MASTER'S THESIS 2019

What is required by software platforms in order to give a good developer experience?

Christoffer MacFie

ISSN 1650-2884
LU-CS-EX 2019-08

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2019-08

What is required by software platforms in order to give a good developer experience?

Christoffer MacFie

What is required by software platforms in order to give a good developer experience?

(A Research Into Qlik Core's Developer Experience)

Christoffer MacFie
dat13cma@student.lu.se

June 19, 2019

Master's thesis work carried out at Qlik AB.

Supervisors: Martin Höst, martin.host@cs.lth.se
Andrée Hansson, andree.hansson@qlik.com

Examiner: Ulf Asklund, ulf.asklund@cs.lth.se

Abstract

This research report was carried out in collaboration with Qlik AB and aimed to identify what aspects are needed from a software platform in order for developers to want to use them. It also aimed to identify how well the software platform Qlik Core had implemented these aspects.

The research used online questionnaires, interviews and data analysis using Qlik Sense. The research compared how different groupings of people had varying needs. It also explored and got a deeper understanding of why some of these aspects were needed.

The research found that the most important aspects of a software platform is API code examples, thorough documentation explanations and that you can have working code quickly. It also found that Qlik Core does not have these three aspects and it needs to mainly work in its documentation and examples.

Keywords: MSc, Developer Experience, DX, Software Platforms, Qlik Core

Acknowledgements

There are several people I want to thank who made this paper possible.

I want to thank the interviewees who took time out of their day to be interviewed for this research paper. I also want to thank all the anonymous people who took the surveys conducted in this research paper.

I want to thank Andrée Hansson and Johan Bjering, who both served as mentors for me at Qlik. I also want to thank Martin Höst, who was my mentor at Lund University. Finally, I also want to thank Ulf Asklund, who is the exterminator of this paper.

Contents

Preface **vii**

1 Background and Purpose **1**
1.1 Goal Of This Research Paper 1
1.2 What is User Experience? 1
1.3 What is Developer Experience? 3
1.4 How Can One Define ”Good” DX? 3
1.5 Who are Qlik and what is Qlik Core? 4
1.6 Popularity of Programming Languages 5
1.7 Kinds of APIs 6
1.8 API User Personas 10
1.9 Standardisation 11

2 Methodology and Preparations **15**
2.1 Deciding consideration aspects 15
2.2 Linking considerations to ISO-9216-1 18
2.3 Surveys 19
2.4 Interviews 28
2.5 Making Recommendations for Software Platforms 34
2.6 Evaluating Qlik Core 34

3 Results and Discussion **35**
3.1 Initial Survey 35
3.2 Survey 2 Results 39
3.3 Interview Results 52
3.4 Recommendations For Software Platforms 62
3.5 Evaluation of Qlik Core 82
3.6 Qlik Core Evaluation Summary 90
3.7 Threats to Validity 92

4 Conclusion **93**
4.1 Further Research 94

Bibliography **97**

A Pilot Survey **101**

B Main Survey **109**

C Material for Interview **125**

Preface

To start this research paper off, I will tell you a story.

Some years ago I believed I was too dumb to understand thermodynamics. I was about 18 years old and was sitting in my class room trying to solve a problem surrounding that very subject. I just could not understand it. My classmates were doing fine, talking and joking with each other while they were solving one problem after another. Meanwhile I just sat there, scratching my head, struggling with the first task.

Then one day, I decided to go and sit by myself instead of in the classroom. I had brought a book, not the one given by the school, but another I had borrowed from the library. I sat in total silence and carefully read the chapter about thermodynamics. I stopped, reflected on the text, and then went back into it. And suddenly, I just understood what I could not understand before.

I think about this from time to time whenever I feel too dumb to learn something. It is not that I was too dumb, it is that the needs I had to do it were not fulfilled. The lesson of the story is that people have different needs in order to have a good experience.

This story can be applied to a lot of things in life. Some time ago when trying to use a software platform when programming, I encountered the feeling off not being smart enough again. But then I remembered this story, and realized that it is not that I am too dumb, it is that my needs are not fulfilled. I needed another explanation, to not sit next to may classmates and to have some silence. And this is what this research paper will be about. What needs do different developers have from software platforms in order to have a good experience using them?

Chapter 1
Background and Purpose

1.1 Goal Of This Research Paper

This research paper is done in collaboration with the company Qlik. The goal of it is to define the needs developers have from software platforms in order for them to have a good experience. It also evaluates how well Qlik's software platform Qlik Core satisfy these needs. The questions this research paper intends to answer are:

1. What aspects are needed by a software platform in order for people to find them favourable, and how important are these aspects?

2. Do different groups of software platform users have different needs?

3. Why are some of these aspects needed or not needed by a software platform in order for people to find them favourable?

4. How favourable is Qlik's software platform Qlik Core to use users, and what can be improved?

1.2 What is User Experience?

To start off, one needs a method of how to even measure experience. One field we can look into, that has been researched a lot, is User Experience (UX). UX is the collective term for many disciplines merged into one that evaluates the overall experience delivered to a user of a system, product or service. The coining of the term is often attributed to Norman et al. (1995), who has a background in the fields of cognitive science and usability engineering. There is not one definitive definition of what UX is. They one who created the term, Norman, defines UX as:

> ”All aspects of the end-user’s interaction with the company, its services, and its products. The first requirement for an exemplary user experience is to meet the exact needs of the customer, without fuss or bother. Next comes simplicity and elegance that produce products that are a joy to own, a joy to use. True user experience goes far beyond giving customers what they say they want, or providing checklist features. In order to achieve high-quality user experience in a company’s offerings there must be a seamless merging of the services of multiple disciplines, including engineering, marketing, graphical and industrial design, and interface design.”

User Experience has since its emergence in the 1990s gotten its own ISO-standard: ISO-9241-210. It is defined by ISO 9241-210:2010 (2010), part of ”Ergonomics of human system interactions”, as

> ”A person’s perceptions and responses that result from the use or anticipated use of a product, system or service”

The World Wide Web Consortium (W3C) is the main international standards organization for the World Wide Web. W3C (2005) defines it as following:

> ”A set of material rendered by a user agent which may be perceived by a user and with which interaction may be possible.”

One common denominator is that UX is related to how something is *perceived*. It can therefor be considered a somewhat subjective quality of a product, system or service. Because of this it can be hard to measure with exact numbers. ISO 9241-210:2010 (2010) has a four-part process for how to evaluate a system or service. The process can be seen in Figure 1.1. It is an iterative process, and depending on if the design solution meets the user requirements or not, an earlier phase in the process needs to be revisited.

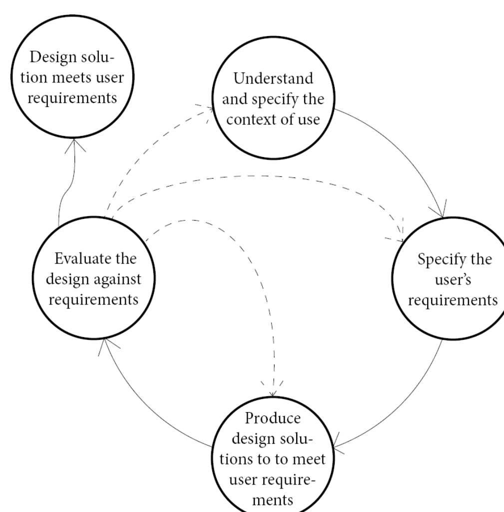


Figure 1.1: The UX evaluation process as defined by ISO 9241-210. Solid arrows shows the next step in the process, and dashed lines shows possible returns to earlier steps if needed.

1.3 What is Developer Experience?

User Experience revolves around the broad and vague group of ”users”. What we are interested however is the experience for one certain group: developers. This is what the term ”Developer Experience” revolves around. Developer Experience, or DX, is similar to the more well known User Experience (UX), but with the user being a software developer using a service or software aimed at developers, such as an API or library (rather than some executable program). Developer Experience has yet to be defined by any standards organization. There is also limited peer-reviewed academic research around the subject. That does not however mean that there does not exist articles about it. DX is defined by Jarman (2017) as

> ”The experience developers have when they use your product, be it client libraries, SDKs, frameworks, open source code, tools, API, technology or service.”

Dhide (2017) has describes in an article how he and a team of developer tries to define DX, and came to the following conclusion:

> ”Developer Experience (DX) is inspired by the User Experience practice and sees developers as a special case of users. Developer Experience Design is the practice of understanding how developers get their work done, and optimizing that experience.”

This report uses the following definition, inspired by the UX-definitions described above:

> ”Developer Experience is the perceived feelings and thoughts of a developer, generated by an interaction with a software or service that is meant to be used by a software developer”

1.4 How Can One Define ”Good” DX?

So how do we figure out what exactly is ”good” DX? There are many potential factors for defining what constitutes ”good” DX. The website Every Developer (2019) has developed a *DX Index* from 1-10, where they consider four factors:

1. Are the libraries available in popular languages?
2. How prominent, in-depth are the starting guides?
3. Are the solutions self-serving, without need of demos or ’call us’?
4. Is the pricing clearly stated?

Jarman (2017) has other factors that he uses to evaluate if something gives a good DX. He for example puts emphasis on communication between the product provider and the developer. The dialog between the product provider and the community needs to be authentic, open and honest in order the give a good developer experience, according to Jarman.

Dhide (2017) did a workshop, which generated a list of things that they think should be considered when designing software or services with good DX. Their key aspects are simplicity, usability, innovation and ”delightfulness”. They put emphasis on ”Less is more”, stating that ”Any element that is not helping the user achieve their goal is working against them”. Some other things they say are important are ”Always keep the target users in mind as the product is designed”, ”Know what type of problem you’re solving” and ”Use concepts familiar to the user rather than system-oriented terms”.

It has been researched by Graziotina et al. (2018) what makes a developer happy and unhappy, and they found several indicators, both internal and external factors. The internal factors were things like stress, fatigue, low motivation, etc. They also pointed to external factors. These were things like low productivity, delays, broken flow, and more. According to their findings, low productivity is the most common cause of unhappiness among developers.

A similar study researched by Wróbel (2013) showed that positive emotional states give an increase in productivity, and negative emotional states gives a decrease in productivity. The research also calculated the risk/opportunity for each emotional state, in relation to productivity. The calculation showed that the most impactful emotional state to productivity is ”frustration”, being the most the most common negative emotional state for programmers.

The issues pointed out by Graziotina et al. (2018) and Wróbel (2013) is something that may be improved by giving developers a better DX. This research shows that there is a clear need for good DX.

1.5 Who are Qlik and what is Qlik Core?

This research report is done in collaboration with the company Qlik. Qlik is a software company in Lund, founded in 1993 (Qlik Media Representation, 2010). They offer several products, with the two most popular ones being QlikView and Qlik Sense. These two programs are used for discovery of insights about data that the user has. The programs take large quantities of data and link them together into a data-model, which then can be easily interacted with to help the user find insights, patterns and anomalies. This new information can then help companies make decisions for the next step for the company.

Let me take an example. A telemarketing company is trying to figure out what employee deserves a Christmas bonus. The company has several thousands of excel sheets of all sold products during the year and who sold what. Going through all these sheets would take many days, and would be very difficult. However, when they load it into Qlik Sense, they can within minutes see the sales per person and date. They find that two employee has the same sale amount. However, one of the employees had zero days throughout the year where they had no sales, whereas the other had 95 days. The company decides to give the Christmas bonus to the employee who had the most consistency in sales.

This is all possible because of the Qlik Associative Engine (QAE). The engine links all the data together and makes a data model, which can then be easily displayed in different ways. The target audience for QlikView and Qlik Sense are businesses.

Qlik has now launched a new product called Qlik Core (QC). The target for this product is developers, rather than ”ordinary people”. The main selling point of this new product is that it gives the powerful QAE to developers to use, and make their own products on top of it. Qlik Core is, as described on the official website, ”an analytics development platform built around Qlik Associative Engine and Qlik-authored open source libraries”(Qlik, 2019). The Qlik Core platform consists of several components, with its central part being the QAE. To protect intellectual property (IP), the QAE is close-source.

The Qlik Core platform also provides libraries and services to make it possible to use the QAE, which almost all of them are open-source. The libraries and services that are included with Qlik Core are ”Mira”, ”Halyard”, ”Enigma” and a licensing service.

Mira is an open-source JavaScript service which is used to generate insights about the data. Halyard is a JavaScript library which makes loading data into the QAE easier. Enigma is an open-source service which makes it possible to communicate with the QAE. This library is offered in both JavaScript and Go. Lastly, the licensing service is a close-source service that authenticates that the user has paid for the use of the platform.

Furthermore, Qlik also offers a unified testing framework called ”after-work.js” and visualisation library called ”picasso.js”. These are also open-source, but are still classified by Qlik as ”experimental” as they are not yet stable, at time of writing.

It should be mentioned that QC takes use of a third-party software called Docker. As described by OpenSource (2019), Docker works a bit like virtual machine, but instead of having a whole operating system run, it simulates just a folder-system. Docker works with something they call ”Containers”, which makes it possible to package an application with all its dependencies, libraries, etc, into one package. By using Docker, Qlik has made QC portable and runnable in most operating system.

1.6 Popularity of Programming Languages

It should said a few things about the state of programming languages in 2019, since it is an aspect that affects developers as we will see later in this report. There are new languages emerging all the time. Some disappear just as quickly as they appeared, while others are here to stay. GitHub (2019) releases a yearly review of people’s usage of the web-based hosting service. In their review they talk about, amongst other things, what languages are the most popular. In Figure 1.2 we can see the most popular languages from the end of 2014, to the end of 2018. As mentioned, some languages come and go. One example is Ruby, which has dropped from 5th place, to 10th in just three years. The undisputed winner however is JavaScript, which is also what Qlik Core mainly uses. It should also be mentioned that Go, which QC also offers one of their libraries in, is on the rise and saw

an increase by 150% during 2018(GitHub, 2019).

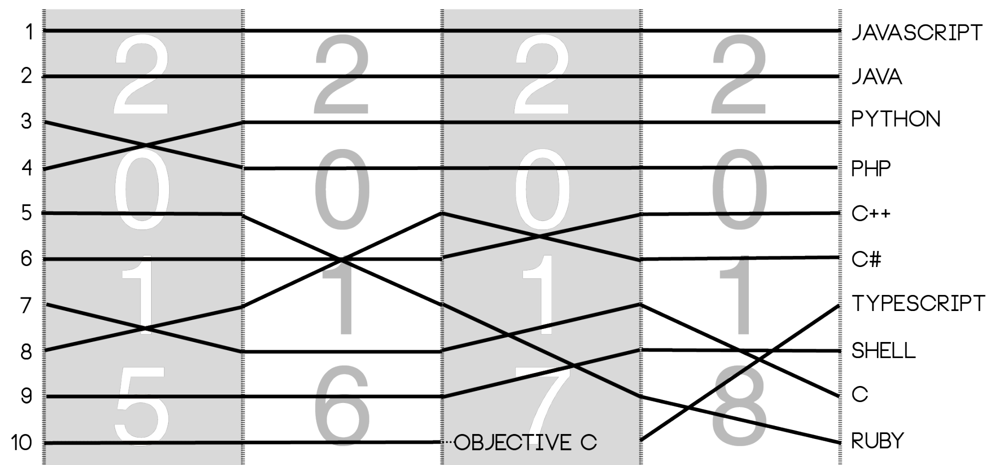


Figure 1.2: The most popular programming languages in GitHub repositories over time.

Source: Data from GitHub (2019)

1.7 Kinds of APIs

Application Program Interfaces (APIs) are, simply put, a software that lets one application interact with another application's inner data and services. Applications are in need of an interface to interact with its inner parts to create, read, update and read (CRUD) as well as execute commands. APIs provide this link between the two pieces of software that lets them communicate. Because of APIs broad nature, there are many types of APIs(Sturgeon, 2016).

1.7.1 Internal, Public and Partner APIs

APIs have different level of openness, depending on who is going to access them, as described by Levin (2017). They are usually divided into three groups: Internal APIs, Public APIs and Partner APIs. These three groups are explained below.

Internal APIs

Internal APIs are APIs that are meant to be used in internal productions and within an organisation or company. They are often developed to be used between different teams in the company to be able to connect software components in the application, without having to actually know the code of the component. The benefit of this is that the team can open up certain needed functionality of the software to other teams while still being in control of their own code. This kind of APIs are protected and require internal API keys to access to ensure that people outside of the company or other non-company authorised persons are not able to access them.

Public APIs

Public APIs is another kind of API. This is a way for the company to open up functionality of the software's inner workings to the world so that anyone may build new applications that are built upon the original software. This kind of interface often only has a small percentage of the functionality that the internal API has, since the circuitry of the software must be protected for security reasons as well as from business intelligence theft. If the internal API was open to the public anyone could build their own copy of the program or find vulnerabilities. These kind of APIs either do not require any API key to access, or have an API key that is open for anyone to acquire (either through payment or for free).

Public APIs also helps with stability for software. ''Martin's Metrics for Instability" measures how ''stable" a piece of software is, based on how many dependencies it has on other packages (Höst, 2019, pp. 37-38). According to this theory, the more dependent a software is on other packages, the more unstable it is. This can be related to public APIs. If a software is built on a platform with just a few API methods, it is much more stable than if it were to call many different ones since there is a lot more thinks that can break.

Partner APIs

Partner APIs are a third interface that can be shared business-to-business (B2B), with strategic partners to the company. Partner APIs often put some restraints on what can be exposed so that the inner workings of the software is still protected, but is able to be more open than a public API. These kinds of APIs require an API key that is often contracted with terms and condition to protect the company's business intelligence.Levin (2017)

1.7.2 Web APIs

When using services over the internet, there are many different protocols that can be used to communicate. Just like with many other things in computer science, there are many valid approaches whom all have their pros and cons. The world wide web (WWW) is largely built upon the application protocol hypertext-transfer-protocol (HTTP) which, amongst other things, provides CRUD methods to be applied on resources. Resources in this context refers to any *thing*: file, object, document, text, etc, that is provided by a web service. There are however many ways of utilizing this protocol to let a client access and manipulate server-side data, as well as execute commands. Below there are two methods described.

REST

Representational State Transfer (REST) is a software architectural style that is used in web services which acts as a communication bridge between computer systems and the internet. It lets the system interact and manipulate the service it is interacting with. REST solves many issues that had been present in previous implementations of communication between computer systems and the web (Code Academy, 2019; Sturgeon, 2016; Feng et al., 2009).

One of the key factors in REST is that it is stateless. Statelessness in this context means

that the two communicating parties do not need to know anything about each other or have seen previous messages to understand future ones. This feature is possible by limiting it to the use of resources instead of commands. REST-APIs can therefor not ask the server side to execute a specific custom command, but is limited to using CRUD methods. Feng et al. (2009) points out some key constraints that exist in REST. A few of these are that everything is a resource in REST, the identification of resources is done by using URI (Uniform Resource Identifier) and that it uses stateless interactions.

A REST request consists of an HTTP method, a header containing information about the request, the path to the resource and lastly an optional message body consisting of data.

Statelessness makes it possible to separate the client and the server. Code changes to the server will not require changes to the client's code, and vice versa, as long as the message format between the two are kept the same.

Since REST does not use sessions, but simply responds to any incoming requests, it is easy to scale up. It just requires more bandwidth and processing power to be able to handle more requests per second.

If you for example want to post a message as a user with the `userID 1`, it could look something like what we can see in Listing 1.1.

Listing 1.1: An example of how an API call can look using REST

```
POST /users/1/messages HTTP/1.1
Host: example.com
Content-Type: application/json
{"msg": "Test123"}
```

Here, the resource of `/users/1/messages` is fetched and then the new message is created and put into the database. The server does not work in sessions and cannot tell clients that a new message is available. The clients has to periodically ask the server if there are any new messages to retrieve.

REST may be appropriate to use when you mostly want to do CRUD-commands or manipulate data.

RPC

Remote Procedure Call (RPC) is a also described by Code Academy (2019); Sturgeon (2016); Feng et al. (2009) as a protocol used to execute commands on remote systems. RPC is, like REST, also built on HTTP, but uses mostly just the `GET` and `POST` commands. RPC is a request-response protocol and is, unlike REST, stateful. Ergo, the protocol works with sessions between a client and a server, and previous messages may be needed in order to understand future ones.

An upside of RPC is that it lets a client request the server to execute a custom command.

Making an RPC-call is much like making a normal function call, in that you simply provide the name of the method and the parameters. A consequence of this is that code changes on server-side, such as method-name or parameter input, may require code changes on the client side as well.

Since RPC has two-way communication, the server can tell the client when something has changed, whereas in REST-based communication the client has to ping the server to check if there are any changes. An RPC based server needs to have a unique session for each client, which can cause problems with scalability.

If we go back to the example used in the previous section: posting a message may look something like what can be seen in Listing 1.2.

Listing 1.2: An example of how an API call can look using RPC

```
POST /SendMessage HTTP/1.1
Host: example.com
Content-Type: application/json
{"userId": 1, "msg": "Test123"}
```

Here, the server has a custom method called `SendMessage`. If the method call is not made asynchronous, the client is put in wait until the server responds with an acknowledgement or the call reaches a timeout. Since RPC uses sessions and custom commands, the method can be implemented as such that other sessions should be notified that a new message has been sent, and the server can send it out to appropriate clients.

RPC may be appropriate to use when you have functionality that can benefit from two-way communication or is mainly command-oriented.

1.7.3 What Type of API is Qlik Core?

Qlik Core consists of several components which utilizes several libraries that has different types of APIs. The QAE also has several different ways of communicating with its inner workings. These APIs also have different types of APIs, namely JSON-RPC, gRPC and REST. JSON-RPC is a variant of RPC that utilizes the JSON format and gRPC is Google's implementation of RPC (Qlik, 2019). The libraries is listed in List 1.1.

List 1.1: Libraries offered by Qlik with Qlik Core

Mira - Discovery service library included in QC, based on REST

halyard.js - Data loading service that works as a wrapper around the load scripts to make data loading easier for a user. This service does not use an API standard.

enigma.js - QAE communicating service included in QC, based on gRPC

Qlik Associative Engine - QAE has several different APIs offered for communication with it

  QIX API - An API based on JSON-RPC

Data Connector API - An API based on gRPC

Analyitical Connector API - An API based on gRPC

1.8 API User Personas

There are many types of people using platforms, whom all have different requirements and personalities. To get a better understanding of what the needs of a system will have one can take help of the concept of ”User Personas”, which has its origin in UX. It described by Bernhaupt et al., p. 191 as a fictional character which helps understands a user’s goals, environment, needs and problems to achieve a certain task. User personas helps a designer of a system to understand what needs the system’s user will have.

One way of dividing people is into the two groups ”Decision makers” and ”Non-Decision Makers”. These both need to be catered to in order to have a successful platform: if the decision makers are ignored the platform will not be implemented by companies in the first place. If the users are ignored, the platform will be quickly dropped since its usage is not good enough.

Nottingham (2012) lists several personas that will affected by how HTTP-based APIs are developed. A persona in this context is a group of people who share certain characteristics. He tries to cover all thinkable personas that would be affected by APIs. He looks both from the perspective of how to satisfy the consumer of APIs, and the creator of APIs. For this research paper, the view is that of what makes consumers of software platforms satisfied, and it is therefore the viewpoint of the consumer that is interesting to us.

Established Company Employees - These people are limited in their freedom of what they can adopt, prohibited by the company’s policies, ways of working and legal requirements. They are more likely to use more established programming languages, rather than the latest fads or up-and-coming languages. They are more likely to demand good documentation with examples in their languages, and less likely to spend time trying to understand new ways of working.

Startup Company Employees - This person is in many ways the opposite of the person described above. This person is more likely to use newer programming languages, closely following what’s ”hot and new”. This person also prefers quick solutions, going for existing implementations rather than developing their own. They’re pragmatic, looking for fast results and does not have time to spend too much time on ”fancy” solutions.

Mobile Developers - This person has concerns that developers for computer-based API consumers does not have to the same extent. Even though it was more relevant a few years ago, when mobile phones were considerably less powerful and PCs, the mobile developer has to take into account CPU and memory usage more than the developers for PCs. They also care about energy usage, since mobiles uses batteries. API creators can help this developer out by specifying how heavy API calls are to make.

Poorly Connected Users - These users has concerns that the others don't. With poor connection, they're taking into account how big the responses from API calls are, how many calls has to be made, etc. This developer needs specifications on these things when developing.

1.9 Standardisation

As mentioned in Section 1.5, there is no standard for DX given by The International Organisation for Standardisation (ISO). There are however other standards from ISO that are interesting to examine and compare with. One is *ISO 9126 Software engineering - Product Quality*. One part of this ISO-standard concerns how to measure the quality of software. It has six different characteristics: functionality, reliability, usability, efficiency, maintainability and portability. Each of these characteristics has sub-characteristics. The definition of each characteristic is listed, and their sub-characteristics, is in Table 1.1 and Table 1.2. This standardisation is presented as it will be used in Section 2.2 to relate the ISO-standard to the aspects that are to be considered in this research.

Table 1.1: Part 1: ISO-9216

	Functionality	
F1	Suitability	The capability of the software product to provide an appropriate set of functions to specified tasks and objectives
F2	Accurateness	The cap... / / ... to provide the right or agreed results or effects with the needed degree of precision
F3	Interoperability	The cap... / / ... to interact with one or more specified systems
F4	Security	The cap... / / ... to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them
	Reliability	
R1	Maturity	The cap... / / ... to avoid failure as a result of faults in the software
R2	Fault tolerance	The cap... / / ... to maintain a specified level of performance in cases of the software faults or of infringement of its specified interface
R3	Recoverability	The cap... / / ... to re-establish a specified level of performance and recover the data directly affected in the case of a failure
	Usability	
U1	Understandability	The cap... / / ... to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use
U2	Learnability	The cap... / / ... to enable the user to learn it application
U3	Operability	The cap... / / ... to operate and control it
U4	Attractiveness	The cap... / / ... to be attractive to the user [visually]
	Efficiency	
E1	Time Behaviour	The cap... / / ... to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions
E2	Resource Utilisation	The cap... / / ... to use appropriate amounts and types of resources when the software performs its function under stated conditions

Table 1.2: Part 2: ISO-9216

	Maintainability	
M1	Analysability	The cap... / / ... to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified
M2	Changeability	The cap... / / ... to enable a specified modification to be implemented
M3	Stability	The cap... / / ... to avoid unexpected effects from modifications of the software
M4	Testability	The cap... / / ... to enable modified software to be validated
	Portability	
P1	Adaptability	The cap... / / ... to be adapted for different specified enviroments without applying actions or means other than tose provided for this purpose for the software considered
P2	Installability	The cap... / / ... to be installed in a specified environment
P3	Co-existence	The cap... / / ... to co-exist with other independent software in a common enviroment sharing common resources
P4	Replaceability	The cap... / / ... to be used in a place of another specified software product for the same purpose in the same environment
	All characteristics	
AC	Compliance	The cap... / / ... to adhere to standards and conventions relating to the characteristic

The most relevant part of this ISO-standard for this report is the part about usability, with understandability and learnability being the most central for DX. However, all the characteristics are relevant for providing good DX.

This ends the chapter about Background and Purpose. As has been shown, there is a great need for good DX but it has yet to be standardized by any organisation or researched thoroughly in any big publication. This emphasis the importance of this report.

Chapter 2
Methodology and Preparations

There could have been many approaches in this project to try to find out what people require from software platforms in order to get a good developer experience. The process chosen in this research paper was to make a survey followed by interviews to triangulate the results. The importance of data triangulation is described by Runeson and Höst (2008). One of the triangulation methods described in that publication is methodological triangulation, which triangulates the data by using different types of data collection, e.g. qualitative and quantitative data. This method is the one used in this report.

First of all it was decided what aspects were going to considered. This was followed by an initial pilot survey to find what parts to investigate further. Then the main survey was conducted. After that the results were analyzed to find patterns and interesting aspects. This lead to a series of questions, which was then used when conducting interviews with people with different experience and job titles. The result from the interviews were put together and related to the findings in the survey. This finally resulted in a list of recommendation on what is needed by a software platform in order to give a good DX. Lastly, this list was used to analyze how well Qlik Core follow these recommendations.

2.1 Deciding consideration aspects

There are a lot of aspects that could have been considered as requirements for good DX. The first section presents the list of aspects used, followed by a description of how it was created. It should be said that this list of aspects was revised between the pilot survey, and the main survey. The list of aspects can be seen in Table 2.1.

The original version of the list contained 14 aspects. They were decided in a combina-

tion of reading literature, my own experience of what I would consider when choosing software platforms, and a brainstorm meeting with more experienced people at Qlik, consisting of an architect and developers. The revision of the list is discussed in Section 2.3.5.

	List of Aspects
1	How often the software is updated
2	I can have working code quickly
3	The API documentation gives thorough explanations on how it works
4	The API has code examples
5	The documentation doesn't assume any prior expertise
6	The documentation has consistent language
7	The documentation is easy to navigate
8	The official website looks professional
9	The pricing of the software
10	The release- and change notes are thorough
11	The software has the same features on all different platforms
12	The software is compatible with different platforms
13	The software is offered in more than one programming language
14*	The software is open source
15*	The software uses the programming language I am most comfortable with
16	There exists an active online community around the software
17**	The creator of the software has good communication with it is users
18**	The creator of the software has high transparency with its issues, ways of working, future plans, etc.
19**	The creator of the software seems professional
20**	The creator of the software has a good reputation online
21**	I have heard of the creator of the software before
22**	I have heard of other software the creator of the software has made

**Not part of the first survey, **Not part of the second survey*

Table 2.1: List of aspects considered to give good DX

The process of deciding the aspects consisted of three parts: reading literature, self-reflection and a brainstorming meeting, which each part generated a few of the aspects used in the end. An overview of the process can be seen in Figure 2.1. The next coming sections talk in detail about how the aspects were derived.

Jarman (2017) is the source for some of these aspects. One of the points he makes is the importance of a great documentation. He says the documentation should always be written as if the developer is a beginner, which lead to aspect ”5 - The documentation doesn't assume any prior expertise” in the list. He also says great documentation is consistent, ergo it does not use different words to mean the same thing, which lead to aspect ”6 - The documentation has consistent language”. A third aspect he says is needed for great documentation is that is has a logical structure, which lead to aspect ”7 - The documentation is easy to navigate”. The last part he considers important for a great documentation is verbosity. As he puts it, ”You can never say too much”. This resulted in ”3 - The API

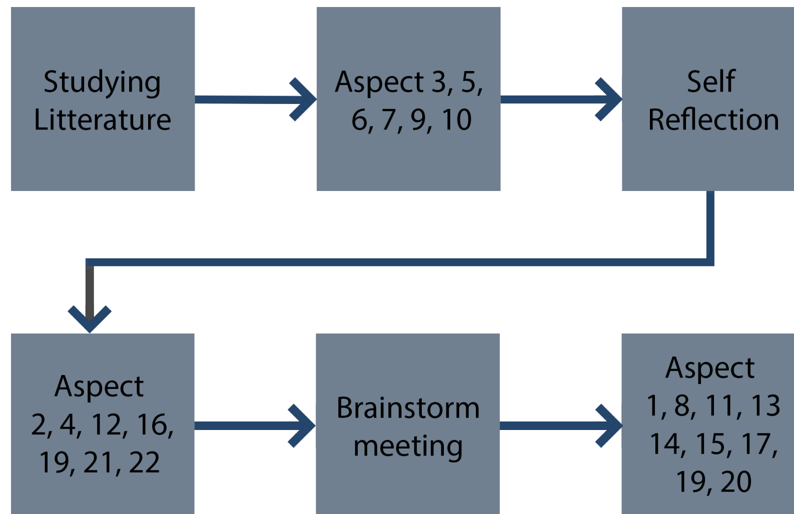


Figure 2.1: The process of deciding the aspects to be considered in this report

documentation gives thorough explanations on how it works". Jarman also thinks it is important to have good release notes. He goes on to present what release notes should consist of. According to him, not only should the release notes consist of the expected, such as what's new, updated, deprecated, fixed, etc, but also point out possible risks of the new release, such as things that might break with it. Although these sub-features of release notes might be interesting to list as their own aspects, the list had to be kept short and the aspect that was ended up with was "10 - The release- and change notes are thorough".

Jarman also talks about pricing. He puts emphasis on that pricing of the software should be easy to find for the developer. This was also listed by Every Developer (2019) as an aspect they consider. During the brainstorming, this aspect was also discussed. The aspect was ended up with was the somewhat vague "9 - The pricing of the software". This was deliberately chosen to be a somewhat open-ended aspect, since there's a lot of things that you can consider around the pricing. The information that was searched for was simply if pricing was something that was often considered in general. In hindsight, it might have been better to have divided this into several aspects, since it is difficult to know how the survey taker interpreted the aspect.

When this short list existed, I sat down and thought of things that I consider myself when picking software. The list was extended further, with aspects related to API examples, online community and platform compatibility.

After this, a brainstorming meeting was held with two senior developers and a senior architect to further extend the list of aspects. The meeting attendees can arguably be considered to have expertise within the field, having worked for many years within the industry. The architect had worked at Qlik for five years, and within the industry for 15 years. He works mostly with back-end development. He is a very outspoken person with lots of ideas, and felt like a valuable person to have at the brainstorming meeting. One of the developers is a fairly new employee at Qlik which had recently moved over to working with web develop-

ment. She felt like she could be valuable, having fresh eyes on web platforms. The second senior developer had extensive experience in both back and front end development and was also part of the team developing Qlik Core. Since she is part of developing a software platform and have thus thought about these aspects a lot, it felt natural to include her in the brainstorming meeting.

Firstly, all three were given five minutes to write down as many aspects as they could think of. These aspects were then presented in turn and written on a whiteboard. Any further ideas that the three experts got when they saw the others' ideas were also added. I then presented the list I had created myself earlier and the aspects not yet listed were added as well on the whiteboard. The list was then discussed, as well as the phrasing of each aspect and the importance of them.

Finally the list contained fourteen aspects, as seen in Table 2.1. Being open source is pointed out as important by Jarman (2017), and was discussed in the meeting. It was however not included in the final list. After the first survey was conducted, it was also pointed out by survey takers as an aspect that they considered. It was then added to the list to be used in the main survey. The list also contained the aspect of how often the software is updated. Linares-Vásquez et al. (2013) researched how the stability of an API, i.e. change-proneness, impacts the success of software using that API. The study suggests that software using APIs that are not prone to change were more successful.

Aspect number 15 on the list was added for the main survey as well. The reasoning here being that the aspect ”13 - The software is offered in more than one programming language” felt like it needed a parallel question: ”15 - The software uses the programming language I am most comfortable with”, to see if the importance of several language was solely based on the fact that people wanted their favourite programming language.

Robillard (2009) conducted a similar survey as the one being made in this report, which had more open-ended questions where the respondents would list what obstacles make it difficult for them to learn an API. This survey found that there were five areas that may cause difficulties in learning an API. They were ”Resources”: Obstacles caused by inadequate or absent resources for learning the API (for example, documentation), ”Structure”: Obstacles related to the structure or design of the API, ”Background”: Obstacles caused by the respondent's background and prior experience, ”Technical Enviroment”: Obstacles caused by the technical environment in which the API is used (for example, heterogeneous system, hardware) and lastly ”Proces”: Obstacles related to process issues (for example, time, interruptions). The most common obstacle were insufficient or inadequate examples, followed by issues with the API's structural design and thirdly unspecified issues with the documentation.

2.2 Linking considerations to ISO-9216-1

As described above, there is no standard for DX. ISO-9216-1 is however a standard to describe and measure the quality of software, so comparing the aspects to this list can be

interesting. In Table 2.2, the aspects are linked to characteristics they relate to. The aspect ID's can be found in Table 2.1 and the references to the sub-characteristics can be seen in Table 1.1 and 1.2.

Table 2.2: Relation between ISO-9216-1 and DX-aspects

Aspect ID	Functionality	Reliability	Usability	Efficiency	Maintainability	Portability
1		R1			M3	P4
2	F1, F2		U1, U2, U3		M2	
3	F1, F2, AC	AC	U1, U2, U3, AC	AC	M1, AC	AC
4	F1, F2, AC	AC	U1, U2, U3, AC	AC	M1, AC	AC
5	AC	AC	U1, U2, U3, AC	AC	AC	AC
6	AC	AC	U1, U2, U3, AC	AC	AC	AC
7			U1, U2, U3, AC			
8			U4			
9			U1			
10	F1	R1	U1			P4
11						P4
12	F3					P1, P2
13			U2			
14	AC	AC	AC	AC	M1, M2, M4, AC	AC
15	F1, F2		U1, U2, U3			
16			U1, U2			

2.3 Surveys

2.3.1 Reasons To Do Surveys

Surveys are a popular way to collect quantitative data. There are several situations when surveys, or questionnaires as they are also referred to, are a good method to use when one wants to collect quantitative data. Denscombe (2010) has described it as a good method when you are working with data that is of non-sensitive subjects, when the data sources are spread out and when the data pool is big. He also states that questionnaires are a good way to collect data of both factual nature, as well as opinion-based. Denscombe goes on to say that a questionnaire is likely to contain collection of both of these kinds of data and that it is important to make it clear to the person taking the questionnaire whether or not the question is asking for an opinion, or a 'fact'. Since this research project concerns itself with a mixture of both factual data, such as people's work experience, job titles and responsibilites, as well as opinion-based data relating to developer experience, questionnaires are a good methodology to use.

2.3.2 Construction of a Questionnaire

Denscombe (2010) has a section where he lays out the vital parts of a questionnaire. According to him, a questionnaire should always contain the following parts.

- **The Sponsor** - Clearly state who this questionnaire is from and for.
- **The Purpose** - Why is this questionnaire being made and for what will the data be used. He warns however that one should not go into too much detail, as to lead the questionnaire taker into answering in a certain way.
- **Confidentiality** - Assert the questionnaire taker that the data collected will not be publically available or be directly linked to the him or her, if him or her so not wishes.
- **Voluntary Responses** - Convey that the questionnaire is completely voluntary to take.
- **Thanks** - Make sure to extend your thanks to the questionnaire taker for voluntarily taking the questionnaire.

Denscombe (2010) also states that there is no good, defined number of questions that should exist in a questionnaire. However, it should be *as brief as possible*. He warns of trying to ask too many questions, for anything that *might* be important. This is not a good approach. A questionnaire constructor should always try to keep the scope as tight as possible. He gives four tips for when trying to do this.

- Only ask questions that are absolutely vital for the research
- Proof-read to make sure you do not ask any duplicate questions
- Make it as fast and easy as possible to answer the questionnaire
- Have a test-round for your questionnaire

Furthermore, Denscombe (2010) stresses the importance of the phrasing of questions and knowing your target audience. He gives a comprehensive list of things to think of when doing this. Some key parts are that using words that are suitable for your target audience, avoid leading questions and make sure to include sufficient alternatives. He also puts emphasis on the ordering of questions. He states that ”easy” questions, that don’t require much consideration from the questionnaire taker (Such as job title) should come early in the questionnaire.

In his book, Denscombe (2010) also talks about consistency in how you state your questions. There are pros and cons of using a variety of style for your questions, according to Denscombe. He suggests that using a variety of styles will stop the questionnaire taker from becoming bored, and also stops them from falling into a pattern, where they simply answer the same way every time without considering the question. The upside of using the same question style is that it will make the questionnaire taker used to how the questions

should be answered and limits the likelihood for confusion.

The first questionnaire had a mixture of question style whereas the second one had a more standardized style for the questions. The reasoning for this is discussed in Section 2.3.5.

2.3.3 Making a Test Questionnaire - Survey 1

In the early stages of the research project, it was still not decided what part of DX should be thoroughly researched. A test-survey was therefor constructed with what was known to be a very broad scope. The goal of the survey was not to collect useful data per say, but rather to find what sub-field of DX was interesting to pursue. The goal was also to see how well defined the list of aspects was (see Table 2.1), if there were better ways of phrasing the questions and if people even knew what developer experience was. The survey was only sent internally to people working at Qlik. I was weary of Denscombes warning that people are hesitant to do more than one survey, but knew that the main survey would be much later. I also made sure, when the survey was sent out, to keep the tone casual to only catch people whom were actually interested in taking this survey, to ”save” the more hesitant people for the main survey, hoping that the interested people might take the main survey later as well. A last step to prevent this feeling of ”Only having one shot” was to only keep the survey open for 24 hours. The questions in the survey concerned how people find new software, what they consider when choosing software, if there are any deal breakers for them when choosing a software platform and if they had heard of the concept of DX. The The survey can be found in appendix A.

Survey 1 Subjects, Questions and Takeaways

As stated before, the scope was kept very broad. The survey consisted of three main parts: Background information, how developers find new software and lastly their relationship to DX around software. Both close- and open-ended questions were mixed in this survey. From this, two things were found:

1. More aspects suggestions of things to consider around DX and how people find new software.

2. The process needs to be more streamlined in the main survey with close-ended questions to more easily be able to compare the data.

Exploring how people find new software would have been interesting, but the directly DX-related questions on what people want from a software was more interesting to research further. The test survey also found that the majority of developers had not heard of the concept of ”Developer Experience” before, or could not define it.

2.3.4 Follow-up Interview to Survey 1

After the results from the initial survey had been evaluated, two follow-up interviews were conducted in order to get insights if there were any issues with the survey. Results from the

research are not normally presented in this section. However, as the results of the follow-up interview affected how the main survey was constructed, some results will be presented here right away anyway.

The overall takeaway from the interviews was that mindset, context and definitions was a concern. The two interviewees realized during the interview that they had not given consistent answers, due to having thought of different situations for different survey questions. The survey also used technical jargon, that while it had been explained in the survey, had not been read by the interviewees. This leads to questioning the integrity of the data collected in the first survey, since one cannot know if the survey takers had used their own understanding of the jargon, or read the definition given in the survey. During the two interviews some questions were discussed as well as being too open-ended to be answered with close-ended alternatives. Lastly, the interviewees had an issue with the questions having too little context. As stated by one of the interviewees, ”Well, it depends on the context.”, was the answer to a lot of the survey questions according to him. He stated that it depends if he is working on a hobby project, or professionally. He also stated that it depends on if he’s the only one that will use the platform, or if his co-workers will as well.

Takeaways from Follow-up interview

Some takeaways are that the context is key to many of the questions that are trying to be asked, and it needs to be clearly stated in the main survey. Another insights is that it needs to be confirmed that the survey taker has read the definitions of certain jargon to give comparable data when analyzing the results. Lastly, a context given by an example would be good to have, to put all the survey takers in the same mindset.

2.3.5 Making the Main Questionnaire - Survey 2

Having made a pilot survey and a follow-up interview, the main survey for the research project was ready to be constructed. When the second questionnaire was constructed, it was sent out internally on Qlik and shared on Qlik’s official twitter, which is followed mainly by customers of Qlik. This is discussed further later in this chapter.

The questionnaire can be seen in Appendix B.

Re-scoping and Changes

Overall, the width of the project had to be scoped down, and some things had to be dropped from exploration in the project. The first survey found that the creator behind a software was less considered than had been anticipated. Although it would have been interesting to explore, the decision was to drop these questions for the main survey. The questions about how they find new software and how long time they spend on this was also be dropped from exploration. The re-scoped focus of the main survey instead became what developers are considering when looking at software platforms.

It was clear that most people had not heard of, or were not sure about, what developer

experience was. The main survey therefor included an even more comprehensive definition of exactly what DX is. The follow-up interview made it clear that people may skip texts about definitions and context, which lead to the change of making these texts more in focus and even having confirmation questions, assuring that they had indeed read the text.

With the need for more context, three different situations were defined.

> **Group:** - When working professionally and choosing a software platform for a group of people.
>
> **Single:** - When professionally choosing a software platform solely for yourself.
>
> **Hobby:** - When working non-professionally on a hobby project.

From now on, these three situations will be referred to as 'Group', 'Single' and 'Hobby'.

As many of the questions as possible were made close-ended to give easily comparable data. In the first survey there was also the very broad phrase of referring to "Tools and Framework". This term felt too broad, and since Qlik Core is a software platform, in the end people's thought around only this concept was pursued.

Furthermore, when extending the questioning around software considerations, it was split it into two parts: How often they consider something, as well as how it affects them emotionally, since this is more central to how DX is measured. Lastly, as mentioned before, two new entries were added to the list of aspects (See Table 2.1).

Survey 2 Structure

The survey consisted of three parts:

> **Part 1: Background** - A screener around who the survey taker is.
>
> **Part 2: Software Considerations** - A three-part section around of how often they consider the different aspects when choosing a software platform
>
> **Part 3: Developer Experience** - A two-part section around how likely the different aspects are to cause a positive/negative feeling

This first part of the survey contained definitions and explanations. It also consisted of a background-check of whom the respondent was, asking questions like job title, years of experience and size of the company he/she is working at. It also had a question on whether or not the respondent was in a position to make decisions on what software other people would use. This question was relevant out of two reasons. The first was to be able to have data on whether or not decision makers have other priorities. It also made sure that non-decision makers would not answer the part about the context of 'Group', since they did not have the authority to make those decisions anyway.

The second part of the survey focuses on what people consider when choosing a software platform. This part was divided into the three contexts Group, Single and Hobby.

The questions were the same in these three parts, the only difference was the context given. The question was:

> ”When `<context>`, which of these traits or aspects do you usually consider when using a software platform?”

The respondent would then rank each aspect (see Table 2.1) on a scale. They had the following alternatives.

- Never consider
- Rarely consider
- Sometimes consider
- Often consider
- Always consider

The last part of the survey focused on developer experience. It had two sub-parts, how likely a factor is to cause them to leave an interaction with a software platform with a positive feeling, and how likely a factor is to cause them to leave an interaction with a software platform with a negative feeling. The question were closely linked to the questions asked in the consideration part, but focused on the *feeling* rather than if they usually consider the factor when choosing a software platform. For this section, the survey takers had to consider the following questions, asked twice with one version being `POSITIVE` and the other being `NEGATIVE`.

> On a scale from 1 - 5, how important is it that these aspects exist in order for you to leave the interaction with the software platform with a `<POSITIVE / NEGATIVE>` FEELING?

The survey taker was then presented with the following alternatives:

1 - Not very important

2 -

3 - Neutral

4 -

5 - Very important

So to put it more clearly, one example could be: "If the software platform was open source: on a scale from 1 - 5, how likely are you to leave the interaction with a positive feeling?". The same question was then asked again, in it’s opposite form: "If the software platform was not open source, on a scale from 1 - 5, how likely are you to leave the interaction with a negative feeling?".

In this paper this type of question will from now on be referred to as ”DX Impact”, in reference to an aspect having a certain amount of positive impact on developer experience if it exists, and a certain negative impact on developer experience if it does not exist.

Implications of Data Pool

The survey was sent out internally to Qlik employees. It was also shared with customers of Qlik through their twitter account. This is a quite homogeneous group of people, all working with a certain type of software. While this could be seen as concerning for the data, it was decided that a wider audience than this should not tried to be reached. The reasoning behind this was that there were no reliable or guaranteed way of widening the data pool by any big numbers. However, if the survey was to be shared online and ask 'random' people to answer, one could not know who is answering. By keeping it limited to these two sources, one can have control of who are in the data pool and can take this into account when drawing conclusions.

Analyzing Survey 2

With all the questions, a part from people's job title, being close-ended it was quite easy to streamline and compare the data collected from the survey. The data was loaded into Qlik Sense, which made it easy to find patterns and insights of the data. In order to compare the questions where people had to choose alternatives on a scale, each answer was given a number of points, see Table 2.3. This was based on Denscombe (2010, p. 243) description on how to compare ordinal data. The decision to make it go between 0.00 and 1.00 was arbitrary, the importance is that the distance between the numbers are equal to make them comparable.

Table 2.3: Scaled answers and their given points when analyzing the data

Answer		Points
Never Consider	1 - Not very important	0.00
Rarely Consider	2	0.25
Sometimes Consider	3 - Neutral	0.50
Often Consider	4	0.75
Always Consider	5 - Very important	1.00

Comparison of Different Types of Questions

In the main survey there was one section with consideration questions, and one section with DX-related questions. These were closely related, but some differ in what they are asking. In Table 2.4 and Table 2.5 all the questions grouped together. The most noteworthy differences between DX-question and consideration question is the pricing of the software, where the consideration question is very broad whereas the DX-question specifically asks how easy it was to find the price. Another question that differs is the broad consideration question of how often the software is updated, whereas the DX-question specifically asks about how quickly the software addresses bugs.

Table 2.4: Part 1: The questions asked in the main survey, grouped by aspect category.

AMOUNT OF PROGRAMMING LANGUAGE OFFERS	
The software was offered in more than programming language	P
The software was only offered in one programming language	N
The software is offered in more than one programming language	C
ONLINE COMMUNITY	
The online community was helpful with up-to-date discussion threads	P
The online community was not helpful and the discussion threads were out-of-date	N
There exists an active online community around the software	C
RELEASE NOTES	
The release notes for what's new/updated/deprecated/etc in an update were well-written	P
The change- and release logs for what's new/updated/deprecated/etc in an update were poorly written	N
The release- and change notes are thorough	C
DOCUMENTATION LANGUAGE CONSISTENCY	
The documentation had a consistent language, not using different words to mean the same thing	P
The documentation did not have a consistent language, using different words to mean the same thing	N
The documentation has consistent language	C
DOCUMENTATION NAVIGATION	
The documentation was easy to navigate	P
The documentation was hard to navigate	N
The documentation is easy to navigate	C
PLATFORM COMPATIBILITY	
The software was compatible on different platforms	P
The software was compatible on only one platform	N
The software is compatible with different platforms	C
BEING OPEN SOURCE	
The software was open source	P
The software was not open source	N
The software is open source	C
OFFICIAL WEBSITE LOOK	
The official website looked professional	P
The official website did not look professional	N
The official website looks professional	C

P: Positive DX Impact Question
N: Negative DX Impact Question
C: Consideration Question

Table 2.5: Part 2: The questions asked in the main survey, grouped by aspect category.

DOCUMENTATION PRIOR EXPERTISE	
The documentation did not assume that I had any prior expertise with the software, not referencing software-specific things without explaining them	P
The documentation assumed I had prior expertise with the software, referencing software-specific things without explaining them	N
The documentation doesn't assume any prior expertise	C
FEATURES ON ALL PLATFORMS	
The software's features existed and acted the same on different platforms	P
The software's features did not exist or acted differently on different platforms	N
The software has the same features on all different platforms	C
UPDATES	
The software quickly released updates to address bugs	P
The software was slow to release updates to address bugs	N
How often the software is updated	C
WORKING CODE QUICKLY	
I could quickly have working code when starting from scratch	P
I took a long time before I had working code when starting from scratch	N
I can have working code quickly	C
API DOCUMENTATION THOROUGHNESS	
The API was thoroughly explained so that you could understand how it worked	P
The API was poorly explained so that you could not understand how it worked	N
The API documentation gives thorough explanations on how it works	C
API CODE EXAMPLES	
The code examples for the API were good	P
The code examples for the APIs were bad	N
The API has code examples	C
FAVOURITE PROGRAMMING LANGUAGE	
The software used a programming language I am skilled in	P
The software used a programming language I am not very skilled in	N
The software uses the programming language I am most comfortable with	C
SOFTWARE PRICING	
The pricing of the software was easy to find	P
The pricing of the software was hard to find	N
The pricing of the software	C

P: Positive DX Impact Question
N: Negative DX Impact Question
C: Consideration Question

2.4 Interviews

2.4.1 Reasons to do Interviews

There are several reasons why interviews are a suitable method use in this research project. The quantitative data collected by the two survey gives a good idea of what is needed, but does not answer *why* it is needed. Interviews will therefor give a depth to the quantitative data that has been collected. Denscombe (2010) recommends using interviews as a follow-up to questionnaires. As he puts it, questionnaires can generate some interesting results that interviews can pursue in greater detail and depth. He also states that interviews should be seen as a good way to corroborate data found with other methods. By using interviews, one can triangulate data collected by the questionnaires to confirm the facts once more with another approach. Furthermore, Dencsombe talks about interviews being well-suited for certain kinds of data. He gives three main data types when interviews are a good method, two of which are applicable to this research project. The first reason is when data is based on emotions. DX is, as described before, based on the *feeling* of a good interaction, something that is hard to get a understanding of through surveys. The second reason given by Denscombe is when you have access to 'key players' with in a field. That is, when you have the possibility to interview people that have great insight into a field, interviews are a good way to collect that data. With these interviews, one have access to people of different experience, job titles and level of decision power. By doing interviews, one can get an understanding of why different job titles desire different things, why more experienced people want certain aspects and why persons with a lot of decision power within a company have needs that others do not.

Interview Instead of Observation Study

An observation study was discussed as a method to be used to evaluate DX. This type of data collection is presented in Denscombe (2010, pp. 196-215). When evaluating UX, observation studies are used to see how people interact with software. It is mainly used to see if an UI is intuitive. The conductor of the research (the observer) let's a user (the observed) interact with a piece of software. Meanwhile, the observer asks non-intrusive questions, such as "What are you thinking now?" and "What do you expect to happen when you click this?". The observations, such as how long people are stuck or if they made any assumptions that were not expected, are written down. A discussion around this approach for evaluating DX as well was had. However, when diving deeper into what could be observed, it was found that this approach was quite limiting. Documentation navigation, consistency, examples and deception thoroughness could be explored with an observation study, whereas the other aspects would be more difficult to explore with an observation study. Since interview had no limitations on what could be explored, it was decided that this approach was superior.

2.4.2 Interview Decisions

There are several things that had to decided when interviews were to be conducted. This included what type of interview style was to be used, what structure the interview should

have, what subjects should be explored and what questions should be asked.

Interview Style

There are many different ways of conducting interviews. Denscombe (2010) groups it into three different approaches: one-on-one interviews, group interviews and focus groups. These methods all have pros and cons.

One-on-one interviews are easy to arrange, since only two people's (Interviewer and interviewee) schedules have to coincide. Compared to group interviews and focus groups, it is also easy to control the interview, dive deeper into questions when needed and it is easy to link the results to the specific person.

Group interviews have advantages as well. It helps to find what a consensus around a topic is, where people's opinions can be challenged right away by other group members. There are also risks of group interviews, where 'quieter' people may be silenced by members of the group who are more dominant. Group interviews are also less suited for topics where some answers that are more 'accepted' than others.

Finally, focus groups is quite similar to group interviews. The major difference is in the moderation, where group interviews is closer to one-on-one interviews where the interviewer asks the group very specific questions, whereas in focus groups the participants are less moderated and discuss more amongst themselves. Focus groups have the same issues as group interviews does. It also requires a more skilled moderator, since focus groups is more of an ongoing discussion of experts within a field, that can easily get out of hand if the moderator does not know how to steer the group. Transcribing focus groups are also more challenging since it is natural that people talk over each other. Linking opinions to certain people may also be harder when everyone's statements are blurred together. Focus groups is pointed out by Kontio et al. (2004) as a very cost efficient and quick method for collecting this type of data. That article however also warns that focus groups requires a lot of planning and instrumentation. If it lacks this, it may result in biased results.

For this research project, one-on-one interviews were chosen. The disadvantages of them are small, and one the most challenging things about the interviews is finding time to arrange them. It was not feasible to try to find enough people for a group interview where everyone's schedules could coincide.

Interview Structure

Dencsombe describes three different types of interviews: structured, semi-structured and unstructured interviews. Structured interviews are much like questionnaires, where the interviewer holds a tight grip on the interview and does not expect free form answers. Structured interviews looks to standardize the results for easy comparison. This interview structure is more for 'checking' rather than 'discovery' and is therefor more of a quantitative data collection rather than qualitative.

Semi-constructed interviews has, just like constructed interviews, a clear set list of questions. However, in a semi-structured interview, the interviewee is not presented with yes or no questions, or questions with alternatives. The questions are instead open-ended. As described by Denscombe (2010), the interviewee is expected to develop their own ideas and speak widely about the given issue or question. Here, the emphasis is more on 'discovery' than checking.

Unstructured interviews is the third alternative and is the loosest of the three styles. With this structure, the interviewer wants to get the interviewees general thoughts on a subject or issue and tries to be as unobtrusive as possible. As described by Denscombe (2010), the interviewer presents a specific issue or subject and hopes to get a ball rolling. Semi-constructed and unstructured exist on a continuum. The more open-ended the questions are, the more you move from semi-structured to to unstructured.

For this research project, the style of semi-constructed was used. The goal is to discover new things, rather than check and confirm, which removes the option of a structured interview. However, there is a very clear set of questions that need answers.

2.4.3 Interview Subjects

There are many things that be explored for these interviews. Due to limited time, it had to be narrowed down to a few subjects. The results of the survey will presented in Section 3.2, but they served as a basis for choosing interview subjects. The three subjects that were chosen are related to release notes, APIs and online community.

Release notes is noted as a very important aspect by literature, but the survey results showed that this is one of the least considered aspects. It was therefor decided to be futher explored.

The API documentation, examples and quick-working code were pointed as the most important aspects by the surveys. Therefor they were decided to be explored more.

The last subject that was chosen to be explored was online communities. This is personally a very important aspect, which I was interested in exploring more.

2.4.4 Interview material

I concluded, after discussion with mentors, that it would be good to have reference points during the interviews, that there should be some material that the interviewee could use when talking. It was discussed using some API documentation and release notes from existing software platforms, more precisely Qlik Core. However, Qlik Core is a very advanced platform which takes a long time to grasp. Since the interviewees have limited time to put aside for the interviews, it seemed like making them read and understand documentation of a whole new software platform was too much to ask. Instead an API documentation and release notes for a made up software platform called 'MyBakery' was created. Everyone already have a good mental picture of how a bakery works, so even if you give

them limited amount of documentation they still have a mental picture of what this software platform does. To make the request of making them read documentation less boring, I tried to make a gamification of the task. The interviewees were given the material, and three questions they needed to answer. This forced them to understand the material, rather than just read through it. The material can be seen in appendix C.

2.4.5 Interview Questions

Ultimately, the question that needs to be answered is ”Why are the aspects needed or not needed to give a good DX and what happens if they do/do not exist or are poorly/well implemented?” The questions for the interviews were constructed to be able to answer this question. Since the interview is semi-constructed, all questions will not necessarily be asked. Depending on how the interviewee answer, some questions may already have been answered, some questions may be uninteresting to dive into, etc.

A - APIs

For the API documentation, API examples and to have working code quickly, it was already known that it was important. It was not very surprising that it was. For this, the goal was rather trying to understand how one can define ”good” API related things.

AA **General**

- AA1 How important would you say API documentation and examples are?

AB **Api Documentation**

- AB1 When you look at API documentation, what are you usually looking for?
  - What should the documentation look like?
  - Is there anything in the material you always look for, or something that is missing?
  - When you come across an API documentation, are there any red flags you look for?
- AB2 How does the API documentation quality affect you in your work?
  - Do you abandon a software platform if the documentation is poor?

AC **Api Examples**

- AC1 What is your goal when looking at API examples?
  - For copy-pasting?
  - To understand underlying structure?
  - To simply see how it is used?
- AC2 What should API examples look like?
  - Short examples or long examples?
  - Should it be runnable or concise?

- Within a big context or concise?

AC3 How does the API examples quality affect you in your work?

- Do you abandon a software if the examples are poor?

AD **Working Code Quickly**

AD1 Is it important for you to have working code quickly?

- Why is it/is it not?

AD2 When can you accept to not have working code quickly?

AD3 Can you have working code quickly if the API examples and documentation is bad?

B - Release Notes

For release notes, it is known through the survey that they are *not* considered. The questions have been divided into three groups: "When are [release notes] needed" and "What should [release notes] look like?". A final question to see if the interviewee had any ideas about anomalies in the surveys was also asked.

BA **BA - When are they used?**

BA1 How often do you look at release notes?

BA2 In what circumstances do you look at release notes?

- Do you look at release notes before deciding to use a platform?

BA3 What do you look for when looking at release notes?

BB **What should they look like?**

BB1 Would you say it is important that software platforms have release notes?

- Do you find that information in some other way?

BB2 How detailed should they be? / What should they look like?

- How important is it for you that the release notes are thoroughly written?

BB3 Is it worth a company's time to make thorough release notes?

BB4 How do poor release notes affect you?

BC **Survey Anomaly**

BC1 Release notes is ranked as the least, or amongst the least important aspect for all groups. Why do you think that is?

C - Online Communities

For online community, the goal is to try and understand why it was ranked so differently in the two surveys. A goal is also try to understand why, in general, an online community matters.

CA **General**

- CA1 When talking about software platforms, what is an online community to you?
- CA2 If the documentation was flawless, would you not need an online community?

CB **When are they used?**

- CB1 How often would you say you take help from online communities?
- CB2 Do you check out the online community before choosing a platform?

CC **What should it look like?**

- CC1 What do you want from an online community / what should it look like?
  - How important are online communities?
  - What do communities that you like have in common?
  - Is it important that a community feels alive?
  - Is it important the community feels helpful?
  - Is it important that the tone used in the community is positive?
  - Is it important that the company behind the software are part of the community?
- CC2 If we compare software platforms to something smaller, such as a library. Would you say it is more important or less important to have a software community around it?
  - Why is it more/less important?

CD **Survey Anomaly**

- CD1 In the first survey, having an online community was ranked as the second most important aspect. In the second one, it's ranked in the middle. Why do you think that is?

2.4.6 Analysis of Interviews

After all the interviews had been completed, they were transcribed from auditory format into text form manually. Each statement from the interviewees were coded with what topic was being talked about and what question they were answering with that statement. Once this had been done, each question in Section 2.4.5 was looked at. By having all the statements coded, one could quickly see what each interviewee had answered for every question. The answers for each question from the different interviewees was then summarised, to try to get an overview of what the the concerns around each topic was. Any anomalous answers that went against the other interviewees answers were also noted. The results of the interviews were then summarised in Section 3.3.

2.5 Making Recommendations for Software Platforms

Using the results from the surveys and the interviews, this report gives recommendations to software companies making software platforms. The result for each aspect is summarised, and then given a result where it is presented how important the aspect is to a software platform, how big of an effort it is to make a software platform have the aspect, and how big of a payoff it is for a company to implement the aspect. The importance of the aspect is based on combination of the effort and payoff. The effort to make the software platform have the aspect is estimated, and the payoff is based on a combination of the scores it got from the results in the surveys and interviews. The importance is scored on a scale from ”Not important”, ”Not very important”, ”Somewhat Important”, ”Important” and ”Very Important”. Payoff and effort is scored on a scale from ”Low”, ”Low-Medium”, ”Medium”, ”Medium-High”, ”High”. Each aspect is also presented with a recommendation on what a software company making a platform should do with the aspect.

2.6 Evaluating Qlik Core

One of the goals for this research paper is to evaluate how good of a software platform Qlik Core is. In section 3.4, the recommendations I have for companies who provide a software platform is presented. These recommendations served as a basis for evaluating Qlik Core. I went through each aspect and determined how well Qlik Core followed the recommendations given. I discussed in what ways QC does, or does not, follow the recommendations and what issues it had. Finally I gave a verdict of either ”Needs to be changed” or ”No need for change”. In the case of giving the verdict ”Needs to be changed”, I presented what needs to be done in order to get a passing grade. After it all a summation is given in Table 3.16. If the aspect is passed, the score of aspect is counted, otherwise the aspect gets 0 points. The sum of points is divided by the total possible points to give a score between 0.00 and 1.00.

$$\frac{\text{Sum of points from passed aspects}}{\text{Sum of points from all aspects}} = \text{Score}$$

The results were viewed from different groupings, which is discussed later in this paper. The groupings had a different number for the maximum scoring, and the result had to be put into an interval so one could compare them. The interval of 0.00 and 1.00 was chosen as it can be seen as a percentage of many of the total possible points each grouping scored, with 0% being no points scored and 100% being that they passed all aspects.

Chapter 3

Results and Discussion

3.1 Initial Survey

Although the first survey only was a pilot version, it can be interesting to briefly evaluate the results of it. There were also things asked in the first survey, that was not in the main one. The survey got 38 responses in total. Initially it was intended to look at the outcome from four different perspectives, and compare them to the overall group. Perspective in this context means grouping the respondents by some common denominator. The four perspectives were:

People with more than 5 years experience in the industry

People who are developers within the industry

People who are architects within the industry

People in companies with more than 200 employees

The last perspective with the larger companies was removed from the report since the majority of all answers were from Qlik, and the group was therefor a too big majority of the whole group.

The survey found that the majority of people do not know what DX is. In Table 3.1 we can see how many people felt they could define DX. It can be seen that it is quite an unknown term. Almost half of the respondents had not heard of it, and only 7.9% felt they fully knew what it was.

Table 3.1: Percent of people who knew what Developer Experience was.

Have you heard of the term 'Developer Experience'?	
Answer	**Percent**
Yes, and I could comfortably give a definition of it	7.9%
Yes, and I think I could give a definition of it	21.1%
Yes, but I could not give a definition of it	18.4%
No	47.4%
I don't know	5.3%

3.1.1 Consideration Factors

The scores from consideration question in the first survey in Figure 3.1. The table is divided into the different perspectives. "Everyone" is simply the average result, "Experienced" is people with more than 5 years of experience in the industry, "Developers" are those who had the job title of developer or software engineer, and "Architects" are those who were some for of software architect. The score columns are coloured from green, to yellow, to red, with a higher score being green and a lower score being red. The columns for the difference between the perspective and the average are coloured with an increase of more than 0.05 being green, a change of 0.25-0.49 being yellow, a change of 0.00-0.24 being white and a decrease of more than 0.05 being red. As we can see, all but 3 factors scored equal to or above 0.50 points, meaning that they were factors that were more often than not considered.

Num of answers	38	24		27		5	
Group	**Everyone**	**Experienced[1]**		**Developers[2]**		**Architechts**	
Software Considerations	**Score[3]**	**Score**	**Diff[4]**	**Score**	**Diff**	**Score**	**Diff**
The API has code examples	0.87	0.84	-0.03	0.87	0.00	0.70	-0.17
There exists an active online community around the software	0.80	0.81	0.01	0.80	0.00	0.70	-0.10
The API gives thorough explanations	0.78	0.77	-0.01	0.77	-0.01	0.65	-0.13
I can have working code quickly	0.76	0.73	-0.03	0.78	0.02	0.75	-0.01
The pricing of the software	0.74	0.79	0.05	0.73	-0.01	0.70	-0.04
The software is compatible with different platforms	0.72	0.75	0.03	0.70	-0.02	0.65	-0.07
The software uses the programming language I am most comfortable with	0.68	0.67	-0.01	0.67	-0.01	0.65	-0.03
How often the software is updated	0.63	0.63	0.00	0.59	-0.04	0.85	0.22
The software has the same features on all different platforms	0.62	0.66	0.04	0.58	-0.04	0.60	-0.02
The documentation is easy to navigate	0.61	0.54	-0.07	0.65	0.04	0.50	-0.11
The official website looks professional	0.59	0.63	0.04	0.58	-0.01	0.55	-0.04
The documentation doesn't assume any prior expertise	0.50	0.45	-0.05	0.53	0.03	0.45	-0.05
The documentation has consistent language	0.54	0.39	-0.15	0.48	-0.06	0.34	-0.21
The release- and change notes are thorough	0.41	0.44	0.03	0.37	-0.04	0.50	0.09
Creator Considerations							
The creator of the software seems professional	0.63	0.60	-0.03	0.64	0.01	0.60	-0.03
The creator of the software has a good reputation online	0.59	0.54	-0.05	0.58	-0.01	0.65	0.06
The creator of the software has high transparency with it's issues, ways of working, future plans, etc.	0.55	0.56	0.01	0.52	-0.03	0.70	0.15
The creator of the software has good communication with it's users	0.50	0.48	-0.02	0.49	-0.01	0.55	0.05
I have heard of other software the creator of the software has made	0.43	0.41	-0.02	0.44	0.01	0.40	-0.03
I have heard of the creator of the software before	0.39	0.34	-0.05	0.43	0.04	0.30	-0.09
[1]: People with +5 years experience in the software industry		[3]: Score goes between 0 and 1					
[2]: People with the job title Developer or Software Engineer		[4]: Difference relative to the whole data pool					

Figure 3.1: Scores from consideration aspects in survey 1, divided into different perspectives.

If we start by looking at the software consideration aspects, we can see that the most important factor was 'The API has code examples', followed by the importance of an active community around the software, and thirdly that the API explanations were thorough. The least two important factors were 'The documentation has consistent language' and 'The release- and change notes are thorough'. This is an interesting find, since literature highlights these to factors as being very important. Many companies also spend a lot of resources to keep documentation consistent and release notes thorough. Here we see that most people, more often than not, do *not* consider this factor.

For the more experienced users, the result differ somewhat when compared to the 'Everyone' group. They take less consideration to factors surrounding APIs, documentation and time to get started, and care more about pricing, cross-platform compatibility and the thoroughness of release notes.

With the group consisting of mainly developers, we see almost the inversion of some trends in the more-experience-group. The developer group cares about documentation and time-to-get-started, and do not care about release notes and cross-platform compatibility.

Looking at the part about the creator behind the software, we can deduct a few things. Overall, the creator behind the software seem to be not completely non-important, but

also not very important, with all factors scoring close to 0.50. The most commonly considered factor was that the creator seemed professional, but that only ranked 0.63 on average. The least considered factor was 'I have heard of the creator of the software before', which ranked only 0.39.

With the more experienced users, they considered the factors even more rarely than the 'Everyone' group, apart from transparency, which increased slightly. Still, the scores are close to 0.50, and the creator once again seem to be only a somewhat important factor.

With the group consisting of only developers, it is much the same as before, with the one exception that 'I have heard of the creator of the software before' had an increase by 10%. It is however still the least considered factor.

3.1.2 Deal Breakers

The first survey also had a section of what they considered to be a deal breaker for trying out a new software. In Table 3.2 we can see the result of this.

Table 3.2: Percentage of people who found aspects to be a deal breaker

	Aspect
70%	I have to pay to be able to fully evaluate the software
65%	It takes a long time to get initially started
65%	The API is poorly explained
62%	The API has poor or no code examples
59%	The online community around the software is dead or has little activity
27%	The creators behind the software feel like they cater to businesses, not developers
27%	The online community around the software is unappealing
16%	The website for the documentation is hard to navigate
11%	The documentation uses inconsistent language
11%	The software is not open source
8%	The creators behind the software are not transparent
8%	The documentation assumes prior experience with the software
3%	The release notes are poorly written

As we can see many of the aspects that people consider when choosing a software is also a deal breaker if it is poorly implemented. A very clear result from this part is that most people are *not* willing to pay for a software before they can try it out.

3.1.3 Ways of Finding New Software

In the test survey there was also a part about how people find new software. The answers can be seen in Table 3.3. As we can see, the most common way of finding new software is either through people you know, or online discussions. Since people spend the majority of their working time online and with coworkers, this is quite a natural result. Conferences

and online courses are not part of people's daily lives in the same way, and naturally it is not a common source of new discovering new software.

Table 3.3: The most common ways of finding new software

How do you usually discover new tools and frameworks?

Answer	**Percent**
Friends or coworkers telling me about it	86.8%
Online communities and forums	86.8%
Reading blog posts or articles	55.3%
Searching for related key words online	50.0%
Social media	34.2%
Conferences	26.3%
Online Courses	13.2%

3.2 Survey 2 Results

There are many ways you can view the results. In the research it has been looked at from many different angles, and compared with different groups.

In total, the main survey yielded 39 responses. The results were loaded into Qlik Sense where one could find different groups and patterns. The results were looked from the overall average result, result depending on your job title, result depending on how much experience you have in the industry and if you have the authority to make decisions of what software a group of people will use.

The job titles were divided into four groups: Architects, Developer and Engineers, Managers and Other. The quantity and percentage of total can be seen in the Table 3.4. The "Other" group consisted of a CTO, a Product Designer, a Head of Consulting and Development and a Principal Consultant. This group of people will not be investigated in the results.

A summary of how the different job titles ranked the aspects can be seen in Table 3.5. The ranking is based on how many points they received. Having an interval as ranking means that several aspects had the same score and are in joint places.

Table 3.4: Main survey responses grouped by job titles

Grouped Job Titles	**# of answers**	**% of total**
Architects	10	25.6%
Developer and Engineers	21	53.8%
Managers	4	10.3%
Other	4	10.3%

Table 3.5: List of aspects with their ID, and points and rankings of aspects, grouped by job title

Aspects with their ID

ID	Aspect
1	How often the software is updated
2	I can have working code quickly
3	The API documentation gives thorough explanations on how it works
4	The API has code examples
5	The documentation doesn't assume any prior expertise
6	The documentation has consistent language
7	The documentation is easy to navigate
8	The official website looks professional
9	The pricing of the software
10	The release- and change notes are thorough
11	The software has the same features on all different platforms
12	The software is compatible with different platforms
13	The software is offered in more than one programming language
14	The software is open source
15	The software uses the programming language I am most comfortable with
16	There exists an active online community around the software

Scores and rankings

Aspect ID	Everyone Rank	Avg Everyone	Architects Rank	Architects Score	D&E* Rank	D&E* Score	Managers Rank	Managers Score	Other Rank	Other Score
4	1	0.90	1	0.92	1	0.91	1	0.79	1-2	0.94
3	2	0.82	3	0.82	2	0.87	3	0.73	4	0.74
2	3	0.80	2	0.85	4	0.80	4-5	0.71	1-2	0.94
9	4	0.80	5	0.76	3	0.84	2	0.75	3	0.85
15	5	0.68	6	0.69	5	0.73	6-7	0.63	5	0.69
14	6	0.68	4	0.77	6	0.69	8	0.60	8	0.58
16	7	0.65	8	0.63	7	0.67	9-11	0.54	6	0.65
7	8	0.60	7	0.66	8	0.61	12	0.52	10	0.52
8	9	0.60	9	0.63	10	0.56	4-5	0.71	12	0.44
12	10	0.58	10	0.58	11	0.52	9-11	0.54	7	0.59
1	11	0.58	11	0.52	9	0.57	6-7	0.63	11	0.51
5	12	0.47	13	0.47	13	0.44	9-11	0.54	9	0.52
11	13	0.46	14	0.41	12	0.45	13-15	0.44	13	0.42
6	14	0.41	12	0.49	14	0.34	16	0.42	14	0.39
13	15	0.36	15	0.37	15	0.31	13-15	0.44	16	0.23
10	16	0.33	16	0.26	16	0.29	13-15	0.44	15	0.35

*D&E: Developers and Engineers

3.2.1 Overall Result

Overall, there were quite a lot of interesting findings when analyzing the results from the main survey. In general, people are more likely to consider things when they are choosing for a group rather than for themselves. For the three categories, Single is the closest to the average result, with Group and Hobby existing as opposites on either side of Single. In all but one case, if it is often considered when choosing for a group, it is *not* often considered when choosing for a hobby project, and vice versa.

For the part of the survey regarding DX and people's feelings around software platforms, it can be said that there is a direct correlation between if an aspect in it is good form has a positive impact, that same aspect in its bad form will have about the same level of negative impact. However, in all but once case, the positive effect is greater than the negative effect, if only slightly. In general it is closely related to how often something is considered. That is to say, if something will cause someone to have a non-neutral feeling regarding an aspect, they will also consider it when choosing a platform. Tahat is, there is no uncoupling between what causes them to have a good DX, and what they consider when choosing a platform. There are some outliers to this, but that is the general case.

The top three most considered aspects for each context can be seen in the Table 3.6.

Table 3.6: Top 3 considered aspects by each context

	Average
1	The API has code examples
2	The API documentation gives thorough explanations on how it works
3	I can have working code quickly
	Group
1	The API has code examples
2	The API documentation gives thorough explanations on how it works
3	The software is compatible with different platforms
	Single
1	The API has code examples
2	The API documentation gives thorough explanations on how it works
3	I can have working code quickly
	Hobby
1	The pricing of the software
2	The API has code examples
3	I can have working code quickly

The overall result concluded that *the* most considered aspect overall is 'The API has code examples'. For hobby, it is the second most important aspect, only 0.03 points behind the first place. Further, good code examples had the biggest positive impact on developer experience, and bad code examples had the biggest negative impact on DX. The negative impact if the examples are bad are not as extreme as the positive impact is if they are good however. In conclusion, API examples are a key factor for software platforms' quality.

This aligns with the result from Robillard (2009), which also found that inadequate API examples is the most common obstacle for learning an API.

The thoroughness of the documentation is also one of the most important aspects, coming in as the second most important aspect for all categories except for Hobby. The positive DX-impact is as big as the negative one, and it is also reflected in that its consideration points are as high as the DX-impact points.

Having working code quickly is also in the top three for all but the group category, where it is in fourth place. The positive DX-impact if you can have working code quickly is slightly higher than the negative DX-impact if it takes a long time before you have working code. The positive impact is also stronger than if you compare it to how often it is considered.

Having the software be compatible with different platforms is a big divider. It is the third most important aspect for Group, but one the least important aspects for Hobby, and somewhere in the middle for Single. The positive impact is lower than how often it is considered, and the negative DX-impact is even less. If you exclude the Group-category, it places itself on average as the 11th most important aspect, out of 16.

The pricing of the software is important to everyone, placing itself on average as the 4th most important aspect overall, but it is *the* most important aspect for Hobby. For group and single, it is both the 5th and 4th most important aspect respectively. In Table 3.7 we can see the full score and rankings for the different contexts.

Table 3.7: Scoring for each aspect by each context

Aspect ID	Aspect	Group	Single	Hobby
1	How often the software is updated	0.69	0.59	0.46
2	I can have working code quickly	0.75	0.80	0.86
3	The API documentation gives thorough explanations on how it works	0.85	0.82	0.79
4	The API has code examples	0.93	0.88	0.88
5	The documentation doesn't assume any prior expertise	0.41	0.47	0.53
6	The documentation has consistent language	0.42	0.42	0.39
7	The documentation is easy to navigate	0.61	0.60	0.60
8	The official website looks professional	0.67	0.60	0.53
9	The pricing of the software	0.74	0.76	0.91
10	The release- and change notes are thorough	0.40	0.31	0.27
11	The software has the same features on all different platforms	0.63	0.47	0.28
12	The software is compatible with different platforms	0.81	0.58	0.37
13	The software is offered in more than one programming language	0.48	0.35	0.25
14	The software is open source	0.64	0.66	0.74
15	The software uses the programming language I am most comfortable with	0.56	0.71	0.78
16	There exists an active online community around the software	0.69	0.62	0.65

In Figure 3.2 it is shown how the consideration points compares to how much of a positive and negative DX impact it has. Table 3.8 shows the exact numbers for the DX impacts, and the average scores. The difference in positive- and negative DX impact is quite small.

Table 3.8: The average scoring for the consideration questions, its positive DX impact and its negative DX impact and their difference, by each aspect category

Aspect ID	Aspect Category	CQ* Score	NQ** Score	PQ*** Score	DX Score Diff
4	Api Code Examples	0.89	0.85	0.89	±0.04
3	Working Code Quickly	0.85	0.83	0.82	±0.01
2	Documentation Thoroughness	0.83	0.80	0.81	±0.01
9	Pricing	0.76	0.74	0.81	±0.07
15	Documentation Navigation	0.72	0.64	0.70	±0.06
14	Updates	0.72	0.63	0.69	±0.04
16	Online Community	0.71	0.62	0.65	±0.03
7	Favourite Programming Language	0.67	0.62	0.60	±0.02
8	Being Open Source	0.66	0.59	0.59	0.00
12	Official Website Look	0.63	0.56	0.56	0.00
1	Documentation Language Consistency	0.63	0.53	0.55	±0.02
5	Features On All Platforms	0.63	0.53	0.48	±0.05
11	Documentation Prior Expertise	0.57	0.47	0.43	±0.03
6	Platform Compatibility	0.47	0.42	0.41	±0.01
13	Release Notes	0.44	0.40	0.34	±0.06
10	Number of Programming Languages	0.39	0.34	0.32	±0.02

*CQ: Consideration Question
**NQ: Negative DX Question
***PQ: Positive DX Question

1	How often the software is updated
2	I can have working code quickly
3	The API documentation gives thorough explanations on how it works
4	The API has code examples
5	The documentation doesn't assume any prior expertise
6	The documentation has consistent language
7	The documentation is easy to navigate
8	The official website looks professional
9	The pricing of the software
10	The release- and change notes are thorough
11	The software has the same features on all different platforms
12	The software is compatible with different platforms
13	The software is offered in more than one programming language
14	The software is open source
15	The software uses the programming language I am most comfortable with
16	There exists an active online community around the software

Figure 3.2: The scoring of the consideration points, with how much of a positive- and negative DX impact it has. The interval goes between the lowest score and the highest score. The consideration question's score is blue, the positive DX impact question is green and the negative DX impact question is pink.

There are several angles you can view the results at. In the following sections, a few are discussed.

3.2.2 Developer and Engineers

Developer and Engineers are the majority of people who are going to use a software platform, since they make out the majority of a development team. In the survey, they also made out the majority of the respondents. In Table **??** we can see how they ranked the

considerations. According to this, they are people who want thorough documentation and API examples. They want to have working code quickly, and they care about the pricing of the software. They also more often than not care if a project is open-source, and also want to use the programming language they are most comfortable with. Even though they want thorough documentation, less often than not they don't care if the documentation uses inconsistent language or assumes prior expertise. They do not care if release notes are thorough. Developers and engineers have an average score of 0.60, with the difference between the most considered aspect, and the least considered aspect, being 0.62.

Table 3.9: The ranking of the aspects, sorted alphabetically, by all job titles. Rankings in an interval have the same score and therefor the same rank.

Question	Architects	Devs & Eng	Managers
How often the software is updated	11	9	6-7
I can have working code quickly	2	4	4-5
The API documentation gives thorough explanations on how it works	3	2	3
The API has code examples	1	1	1
The documentation doesn't assume any prior expertise	13	13	9-11
The documentation has consistent language	12	14	16
The documentation is easy to navigate	7	8	12
The official website looks professional	9	10	4-5
The pricing of the software	5	3	2
The release- and change notes are thorough	16	16	13-15
The software has the same features on all different platforms	14	12	13-15
The software is compatible with different platforms	10	11	9-11
The software is offered in more than one programming language	15	15	13-15
The software is open source	4	6	8
The software uses the programming language I am most comfortable with	6	5	6-7
There exists an active online community around the software	8	7	9-11

3.2.3 Architects

Architects are a key part of a development team. They are responsible for how the software will be structured and therefor also often have a say in if the software will be using a software platform, what platform will be used. In Table 3.9 we can see how the architects ranked the aspects. According to this result, architects are (just like developers) people who want code examples and thorough documentation. They find it important that software platforms are open-source, and want to have working code quickly. They also, just like developers, do not care if the release notes are thorough. The average score for architects is 0.61, with the a difference between the most considered aspect, and the least considered aspect, of 0.66.

3.2.4 Managers

Managers have responsibilities that differ quite much from developers, engineers and architects. They often have to think of legal aspects and are in contact with other departments, such as sales. They will often also not have the same level of competence when it comes to software development as architects, developers and engineers have. They are also not the key demographic for this research, since we are talking about *developer* experience. They are however a key part for the selection of software platforms, since they will have the most authority and also are the person in control (or is the contact to the person in control) of the money. In Table 3.9 we can see how the managers ranked the aspects. There we can see that even managers care about API examples and that they can have working code quickly. The pricing of the software is, maybe not surprisingly, quite important to them. Managers have an average score of 0.59, with the biggest difference between the most considered aspect, and the least considered aspect, being just 0.37.

3.2.5 All job titles compared

In Table 3.5 we can see all the job titles grouped. If we compare architects with developer and engineers, they are quite similar. On average, the score differences is only 0.06. The biggest difference for consideration questions is 'The documentation has consistent language', where Architects average out at 0.49/1.00 in points, making it 'Sometimes considered', whereas Developer and Engineers only score 0.34/1.00, placing it between 'Sometimes consider' and 'Rarely consider'. Their ranking does not differ much, with the biggest ranking difference being two spots. It can also be said that architects on average have a score of 0.61, and developers 0.60. So by a fine margin, architects consider more things than developers and engineers. Architects are also more extreme, where the most considered aspect, and least considered aspect have difference of 0.66 points, whereas developers and engineers have 0.62. They are however, like stated before, quite close and similiar.

If we compare architects and managers, the differences are bit more extreme. On average, they differ by 0.09 points. The biggest divider for them is 'The release- and change notes are thorough', where they differ by 0.18 points. The rank difference is only one spot though. Their biggest dividers in rank differ by five spots. They are 'The official website looks professional' and 'The documentation is easy to navigate' where the first one is more important to Architects, and the second one is more important to Managers. We can also see that managers are much less extreme in their opinions than architects, where the difference between the most considered aspect, and the least considered aspect, being just 0.37, whereas it is 0.66 for architects. Since architects will be working more directly with software platforms, they may have stronger opinions on what is needed, and not needed for them.

If we compare Developer and Engineer with Managers, we also find that their quite different. On average, they differ in points by 0.10 points. Their biggest differ in points is for 'The official website looks professional', where they differ by 0.15 points, which is also their biggest differ in ranking: 6 spots difference. Just like with architects vs managers, they're differences between the extreme are very different. 0.37 difference for managers,

and 0.62 for developers and engineers. Once again, this seems quite natural since developers and engineers will be working with the platform more and have stronger opinions on what they need.

3.2.6 Experience

The responses were also divided into five groups, depending on how much experience in the software industry the had. There were 5 people with less than 5 years experience, 10 people with 5 - 10 years experience, 10 people with 10 - 15 years experience, 12 people with 15 - 25 years experience and 2 people with 25+ years of experience in the software industry. How people ranked the aspects, depending on years of experience, can be seen in Table 3.10. The overall take away is that years of experience is *not* a good indicator of people's needs. There doesn't seem to be any relationship with "The more/less experience you have, the more/less something is important".

Table 3.10: The scoring of the aspects, grouped by years of experience.

Aspect	**< 5 years**	**5 – 10 years**	**10 – 15 years**	**15 – 25 years**	**25+ years**
The API has code examples	0.91	0.93	0.89	0.85	1.00
The API documentation gives thorough explanations on how it works	0.91	0.86	0.87	0.72	0.83
I can have working code quickly	0.79	0.86	0.82	0.77	0.92
The pricing of the software	0.73	0.86	0.81	0.83	0.67
The software uses the programming language I am most comfortable with	0.68	0.73	0.68	0.74	0.50
The software is open source	0.67	0.62	0.74	0.71	0.67
There exists an active online community around the software	0.84	0.53	0.73	0.55	0.75
The documentation is easy to navigate	0.60	0.68	0.64	0.48	0.71
The official website looks professional	0.53	0.67	0.54	0.59	0.50
How often the software is updated	0.66	0.52	0.52	0.58	0.50
The software is compatible with different platforms	0.41	0.54	0.59	0.57	0.54
The documentation doesn't assume any prior expertise	0.43	0.57	0.50	0.39	0.50
The software has the same features on all different platforms	0.28	0.42	0.53	0.45	0.29
The documentation has consistent language	0.41	0.45	0.40	0.32	0.58
The software is offered in more than one programming language	0.28	0.35	0.35	0.36	0.25
The release- and change notes are thorough	0.26	0.34	0.33	0.30	0.25

3.2.7 Decision Makers

The results were also divided into decision makers and non-decision makers. The groups are of comparable size: There are 22 decision makers for groups and 14 non-decision makers for groups, and 3 who answered that it is not applicable. In Table 3.11 we can see how decision makers and non-decision makers score differently. Overall, non-decision makers and decision makers don't differ a lot.

Aspect	**DM* Rank**	**DM* Score**	**NDM** Rank**	**NDM** Score**
The API has code examples	1	0.91	1	0.86
The API documentation gives thorough explanations on how it works	2-3	0.83	3	0.80
I can have working code quickly	2-3	0.83	4	0.79
The pricing of the software	4	0.80	2	0.83
The software uses the programming language I am most comfortable with	6	0.69	6	0.74
The software is open source	5	0.70	8	0.63
There exists an active online community around the software	7	0.62	5	0.75
The documentation is easy to navigate	8	0.59	7	0.66
The official website looks professional	9	0.58	9-10	0.61
How often the software is updated	11	0.53	9-10	0.61
The software is compatible with different platforms	10	0.55	12	0.54
The documentation doesn't assume any prior expertise	12	0.45	11	0.60
The software has the same features on all different platforms	13	0.43	14	0.46
The documentation has consistent language	14	0.38	13	0.50
The software is offered in more than one programming language	15	0.32	15	0.38
The release- and change notes are thorough	16	0.29	16	0.35

*DM: Decision Makers, **NDM: Non-Decision Makers

Table 3.11: The ranking and scorings of decision makers, compared with non-decision makers.

It could also be interesting to see who the decision makers are. Divided by job title and level, it looks like this:

Job Title	**Level**	**DM***	**NDM****
Architect	Middle	0	0
Architect	Senior	7	2
Developer and Engineers	Middle	3	4
Developer and Engineers	Senior	5	7
Managers	Middle	1	0
Managers	Senior	3	0
Other	Middle	0	0
Other	Senior	3	1
		22	**14**

*DM: Decision Makers, **NDM: Non-Decision Makers

Table 3.12: Caption

As can be seen in Table 3.12, and not surprising, all managers are decision makers. Somewhat more interesting, two architects claim they are not in a position to make decisions on what software others will use. We also see that 50% of Middle level people are decision makers, and 64% of Senior level people are decision makers.

3.2.8 Compared to Survey 1

The sample size is almost exactly the same in the two surveys. The pilot survey had 38 responses, whereas the main one had 39. There are two major differences between the two surveys. The first one had mostly just Developer and Engineers, 74%, only one manager, 3%, and just five architects, 13%. As a reminder, the second survey had 53.8% developer and engineers, 10.3% managers and 25.6% architects. The second difference is that in the main survey, the people were given a context for when they were considering the aspects.

If we compare it to the first survey, we see some anomalies. To be able to compare the rankings, we will exclude the two newly added questions in this part. In Table 3.13 we see the two surveys compared.

Table 3.13: The difference in score and ranking between the first and second survey.

Aspect	S2* Rank	S2* Points	S1** Rank	S1** Points	Difference Points	Difference Rank
The API has code examples	1	0.90	1	0.87	+0.03	0
The API documentation gives thorough explanations [on how it works]	2	0.82	3	0.78	+0.05	+1
I can have working code quickly	3	0.80	4	0.76	-0.04	+1
The pricing of the software	4	0.80	5	0.74	+0.06	+1
The software uses the programming language I am most comfortable with	5	0.68	7	0.68	0.00	+2
~~The software is open source~~	-	0.68	-	-	-	-
There exists an active online community around the software	6	0.65	2	0.80	-0.15	-4
The documentation is easy to navigate	7	0.60	10	0.61	0.00	+3
The official website looks professional	8	0.60	11	0.59	+0.01	+3
The software is compatible with different platforms	9	0.58	6	0.72	-0.14	-3
How often the software is updated	10	0.58	8	0.63	-0.05	-2
The documentation doesn't assume any prior expertise	11	0.47	12	0.50	-0.03	+1
The software has the same features on all different platforms	12	0.46	9	0.62	-0.16	-3
The documentation has consistent language	13	0.41	13	0.45	-0.04	0
~~The software is offered in more than one programming language~~	-	0.36	-	-	-	
The release- and change notes are thorough	14	0.33	14	0.41	-0.09	0

*S2: Survey 2, **S1: Survey 1

An interesting difference is the rank of 'The software is compatible with different platforms'. However, this aspect is very divided in the second survey depending on the situation. The rankings for this aspect look as following: Group ranks it as the third most important, whereas single ranks is at the 10th spot and finally hobby at place number 13. Similarly, the question 'The software has the same features on all different platforms' has a big point difference when comparing the first and second survey. However, this is also a divided question depending on situation in the second survey. It is ranked by Group at 9th, Single is tied between the 11th and 12th spot and for Hobby it is 13th. In the first survey, it is ranked as 9 out of 14. The change in ranking in both these cases can therefor somewhat be explained by the fact that no context was given in the first survey.

Perhaps most interesting is that in the first survey, 'There exists an active online commu-

nity around the software' ranked as the second most important aspect, but is on average ranked as the sixth most important aspect in the second survey. There is no difference on the context here, in the second survey it is ranked at 6 by both Single and Hobby, and is tied between 6th and 7th spot for Group. It *is* slightly more important for Developer and Engineers, but only by 0.04 points, so the fact that the first survey had them as a majority cannot explain this either. There does not seem to be any obvious explanation for this shift.

3.2.9 Perspectives that were not Considered

Data was collected on other things as well. Two of these were the size of the company the questionnaire taker worked at, and what professional level they were at their company.

Because of the way the data pool was shaped, these two groups were not looked at, although they could have been interesting. The first one was company size. 30 out of 39 people worked in companies with more than 1000 people, and only 4 out of 39 people worked in companies with less than 100 people. Therefor the data pool was too small to make any larger claims on patterns.

The other group that could have been interesting to look at was the professional level of the persons. However, 77% were senior level, and 23% were middle, and no one was junior level. The job title level is somewhat arbitrary, and it could be argued that looking at years of experience, where the answers are more divided, can substitute this angle. Most of the people with a middle level have worked less than 5 years. How big part of the work experience group are made up of middle and senior level respectively can be seen in the Table 3.14.

Table 3.14: The link between level in job title and years of experience

Years of experience	Middle level	Senior level
5 years	100%	0%
5-10 years	20%	80%
10-15 years	20%	80%
15-25 years	0%	100%
25+ years	0%	100%

3.3 Interview Results

After the survey was done and analyzed, there were fives interviews conducted. The interviewees all works at Qlik. Their names are kept anonymous and will only be referred to by their job title. The interviewees were:

- The Architect, a person with over 25 years of experience in the industry
- The Software Engineer, a person with four years of professional experience
- The Quality Architect, a person with 20 - 25 years of professional experience

- The Software Developer, a person with 10 - 15 years of professional experience
- The Development Manager, a person with 10 - 15 years of professional experience

The following sections presents the results of the interviewees by each subject field.

3.3.1 API Documentation and Examples

Just like the survey suggested, API documentation and examples are extremely important. The first people look at, according to the interviews, is the examples. That seems to be the way people get started when encountering a documentation. This is, according to the interview subjects, to get an overview of the software platform. Examples give an insight into what the software can do, what its applications are.

> ”The example is a summary without comments”
> — Quality Architect

So it is an easy way for them to quickly see what the platform is intended to do. Interview subjects also state that examples are a good way to see if it is something that they can quickly understand, if it is something they are used to. One of the interview subjects stated that the more generic an API is, e.g. the more applications it has, the more examples are needed.

> ”... which means that if it’s a broad and generic API ... they will have to have a lot of examples”
> – Architect

The interview subjects also said they look at other things to determine if they want to use a software platform. When asked if there were any ”red flags” for API documentation, several different answers were given. One stated that language inconstancy was a big warning, another if the documentation felt auto-generated, and a third if the description of the methods were lacking. People also stated they usually look at things outside of documentation when they are deciding, namely how well-known and popular the software platform seems to be.

> ”In the github you look at, is it starred, how many has downloaded it, and stuff. Is it used? Is it up-to-date?”
> — Quality Architect

> ”And googling a lot. See what others are using.”
> — Senior Developer

> ”I would look both at the API documentation but also how much it’s used and how much it’s maintained, what was the latest changes and things like that”
> — Architect

One of the interview subjects were a manager, which turned out to have quite other priorities when it came to API documentation when he was in his working role. His opinion was that developers tend to think about more what’s fun to use, rather than what is useful.

> ”I think that developers tend to play down the need for production worthy code ... [if] it looks good, if [developers] think it’s fun to work with, it’s good [to the developers]”
> — Development Manager

The manager was more concerned with reliability of the software platform. How likely was it to break down? And if so, what support can the developers get? Should we buy a support contract?

All the interview subjects said that they want to have working code quickly. Some of them attributed this to them being impatient or lazy, but when you started to question a bit more you realized that this is actually their way of figuring out if a software is valuable. When question about why he wants to have working code quickly, the architect stated:

> ”I don’t get the feeling of the API otherwise.”
> — Architect

So examples were used both to understand the underlying structure, how different part of the platform are linked up. But also for copy-pasting into your project to try it out.

The interviewees were also asked if they could ever tolerate not having working code quickly. It turns out that it is acceptable if they already know that the software is valuable and is what they are looking for. This however had the prerequisite that they would work with the software platform for a long time, and they knew that the long time it took to get started would be worth it. It however caused irritation. Consensus was though that they would rather try to find an alternative rather than to spend time to get it working, if they had the option.

> ”So if there is several different alternatives, I think I start out with another.”
> — Quality Architect

> ”Well, if I have a choice to find something else to use then I would just do that.”
> — Software Engineer

> ”...if you go to ... one API and you found it quite bad, then you skip and go the next API.”
> — Senior Developer

So it is quite clear that developers are impatient and are not reluctant to drop platforms for something else, if they have the choice. It is therefor important for a software platform to get started quickly, and to quickly convey to the developer what it is that the software platform can do. Developer do not want to spend a lot of time to figure out if a software platform is useful for them.

In the interviews it was also talked about what the examples should look like. Not surprisingly, most of the people said it depends on the situation. Two cases that were brought up by several interviewees was that there should be ’getting-started’ kind of examples,

where the example is runnable. The other case was for examples that exist in the API's method descriptions. These ones should be as isolated as possible, according to interview subjects.

> ”... as isolated as possible. So you just show one small feature...”
> — Quality Architect

It was also stated by many that if the concepts are complicated, it is preferable to have smaller, step-by-step examples rather than a big example that does the complicated thing. This is especially important if the example introduces a lot of new concepts that needs to be understood.

> ”...it's important to have more step-by-step things ... [if] you need some pre-requisite knowledge in order to [understand] that”
> — Software Engineer

Preferably there should both exist a larger example within a bigger context, and a smaller snippet that just shows how specific methods of the API should be used. If there can only exist one of these, the concise example is preferred over the bigger ones.

> ”If I have to choose then it would be [the] more concise examples...”
> — Software Engineer

One of the interview subjects had a warning on quick-working code and that it can be decieving. They had been working on a project where they found a software platform to use. They got started quickly, and it seemed to be a good platform to use for their purpose. However, once they started to try to do more advanced things using the platform, they realized that the documentation was severely lacking. To do the simple things, and get up and going was no issue. But the documentation was not well documented beyond its simplest cases. They ended up doing and investigation, and decided it was easier to actually throw away months of work and use another platform with good documentation, rather than trying to use the platform with poor documentation.

> ”And you could do the basic things quickly and it worked well. But after a while every time you ran into a problem and you wanted to fix it and you look into the documentation it's like: it doesn't say anything here. Then we realized: 'Okay, if this continues happening you just keep losing time, every time you want to fix something'. So we kind of scraped it and did an investigation: 'Okay what are our other options?. And in this case, let's try to look at the documentation straight away and see if all of the problems we had with the older one are now gonna be fixed.”'
> — Software Engineer

Takeaway from API Documentation and Examples Interviews

The takeaway is that examples are *very* important in documentation. There needs to be examples that explains what the software platform can do and give the developer a way to get started quickly. Developers are impatient, so if you cannot provide a way to quickly give these things, they will choose another alternative. The documentation needs to explain

advanced concepts step-by-step. It is also important that there exists documentation for both how you do the simple things, as well as more special cases. The language needs to be consistent, the documentation needs to be manually written with care and the methods need to have thorough explanations.

3.3.2 Release note

Release notes is part of all serious software platforms. But how much are they used, what do people want from them? According to the interviews, they are not used much at all. All but one interviewee said that they use release notes regularly, and some said they almost never use them, and one had never even looked at release notes in their professional career. When asked how often they read release notes the answers were:

> ”Almost never, I can say ... they are not very important to me.”
> — Quality Architect

> ”I don’t know if I’ve ever looked at release notes”
> — Software Engineer

The software engineer, who even has part of her working duties to present release notes, stated that she never looks at release notes herself. The interviewees’ answers suggests that release notes are highly avoided. The manager said that he will look at release notes when there are major releases, to get a grip on how much time an upgrade would take.

> ”I want to know how big of a change, and how much time do we need to invest in such a change.”
> — Developer Manager

He also said that managers more likely rely on developers and architects to take that roll in general.

> ”The manager would probably ask the architects and developers to tell how much work they needed to put in ... I don’t think the manager himself would look into it”
> — Developer Manager

The architect however said that release notes are very important to him. Out of the interview subjects, he is the one with the most experience, over 25 years. He stated that he earlier in his career did not care much about release notes either. He says:

> ”A lot of year ago when I was new I would probably say the same thing, to be honest. It’s more like afterwards you really realize how important it becomes when you sit there and you have the give the next version ... maybe it’s easy to overlook that fact.”
> — Architect

The architect suggests that the release notes are something that you are taught to appreciate after having trouble with poor release notes. He says there are three different situation when he looks at release notes. The first is the initial situation, when he first encounters a software platform.

> ”...getting the feeling of how well they are documentation things ... it also gives you a feeling [of] how mature or obsolete the platform is. If the the last release note is five years old ... that’s a bad thing. If there [are] new release notes every day that’s also a warning sign, [it] means it’s not mature”
> — Architect

The second situation is the frustration phase, as he puts it.

> ”The other one is ... the frustration phase. ’Why is this not working, it should work’, and then you’re going to release notes. It’s more like finding the issue”
> — Architect

The last situation he presents that causes him to look at release notes is when there needs to be a decision if software should be upgraded or not.

> ”Probably the real reason for release notes, it’s like ’Okay, should we upgrade?’ ... I think that’s the key thing with release notes, that’s why you have them. If it’s worth to upgrade.”
> — Architect

This would suggest that there are in fact many use cases for release notes.

When asked how important they think release notes are, most of the interviewees say that they *are* important, even if they do not use them themselves. The senior developer stated that release notes is not important. But when asked about if they thought it was the general case for developers that they do not care about release notes, the senior developer stated:

> ”No. Because my colleagues wants us to write release notes. So I think ... others are actually reading them.”
> — Senior Developer

The software engineer, who had as part of the duties to present Qlik Sense’s changes between versions, stated:

> ”Looking at how much other people are interested in having us present that, then it seem like it’s quite important”
> — Software Engineer

For these two, it seems that release notes is something that they are not interested in, but have learned that others apparently do. The architect even stated, as mentioned previously, that he has been taught that they are important.

> ”So I’ve been taught in that sense that [releaes notes] are very important.”
> — Architect

For those interviewees who do look at release notes, it’s quite different things they look at. The quality architect and manager states they only care about things that may break anything. The architect, as stated before, looks at several different things. The senior developer stated that she may be interested in new features, but says she finds those things through online discussions and blogs rather than release notes. The quality architect, who is part of the decision if software should be upgraded, stated he doesn’t even look at release notes before that decision. Instead, they do test builds with the newer version, and if none of the tests breaks with the new version, they simply upgrade to that version.

> ”If there’s a new [version], we take it. And then the release notes [are] not important. Unless the new bump breaks something. Then you have to try to figure what has changed ... if our pipeline is green, we merge to master.”
> — Quality Architect

The architect is also the only interviewee that looks at release notes before choosing a software platform. When asked if they check the release notes before choosing a software platform, they stated:

> ”I don’t think I have”
> — Quality Architect

> ”Probably not ... Unfortunately not ... I probably should. But I would rely on the architects and developers to tell me if they were good enough”
> — Development Manager

When asked why they think release notes are not very important to people, according to the surveys, a few theories were presented. Mainly it was release notes are not part of every day work, and you can most of the time use a software platform without looking at release notes.

> ”It’s not so often you use them ... most things don’t change that much”
> — Architect

> ”You can use an API without the release notes”.
> — Senior Developer

When asked how poor release notes affect people, all but the architect naturally said that it does not affect them that much, since they do not use them. They however said it affects their opinion on the company.

> ”I think it is bad when it comes to ... the trust ... between whoever is using the API and whoever is developing. I think it reflects poorly on the company.”
> — Software Engineer

> ”You get a more serious feeling about the API if you have release notes. [Poor release notes] reflects poorly on the company”
> — Senior Developer

Takeaway from Release Notes interviews

The takeaway seems to be that most people don’t use release notes, even people who ”should” use release notes. People go to great lengths to not read them. The manager relies on others to read them and the quality architect relies on his tests. The senior developer and software engineer simply don’t do it. The architect cares a lot about it, that is however something that years of experience has taught him. Release notes seem to be used more as a last resort when something breaks, as is to be avoided at every cost. The release notes are however thought of by the interviewees as something very important. The mentality is that they should be there, and the lack of them makes the software creator seem unprofessional.

3.3.3 Online Communities

Online communities are places on the internet where people can discuss issues or questions they have about software. They can exist on many places. Some software companies host their own forums, other communities naturally emerge on online forums such as Stack Overflow. Sometimes they exist on both of these places. Exactly *what* defines an online community can be somewhat vague. When the interviewees where asked what an online community was to them, there were several different answers given. The consensus seemed however that *where* it existed, did not matter.

> ”When it comes to online community, it’s like anywhere on the internet.”
> — Software Engineer

> ”Well it’s somewhere I can find something specific about [the software platform]. It could be their own community like Qlik community, [or] Stack Overflow or something where I can find a lot of information. A lively debate somewhere else where I would typically find discussions, I would say that they’re pretty equal to me. In reality I don’t care [where I find the information].”
> —Architect

So online communities can exist in many places. The answers given by the interviewees would suggest that they are not loyal to any specific forum site but would use whatever source fulfilled their needs. When asked how much they used online communities, it was clear that it was a very central part of their daily life.

> ”I use them very often just to find answers. [And] asking questions I would probably say like a couple of times a month.”
> — Architect

> ”All the time”
> — Software Engineer

So an aspect that is used daily by developer is clearly something that is important.

Another aspect of a community that was asked about was if it needed to ”feel” alive. This turns out was a very central part of an online community. It was also concluded that it needed to feel helpful. The interviewees went so far as to say that they would not even consider it a community if it did not feel alive or helpful.

> ”I wouldn’t call a dead community a community. That’s an aspect of the community, that there really is somebody there. If the community isn’t able to answer questions I put there then it’s not [a] community I would say. That’s to me the key thing about it, that if I reach out and ask something I will get an answer.”
> —Architect

The development manager puts emphasis as well that the community needs to have a positive tone in how discussions are conducted.

> ”That they are polite answers and they are elaborate answers to questions. [That] there are people answering to the right level. Sometimes there are people who misuse this, like ’I have to show something to my manager, please do this for me’. I don’t think you need to answer those but you should least support: ’Here’s some help to get you started’ ”
> — Development Manager

When asked if they needed to feel like they’re apart of the community themselves, the answer seemed to be no from the interviewees. Their participation was not something that was important, the key thing was just that they got help with what they needed from the community.

> ”No. But it’s great that others want to be part of the community because otherwise I wouldn’t find my answers.”
> — Senior Developer

The interviewees were also asked if they felt like it was important that the company behind the software was part of the community. Here, some mixed answers were given as well. The architect stated that if it is a big company, it is needed. But for smaller, open-source things, it was not needed. However, the consensus seemed to be that although it was not necessary per say, it had a positive impact.

> ”Yeah, I think that’s always good. I don’t think it’s necessary that they’re part of the community but I think it would make the consumer feel better about using the product. Again, coming back to the trust and relationship between the product and the user.”
> — Software Engineer

The senior developer stated that if the company behind a software is part of the online community, it gives the feeling of the company actually caring about their users. The answers from the interviewees would suggest that

> ”...that [the company] actually take [their] responsibility for [their] users. For the community.”
> — Senior Developer

However, when asked if it has the opposite effect if they’re *not* part of the community, she changes her answer.

> ”Actually, I don’t think I care who answers the question. [But] if there are questions that are not answered, that you know that a [company]-person can answer, and doesn’t do that, that’s not good.”
> — Senior Developer

When further asked if it is worth a company’s time to answer community questions, the manager elaborated his answer.

> ”I think it goes down to a money question, actually. If you can get by without, because there is a lot of people answering, then you’re a lucky company, and you don’t need to do that. Then you can back away if you think that the community is working by itself. [Otherwise] you probably need to be there.”
> — Development Manager

In the interviews it was also explored if online community was just a substitute for poor documentation. When asked if they needed a community, even if the documentation was close to perfect, the general answer was yes. The reasoning behind this was that online communities can answer very specific questions, whereas documentation are more general. Online communities can also answer why you can *not* do something, which a documentation most often will not. Online communities was also stated as being comforting, that it feels safe to know that lots of other developers are using the same thing.

> ”It would lower my thoughts about it. It has to be a really good [platform] to get started without [an online community]. [Online communities] can put the solution in perspective to the question, which a documentation couldn’t do. If you have an area which is very generic, a community could catch that person easier than a documentation.”
> — Development Manager

The architect was the only one who checked online communities before deciding if he wanted to use a software platform. He stated that it was quite hard to get a grip on if a community is useful or not, when you’re not using the platform yet. But he said that reading threads was still helpful, to see what the community was like.

> ”You can get a feeling if you follow the discussion threads and things like that to see that... If most of them end in... like people don’t understand anything; that’s a bad thing. That’s a very important answer [too], that there is no answer.”
> —Architect

Several of the other interviewees stated that they usually just use online communities when they are stuck on something, and only then. The quality architect stated that he didn’t feel the need to check the online community before deciding.

> ”Not if I find good information on the github page or something. Then I don’t have any reason to check out the community.”
> — Quality Architect

> ”No I don’t [check the online community before deciding].”
> — Senior Developer

When finally asked to put concisely what they want from an online community, it was clear that they mostly just want answers and their problems solved easily.

> ”Structured, indexible solutions”
> — Quality Architect

> ”I want answers to my questions, I think that’s it.”
> — Development Manager

> ”Activity and people interacting with each other. And having questions and examples.”
> — Software Engieneer

Takeaways from Online Communities interviews

The takeaway from the interviews around online communities shows a few things. One is that online communities is something primarily used by people when they're stuck and have a specific problem that the documentation may not cover. Where this community exists does not really matter to people, the priority is if the information they need exists or not. Most of them however don't care if *they're* part of the community. It simply acts as a problem solver. They also point out that it is not necessary that the company behind the software is part of the community, but that it doesn't hurt and can build a trust between the company and the user. As much as it is a problem solver for when the documentation doesn't give them the answer, they still want an online community even with a very good documentation. The primary reason for this is that there is always problems and situations that are too specific for a documentation to cover. Online communities are used by the interviewees very often, and is therefor quite important to them. However, documentation is *even more* important.

3.4 Recommendations For Software Platforms

Having presented the results I will now give my personal recommendation for companies who make software platforms. These are my personal recommendation, based on the results that the research yielded. The are presented in alphabetical order.

3.4.1 How often the software is updated

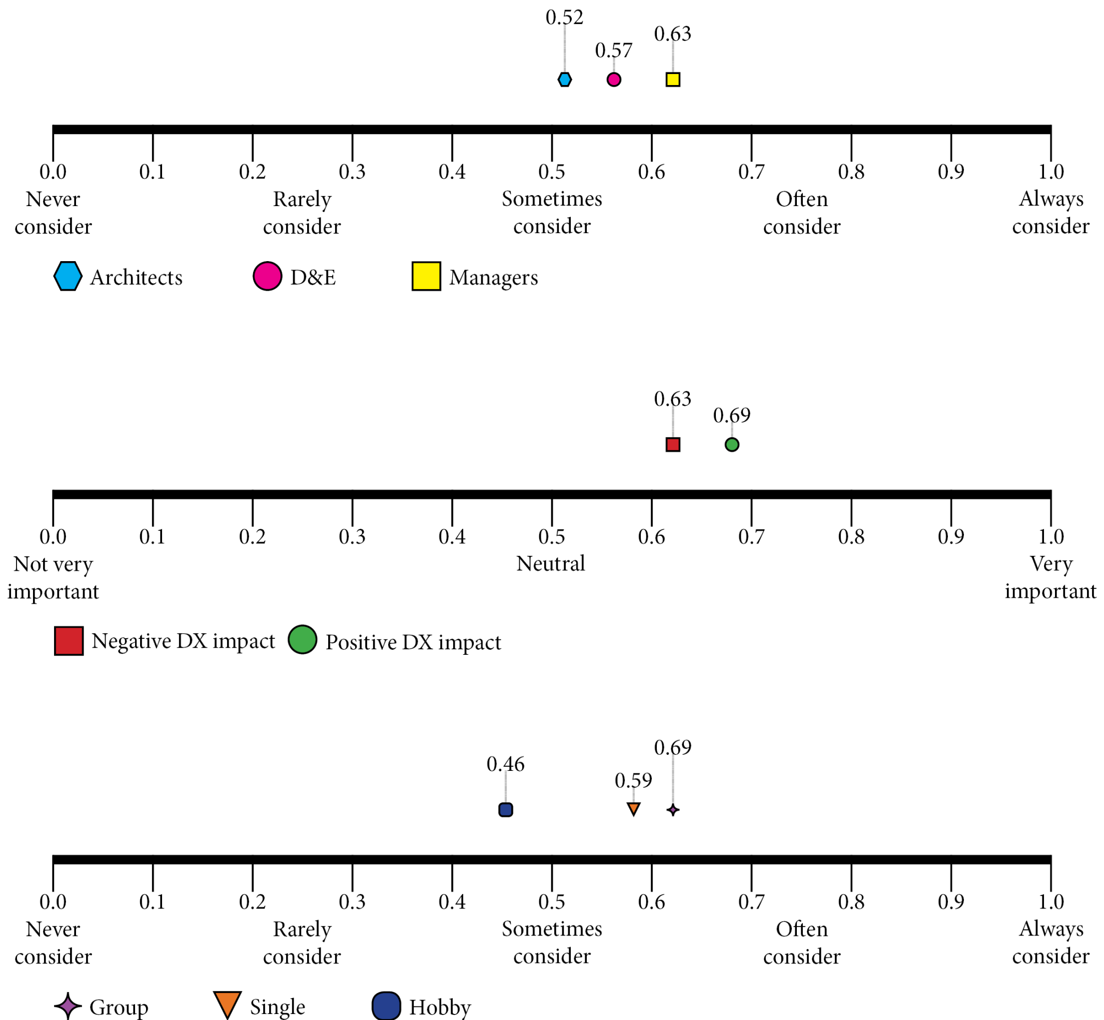


Figure 3.3: Scoring for ”How often the software is updated”

Updating of the software platform is of course important. Bugs need to be fixed, the compatibility for different platforms improved and new features might be added from time to time. In Figure 3.3 we can see how it is ranked. As we can see it is ranking somewhere in the middle. We also see that the DX impact is bigger than the consideration. As mentioned in section 2.3.5, the DX-question and consideration question differed quite a bit for this aspect, and we should be careful to correlate these two. The DX impact shows however that being quick to address bugs has a bigger impact than the negative impact of being slow to address bugs.

The interviews showed that how often the software is updated can be seen as an indicator for how mature the software platform is. Too often and it will scare people away, as it is an indicator that there is a lot of bugs or the software is immature. If the software

platform is updated not often enough it is an indicator that the software platform feels abandoned or not prioritised. The recommendation is to plan your updates carefully, try to lump small updates together into bigger ones, as to not update too often. Exemptions from this is critical bug fixes, such as relating to security or breaking bugs.

Result: Somewhat important. Medium effort, medium payoff.

3.4.2 I can have working code quickly

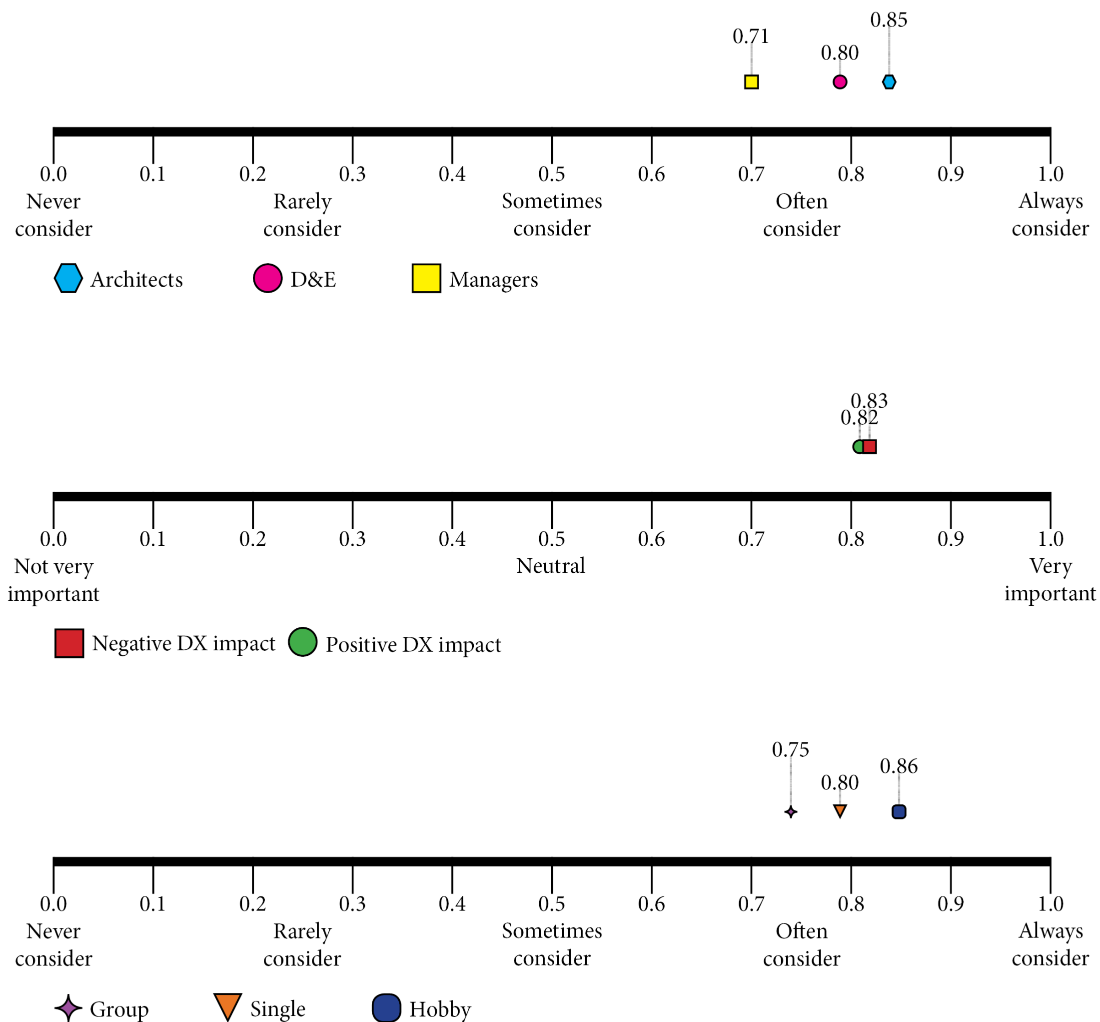


Figure 3.4: Scoring for ”I can have working code quickly”

To have working code quickly has been shown to be important to developers. In Figure 3.4 we can see the scoring. With an average score of 0.80/1.00 for the job titles, this is an important aspect that should always be considered when creating a software platform. There’s also no major difference depending on the context. We also see that it has a strong DX impact. Not only are developers impatient people that want results quickly overall,

working code quickly is developers way to figure out if a software platform is useful. Software developers quickly abandon software if they don't see its value. Because working code quickly is their way of evaluating new software, it is extremely important to be able to provide this. The recommendation is to easily show how to get started. It should both be front and centre when you visit the website, and the example should be easy to follow without being too simple. The effort to have an example that is easy to follow and makes the user understand it can take time, and be a bit of an effort, but is definitely worth it.

Result: Very important, always keep in mind. Medium effort, high payoff.

3.4.3 The API documentation gives thorough explanations on how it works

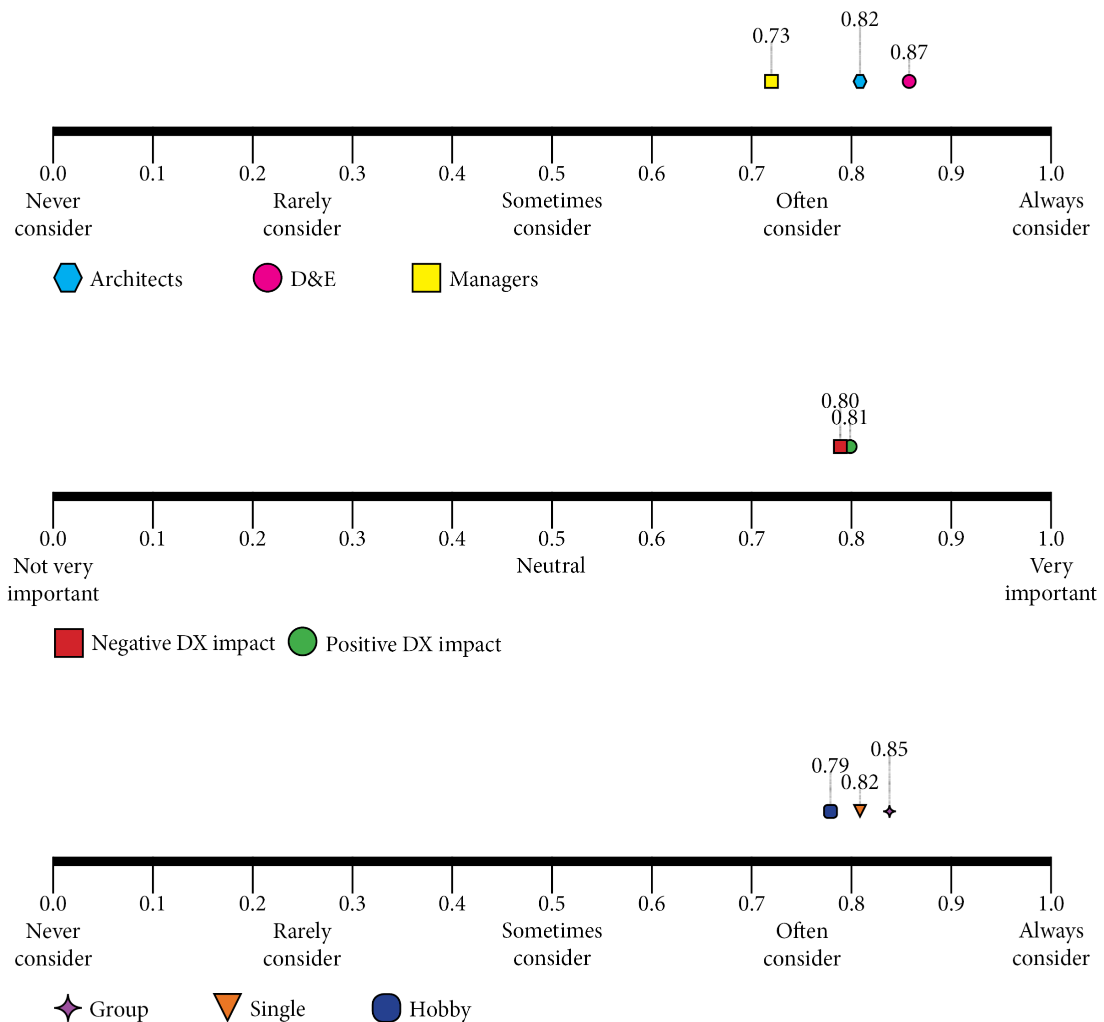


Figure 3.5: Scoring for "The API documentation gives thorough explanations on how it works"

Documentation is the heart of a software platform, providing the explanation of how to use it. In Figure 3.5 we can see how it scored. With an average score of 0.82/1.00 for the job titles it is paramount that this aspect is taken care of. We also see that it is important for all contexts. This is also reflected in the DX impact result, where we see that poor documentation has a strong negative impact, and vice versa. Poor documentation was shown to cause developers to quickly abandon software. It doesn't matter how good your software platform is. If you don't have good documentation that clearly explains how you're suppose to use it, people will not use your platform. You *must* make sure you explain all parts of your platform. The effort to have thorough documentation is big, but is worth it when you see how important it is. While documentation needs to be thorough, it is important to ease the reader into it. Presenting all information at once will overwhelm the reader. Images are often a good way to explain how things are interacting. The recommendation is to put a lot of effort into this. Before going into too much detail, give the reader an overview of what the documentation will convey, and how things are interconnected. User pictures and models for this. Listen carefully to any questions you get from users. If a lot of users find the same things difficult, it can be an indicator that the documentation is not thorough enough. A good method could be to read online discussions, and see what people have difficulty with.

Result: Very important, always keep in mind. High effort, high payoff.

3.4.4 The API has code examples

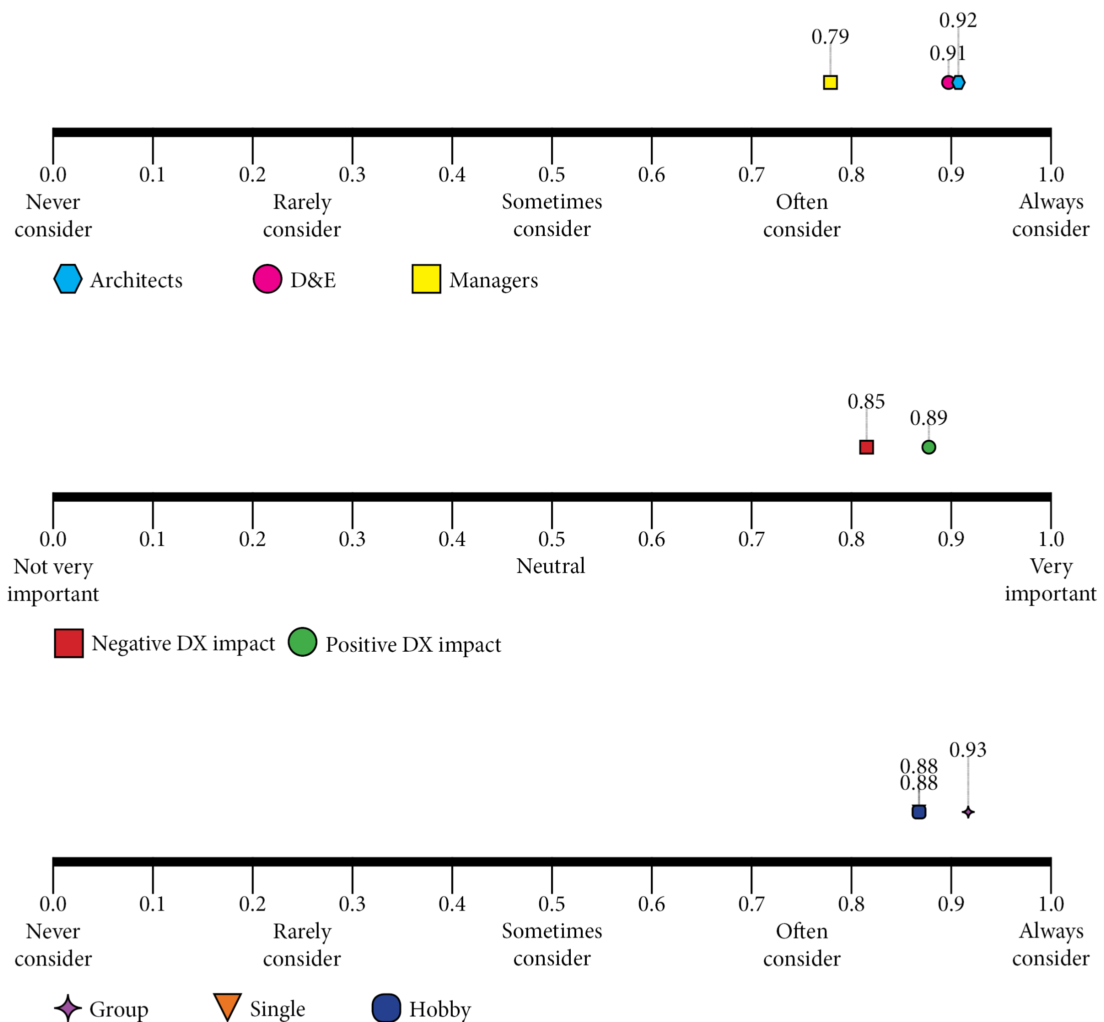


Figure 3.6: Scoring for ”The API has code examples”

You can explain concepts and methods in text, but often an example is the best way to convey something quickly. The interviews showed that examples is the first thing people look at when encountering documentation. It is therefor important that the example is front and centre in documentation. The examples are used for many things too. It is for copy-pasting into people’s projects, getting an overview how things work and are linked together as well as to simply see how things should be used. The effort to construct good examples is quite big. The recommendation is to always have a simple example with all methods and concepts, and if possible more advanced examples too. The simple example should be concise, and show the standard situation. It could be tempting to show something fancy. This however increases the risk for confusion. If you’re going to show advanced situations, do it step-by-step as to not confuse the user. In Figure 3.6 you can see how it scored. With an average score of 0.90/1.00 for the job titles and a strong DX impact it is paramount that

this aspect is taken care of. It is also important in all contexts. It is the first thing users look at to get an overview, a mental model, and if they don't understand you risk losing them.

Result: Very important, always keep in mind. High effort, high payoff.

3.4.5 The documentation doesn't assume any prior expertise

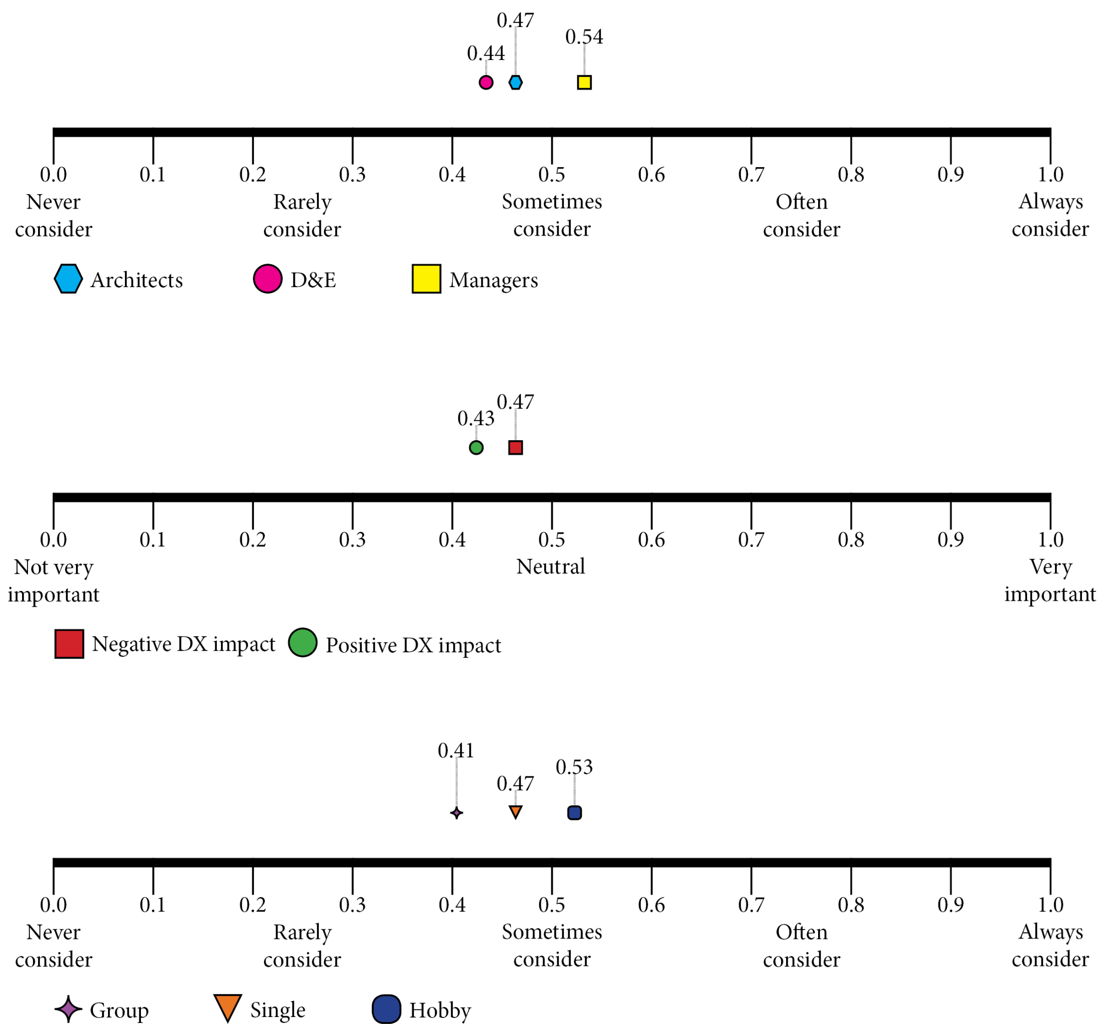


Figure 3.7: Scoring for "The documentation doesn't assume any prior expertise"

Software platforms will naturally have concepts that are new to people. Whenever a new concept is used, you risk confusing the user if it is not explained. In Figure 3.7 you can see that it does not score very high, with an average of 0.47/1.00 for the job titles. It is

the same for the contexts. We also see that the DX impact scores about the same as the consideration questions. This doesn't mean that it can be completely ignored, but it seems developers are not deterred by new concepts. The recommendation is to link to an explanations of new concepts where ever they're used. This effort is not very big, but solves the problem.

Result: Not very important, but don't ignore. Low effort, medium payoff.

3.4.6 The documentation has consistent language

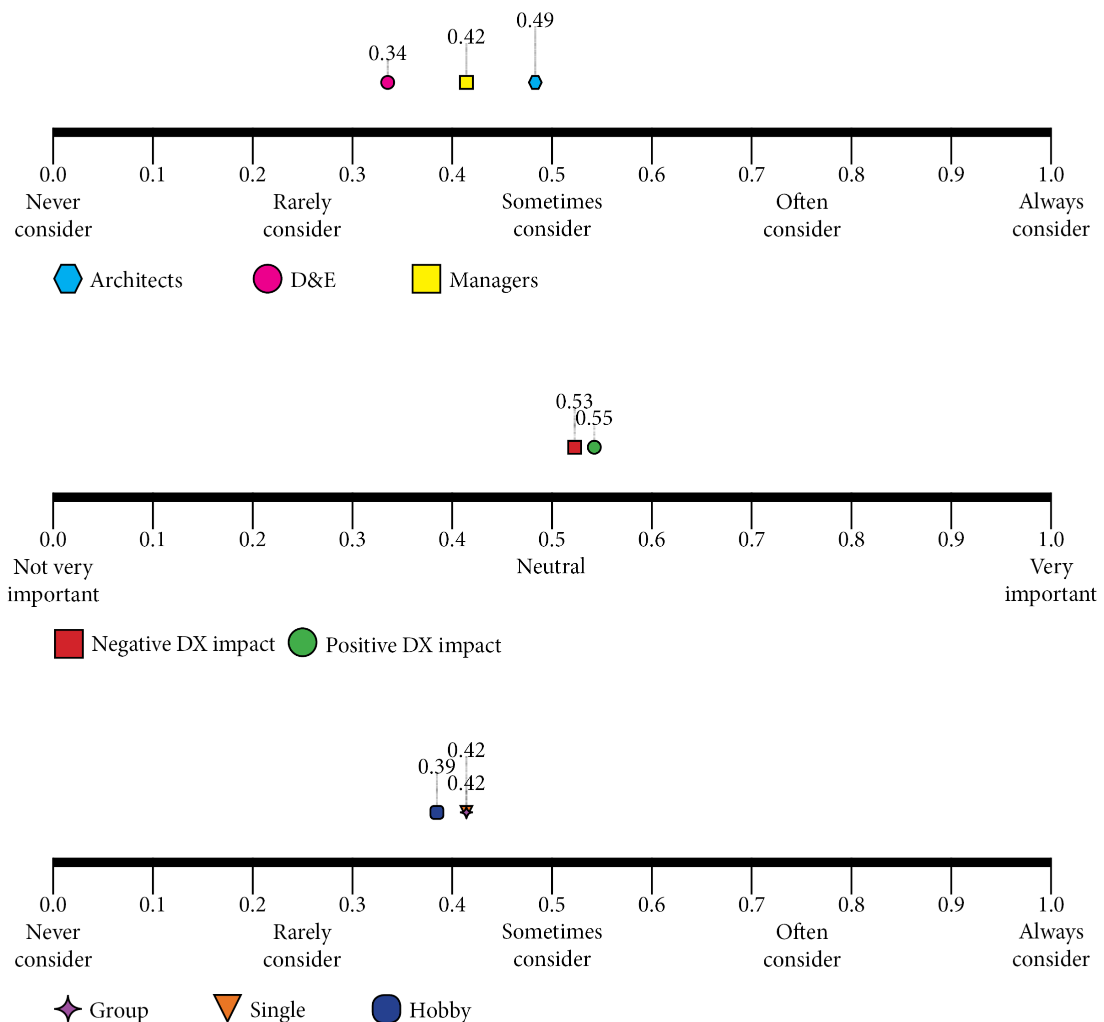


Figure 3.8: Scoring for "The documentation has consistent language"

Having consistent language is an indicator of documentation that has been thoroughly reviewed and worked with. In Figure 3.8 however we can see that it scores quite low, with an average of 0.41/1.00 for the job titles. It is about the same for the contexts. We also see

that although it is not quite often considered, it has a stronger DX impact. It is however still close to neutral. A documentation with inconsistent language is still very much usable, it just increases the risk for confusion. The effort to fix an already inconsistent documentation is can sometimes be very high. The recommendation is to define a vocabulary that should be used by the documentation writers before starting to write documentation, or from now on if the documentation has already been written. This will ensure that new documentation has consistency, as long as the documentation writers make sure to use the vocabulary. Because of the high effort, and low importance according to this research, going through documentation and fixing inconsistency should not be of very high priority.

Result: Not very important, but don't ignore. High effort, low payoff.

3.4.7 The documentation is easy to navigate

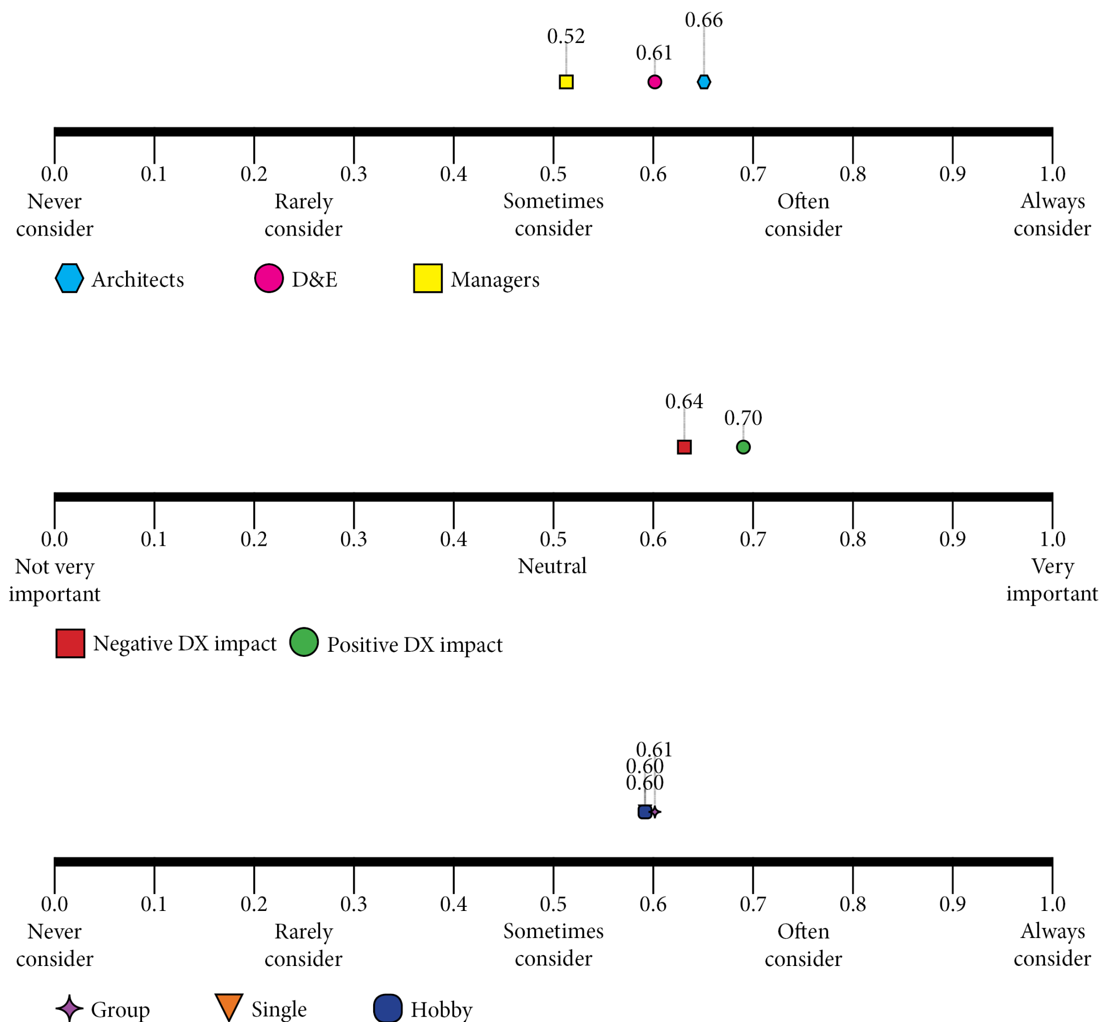


Figure 3.9: Scoring for "The documentation is easy to navigate"

As mentioned before, developers are impatient people. A documentation that is easy to navigate, means developers can find what they're looking for quicker. In Figure 3.9 we see that it scored somewhere in between 'Sometimes consider' and 'Often consider' for both the job titles and the contexts. It also has a DX impact that is above neutral. The recommendation is therefor to have a clear navigation structure, with carefully chosen titles that makes the user understand what each link will lead them to. For larger pages, having a table of contents at the top that shows what sections it contains is also recommended. For smaller sub-pages a short paragraph could be good to explain to the user what he or she can expect to find on the page. This gives the possibility to the user to be more efficient when searching for specific things. The effort to do this can be somewhat high, especially the careful title naming. The titles can easily be too vague or ambivalent. Having an efficient search function for your documentation will also make the navigation more easy. To implement this is a higher effort than structuring the documentation. The recommendation is therefor to start with the structure, and after that work look into the possibilities to create a search function.

Result: Somewhat important, keep in mind. Medium-high effort, medium payoff.

3.4.8 The official website looks professional

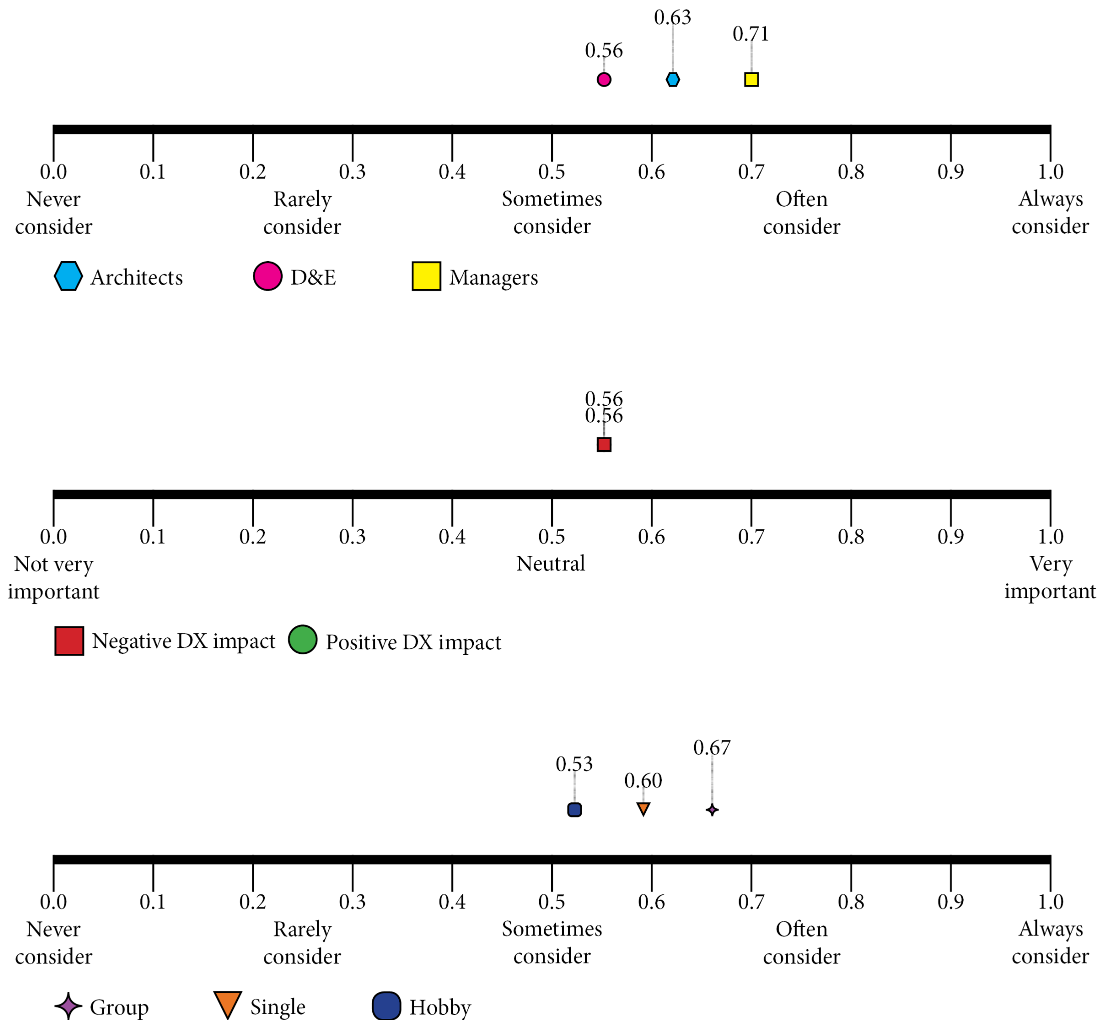


Figure 3.10: Scoring for "The official website looks professional"

The official website is often a company's face outwards, and a user's first encounter with the company. You should not downplay the importance of first impressions. In Figure 3.10 we can see that it scored somewhere in between 'Sometimes consider' and 'Often consider' with the average score of 0.60/1.00 for the job titles. It is the same for the contexts. The DX-impact for a good or bad website is however neutral. The recommendation is to spend some time to make sure your official website looks professional. Test it for different browsers and devices. The effort to do this should not be high for a software company who creates software platforms. If a company whom provides software platform solutions cannot provide a professional looking website, the impression it gives is that they won't provide a good platform either.

Result: Somewhat important, keep in mind. Medium effort, medium payoff.

3.4.9 The pricing of the software

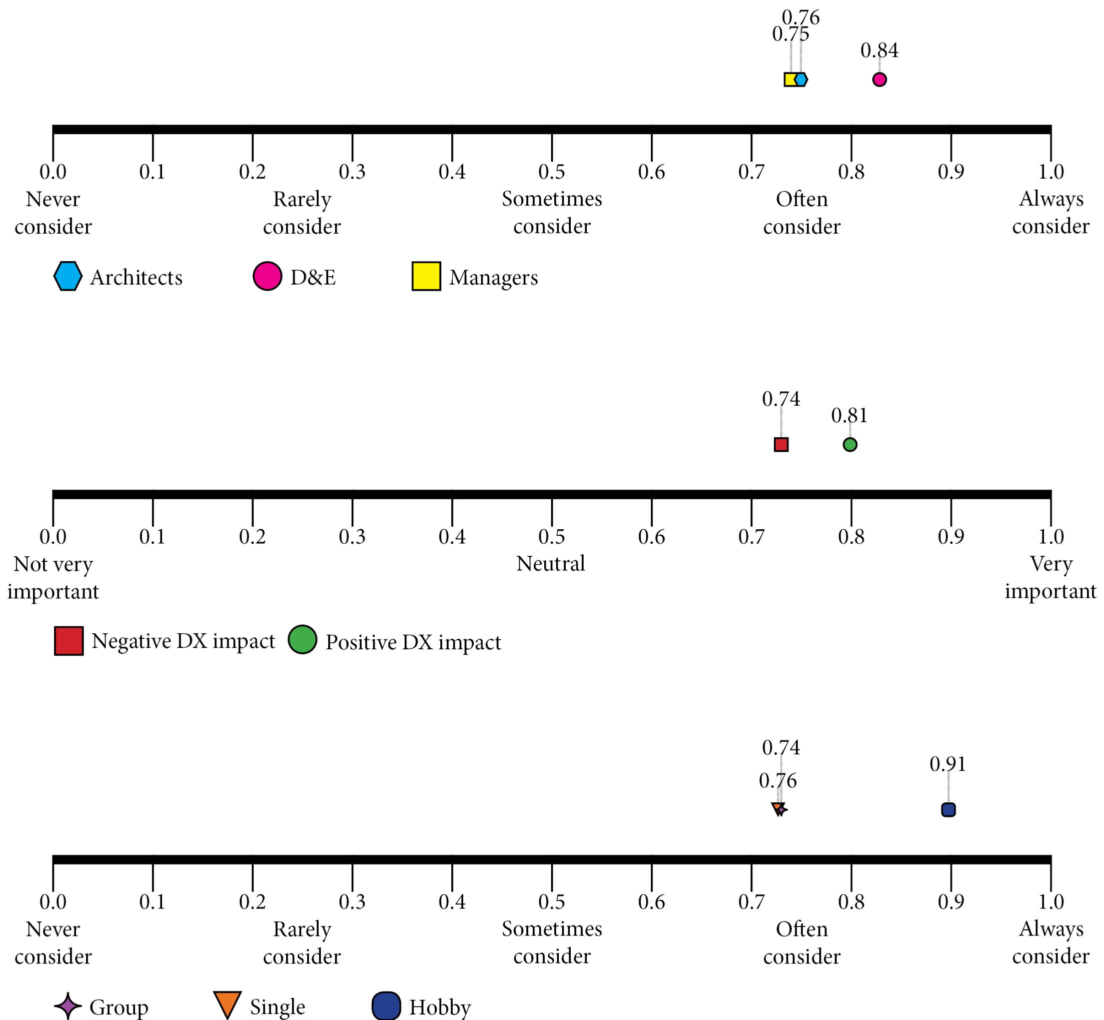


Figure 3.11: Scoring for ”The pricing of the software”

Money always plays a roll in any company, and the price of a software cannot be ignored. You can make money from your software in many ways, through licensing, subscriptions, support contract, and more. In Figure 3.11 we see that it is quite important to people. It is especially important for people working on hobby projects. This is another aspect, as mentioned in section 2.3.5, where the DX-questions and considerations questions are a bit different, and one should be careful to draw parallels. We can see however in the consideration scoring that pricing is very important. We can tell by the DX-scoring that clearly showing the pricing of the software has a very strong positive impact, and hiding the pricing away has a strong negative impact. The recommendation for pricing, in order to give a good experience for developers, is to clearly state the pricing of the software platform. Especially if your target audience is people working on hobby projects. Whichever money making model the software platform uses, the cost to inquire it for a developer should not

be hidden away. Hiding the pricing can not only cause irritation from the developer, but also hurt the trust between the customer and the company.

Result: Important, keep in mind. Low effort, high payoff.

3.4.10 The release- and change notes are thorough

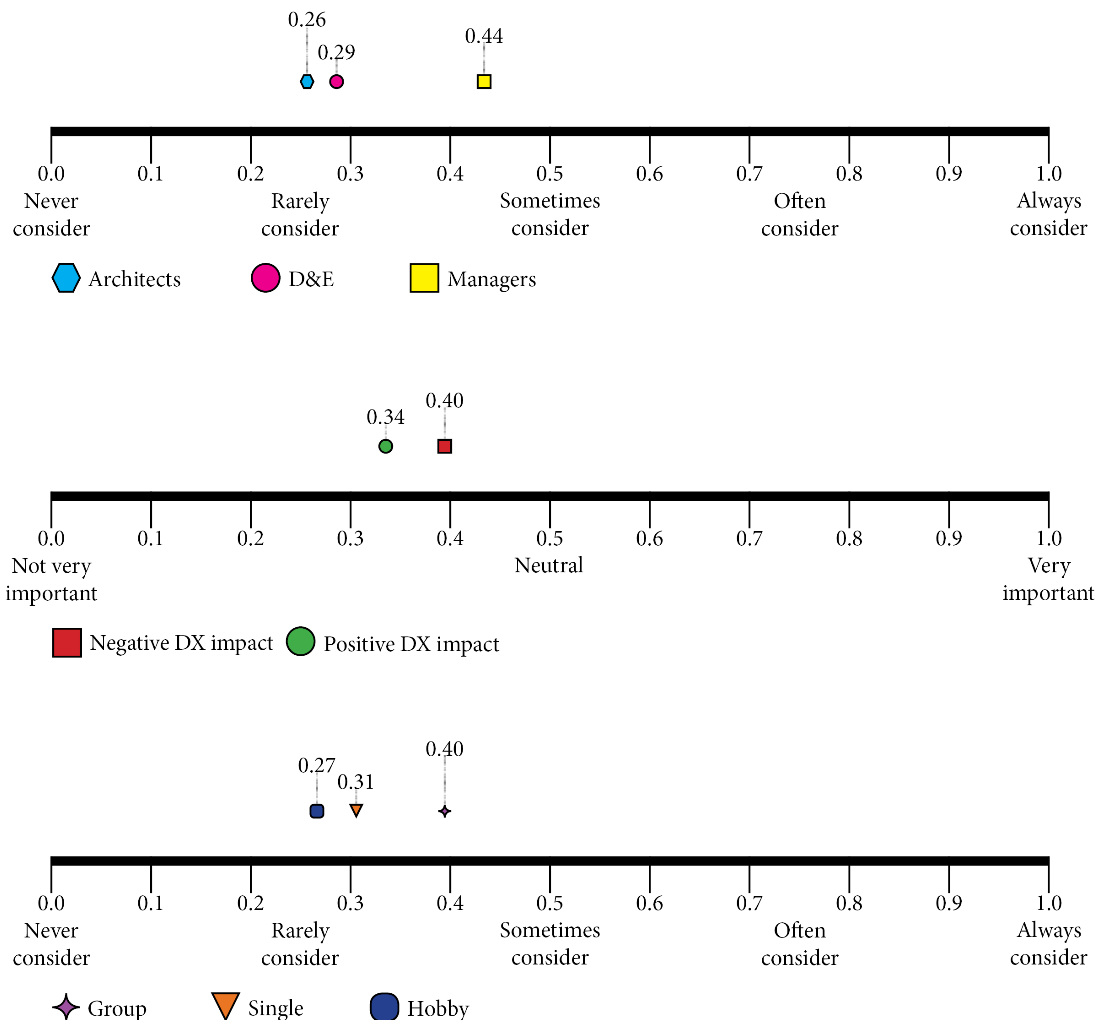


Figure 3.12: Scoring for ”The release- and change notes are thorough”

Contrary to what litterature would suggest, release notes was shown to be a quite non-important aspect, as seen in Figure 3.12. Especially for the people who are supposedly using them: developers and architects. We can see by the DX-impact that poor release notes has a bit of a negative impact. It is however still below neutral. The interviews showed that release notes are a last resort for people, and ignored as much as possible. The recommendation is therefor to not spend too much time on release notes. But make

sure to not ignore them since it reflects poorly on the company, according to the interviews. If you make sure to have good commit messages, these can act as a good base for writing the release notes. This minimises the time that has to be spent on writing the release notes.

Result: Not important, but don't ignore. Medium effort, low payoff.

3.4.11 The software has the same features on all different platforms

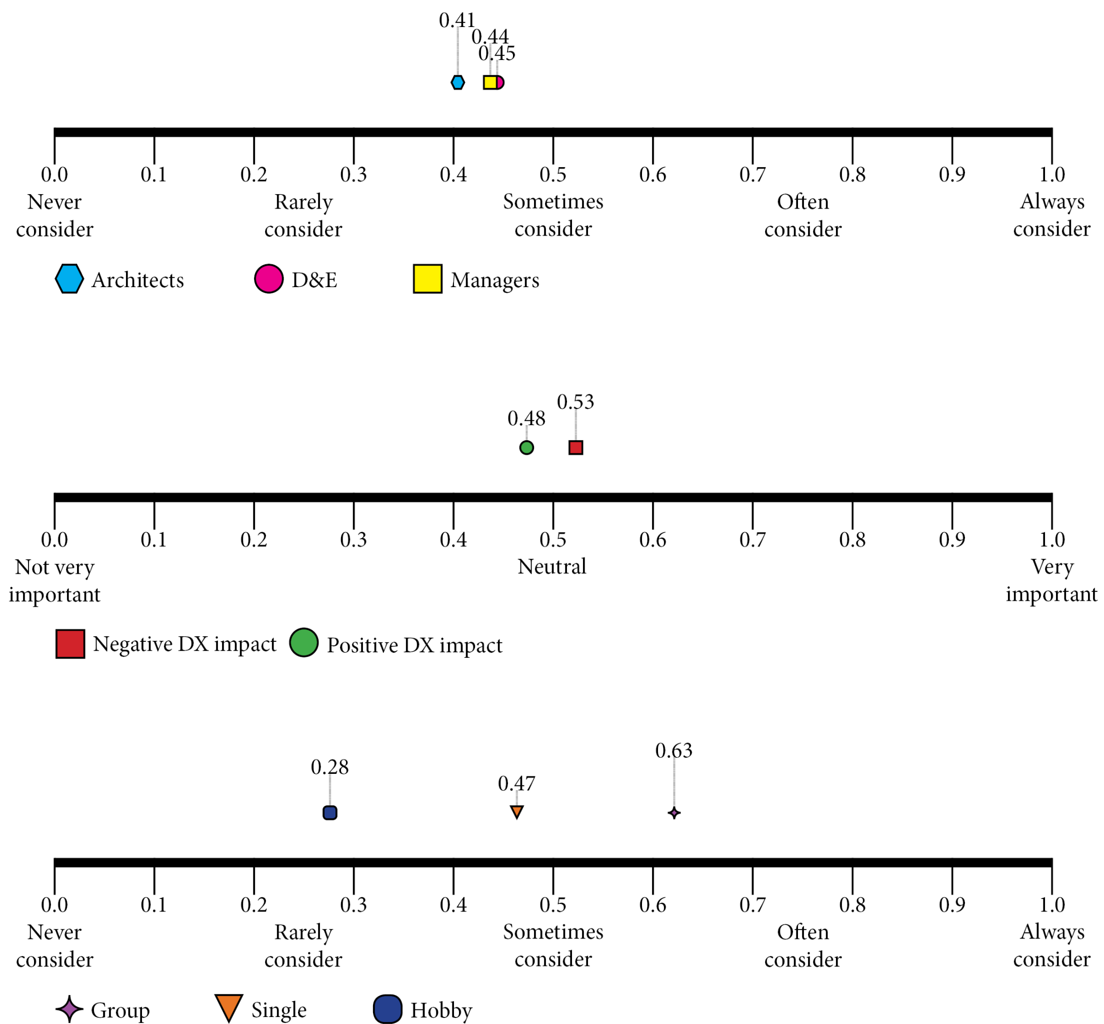


Figure 3.13: Scoring for "The software has the same features on all different platforms"

If your software platform supports many different platforms, such as operatingf systems and browsers, it takes time and effort to make sure that they work the same in all situations. In Figure 3.13 we see that this scores kind of low, and the DX-impact is neutral. It

is somewhat higher in the context of deciding for a group. The effort to make sure that all features exist on all platforms is very high. The recommendation is therefor to clearly state to the users what features are offered on what platforms. If you decide to make sure that the features are consistent on all platforms, be aware that the effort to do this is very high and may not have a big payoff.

Result: Not very important. High effort, low-medium payoff.

3.4.12 The software is compatible with different platforms

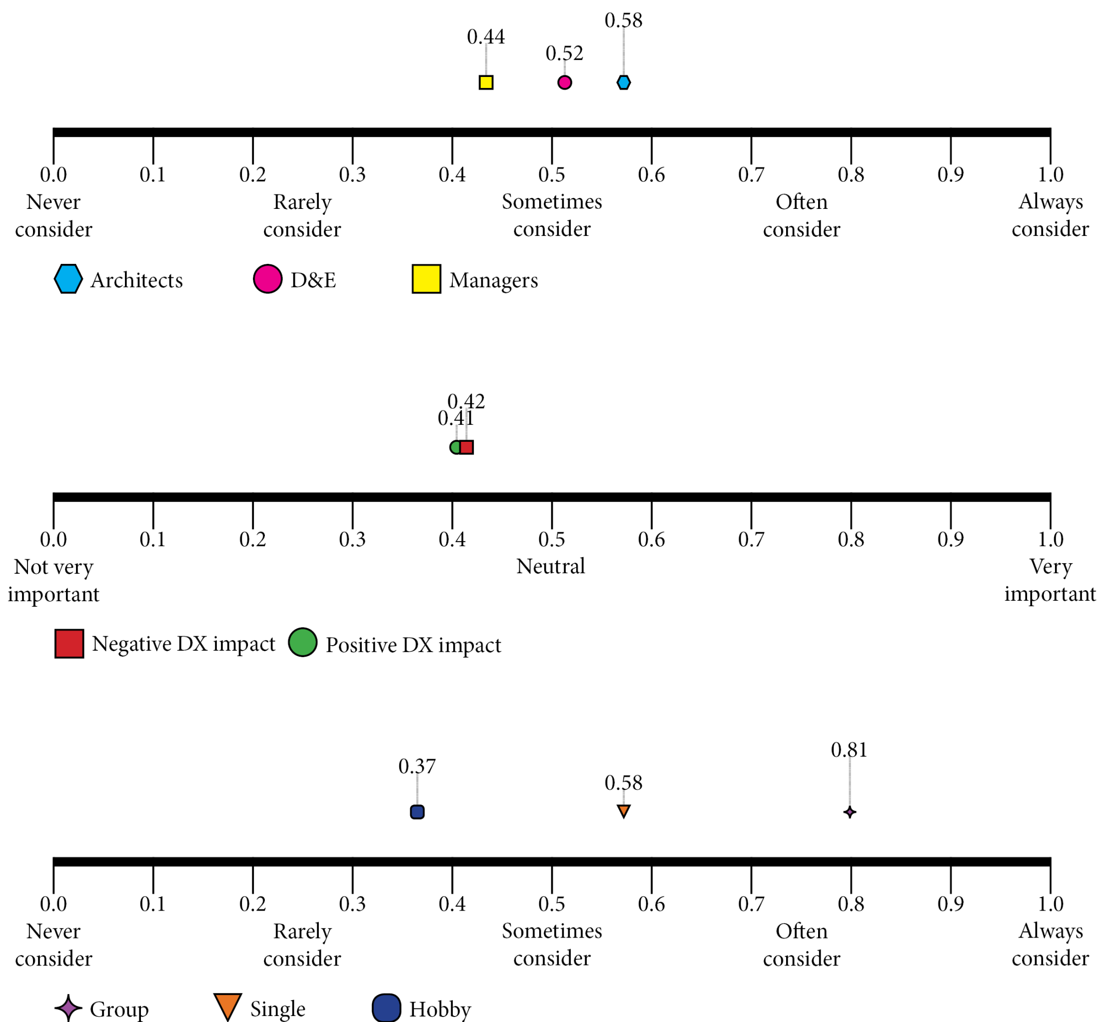


Figure 3.14: Scoring for ”The software is compatible with different platforms”

Having your software platform be compatible with many different platforms is a requirement for some users, a bonus for some. The effort to do this is however high. If you plan to

implement this, make sure that the increase in potential users is worth the effort to do this. You need to be aware that it highly increases all future maintenance for your platform. As we can see in Figure 3.14, the scoring is low to average for both job titles and DX impact. The one outlier is in the context of deciding for a group, where it is very important. The recommendation is to skip this aspect, if you are not sure that the increase in users is worth the effort.

Result: Not very important. High effort, low-medium payoff.

3.4.13 The software is offered in more than one programming language

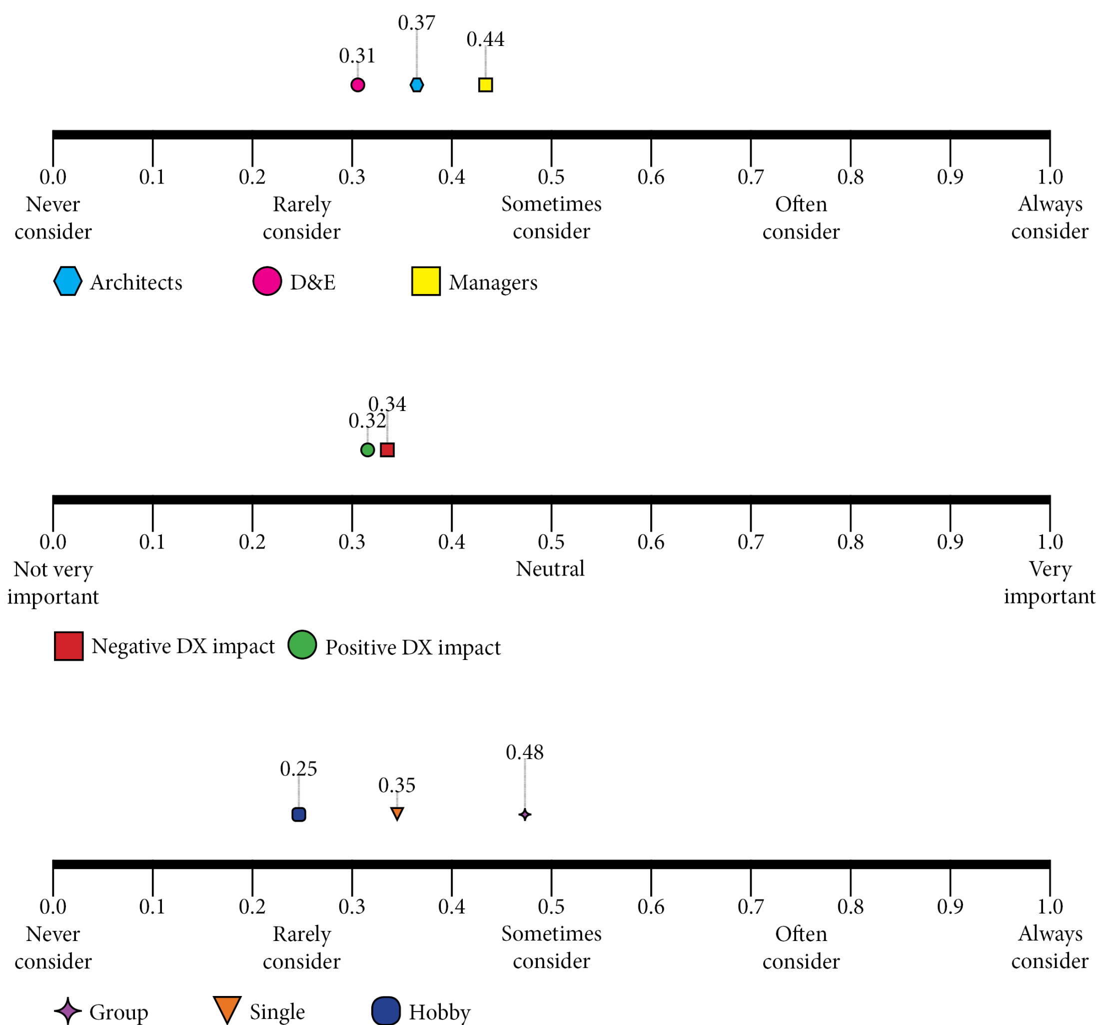


Figure 3.15: Scoring for ”The software is offered in more than one programming language”

Having your platform be compatible with more than one language takes a big effort. It takes a long time to develop and doubles the maintenance needed. As we can see in Figure 3.15 it is not ranked high at all, both for the consideration part and the DX part. The recommendation is therefor to ignore this aspect. The effort compared with the payoff is not worth it.

Result: Not important. High effort, low payoff.

3.4.14 The software is open source

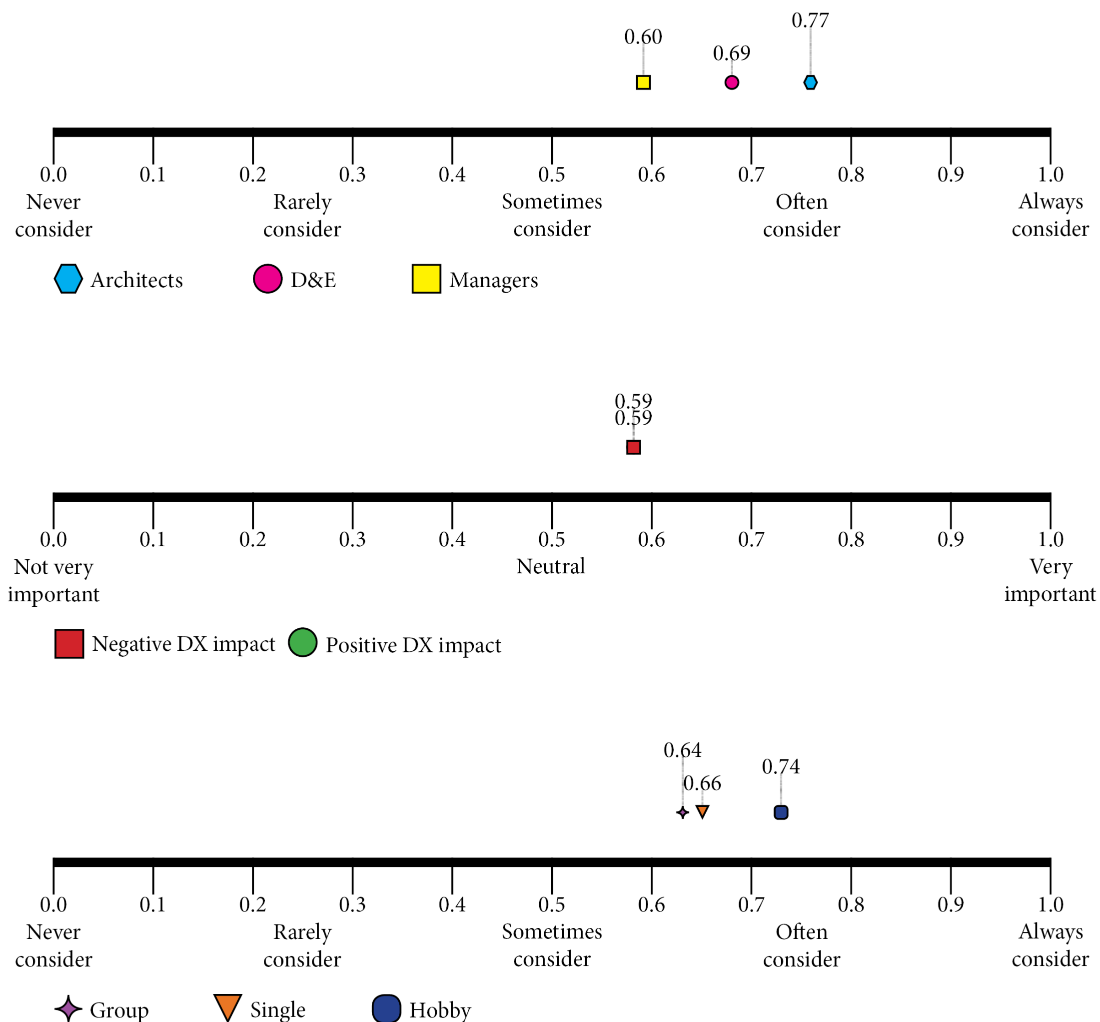


Figure 3.16: Scoring for "The software is open source"

Being open source has both benefits and limitations. It is out of the scope of this research paper to evaluate these. But as we can see in Figure 3.16 it is quite important to developers and architects. The DX impact shows however that being close-sourced doesn't have a very big impact. The recommendation is therefor to try to be open source if you can,

but it is not necessary. If it is not something that the company is used to, be aware that it requires other ways of working than conventional software that is close-source. If you decide to make an open-source software platform, make sure you have the expertise in the company to be able to handle this type of software. It has a potential to increase the amount of users, but if you don't use the benefits that open-source gives, it may not pay off.

Result: Important. Medium-high effort, high payoff.

3.4.15 The software uses the programming language I am most comfortable with

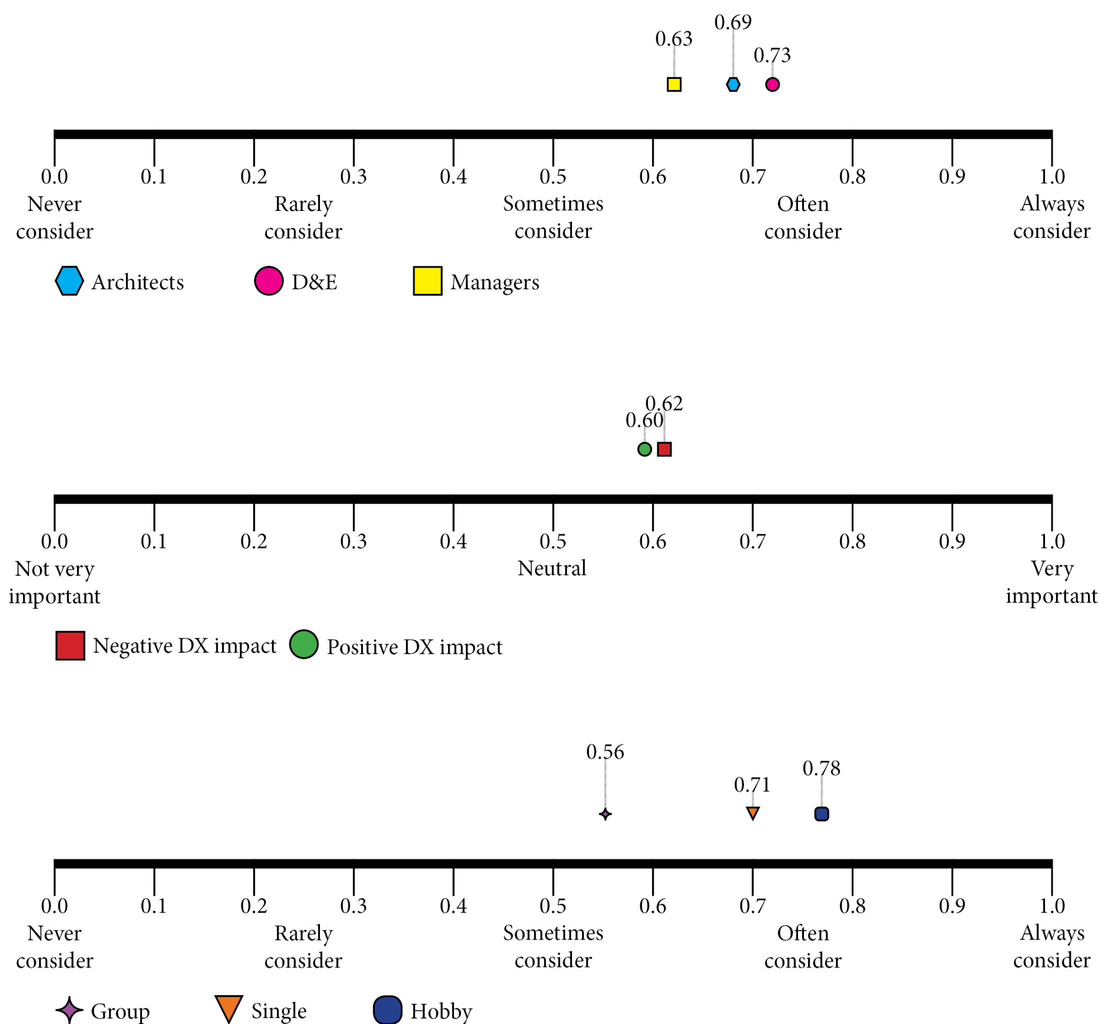


Figure 3.17: Scoring for "The software uses the programming language I am most comfortable with"

Developers tend to be more comfortable working with certain languages, and it is unique for every person. As we can see in Figure 3.17 it is definitely something people consider.

The DX impact shows however that they're not immediately deterred by software platforms that is not in their favourite language. As stated before, providing several languages is not worth the effort. The recommendation is therefor to be attentive to what programming languages are popular, and to see how the trends change. Be aware however that these trends are just that, trends. New languages emerge all the time as "the new hot thing". The safer bet is to offer your software platform in a popular language, that is predicted to be popular for quite some time.

Result: Important. Medium effort, high payoff.

3.4.16 There exists an active online community around the software

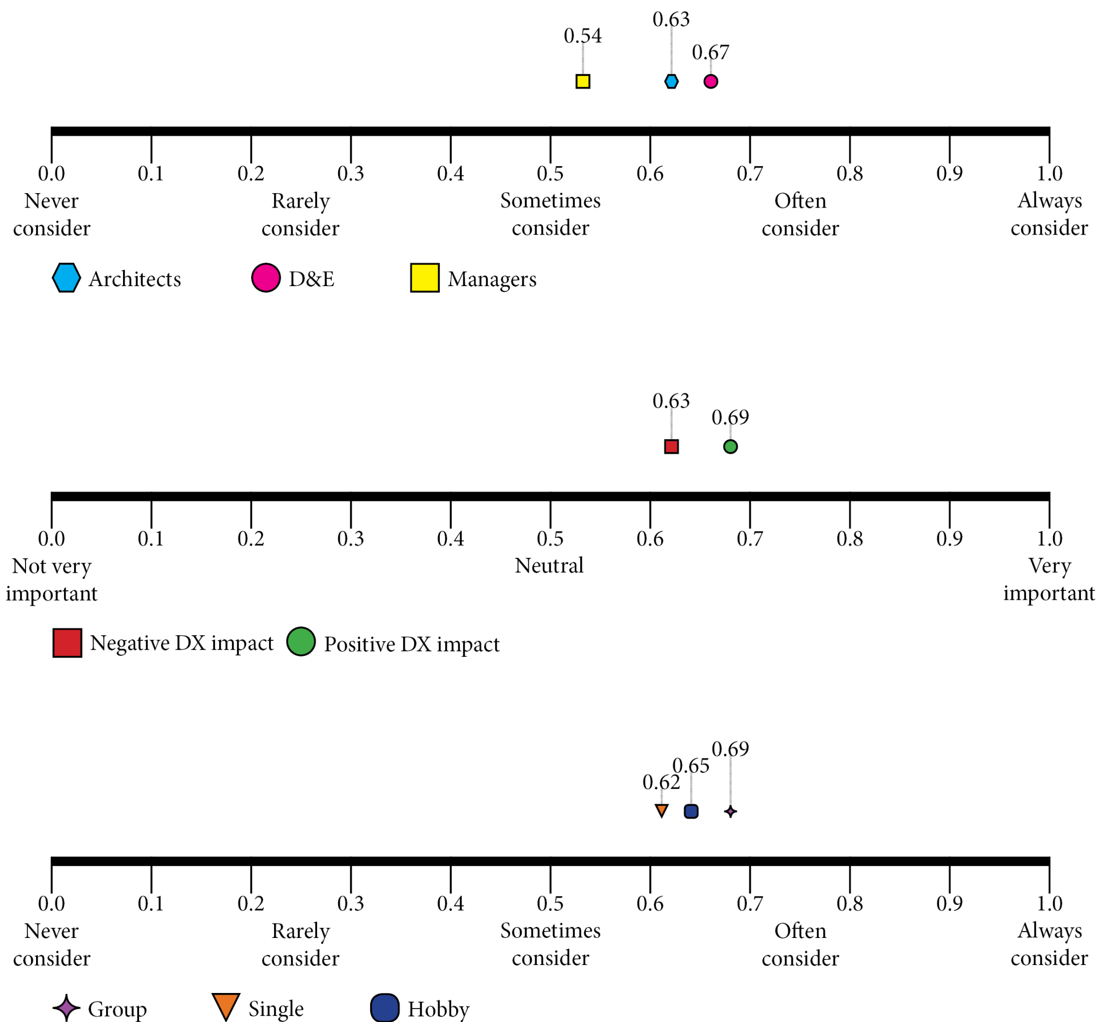


Figure 3.18: Scoring for "There exists an active online community around the software"

It can be difficult, if not impossible, to forcefully create an online community around your software platform. It is not up to the company if people engage in online discussions, it is up to the users themselves. A company can provide platforms for discussions, but also rely on existing community platforms. In Figure 3.18 we see that it is quite often considered, and a good community has above neutral DX impact on developers.

The effort to create an online forum for users to discuss doesn't have to be too big, but acts as a starting point for a community to be created. Often communities will naturally emerge on other places, such as Stack Overflow. The company can help the online community prosper by keeping an eye on these communities as well, and answer question there. This also builds a trust between the user and the company.

Online communities is primarily used for questions and answers to help out when the documentation is not enough. The effort to make sure your online community has the answers they need is somewhat big. The recommendation is to take extra care of your online community when it is new and small, and less when it is big enough that it takes care of itself. Never ignore it though, pay attention to suggestions and issues that your online community has.

Result: Important, keep in mind. Medium effort, medium-high payoff.

3.4.17 Summary of Recommendations

A summary of the relevance, effort and payoff can be found in Table 3.15

Aspect ID	**Relevance**	**Effort**	**Payoff**
1	Somewhat Important	Medium	Medium
2	Very Important	Medium	High
3	Very Important	High	High
4	Very Important	High	High
5	Not very important	Low	Medium
6	Not very important	High	Low
7	Somewhat important	Medium-high	Medium
8	Somewhat important	Medium	Medium
9	Important	Low	High
10	Not important	Medium	Low
11	Not very important	High	Low-medium
12	Not very important	High	Low-medium
13	Not important	High	Low
14	Important	Medium-High	High
15	Important	Medium	High
16	Important	Medium	Medium-High

Table 3.15: Summary of the relevance, effort and payoff for each aspect. The aspect IDs can be seen in Table 2.1

3.5 Evaluation of Qlik Core

The last part of this report looks at the evaluation of how well Qlik Core followed the recommendations given in section 3.4. This was done aspect-by-aspect, and given a verdict based on how important the aspect is compared with how well they follow the recommendation.

3.5.1 How often the software is updated

Qlik Core is a quite new product, which left beta in mid-fall of 2018. The different parts of QC are updated continuously (several commits on GitHub per week for each component of QC). Qlik releases a monthly blog post about what has been updated the last month. At the time of writing this paper, they've made four such blog posts. With this being such a new product, and having so many components, it is natural that there will be a lot of updates. The continuous updating may deter users as it shows that this product is still quite immature, but can also be seen by some that this product is very much taken care of.
Effort: Medium. Payoff: Medium. Importance: Somewhat important.
Verdict: No need for immediate change. It is not feasible for QC to lump the updates together when the product is still so new. But Qlik should be aware of what issues the almost daily updates can be.

3.5.2 I can have working code quickly

When evaluating this aspect, I tried to visit the QC website as if I were a new user. I took the persona of myself, a junior software developer, that had heard of QC and wanted to check out what it was, and what it can do. The following text is the process of doing so, and what issues I had.

Qlik Core provides several tutorials for a user to do when getting started. The first tutorial called 'Hello-Engine' simply sets up the possibility to communicate with the engine. It was not clearly explained in the tutorial what the goal of the tutorial was, so when I finished I did not fully understand what I had finished. I thought that I had done something wrong, my thoughts were "This can't be it?" when I had actually completed the tutorial. Overall I had a lack of achievement, and did not feel I had learned anything about what Qlik Core is or can do. The following tutorial 'Hello Data' suffered a bit from the same issues. The third tutorial 'Hello Visualisation' was the first tutorial where I felt I had learned something about what Qlik Core can do. A tutorial should give the user a sense of achievement, and make the user feel like he or she learned something they did not previously know. In my opinion, these tutorials could be merged into one longer tutorial, divided into three sections.

For the next tutorial 'core-orchestration' I got stuck immediately. The tutorial prompts you to get a licence. The process of figuring out how to configure the license was extremely cumbersome. I had to search for a long, long time on how I should do it. In the end, I had to follow a total of five links away from the tutorial page until I found an ex-

ample, inside a file in a GitHub project. Afterwards, when I talked to the people whom had created the tutorial, it turns out that getting a license was only optional for the tutorial, even though it was stated that it was needed.

The 'core-orchestration' tutorial also has a lot of prerequisites on knowledge. I still tried to follow the guide, which ultimately did not work. When talking to the Qlik Core team, it turns out that this is a very special case for using Qlik Core, and will not be used by most people. The way it is presented, right after the simplest tutorials, is problematic. This will scare users away. It needs to be explained that this is a special and advanced case.

The fourth tutorial was about authorisation. This was also a special, advanced case which did not teach the user about what Qlik Core can do. It was also hard to follow, with explanations that were brief and hard to understand.

The last tutorial is related to data loading. These tutorials did not require as much prerequisite knowledge. It however failed as well. When talking to the tutorial constructor, it turns out that the tutorial was out date with what commands should be run. Ultimately, this tutorial also failed to be completed.

Overall, getting started felt difficult, cumbersome and irritating. I was stopped by errors, getting a licence that I did not actually need, setting up configurations and out-of-date tutorials. I also felt like I was was not actually learning anything about what Qlik Core can do. The tutorials need to be reworked in my opinion, with more simple use cases. And most importantly, tutorials where the user actually gets to change some code and feel like they learn something, rather than running downloaded scripts. The goal when making the tutorials should be "How do we get the users interested in Qlik Core, and make them feel like they are able to use it."

Effort: Medium. Payoff: High. Importance: Important.
Verdict: Needs to be changed. The tutorials needs to be both reworked, and probably extended with more simple examples. The tutorial needs to be created with the goal of getting new users to understand how Qlik Core works, what it can do, and show that them that it solves problems in an easy and efficient way.

3.5.3 The API documentation gives thorough explanations on how it works

Qlik Core is a group of different services running together. How these are linked together, is nowhere explained.
On the QC website one can find the API documentation. It turns out that the documentation there however, is not directly under the Qlik Core teams control. It is pulled from other sources and automatically generated. In the end however, it is the Qlik Core team that is responsible for what is on their website.

At the heart of it all is the QAE, which uses several different APIs, as stated in section

1.5. In the introduction to the QAE it is explained that the different APIs are meant to be used in different use cases, and that each of these APIs has their own API documentation. An issue that exists in all documentation is that there are close to no examples.

QIX API Documentation
For their own QIX API has many issues. The documentation has been split up into several different sub-categories. They are ”Definitions”, ”Global”, ”Doc”, ”Generic Object”, ”Generic Bookmark”, ”Generic Dimension”, ”Generic Measure” and ”Generic Variable”. None of these eight sub-pages have an introduction or description of what the page contains.
The sub-page ”Definitions” is the one that is lacking the most. The definitions page is simply just a long list of a total of 217 definitions. In Figure 3.19 we can see how an entry in ”Definitions” typically looks like.

AppEntry

No description.

Name	Type	Description
qID	string	Identifier of the app.
qTitle	string	Title of the app.
qPath	string	Path of the app.
qLastReloadTime	string	Last reload time of the app.
qReadOnly	boolean	Is set to true if the app is read-only.
qMeta	NxMeta	Meta data.
qThumbnail	StaticContentUrl	App thumbnail.
qFileSize	integer	*No description.*

Figure 3.19: An example of how a method for the ”Definitions” part of QIX API looks like. It has no description for the entry, and one of the fields has no description as well.

This is not an outlier. 36 fields had *No description*, and out of the 217 entries, only 60 has a text explaining what the definition is or does. This does not follow the recommendations given in this paper, and need to be taken care of as soon as possible.
For the other API sub-pages of QIX, most of the entries has descriptions, with just a few exceptions.

REST API Documentation
The REST API seems like the most mature of the four APIs. It has description for all parts, with parameters and responses defined. It also states what permissions are needed. In Figure 3.20 we can see what an entry in the REST API documentation typically looks like.

GET /v1/apps/{appId}/data/metadata

Retrieves the data model and reload statistics metadata of an app.

An empty metadata structure is returned if the metadata is not available in the app.

Required permissions: read

Metadata	Value
Stability Index	Experimental
Produces	application/json

Parameters

Parameter	In	Type	Mandatory	Description
appId	path	string	true	Identifier of the app.

Responses

Status	Description	Type
200	OK	DataModelMetadata

Figure 3.20: An example of how an entry in the REST API documentation typically looks like.

At the top of the page however is a link called ”Qlik Associative Engine API specification”, which is broken. This immediately gives the user a feeling of an immature documentation.

Data Connector API Documentation

In the documentation for Data Connector we once again have the issue of missing descriptions. Its lack of descriptions is however not as bad as the QIX documentation. It is also the only documentation that has a figure that explains how it is interacting with other services.

Analytical Connector API Documentation

This documentation is on par with the REST API documentation in how mature it is. It only has one field without a description. It is however, just like all other documentation, lacking an intro and an explanation on where and when it is suitable to use this.

License Documentation

As discussed in 3.5.2, the licensing caused a lot of confusion. How to actually set up the licensing is not really explained on the page. In Figure 3.21 we can see how they explain of how to setup your license. What a `<License service URL>` should look like, is not explained. The best way to fix it was to open an example in GitHub and look in the configuration file.

Configuration

You need to configure the License service with two environment variables `LICENSES_SERIAL_NBR` with your LEF serial number and `LICENSES_CONTROL_NBR` with your LEF control number.

You also need to configure Qlik Associative Engine where to find it. You can do this by passing the following command line argument to the Qlik Associative Engine.

```
-S LicenseServiceUrl=<License service URL>
```

Figure 3.21: The description on how to pass the licensing to the Qlik Associative Engine.

Effort: High. Payoff: High. Importance: Very important.
Verdict: Needs to be changed. Overall, the documentation feels overwhelming. It is hard to get a grip on when, where and how the APIs are supposed to be used. The documentation is not really explained at all. An introduction, with images, explaining how things are interconnected would be a great way to start. The licensing page needs rework too.

3.5.4 The API has code examples

The APIs have close to no examples at all in the documentation. The few examples that exists are in the introduction part, and are snippets without any context. The recommendation for this aspect is to always have small examples in the documentation, and preferably longer examples as well. As stated in 3.4.4, code examples are used for many things: getting an overview as to understand the platform, getting started and seeing how code is used. Even more alarming, the tutorials that is suppose to get people started is without examples. They simply rely on the user opening and project in GitHub and reading through it. The issue with this is that examples are not only used for having runnable code, but to get a mental image of what something does, interacts with other things and works. API examples is the most important aspect according to this research paper, and the lack of them can been felt heavily. It is hard to get an understandable overview, hard to get started and hard to see how the APIs should be used.

Effort: High. Payoff: High. Importance: Very important.
Verdict: Needs to be changed. The software platform has almost no examples at all, which makes it hard to use in a lot of ways. This needs to be of very high priority to fix.

3.5.5 The documentation does not assume any prior expertise

Qlik states that this documentation requires you to know some things before you begin. Most central is the third-party software Docker. By doing this, Qlik dismiss the responsibility to explain how these things work. However, it feels like the documentation take *too* little responsibility in explaining new concepts. The documentation also uses some

unexplained expressions. One example of this is the term to do something ”On The Fly” that encountered twice, but it was never explained what exactly it means.
Their way of dealing with new concepts however follows the recommendations of linking to places where an explanation is given. Qlik should be aware that the threshold of starting with Qlik Core is high for a new person because of the high requirements in prerequisite knowledge.

Effort: Low. Payoff: Medium. Importance: Not Very Important
Verdict: No need for change. If Qlik Core gives more examples and gets better at explaining its documentation, this aspect does not need fixing. It follows the recommendations. Combined with this aspect's importance being low, this aspects gets a pass.

3.5.6 The documentation has consistent language

As mentioned in 3.5.3, the Qlik Core team are not directly responsible for the documentation for the APIs. When contacting the people responsible for the different parts, it turns out that there is no central vocabulary used, as is recommended. When reading through the documentation, it does however not seem inconsistent in its wording. The effort to produce a vocabulary to be used by many different teams is high, and inconsistency in the language does not seem to be an issue.

Effort: High. Payoff: Low. Importance: Not very important
Verdict: No need for change. The recommendation is to use a central vocabulary. The language is however not inconsitent without it, and because of the high effort to implement it, this gets a pass anyway.

3.5.7 The documentation is easy to navigate

There are a few issues with the navigation for the documentation. The first is that on the start page, see Figure **??**, there is no clear way of how to reach the API documentation. It turns out that the documentation is hidden in ”Services”.

The recommendation is to carefully choose your titles. API documentation being found inside Services is not very obvious, in my opinion. The sub-pages also have some very vague names. The services' APIs are quite big. For example, the QIX Definitions part having 217 entries in just one long list. Navigating the API documentation feels cumbersome and overwhelming. Restructuring this into smaller sections, with aptly named headings would make it easier.
The Qlik Core website has a search function, which searches both their own documentation and third-party software documentation. The search function however is limited. It does not allow you to search for exact phrases, normally done by putting a phrase within citation marks. You cannot tell the search to *not* include certain words or phrases, normally done by adding a minus before the word or phrase. In the end, it is probably easier to use a search engine like Google, than Qlik Core's own search function.

Effort: Medium-High. Payoff: Medium. Importance: Somewhat Important

Verdict: Needs to be changed. The API reference documentation is hard to find, somewhat hard to navigate and the search function is lacking functionality.

3.5.8 The official website looks professional

This aspect is hard to evaluate objectively, since it is up to each user's opinion. According to me, the website does look professional. It is using standard conversions, doesn't have spelling errors or buggy CSS. It works well on mobile as well.

Effort: Medium. Payoff: Medium. Importance: Somewhat important.
Verdict: No need for change.

3.5.9 The pricing of the software

Finding the pricing of Qlik Core is not possible. Going to the licensing page, you can easily try the software for free for a period of time. If you want to obtain an actual license however, you have to contact the sales department of Qlik. They don't give any information of any kind of price range or if it is a monthly or yearly (or any other) payment. Any information you want about pricing, you have to contact the sales department. This clearly goes against my recommendations.

Effort: Low. Payoff: High. Importance: Important
Verdict: Needs to be changed. It is up to Qlik themselves how they choose to sell their product. Not giving any sort of information on the cost however will have a huge negative impact on developer experience for their potential buyers.

3.5.10 The release- and change notes are thorough

The release notes are in my opinion well done. Since it is a very new product, the release notes are more about new features rather than changes. These are presented in the form of a blog post with images and gifs, which makes it easy to get an overview of what's new. They warn about changes they are making that may cause issues, and explain how one should fix these things.

Effort: Medium. Payoff: Low. Importance: Not important.
Verdict: No need for change.

3.5.11 The software has the same features on all different platforms

Qlik Core does not list on their website what features exist on what platforms. They have, according to one of their team members, tested it on different web browsers and found limitations for their services, but have yet to put them on their website. When it comes to the Qlik Associative Engine, that is run in support of the third-party software Docker. Docker is supported by Mac, Windows and most Linux distributions. It is also supports

many different servers. So the heart of Qlik Core, the engine, acting the same on all different platforms is reliant on Docker acting the same on all different platforms.

Effort: High. Payoff: Low-medium. Importance: Not very important
Verdict: Needs to be changed. The recommendation is to list your feature support on your website, which Qlik Core does not.

3.5.12 The software is compatible with different platforms

As mentioned in 3.5.11, the server side of Qlik Core (the engine) supports many platforms through Docker. The other libraries offered in Qlik Core works in all browsers, but may have limitations depending on the browser.

Effort: High. Payoff: Low-medium. Importance: Not very important.
Verdict: No need for change. I recommend however to make the information of platform support more clear on the Qlik Core website.

3.5.13 The software is offered in more than one programming language

As mentioned before, Qlik Core is constructed from a combination of different libraries. The engine is close-source, so the programming languages needed to know is for the supporting libraries that comes with Qlik Core. The majority of the libraries are written in JavaScript, which we saw in section 1.6 was a very popular programming language. However, they've actually gone to the length of offering the library that communicates with the engine, Enigma, is both JavaScript and Go. This is above and beyond what I even recommend.

Effort: High. Payoff: Low. Importance: Not important.
Verdict: No need for change.

3.5.14 The software is open source

Qlik Core is partially open source. The data loading software "Halyard", the discovery service "Mira" and the software communicating with the engine "Enigma" are all open source. The Qlik Core website and all the tutorials are also open-source. The Licensing service and the Qlik Associative Engine are however close-source. Their reasoning behind making the Qlik Associative Engine close-source is to protect intellectual property. It is their main selling point, and also serves as the basis for their other products, and needs to be protected. Their licensing service contains some intellectual property as well and logic that would be vulnerable to open up to the public. Qlik has been in talks however to open up this service as well to the public, but would need some refactoring of the software in that case.

Effort: Medium-High. Payoff: High. Importance: Important
Verdict: No need for change. Qlik has tried to make Qlik Core as open-source as possible, which follows the recommendations.

3.5.15 The software uses the programming language I am most comfortable with

As talked about in 1.5, the libraries in QC mainly uses JavaScript. The recommendation for this aspect is not to offer many languages, but rather keep an eye on what languages are popular, and will keep on being popular. And as we saw in section 1.6, JavaScript is the most popular language on GitHub.

Effort: Medium. Payoff: High. Importance: Important
Verdict: No need for change.

3.5.16 There exists an active online community around the software

Being a fairly new product, there doesn't seem to be any community around Qlik Core yet. Searching for Qlik Core on Stack Overflow yields *zero* results. There are no discussions from users directly in the GitHub repository relating to Qlik Core either. Qlik does have a forum site for its users, `https://community.qlik.com`. This website has different sub-categories for their products. It does not yet however a sub-category dedicated to Qlik Core. On the Qlik Core website, they link to a support channel on Slack (a chat program used commonly by developers). As of writing this paper, this seems to be the best way to get help as a user.
It may be too early to judge Qlik Core for not having an online community. In fact, when talking with a member of the Qlik Core team, they deliberately not tried to push Qlik Core too hard yet. They are still a small team working on the product, and the decision was to try and funnel all questions into one place, namley Slack. He also states that they watch for questions and issues on Stack Overflow. Once they feel ready to push the product more, they should in my opinion dedicate a part of their forum site to Qlik Core.

Effort: Medium. Payoff: Medium-High. Importance: Important.
Verdict: No need for change. It is natural that Qlik Core does not have a community yet, and since they have deliberately chosen not to try and create one, this aspect gets a pass. Qlik should be aware though of the importance of an online community.

3.6 Qlik Core Evaluation Summary

In total, it had no need for change for 10 out of the 16 aspects. In Table 3.16 we can see a summation. We see that it gets av average score of 0.55/1.00. It is suited for all of the

three job titles, but mostly managers. Out of the three contexts, it is most suited for the context of working professionally.

Table 3.16: The summation of how well Qlik Core follows the recommendations

Needs change	Question	Avg Everyone	Architect	D&E	Managers	Group Points	Single Points	Hobby Points
X	The API has code examples	0.90	0.92	0.91	0.79	0.93	0.88	0.88
X	The API documentation gives thorough explanations on how it works	0.82	0.82	0.87	0.73	0.85	0.82	0.79
X	I can have working code quickly	0.80	0.85	0.80	0.71	0.75	0.80	0.86
X	The pricing of the software	0.80	0.76	0.84	0.75	0.74	0.76	0.91
	The software uses the programming language I am most comfortable with	0.68	0.69	0.73	0.63	0.56	0.71	0.78
	The software is open source	0.68	0.77	0.69	0.60	0.64	0.66	0.74
	There exists an active online community around the software	0.65	0.63	0.67	0.54	0.69	0.62	0.65
X	The documentation is easy to navigate	0.60	0.66	0.61	0.52	0.61	0.60	0.60
	The official website looks professional	0.60	0.63	0.56	0.71	0.67	0.60	0.53
	The software is compatible with different platforms	0.58	0.58	0.52	0.54	0.81	0.58	0.37
	How often the software is updated	0.58	0.52	0.57	0.63	0.69	0.59	0.46
	The documentation doesn't assume any prior expertise	0.47	0.47	0.44	0.54	0.41	0.47	0.53
X	The software has the same features on all different platforms	0.46	0.41	0.45	0.44	0.63	0.47	0.28
	The documentation has consistent language	0.41	0.49	0.34	0.42	0.42	0.42	0.39
	The software is offered in more than one programming language	0.36	0.37	0.31	0.44	0.48	0.35	0.25
	The release- and change notes are thorough	0.33	0.26	0.29	0.44	0.40	0.31	0.27
Points		4.39	4.41	4.47	3.94	4.51	4.34	4.32
Max Points		9.73	9.81	9.60	9.42	10.27	9.64	9.28
Score		**0.55**	**0.55**	**0.53**	**0.58**	**0.56**	**0.55**	**0.53**

3.7 Threats to Validity

Runeson and Höst (2008, pp. 71-74) discusses validity of results from studies made in software engineering. As they put it, validity denotes the trustworthiness of the results yielded by a study. They present four main points of validity: construct validity, internal validity, external validity and reliability.

Construct validity is related to the interpretation of questions asked by the researcher, if the person asked interprets the question the way it is intended by the researcher. This study put effort into this issue, trying to make sure as much as possible that the questions were interpreted as intended. The pilot survey showed that there were issues with construct validity, which was fixed for the main survey. In the interviews, the interviewees were asked to clarify ambiguous answers and if it was clear that they were interpreting the question wrong the same question was asked again with more clarification from the interviewer. Although you cannot be positive that the persons taking the survey interpreted the questions asked the way that were intended, construct validity is not of great concern for this report.

Internal validity is related to the examination of causal relations in results, e.g. cause and effect. If it is being studies whether factor A causes factor X, there is a risk that factor X is also affected by an unknown factor B. If factor B is not known to the researcher, false conclusions may be drawn. This report presents the results with this in mind, saying that the results seem to *suggest* that the results leads to a certain conclusion. External validity relates to how the generalization of the results are made. The results are from a certain point in time, by a certain group of people, being in a certain situation and generalizing the results as being true in all cases may cause invalidity. This report is however careful to not make too definite conclusions. The data pool for this report were a very homogeneous group of people and was quite small. There is a certain risk for external validity with this report that it results may not be applicable as a ''truth" for all cases for software platform. Using the results from this report, one should be aware of the nature of the data pool. Reliability, in the context of validity, concerns how dependent the results are on the conductor of the research. A study like this should, in theory, yield the same result if conducted again by another researcher. One way of lowering the risk for invalidity by this factor is to present what question was asked in the interviews and questionnaires, as well as tell how the data was analysed. This has been made in this report, and it is in my opinion a low risk of invalidity caused by reliability.

One way of lowering the risk of result invalidity according to Runeson and Höst (2008) is to triangulate the results, which has been done in this report. The validity of the results is overall not questionable in my opinion.

Chapter 4
Conclusion

So what is the conclusion of this research paper? If we go back to the original questions this research paper intended to answer, they were:

1. What aspects are needed by a software platform in order for people to find them favourable, and how important are these aspects?
2. Do different groups of software platform users have different needs?
3. Why are some of these aspects needed or not needed by a software platform in order for people to find them favourable?
4. How favourable is Qlik's software platform Qlik Core to use users, and what can be improved?

We now have the knowledge to answer each of these questions.

What aspects are needed by a software platform in order for people to find them favourable, and how important are these aspects?
The conclusion of the first question is that the most important aspects for a software platform is that it has thorough documentation, code examples, is easy to get started with, and clearly shows the cost of the software platform. The least important aspects that need to exist in order for developers to find a software platform favourable is that the release notes are thorough, that the software platform is offered in several languages and the documentation has consistent language.

Do different groups of software platform users have different needs?
The conclusion for the second question is that some groupings are irrelevant, while some other show that different groups have different needs. Looking at years of experience or if people has the power to make decisions for other people did not say much about what needs they had from a software platform. Grouping people by their job title or by in what

context they are going to use a software platform showed differences in priorities. The research also found that there is no uncoupling between what people consider and what while give them a positive experience using a software platform. Ergo, they are aware of what they need in order to be happy.
Overall the research also found that developers are impatient and follow the path of least resistance. If a software platform seems cumbersome, they would rather spend time on finding an alternative than trying to understand the software platform.

Why are some of these aspects needed or not needed by a software platform in order for people to find them favourable?
The research found that developers want working code quickly in order to be able to evaluate if a software platform is useful. They do not read about the platform when evaluating, but would rather just try it out themselves. The research also found that online communities are used as a source of questions-and-answers when stuck and developers do not care much about where it is, who is answering or if they feel apart of the community. Release notes were also found to be avoided by developers as much as possible. API examples in documentation was used for several things: copy-pasting into your code, seeing how methods should be used but most importantly as an overview for understanding the platform instead of reading explanatory texts.

How favourable is Qlik's software platform Qlik Core to use users, and what can be improved?
Qlik's software platform Qlik Core was found to have some issues. It lacked API examples and API documentation explanations. It was hard to get started with and did not tell users its pricing. The research found that it was slightly more suited for professional use compared to hobby projects, and was slightly more suited for managers than developers, engineers and architects.

4.1 Further Research

There is a lot of possibility for further research. Here I suggest a few things.

Deeper investigate other aspects than the three in this research paper
Because of time constraints, there was only possible deep dive in to some of the aspects. It would however be interesting to get a deeper understanding of some of the other aspects.

How do people discover new software?
It was investigated a little bit in the test survey how people discover new software. This had to be dropped because of time constraints in this research paper, but would be interesting to research more.

Use another data pool
The data pool for this research paper was quite homogeneous. It would be interesting to

see if other data pools yields similar results.

Do the creators behind the software matter?
In the test survey it was also asked about if people consider the creator behind the software. This was also dropped in the main survey, but would be interesting to research more.

Extend the list of aspects
The nature of this research paper was of the investigating kind. The list of aspects could be extended with considerations that were not brought up in this paper.

Bibliography

Regina Bernhaupt, Girish Dalvi, Anirudha Joshi, Devanuj K. Balkrishan, Jacki O'Neill, and Marco Winckler. *Human-Computer Interaction - INTERACT 2017 - 16th IFIP TC 13 International Conference, Mumbai, India, September 25-29, 2017, Proceedings, Part IV*.

Code Academy. What is rest?, 2019. https://www.codecademy.com/articles/what-is-rest [accessed: 2019.01.15].

Martyn Denscombe. *The good research guide : for small-scale social research projects*. McGraw-Hill Education, 4 edition, 8 2010. ISBN 9780335241408.

Rahul Dhide. Building the Developer Experience (DX) From the Ground Up. 10 2017. https://blog.argoproj.io/building-the-developer-experience-dx-from-the-ground-up-8254d50457f5 [accessed: 2019.01.22].

Every Developer. What is developer experience?, 2019. http://everydeveloper.com/developer-experience/ [accessed: 2019.11.01].

Xinyang Feng, Jianjing Shen, and Ying Fan. Rest: An alternative to rpc for web services architecture. *First International Conference on Future Information Networks*, pages 7–10, 2009.

GitHub. Top languages over time, 2019. https://octoverse.github.com/projects#languages [accessed: 2019.01.31].

Daniel Graziotina, Fabian Fagerholm, Xiaofeng Wangd, and Pekka Abrahamsson. What happens when software developers are (un)happy. *Journal of Systems and Software*, 140:32–47, 2018.

Martin Höst. EDAA35 Utväardering av programvarusystem. 2019. http://fileadmin.cs.lth.se/cs/Education/EDAA35/EDAA35.pdf [accessed: 2019.02.21].

ISO 9241-210:2010 (2010). Ergonomics of human system interaction - part 210: Human-centered design for interactive systems (formerly known as 13407). Standard, International Organization for Standardization, Geneva, CH, 2010. https://www.iso.org/standard/52075.html.

Sam Jarman. The best practices for a great developer experience (dx). 2017. https://hackernoon.com/the-best-practices-for-a-great-developer-experience-dx-9036834382b0 [accessed: 2019.11.01].

Jyrki Kontio, Laura Lehtola, and Johanna Bragge. Using the Focus Group Method in Software Engineering: Obtaining Practitioner and User Experiences. *Proceedings - 2004 International Symposium on Empirical Software Engineering, ISESE 2004*, pages 271– 280, 09 2004.

Guy Levin. Internal vs external apis. 3 2017. https://blog.restcase.com/internal-vs-external-apis/ [accessed: 2019.01.31].

Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. Api change and fault proneness: A threat to the success of android apps. pages 477–487, 2013.

Don Norman, Jim Miller, and Austin Henderson. What you see, some of what's in the future, and how we go about doing it: Hi at apple computer. In *Conference Companion on Human Factors in Computing Systems*, CHI '95, pages 155–. 1995. ISBN 0-89791-755-3.

Mark Nottingham. User personas for http apis. 4 2012. https://www.mnot.net/blog/2012/04/14/user_personas_for_http_apis.

OpenSource. What is Docker?, 2019. https://opensource.com/resources/what-docker [accessed: 2019.01.31].

Qlik. Qlik core, 2019. https://core.qlik.com/ [accessed: 2019.01.31].

Qlik Media Representation. From Swedish Startup to Software Success Story: QlikTech Lars Björk Named Ernst Young Entrepreneur Of The Year® 2010 Winner in the Technology Category, 2010. https://www.qlik.com/us/company/press-room/press-releases/1114-from-swedish-startup-to-software-success-story [accessed: 2019.01.31].

Martin P. Robillard. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, 26(6):27–34, 2009.

Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, Dec 2008.

Phil Sturgeon. Understanding rpc vs rest for http apis. 9 2016. https://www.smashingmagazine.com/2016/09/understanding-rest-and-rpc-for-http-apis/, [accessed: 2019.01.31].

W3C. Glossary of terms for device independence. W3C working draft, W3C, January 2005. http://www.w3.org/TR/2005/WD-di-gloss-20050118/.

Michał Wróbel. Emotions in the software development process. *2013 6th International Conference on Human System Interactions, HSI 2013*, pages 518–523, 06 2013.

Appendix A
Pilot Survey

The initial test survey that was sent out. The original survey was conducted using Google Form, and answered online. The look of it is not the same here as it was online, but the questions are identical.

New Software and Developer Experience

Brief background about you

* Required

1. **What is your job title at your company? ***
   Mark only one oval.
   - I am not currently working at a company.
   - Prefer not to answer.
   - Other: ____________________

2. **How big is the company you are currently working at? ***
   Mark only one oval.
   - Self employed
   - 2 - 5 people
   - 5 - 20 people
   - 20 - 50 people
   - 50 - 100 people
   - 100 - 200 people
   - 200+ people
   - I am not currently working

3. **For how many years have you been working professionally within the software industry? ***
   Mark only one oval.
   - Less than 1 year
   - 1 - 3 years
   - 3 - 5 years
   - 5 - 10 years
   - 10 - 15 years
   - 15+ years
   - I don't work within the software industry.

Skip to question 4.

New software

When we use the word software, we mean anything where the target audience of the software are developers, be it SDKs, libraries, APIs, IDEs etc. We will collectively call these 'tools and frameworks' in the survey.

4. **How do you usually discover new tools and frameworks?** *

Check all that apply.

- [ ] Online communities and forums
- [ ] Friends or coworkers telling me about it
- [ ] Conferences
- [ ] Searching for related key words online
- [ ] Reading blog posts or articles
- [ ] Social media
- [ ] Other: ____________________

5. **What is a common reason you decide not to use a tool or framework?**

____
____
____
____
____

6. **What do you usually do first when you want to evaluate if you want to use a new tool or framework?** *

Check all that apply.

- [ ] Try to make a simple project from scratch (Like "Hello World")
- [ ] Try to integrate it into a simple project I already have
- [ ] Try to integrate it right away into the project I intend to use it in
- [ ] Follow a more advanced step-by-step tutorial
- [ ] Read a lot about it online, before starting any kind of coding
- [ ] Other: ____________________

7. **How quickly do you usually decide if the tool or framework is for you?** *
Mark only one oval.

- Less than 10 minutes
- Less than 30 minutes
- Less than 1 hour
- Less than 3 hours
- Less than 6 hours
- Less than 12 hours
- Less than 1 day
- Less than 2 days
- Less than 3 days
- Less than a week
- Less than a month
- More than a month
- I prefer not to answer

Developer Experience

In this survey we are trying to evaluate what makes a developer give new software a shot!

8. **Have you ever heard of the term "Developer Experience" (DX)?** *
Mark only one oval.

- Yes, and I could comfortably give a definition of it
- Yes, and I think I could give a definition of it
- Yes, but I could not give a definition of it
- No
- I don't know

9. **What is the last new tool or framework that you decided to try that you felt gave you a good experience as a developer?**

10. **Are there any tools or frameworks that you have to use, that you do not like using? Why?**

Last Part!

You just heard about a new tool or framework for your project!

Let's say you just heard about a new software that might be useful in the project that you are working on, and want to check it out.

11. **Which of these traits or aspects do you usually consider when deciding if you want to TRY a new tool or framework?** *
Mark only one oval per row.

	Never consider	Rarely consider	Sometimes consider	Often consider	Always consider
The official website looks professional	◯	◯	◯	◯	◯
I can have working code quickly	◯	◯	◯	◯	◯
The API has code examples	◯	◯	◯	◯	◯
The API gives thorough explanations	◯	◯	◯	◯	◯
The documentation doesn't assume any prior expertise	◯	◯	◯	◯	◯
The documentation has consistent language	◯	◯	◯	◯	◯
The documentation is easy to navigate	◯	◯	◯	◯	◯
The release- and change notes are thorough	◯	◯	◯	◯	◯
How often the software is updated	◯	◯	◯	◯	◯
The pricing of the software	◯	◯	◯	◯	◯
There exists an active online community around the software	◯	◯	◯	◯	◯
The software is compatible with different platforms	◯	◯	◯	◯	◯
The software has the same features on all different platforms	◯	◯	◯	◯	◯
The software uses the programming language I am most comfortable with	◯	◯	◯	◯	◯

12. **What other things do you usually consider with software you might use?**

____
____
____
____
____

The creator behind the tool or framework

Behind the tool or framework is always a creator (be it a company, person or community).

13. **Which of these traits or aspects do you usually consider when deciding if you want to TRY a new tool or framework?** *

Mark only one oval per row.

	Never consider	Rarely consider	Sometimes consider	Often consider	Always consider
The creator of the software has good communication with it's users	◯	◯	◯	◯	◯
The creator of the software has high transparency with it's issues, ways of working, future plans, etc.	◯	◯	◯	◯	◯
The creator of the software seems professional	◯	◯	◯	◯	◯
The creator of the software has a good reputation online	◯	◯	◯	◯	◯
I have heard of the creator of the software before	◯	◯	◯	◯	◯
I have heard of other software the creator of the software has made	◯	◯	◯	◯	◯

14. **What other things do you consider with the creator behind the tool or framework?**

____

____

____

____

____

Deal breakers

Some things just has to exist in order for developers to use a software.

15. **Which of these aspects or traits will make you definitely not use a new tool or framework?** *

Check all that apply.

- [ ] It takes a long time to get initially started
- [ ] The API has poor or no code examples
- [ ] The API is poorly explained
- [ ] The documentation uses inconsistent language
- [ ] The documentation assumes prior experience with the software
- [ ] The website for the documentation is hard to navigate
- [ ] The release notes are poorly written
- [ ] I have to pay to be able to fully evaluate the software
- [ ] The online community around the software is dead or has little activity
- [ ] The online community around the software is unappealing
- [ ] The creators behind the software are not transparent
- [ ] The creators behind the software feel like they cater to businesses, not developers
- [ ] The software is not open source
- [ ] Other: ______________________________

Powered by

Appendix B
Main Survey

The main survey that was sent out. The original survey was conducted using Google Form, and answered online. The look of it is not the same here as it was online, but the questions are identical.

What is required by software platforms in order for developers to use them?

This survey is made as part of a master thesis conducted for Qlik AB. It is aimed at people working with software platforms. It tries to answer the question of what aspects are required by software platforms in order for developers to use them and enjoy them.

By taking this survey, you are not only helping better the quality of Qlik products, but extending the academic research around the understanding of what developers want from software platforms.

As part of this master thesis we will also be conducting some interviews with some selected persons to get a deeper understanding. If you are available to come to the Qlik offices in Lund, and you are interested in taking part in an interview, please answer 'Yes' in the 2nd question below.
You are only saying that you're willing to be contacted. You are not agreeing to an interview, and there's also no guarantee that you will be contacted.

* Required

1. **Have you read and understood the text above?** *
Mark only one oval.

- Yes
- No

2. **Would you be willing to be contacted after this survey to do a more in-depth interview?** *
Mark only one oval.

- Yes *Skip to question 3.*
- No *Skip to question 6.*

Interview Contact Details

You have answered that you would be willing to be contacted.
As stated before, this is not agreeing to taking part in an interview, and there's no guarantee that you will be contacted.

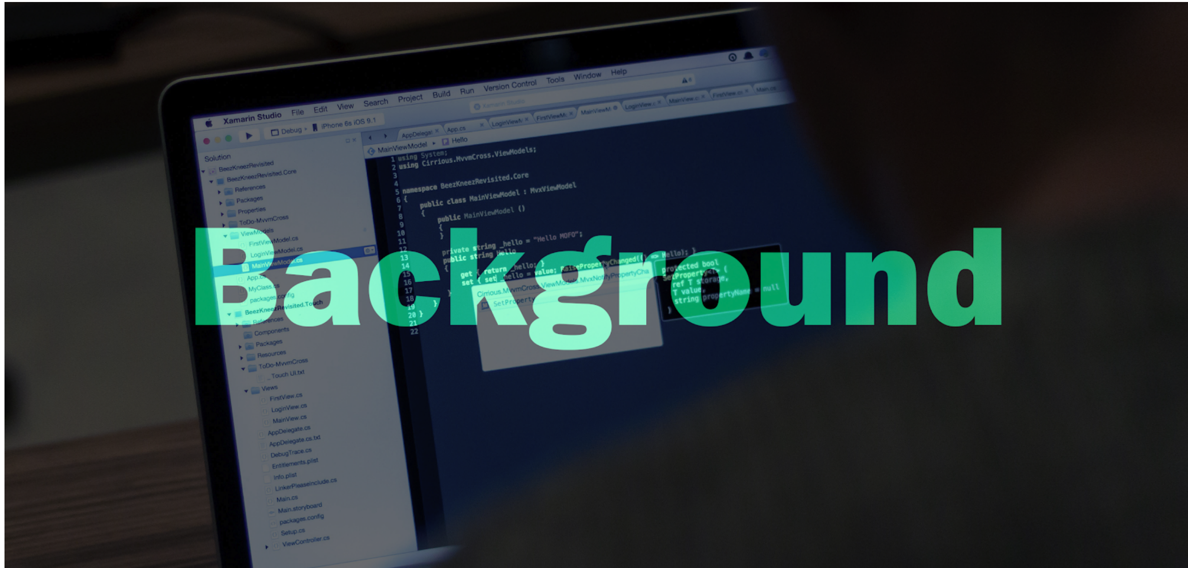
3. **First- and last name** *

4. **Email** *

5. **What company do you work at?** *

Background

Some quick information about yourself.



6. **What is your job title?** *

7. **What is your level?** *
Mark only one oval.

- Junior
- Middle
- Senior
- Don't know / Not applicable
- Prefer not to answer

8. **How many years have your been working professionally within the software industry?** *
Mark only one oval.

- Less than 1 year
- 1 year
- 2 years
- 3 years
- 4 years
- 5 - 7 years
- 8 - 10 years
- 10 - 15 years
- 15 - 20 years
- 20 - 25 years
- 25+ years
- Prefer not to answer

9. **How big is the company you work at?** *

Mark only one oval.

- I'm not currently employed
- I'm self employed
- 2 - 5 people
- 6 - 10 people
- 11 - 30 people
- 31 - 50 people
- 51 - 75 people
- 76 - 100 people
- 101 - 150 people
- 151 - 250 people
- 251 - 500 people
- 501 - 1000 people
- 1001 - 2000 people
- 2000 - 5000 people
- More than 5000 people
- Prefer not to answer
- I don't work within the software development industry

10. **Are you in a position where you can make decisions on what software other coworkers will use, for example what platform a project will be built upon?** *

Mark only one oval.

- Yes *Skip to question 11.*
- No *Skip to question 12.*
- I don't work in a group / Don't know / Prefer not to answer *Skip to question 12.*

Skip to question 12.

Working professionally & deciding for a group

You checked that you ARE in a position to make decisions for a group. Therefore this section concerns when you are going to use software professionally, and your have to take into consideration how it may also affect coworkers. For example, you are choosing what platform a project will be built upon.

11. **When working professionally and deciding for a group, which of these traits or aspects do you usually consider when using a software platform?** *

Mark only one oval per row.

	Never consider	Rarely consider	Sometimes consider	Often consider	Always consider
The API has code examples					
The API documentation gives thorough explanations on how it works					
There exists an active online community around the software					
I can have working code quickly					
The pricing of the software					
The software is compatible with different platforms (Such as different operating systems or browsers)					
The software uses the programming language I am most comfortable with					
The software is offered in more than one programming language					
How often the software is updated					
The software has the same features on all different platforms (Such as different operating systems or browsers)					
The documentation is easy to navigate					
The documentation doesn't assume any prior expertise					
The documentation has consistent language					
The release- and change notes are thorough					
The official website looks professional					
The software is open source					

Working professionally

This part concerns when you are working professionally, but your decision to use software platform is not for a group. For example, the software will be present in a part of a project where you are the sole contributor.

12. **When working professionally and deciding for yourself, which of these traits or aspects do you usually consider when using a software platform?** *

Mark only one oval per row.

	Never consider	Rarely consider	Sometimes consider	Often consider	Always consider
The API has code examples	◯	◯	◯	◯	◯
The API documentation gives thorough explanations on how it works	◯	◯	◯	◯	◯
There exists an active online community around the software	◯	◯	◯	◯	◯
I can have working code quickly	◯	◯	◯	◯	◯
The pricing of the software	◯	◯	◯	◯	◯
The software is compatible with different platforms (Such as different operating systems or browsers)	◯	◯	◯	◯	◯
The software uses the programming language I am most comfortable with	◯	◯	◯	◯	◯
The software is offered in more than one programming language	◯	◯	◯	◯	◯
How often the software is updated	◯	◯	◯	◯	◯
The software has the same features on all different platforms (Such as different operating systems or browsers)	◯	◯	◯	◯	◯
The documentation is easy to navigate	◯	◯	◯	◯	◯
The documentation doesn't assume any prior expertise	◯	◯	◯	◯	◯
The documentation has consistent language	◯	◯	◯	◯	◯
The release- and change notes are thorough	◯	◯	◯	◯	◯
The official website looks professional	◯	◯	◯	◯	◯
The software is open source	◯	◯	◯	◯	◯

Skip to question 13.

Software considerations when working on a hobby project by yourself

This part concerns when you are working on a hobby project by yourself and don't have to concern your self with the issues present when working professionally.

13. **When working on a hobby project, which of these traits or aspects do you usually consider when using a software platform?** *
Mark only one oval per row.

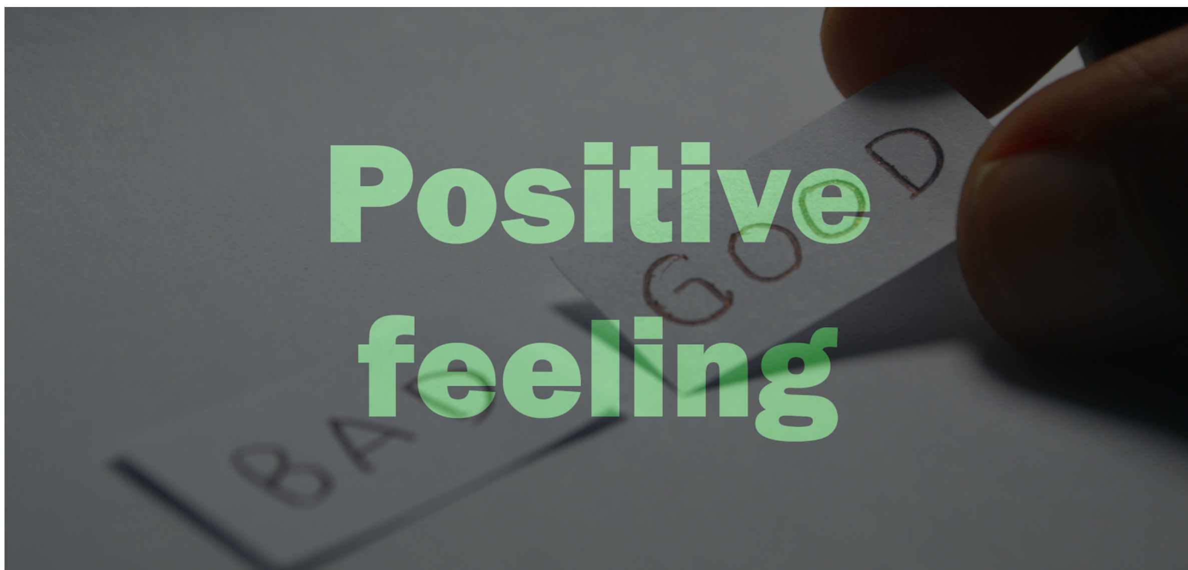
	Never consider	Rarely consider	Sometimes consider	Often consider	Always consider
The API has code examples					
The API documentation gives thorough explanations on how it works					
There exists an active online community around the software					
I can have working code quickly					
The pricing of the software					
The software is compatible with different platforms (Such as different operating systems or browsers)					
The software uses the programming language I am most comfortable with					
The software is offered in more than one programming language					
How often the software is updated					
The software has the same features on all different platforms (Such as different operating systems or browsers)					
The documentation is easy to navigate					
The documentation doesn't assume any prior expertise					
The documentation has consistent language					
The release- and change notes are thorough					
The official website looks professional					
The software is open source					

Developer Experience

Background: Developer Experience (DX) is the equivalent of User Experience (UX) when the target audience are software developers. It concerns itself with the somewhat vague quality of how software makes you "feel". In the following question we're trying to pinpoint what's important from software in order for a developer to have a good experience when interacting with it. When answering this, try to think of moments when software has made you feel good, and times when software has irritated you, to be able to answer how important the aspects are.

14. **Did you read the background text above?** *
Mark only one oval.

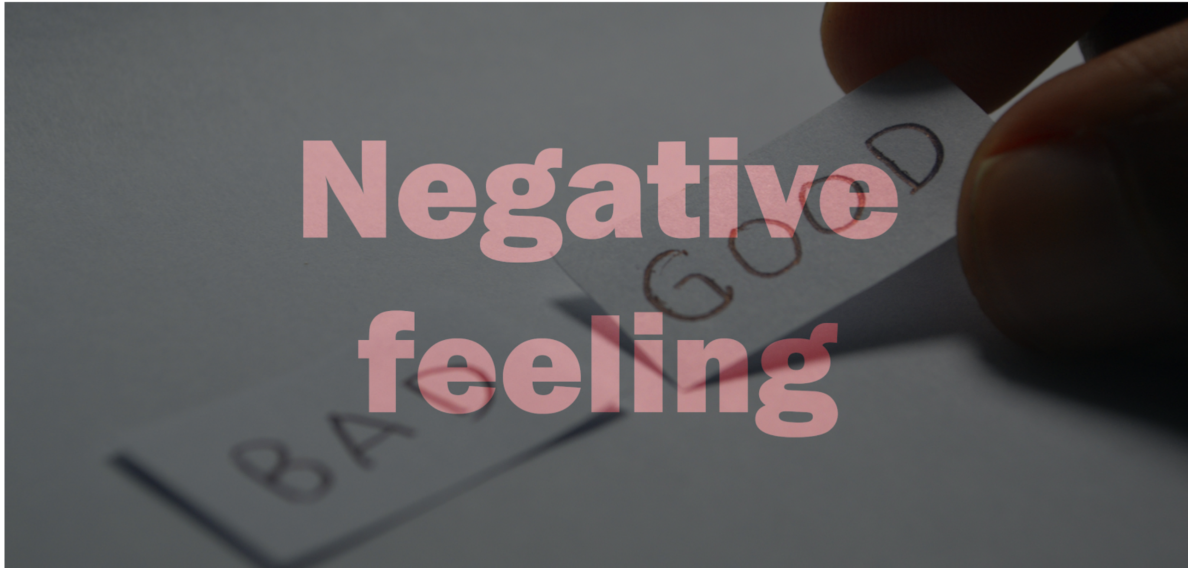
- ( ) Yes
- ( ) No



15. **On a scale from 1 - 5, how important is it that these aspects exist in order for you to leave the interaction with the software platform with a POSITIVE FEELING?** *

Mark only one oval per row.

	1 - Not very important	2	3 - Neutral	4	5 - Very important
The pricing of the software was easy to find					
The code examples for the API were good					
The API was thoroughly explained so that you could understand how it worked					
The online community was helpful with up-to-date discussion threads					
The documentation was easy to navigate					
I could quickly have working code when starting from scratch					
The release notes for what's new/updated/deprecated/etc in an update were well-written					
The software was compatible on different platforms (Such as different operating systems or browsers)					
The software used a programming language I am skilled in					
The software was offered in more than programming language					
The software quickly released updates to address bugs					
The official website looked professional					
The documentation did not assume that I had any prior expertise with the software, not referencing software-specific things without explaining them					
The documentation had a consistent language, not using different words to mean the same thing					
The software's features existed and acted the same on different platforms (such as different operating systems or browsers)					
The software was open source					


Negative
feeling
GOOD
BAD

16. **On a scale from 1 - 5, how important is it that these aspects exist in order for you to leave the interaction with the software platform with a NEGATIVE FEELING?** *

Mark only one oval per row.

	1 - Not very important	2	3 - Neutral	4	5 - Very important
The pricing of the software was hard to find					
The code examples for the APIs were bad					
The API was poorly explained so that you could not understand how it worked					
The online community was not helpful and the discussion threads were out-of-date					
The documentation was hard to navigate					
I took a long time before I had working code when starting from scratch					
The change- and release logs for what's new/updated/deprecated/etc in an update were poorly written					
The software was compatible on only one platform (Such as operating system or browser)					
The software used a programming language I am not very skilled in					
The software was only offered in one programming language					
The software was slow to release updates to address bugs					
The official website did not look professional					
The documentation assumed I had prior expertise with the software, referencing software-specific things without explaining them					
The documentation did not have a consistent language, using different words to mean the same thing					
The software's features did not exist or acted differently on different platforms (such as different operating systems or browsers)					
The software was not open source					

Feedback

Here you can give feedback if there were any issues or if you have anything you'd like to tell the researchers.

17. **Feedback**

Powered by

Appendix C
Material for Interview

Before the interviews were conducted, the interviewees got handed this PDF with a part of a fictive software platform documentation. They had to read and solve three questions, to make sure that they had actually read through it properly.

Material Quiz for MyBakery API

As your disposal you have some material. A section of API documentation, namely the description of the class `Step`, the method `addStep` and the method `makeCake`. You also have a longer version and a shorter version of release notes for the API.

What's wrong in this code?

The code tries to bake a strawberry shortcake. However, there are 3 errors in this code. What are they?

```
1.  let baker = new Baker(createSuperEngine());
2.  let recipeDB = new RecipeDB('recipes/cakes.json');
3.  let recipe = recipeDB.getRecipe('strawberry shortcake');
4.  baker.setRecipe(recipe);
5.  baker.serveCake();
```

Answer:

```

```

Which of these two implementations would you recommend? Why?

Implementation 1

```
1.  let b = new Baker(new SuperEngine());
2.  b.setRecipe(recipeDB.get('chocolate cake'));
3.  b.makeAnySizeCake(500);
```

Implementation 2

```
1.  let b = new Baker(new Engine());
2.  b.setRecipe(recipeDB.get('chocolate cake'));
3.  b.makeLargeCake());
```

Answer:

```

```

What's wrong in this implementation?

Row 4 and 5 are correct, but row 1, 2, 3, 6 and 7, 8 all have errors in them. What are they?

```
1.  let step1 = new Step('add', 'large eggs', 3000);
2.  let step2 = new Step('knead', 5000, 'knead the dough for 5 seconds');
3.  let step3 = new Step('cool', 6000.0);
4.  let recipeDB = new RecipeDatabase('recipes/cakes.json');
5.  let recipe = recipeDB.getRecipe('strawberry cake');
6.  recipe.addStep('after', '1', step1);
7.  recipe.addStep('behind', 3, step2);
8.  recipe.addStep('last', step3);
```

Answer:

```
1.

2.

3.

6.

7.

8.
```

MyBakery Platform

Recipe API

`addStep(string, int, Step)` method

Usage

`recipe.addStep(position, stepNbr, newStep)`

The method calls on the `Recipe` class to add a new `Step`[*Spec*].

Parameters

Name	Type	Description
`position`	string	Specifies if the step should be added before or after `stepNbr`. Acceptable inputs are `'after'`, `'before'`
`stepNbr`	int	Takes a positive int. Specifies the number of the new step. If the number is larger than the last step in the `Recipe`, the step will be added last.
`newStep`	Step	The new step to be added. See Step[*Spec*].

Returns

Return type	Status	Description
`void`	Success	On success, the function will not return anything
`WrongParametersError`	Error	This error is returned when the parameters are of the wrong format.
`NotEnoughArumentsError`	Error	This error will be returned when there are not enough arguments.

Examples

This example makes a strawberry cake, but adds a pause step after step 4.

```
let recipeDB = new RecipeDatabase('recipes/cakes.json');
let recipe = recipeDB.getRecipe('strawberry cake');
recipe.addStep('after', 4, new Step('pause', 300000));
```

MyBakery Platform

Step

Step is a class used by `Recipe` that describes a step of how to bake a cake. The class has no methods, but is simply a structure.

Usage

The class has three different constructors.

`new Step(stepName, duration)`

Intended to be used for a standard step without an ingredient. If `stepName` already exists in the `RecipeDatabase` the description will be fetched automatically.

`new Step(stepName, duration, ingredient)`

Intended to be used for a standard step with an ingredient.

`new Step(stepName, duration, description)`

Intended to be used when adding a custom step that does not exist in the database.

Parameters

Name	Type	Description
`stepName`	string	Name of the step.
`duration`	int	Specifies the duration of the step, in milliseconds. Parameter must be a positive int.
`ingredient`	string	The name of the ingredient
`description`	string	Specifies what to do in the step.

Standard Step Names

These are the standard step names and their respective descriptions. They all exist in `RecipeDatabase`

Step Name	Description
`'pause'`	Pauses all execution for the specified duration.
`'mix'`	Mixes the ingredients

`'knead'`	Kneads the dough, if it exists
`'add'`	Adds the specified ingredient
`'repeat'`	Will loop once the steps between `repeat` and `stopRepeat`
`'stopRepeat'`	Will stop the `repeat` loop.
`'cool'`	Cools the mixture.
`'putInOven'`	Puts the mixture in the oven
`'addToSmallPan'`	Adds the mixture to a small pan
`'addToBigPan'`	Adds the mixture to a big pan

Errors

Error Type	Description
`doubleStepNameError`	This error is returned a custom step is trying to be added that already exists in the `RecipeDatabase`
`wrongParametersError`	This error will be returned the wrong amount of parameters is put in, the parameters are not of the correct type or the duration is negative.

MyBakery Platform

Baker API

The `Baker` is the central part of the MyBaker Platform, creating all `Cake` objects.

`makeCake()` method

Usage

`baker.makeCake()`

The method calls on the class `Baker` to follow the exact instructions and amounts given by `Recipe`*Spec*. It will automatically change `mode` of `Engine`*Spec* if needed. It returns an object of the class `Cake`*Spec* or throws an error.

Parameters

This method takes no parameters.

Returns

Return type	Status	Description
`Cake`	Success	On success, the function will return an object of the class `Cake`
`NoEngineError`	Error	This error is returned when either `Baker` has not been assigned an `Engine` or the assigned `Engine` is busy
`LowEnergyError`	Error	This error is returned when the `Baker` class has less energy than required by the `Recipe`
`WrongEngineModeError`	Error	This error is returned when the mode of the `Engine` `mode` does not match the mode required by `Recipe`

Examples

Example

This example makes a strawberry cake without any changes.

```
let engine = SuperEngine();
let baker = new Baker(engine);
let recipeDB = new RecipeDatabase('recipes/cakes.json');
baker.setRecipe(recipeDB.getRecipe('strawberry cake'));
```

```
let cake = baker.makeCake();
baker.serveCake(cake);
```

Example with modifications

This example makes a strawberry cake, but doubles the amount of flour, changes the speed of the engine and adds a pause step after step 4.

```
let engine = SuperEngine();
let baker = new Baker(engine);
let recipeDB = new RecipeDatabase('recipes/cakes.json');
let recipe = recipeDB.getRecipe('strawberry cake');
recipe.setIngredientAmount('flour', recipe.getAmount('flour')*2);
recipe.addStep('after', 4, new Step('pause', 300000));
baker.setRecipe(recipe);
baker.setEngine(getEngine().setSpeed(400));
let cake = baker.makeCake();
baker.serveCake(cake);
```

Change Log to MyBakery API

Version 5.3

New/Updated

`SuperEngine`

`makeAnySizeCake(int: size)`

`servePlace(string: place)`

Deprecated

- `makeHorribleCake()`
- `makeLargeCake()`
- `makeMediumCake()`
- `makeSmallCake()`

Bugs

The egg bug is fixed.

The overheating bug is fixed.

Change Log to MyBakery API

Version 5.3

7 November 2018

We are happy to announced that this latest release introduces our new, more efficient, `SuperCakeMachineEngine` that addresses many of the issues the community have pointed about. This release sees new methods, updated methods, deprecated methods and bug fixes. We also talk about known bugs that we are still working on.

New Class

With the release we present a new class.

`SuperEngine()`

This is the new cake engine. It replaces the now deprecated `Engine()`

Read specifications here: ***SuperEngine specifications***

New Methods

This release also sees a new method.

`makeAnySizeCake(int: size)`

This method replaces the methods `makeLargeCake()`, `makeMediumCake()` and `makeSmallCake`.

Read specifications here: ***makeAnySizeCake spec***

Updated Methods

`servePlace(string: place)` now also supports the input `'inFace'`.

Read specifications here: ***servePlace spec***

Deprecated Methods

Some methods will be deprecated with this release.

List of deprecated methods

- `Engine()`, instead use `SuperEngine()`

`Engine()` will not be supported with the introduction of our new engine.

- `makeLargeCake()`, instead use `makeAnySizeCake(500)`

- `makeMediumCake()`, instead use `makeAnySizeCake(300)`
- `makeSmallCake()`, instead use `makeAnySizeCake(100)`

The reasoning behind removing the methods are that they are reliant on our old `BadCakeMachineEnginge` that had performance issues. With the introduction of `makeAnySizeCake`, the baker can make any sized cake.

- `makeHorribleCake()`

This method has been deprecated since it's usage has been close to zero. If you still wish to use it the following code will yield the same result:

```
let engine = SuperEngine();
let baker = new Baker(engine);
let recipeDB = new RecipeData(cakes.json);
baker.setRecipe(recipeDB.get('weddingCake'));
baker.getIngredients().forEach((ingredient) => {
   ingredient.setAmount('random');
});
baker.makeCake();
```

Bug fixes

The egg bug

The method `addEggs(float: amount)` had a bug where you could only add even amount of eggs. This was due to a bug in the `Baker` class where it used the `mainBowl` class instead `separateBowl` and then retrieving the correct amount. This has now been fixed.

The bug was introduced in version 5.0. Read more about the release here: *5.0 - A Better Baker!*

The overheating bug

The method `setSpeed(int: speed)` had an issue where setting a speed to over 9000 would overheat the engine. This was due to a bug in the old engine where `extremeCoolingSystem` was not set to `true`. This has now been fixed.

The bug was introduced in version 5.2. Read more about that release here: *Version 5.2 - Power savings*

Known bugs

BakerPropertyError

We are currently still working on the bug of the baker returning the error code of:

```
BakerPropertyError: {
    Name: John Smith
    State: HungOver
    EnergyLevel: -1
```

```
}
Expected EnergyLevel to be 100.
Could not bake cake.
```

The bug has existed since *Version 5.0 - A Better Baker!* and occurs when using the method `setDay('sunday')`. We recommend not using this method in production at the moment.

We have made a blog post on a workaround for this issue that you can read about here:
BakerPropertyError: A workaround with Baker.enforce('sobriety')

The bug is still not solved, but we are working on it and hope to fix it shortly.