.	The team has signed a Non-Disclosure Agreement (NDA) and are aware of data protection and consequences of failure.
	Know when to conduct a data protection impact assessment, and have a process in place to carry it out.	We will consult the security team/guild such that we can have it verified by experts.
	Have a process in place to notify the authorities and your data subjects in the event of a data breach.	We will contact the Security team/guild, the legal team and our supervisors first who can pass on the authorities.
Accountability and governance	Designate someone responsible for ensuring GDPR compliance across your organization.	The legal team at TOPdesk does that.
	Sign a data processing agreement between your organization and any third parties that process personal data on your behalf.	The consultants gets that agreement with customers.
	If your organization is outside the EU, appoint a representative within one of the EU member states.	Within the legal department, there are employees at TOPdesk who are already responsible for the outside regions.
	Appoint a Data Protection Officer (if necessary)	Not necessary in this project.

Continued on next page

Table B.1 – continued from previous page

Group	Regulation	Our application
Privacy rights	It's easy for your customers to request and receive all the information you have about them.	The prototype displays reports of their data and they also see it from their SaaS product.
	It's easy for your customers to correct or update inaccurate or incomplete information.	The customers have power to edit their information within their SaaS product.
	It's easy for your customers to request to have their personal data deleted.	TOPdesk SaaS product handles that for us.
	It's easy for your customers to ask you to stop processing their data.	The customer just has to ask the consultants.
	It's easy for your customers to receive a copy of their personal data in a format that can be easily transferred to another company.	Since our product is about data presentation, it will be possible.
	It's easy for your customers to object to you processing their data.	The customer just has to ask the consultants.
	If you make decisions about people based on automated processes, you have a procedure to protect their rights.	Does not apply to us.

Table B.1: GDPR compliance explanation table

B.2. Example of KPI assement




Selfservice KPI's	
<p>Omschrijving: <i>In hoeverre wordt voor het proces van meldingenbeheer gebruik gemaakt door klanten/eindgebruikers van het selfservice portaal om hun meldingen in te dienen. Hierbij hanteert TOPdesk Consultancy een gezondheidsnorm van minimaal 50% bij meldingen.</i></p>	
<p>Bevindingen benutting van TOPdesk:</p> <p>Het selfservice gebruik percentage is: > 60% Totaal aantal meldingen: 10.000</p> <p>Vergeleken met andere onderwijsinstellingen is dit een hoog percentage van selfservice gebruik dat naast de overige communicatiekanalen wordt gebruikt, en scoort daarmee boven de gezondheidsnorm.</p>	
<p>Bevindingen benutting van TOPdesk:</p> <p>Het selfservice gebruik percentage is: 40% - 60% Totaal aantal meldingen: 10.000</p> <p>Vergeleken met andere onderwijsinstellingen is dit een gemiddeld percentage van selfservice gebruik dat naast de overige communicatiekanalen wordt gebruikt. Er is zeker ruimte voor verbetering, waarbij een verdere kwalitatieve analyse benodigd is om concrete groei- en winstpunten te benoemen, waarmee het percentage hoger uitkomt in de toekomst.</p>	
<p>Bevindingen benutting van TOPdesk:</p> <p>Het selfservice gebruik percentage is: < 40% Totaal aantal meldingen: 10.000</p> <p>Vergeleken met andere onderwijsinstellingen is dit een laag percentage van selfservice gebruik. Dat betekent dat de klassieke communicatiekanalen veel meer toegepast worden in de praktijk, en dit percentage is dan ook onder de gezondheidsnorm.</p> <p>Er is veel ruimte voor verbetering, waarbij een verdere kwalitatieve analyse benodigd is om concrete groei- en winstpunten te benoemen, waarmee het percentage hoger uitkomt in de toekomst.</p>	

Figure B.1: Self-service portal usage assessment guide

B.3. MHC Manual Steps to export KPIs

Figure B.2: Overview of instruction set to export KPIs for the MHC.

B.4. Benchmarking Plots

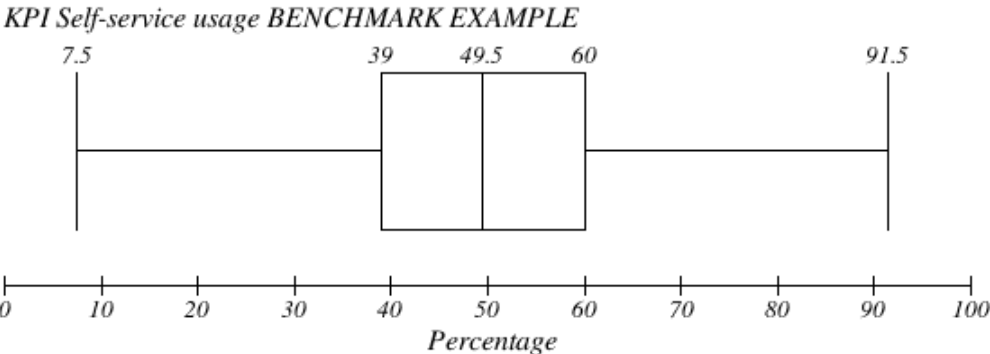


Figure B.3: Example of Self-Service usage KPI with mocked data with a Box Plot

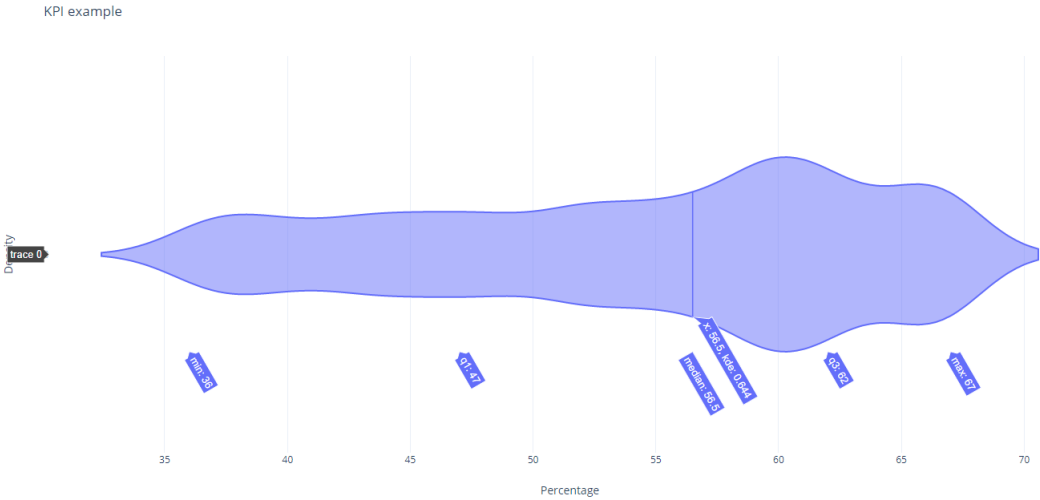


Figure B.4: Example of Self-Service usage KPI with mocked data with a Violin Plot

B.5. SonarQube results

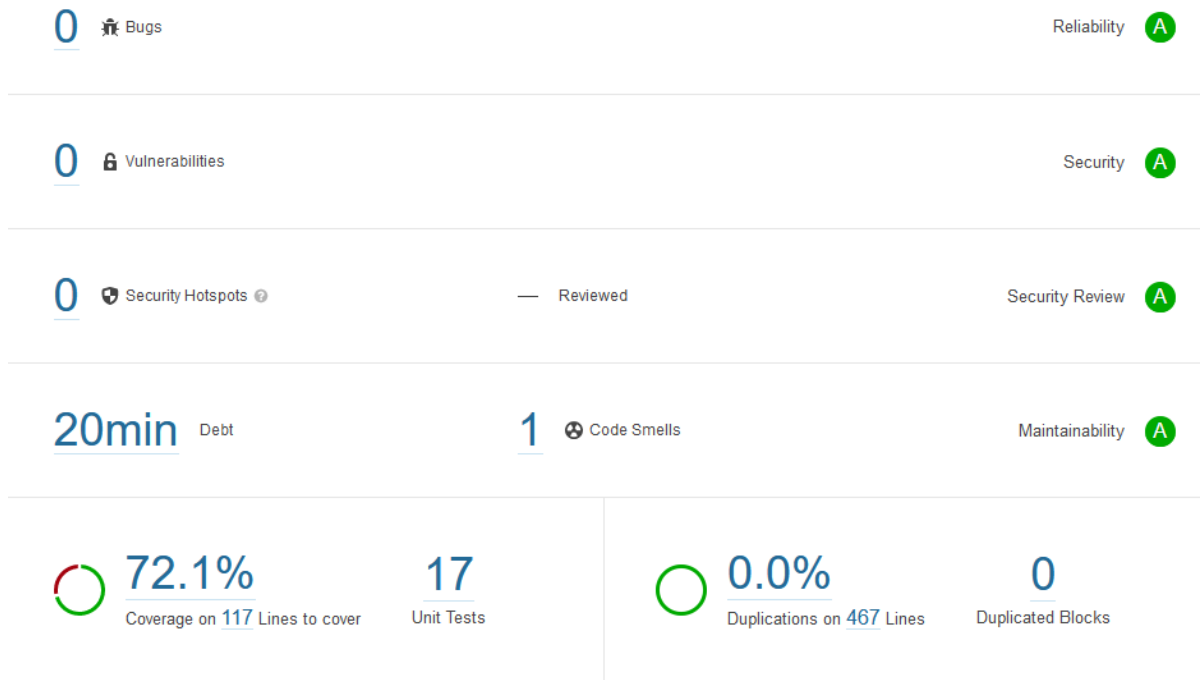


Figure B.5: Vue SonarQube quality results.

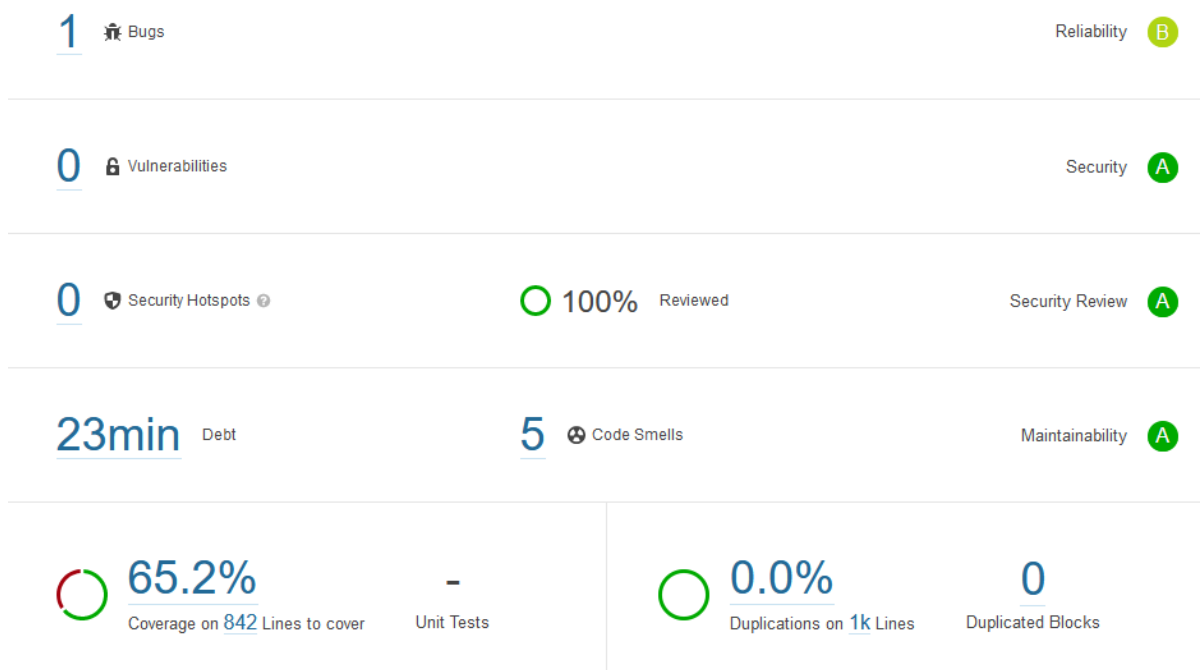


Figure B.6: Django SonarQube quality results.

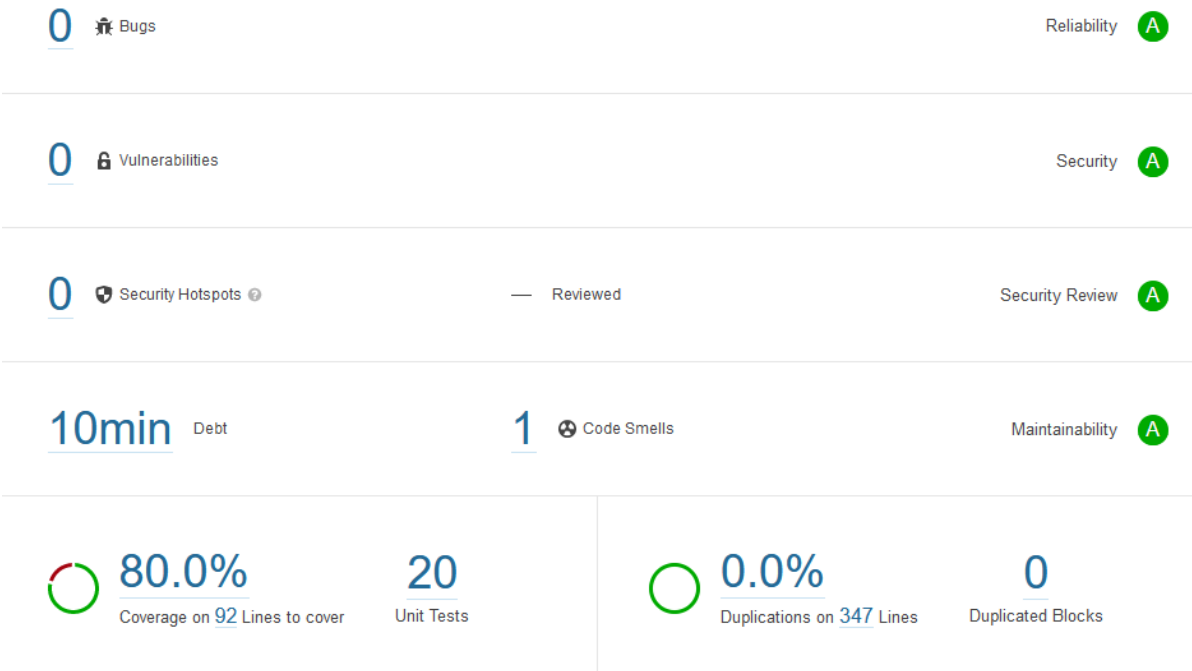


Figure B.7: Firkin SonarQube quality results.

C

Original Project Description

Analyzing how efficiently customers manage their services with TOPdesk

Background

TOPdesk is a company that provides enterprise service management software. Our goal is to help organizations provide a great service experience to their customers and to their own employees. We focus among others on IT, Facilities and HR services.

Besides creating software, we also aim to advise organizations about how they can make their services more efficient and provide a better customer experience.

To assess the maturity level of an organization, the TOPdesk consultants conduct interviews with stakeholders from the organization and also analyze statistical data from TOPdesk. Examples of data that we analyze are: the average time of resolving an incident, the number of incidents that are registered via our self-service portal (which is more efficient than via a phone call), the number of incidents that are just questions (which could have been resolved more efficiently by having published the information in advance).

The project assignment

When assessing the maturity model of a customer, for the data analysis part our consultants currently run a number of reports manually on the TOPdesk database. With this project, we would like to automate this process and make it easy to get the relevant statistical data without a lot of manual effort. Optionally, we would also like to investigate if it is possible to compare the results of a customer with the average of other customers from the same industry or domain.

For this project, the students will need to design a solution that fits well technologically with the rest of our product. Another challenge is that some TOPdesk customers have very large amounts of data, and this needs to be taken into account when creating reports.

D

Manual

Redacted by TOPdesk because of non disclosure agreement.