UNIVERSITY OF VAASA

SCHOOL OF TECHNOLOGY AND INNOVATIONS

SOFTWARE ENGINEERING

Minne Paljakka

REQUIREMENTS SPECIFICATION FOR A WEB APPLICATION

Master´s thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, 29th March 2018.

Supervisor	Prof. Jouni Lampinen
Instructor	M.Sc. Kenneth Widell

ACKNOWLEDGEMENT

The research behind this master’s thesis was made in collaboration with Wärtsilä Finland Oy. As a result, the research produced a requirements specification for an internal web-based business application that will next continue to be developed. Overall the whole requirements engineering process required a lot of interactions with the different stake-holders of the target software as well as support from others as well, being quite a human-centered process. Therefore, the research would not have been possible without all the people who participated in the realization of this project.

In conclusion, I would like to thank everybody who were part of the meetings where this research topic or the related subjects were discussed and the requirements were discov-ered. In addition, special thanks go to Mikael Ehrs and Asko Vakkila, who gave me the topic for this thesis, and most of the support and co-operation during the whole research and its requirements engineering activities. Thank you also goes to the thesis instructor Kenneth Widell, as well as the supervisor Jouni Lampinen for their support and partici-pation.

CONTENTS

ABBREVIATIONS

ADO	ActiveX Data Objects
ALM	Application Lifecycle Management
API	Application Programming Interface
ASP	Active Server Pages
DB	Database
DBMS	Database Management System
DLL	Dynamic-link library
EF	Entity Framework
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IIS	Internet Information Services
MDA	Model-Driven Architecture
MDAC	Microsoft Data Access Components
MS	Microsoft
MVC	Model-View-Controller
NFR	Non-Functional Requirement
ODBC	Open Database Connectivity
OLE DB	Object Linking and Embedding Database
PI	Product Information *(Wärtsilä organizational unit)*
RE	Requirements Engineering
RDBMS	Relational Database Management System
RDS	Remote Data Service
RUP	Rational Unified Process
SE	Software Engineering
SNAC	SQL Server Native Client
SQL	Structured Query Language
SRS	Software (or System) Requirements Specification

SSO	Single Sign-On
SysML	Systems Modeling Language
TCP/IP	Transmission Control Protocol / Internet Protocol
TDS	TechDataSearch *(Target application)*
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VB	Visual Basic
VPN	Virtual Private Network
WWW	World Wide Web
XP	Extreme Programming

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö

Tekijä:	Minne Paljakka	
Diplomityön nimi:	Vaatimusmäärittely verkkosovellukselle	
Valvojan nimi:	Prof. Jouni Lampinen	
Ohjaajan nimi:	M.Sc. Kenneth Widell	
Tutkinto:	Diplomi-insinööri	
Koulutusohjelma:	Tietotekniikan koulutusohjelma	
Suunta:	Ohjelmistotekniikka	
Opintojen aloitusvuosi:	2012	
Diplomityön valmistumisvuosi:	2018	**Sivumäärä:** 97

TIIVISTELMÄ

Tämä tutkimus on osa ohjelmistotuotantoprojektia, jonka tarkoituksena on parantaa jo olemassa olevaa yrityskäyttöön suunnattua verkkosovellusta, jota käytetään moottoriteknisen tiedon hakuun moottorityypeittäin. Tarve täysin uudelle järjestelmälle ja projektille on syntynyt, kun yrityksen liiketoiminta on kehittynyt ja sitä myöten synnyttänyt uusia ohjelmistovaatimuksia, jotka ylittävät olemassa olevan ohjelmiston ylläpidolliset rajat, sillä järjestelmän nykyinen teknologia on vanhentunut eikä näin ollen enää tue tarvittavien muutosten toteuttamista. Tutkimuksen päätavoitteena on tuottaa sekä toiminnalliset, että ei-toiminnalliset vaatimukset sisältävä vaatimusmäärittely tälle uudelle parannetulle ohjelmistolle. Lisäksi, tavoitteena on tarjota suosituksia projektin jatkumiselle sekä ohjelmiston toteuttamisessa käytettäville teknologioille ja työkaluille.

Tutkimus jakautui teoreettiseen ja empiiriseen osaan. Teoreettisessa osassa tutustuttiin sekä verkkosovellusten, että ohjelmistotekniikan teoriaan, keskittyen tarkemmin ohjelmistojen vaatimusmäärittelyyn. Empiirisessä osassa, tutustuttiin ensin olemassa olevaan järjestelmään, jonka jälkeen tehtiin yksityiskohtainen tutkimussuunnitelma, joka edelleen toteutettiin. Käytännössä ohjelmiston eri sidosryhmät tunnistettiin, jonka jälkeen vaatimukset kartoitettiin hyödyntämällä keskustelumuotisia haastatteluita yhdessä nopean prototypoinnin kanssa. Tuloksena saatiin materiaalia ohjelmiston vaatimuksista, josta analysoinnin, dokumentoinnin sekä vahvistamisen vaiheiden kautta toteutettiin lopullinen vaatimusmäärittelydokumentti. Lopuksi, esitettiin suositukset projektin jatkumiselle.

Tutkimuksen tärkeimpänä tuloksena saavutettiin vaatimusmäärittely uudelle parannetulle verkkosovellukselle. Toteutettu vaatimusmäärittely esittää sekä toiminnalliset, että ei-toiminnalliset vaatimukset järjestetyssä ja priorisoidussa luonnollisen kielen muodossa, sekä sisältää lisäksi tuotetun prototyypin, eli eräänlaisen paperimallin ohjelmiston käyttöliittymästä. Prototyyppi vaatimusmäärittelyn osana tarjoaa vaatimuksille visuaalisen esitystavan helpottamaan kommunikointia eri sidosryhmien välillä. Lisäksi, tutkimus tarjoaa suositukset ohjelmiston toteutuksessa hyödynnettäville verkkoteknologioille, sekä projektin etenemiselle. Kaiken kaikkiaan, tulokset toimivat syötteenä seuraaville ohjelmistotuotantoprojektin vaiheille, ja antavat vahvan pohjan projektin jatkumiselle.

AVAINSANAT: Ohjelmistotuotanto, verkkosovellus, vaatimusmäärittely, ohjelmistoprototyyppi

UNIVERSITY OF VAASA
School of technology and innovations

Author:	Minne Paljakka
Topic of the Thesis:	Requirements specification for a web application
Supervisor:	Prof. Jouni Lampinen
Instructor:	M.Sc. Kenneth Widell
Degree:	Master of Science in Technology
Degree Programme:	Degree Programme in Information Technology
Major of Subject:	Software Engineering
Year of Entering the University:	2012
Year of Completing the Thesis:	2018 **Pages:** 97

ABSTRACT

This research is part of a software development project that aims to improve an existing web-based business application that is used to access engine technical data per different engine types. The need for a completely new application and development project has occurred, because the organization's business has evolved and emerged new requirements that go beyond the maintenance of the existing system as the currently used technology is outdated and does no longer support the needed changes. The main intention of this research is to provide a requirements specification for the new improved application, including both the functional and non-functional requirements. Other objectives include giving recommendations for the continuation of the project as well as proposing the technologies and tools to be used in the actual implementation.

The research was divided into theoretical and empirical research. In the theoretical part the theory behind the web applications and software engineering were explored, concentrating more in detail on the requirements engineering activity. In the empirical part, the existing application was first inspected, and then the detailed research design was made and executed. In practice, the different stakeholders for the application were identified, and requirements were discovered by utilizing conversational interviews in combination with early prototyping. As a result, the requirements in their raw form were discovered, and finally turned in to the final requirements specification through analysis, representation and validation. Last, the recommendations for the project's continuation were given.

As a main result of this research, a requirements specification for the new enhanced web application was established. The produced specification gives both the functional and non-functional requirements in a prioritized and organized natural language form, but also includes the produced user interface mock-up prototype to provide more visual representation to easy the communication between the different stakeholders. In addition, the research gives recommendations for the web technologies and tools to be used in the implementation of the software, and provides suggestions for the continuation of the development project. Overall, the results will work as an input for the following development activities and give a good base for the project to proceed.

KEYWORDS: Software engineering, web application, requirements engineering, software prototyping

1 INTRODUCTION

Changes in software systems are inevitable as requirements change and new technologies become available (Sommerville 2011: 43). This is also the case behind this specific software development project: the emerged requirements go beyond the maintenance of the existing system, as the used technology does no longer support the required changes needed to be done to the system. Therefore, the need for a complete new application and development project has occurred. (See Maciaszek 2005: 26–27.)

1.1 Background

The research topic is assigned by the Product Information (PI) team that is part of Wärtsilä Marine Solutions. PI team’s work requires a lot of information collecting and handling, as they are responsible for the reporting and development of internal product information e.g. engine technical data which is stored in the technical database. This information and data is needed to give the other business units such product specific information which can be reused for example in the offering phase, product guides, manuals and installation planning instructions.

Digitalization is the hot topic in Wärtsilä today, and keeping up with the constantly evolving technology has always been important for the PI team. Therefore, the used databases, tools and applications must be updated every now and then. Furthermore, naturally also the constantly evolving business, as well as the growing number of products and therefore constantly increasing information needs have emphasized the existing development and upgrading needs even more.

This research is part of a one specific development project, which goal is to redesign an existing web-based business application, called TechDataSearch (TDS). This application is used to access engine related technical data per different engine types. This data is retrieved from an existing Performance database. The application is old: first introduced in 2001, and the technology used is therefore outdated which makes the application difficult to maintain and for instance to implement new functions to it. The most obvious problem however being that the application is seemingly slow which makes its usage too time consuming and stressful. The technology upgrading is certainly needed and it is at the same time seen as an opportunity to redesign the whole application as some improvement ideas have already emerged over the years.

1.2 Objectives and structure

The main goal of this research is to develop a requirements specification for the new and improved TDS web application – or more precisely for a *database driven web-based business application software*. This specification shall include both functional and non-functional requirements for the application. Moreover, other objectives of the research include: proposing the technologies and tools to be used in the actual implementation of the software solution, and giving recommendations for the continuation of the application development. In conclusion the research questions for this research could be formalized as follows, with the first question being the underlying main research question:

1. *What are the functional and non-functional requirements for the new application?*
    1.1. *What technologies and tools should be used in the implementation of the software?*

    1.2. *How should the application development continue after this research?*

Moreover, this research can be seen as being part of a software development project, involving the first phase of the requirements engineering, and enabling the project to continue from there to the next development phase, but also giving input to the other coming phases, as the information is important for all the actors in the development process: for the designers, programmers, testers as well as for the maintenance engineers. Furthermore, the development project concentrates only on the improvement of the application software, leaving out the related database, which either way may create some limitations for the actual realization of the software product.

The research consists of theoretical and empirical research. In the theoretical part, the basic theory behind the web applications and software engineering, including more detailed description of the requirements engineering, are going to be presented by exploring the related scientific literature. In the empirical part, the existing application is first going to be inspected, and based on that the more detailed research design shall be made. On a general level, the research will include discovering the requirements from the recognized stakeholders and then turning these raw requirements through different requirements engineering activities in to the final requirements specification document. In the end, the research will give the recommendation for the technologies and tools to be used in the implementation of the software, as well as for the continuation of the project.

2 THEORY

In this chapter, the research related scientific literature will be studied, and based on that the fundamental theory will be described, enabling the execution and better understanding of the thesis. Keeping in mind that although the world of software is constantly evolving, the basic principles still remain the same (Kleppmann 2017: x).

The target application being a database driven web-based business application, or briefly put *a web application,* this chapter covers the very basics around the Web and web-based database applications. However, the reader of this thesis is expected to have some prior knowledge about computer science in advance. Also, because this research is part of a software development project, the theory of software engineering, and more in detail the requirements engineering will be described.

2.1 Web applications and the Web

2.1.1 The Web

As we all know, *the Internet* is connecting together thousands of networks and millions of computers, enabling today’s modern society (Laudon & Traver 2016: 96–97). The Internet can be referred as a global set of interconnected networks (Block, Cibraro, Felix, Dierking & Miller 2014: 1), the most popular service of the Internet being *the Web,* also referred as *WWW* (Laudon et. al 2016: 96). It is the underlying technology that makes all the visual elements, like formatted text, pictures and videos possible and enables the non-technical users to benefit from the Internet (Laudon et al. 2016: 139).

The four fundamental components of the Web are *HTML*, *HTTP*, *a web server* and a *web browser* (Laudon et al. 2016: 140). With the Web, users can access linked documents and other *resources*, commonly written in HTML, via Internet with a web browser, such as Internet Explorer (Block et al. 2014: 1). This user side is usually referred as *a web client* or *a user agent* (Laudon et al. 2016: 107–108, 147–149).

The navigation of those resource requires the web client to send a *request* to the correct *web server* where the resource is stored. The server can then in return send a *response* to the client's request. This happens with a use of a standard *protocol*, a set of rules that enable the client and server to communicate, the most common protocol being HTTP. In other words, HTTP provides the methods that form the interface to the resource. HTTP is actually part of a bigger TCP/IP concept that includes the core communications for the internet. Every resource has also an URI – most commonly URL – that identifies the resource and enables the web client to find it. (Laudon et al. 2016: 104–108, 141; Block et al. 2014: 1–5, 12.) The server's response can be referred as a *representation*, which is the data returned from the specific resource, but not the actual resource (Block et al. 2014: 2, 5).

The resource is nowadays not limited for being just a static file or a ready-made web page stored in the web server and returned as a snapshot from the existing resource (Block et al. 2014: 3–5; Polvinen 1999: 176). It can also be a dynamic service, in which case the representation is created on the fly and the resource itself is *the logic* running on the server and the needed *data* is retrieved from *a database* or another data storage (Block et al. 2014: 5; Polvinen 1999: 176; Henderson 2006: 1). Using a database makes the data processing easier, especially when there are large amounts of data involved. The figure below presents the basic components and architecture of the Web.

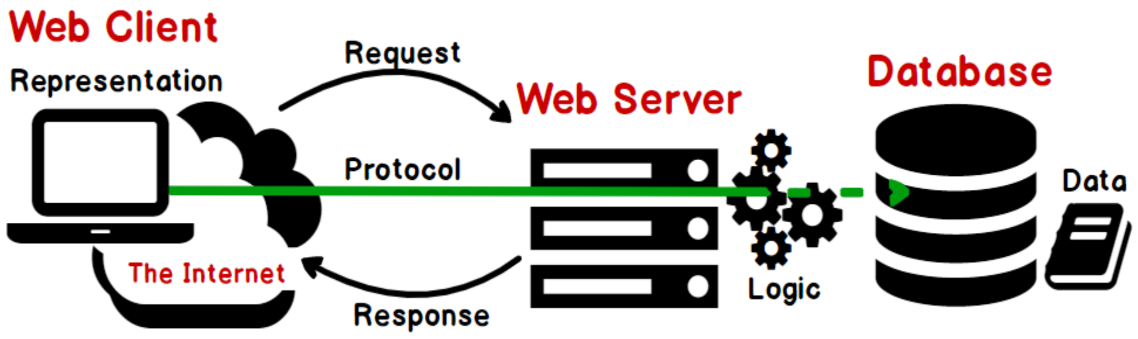


Figure 1. The basic Web architecture (see Laudon et al. 2016: 108).

2.1.2 Web applications

As the Web has grown, so has the amount of services accessed over it (Williams & Lane 2002: 1). A web application, or a web database application, is a service that is driven from a data stored in a database and accessed over the web. A web application sits somewhere between a web site and a desktop application, being kind of like the combination of the both, as it simply integrates the Web – characteristic of a web site – and databases – characteristic of a desktop application (Williams et al. 2002: ix, 1; Henderson 2006: 1). A web application enables users to access the data easily, by offering good accessibility and usability, and the database makes it possible for the application to store, manage and retrieve data. (Williams et al. 2002: ix.)

The Web and databases are many times brought together with a so called *three-tier architecture model*: three layers of application logic (Williams et al. 2002: 1). At the base of the three-tier model is a *database tier* that consists of a database management system (DBMS) and a database itself that is managed by the DBMS (Williams et al. 2002: 1). On top of the database tier is a *middle tier*, where is most of the application logic. The middle tier also communicates the data between the other tiers. And finally, on the top of the others there is a *client tier*, where usually is the web browser software that interacts with the application. (Williams et al. 2002: 2.) In the next figure, the basic web database application architecture is presented using the three-tier architecture model.

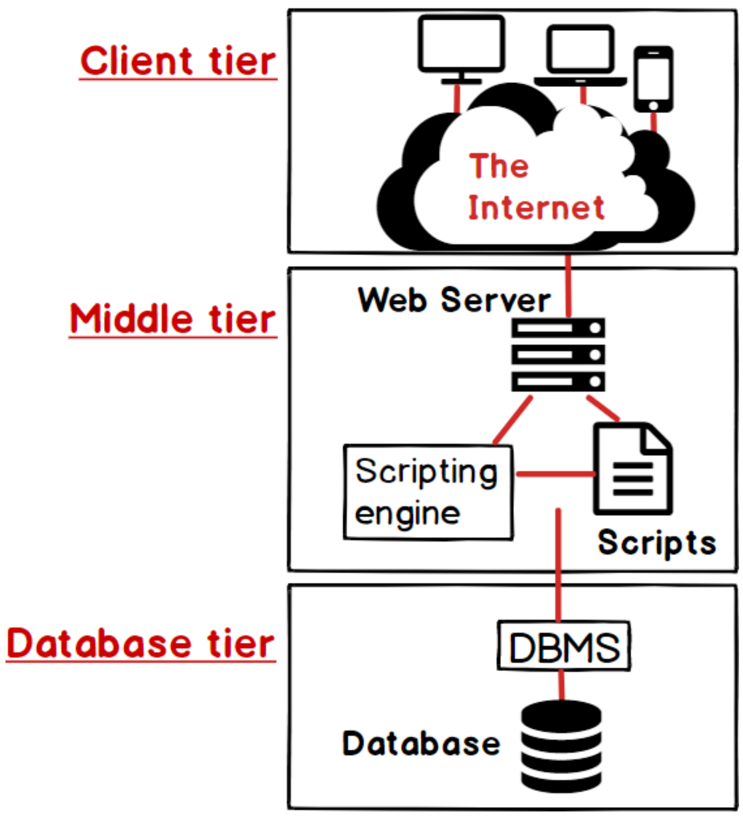


Figure 2. Web application architecture using the three-tier model (Williams et al. 2002: 2).

The model provides a conceptual framework for the web applications (Williams et al. 2002: 3). The benefits of the three-tier architecture model include that any user that has a web browser can access the web database application without having to build or install any additional software or for instance use a specific operating system. In this case the browser can be called a *thin client*, which means that only a little application logic is needed on the client side, and that is to send the HTTP requests and display the responses. (Williams et al. 2002: 5.)

Java applet is one alternative for the thin client, which enables more customization compared to the traditional browser solution. Also, two-tier models can be used, where there is only a client and a server tier. In this case the application logic is mostly on the client side and the server tier is directly the DBMS. However, this requires the software to be provided for each user separately. (Williams et al. 2002: 6). In this case the client side can be referred as a *fat client* (Polvinen 1999: 22).

Moreover, the three-tier architecture model can also be extended to *a multi-tier* alternative – also referred as *n-tier* or *layered* model. For instance, a four-tier model can include separate tiers for web client, web server, application server and database, which for instance in case of complex applications enables the workload to be distributed among multiple servers. However, the tiers in these models are more or less arbitrary, which means that the different layers could be split further into more layers. (See Sommerville 2011: 157–158.)

In the three-tier solution, the middle tier usually holds the majority of the application logic, and therefore it is a complex one. It usually consists of a web server, a web scripting language and its scripting language engine, as can be seen from the previous figure (figure 2). (Williams et al. 2002: 7.) The middle tier processes the input coming from the client in the form of a query to create, delete, modify or read data, and it also forms the structure and content of the data to be displayed on the client side (Williams et al. 2002: 2, 7).

The database tier manages the data: stores and retrieves it, as well as manages updates, allows multiple middle tier processes simultaneously, provides security services like data backup, and so on (Williams et al. 2002: 7, 12). It is the base of the whole three-tier architecture and therefore the database is usually the first one to be designed and build in a development process (Williams et al. 2002: 11). Managing the data requires the DBMS software, where usually SQL is used as a query language to access the data (Williams et al. 2002: 12).

In most cases, the DBMS is more specifically defined to be a relational DBMS (RDBMS), which is based on the relational model invented by Edgar F. Codd, but there are also non-relational software choices, such as search engines, document managements system, and so on (Williams et al. 2002: 12; Polvinen 1999: 2, 6). In the case of a relational database, the data is organized in tables that have relationships that link them (see Polvinen 1999: 2–3).

The DBMS servers as an interface between the database and application, or directly with the user. However, there are also other standardized interfaces available that utilize this existing interface, and enable applications to access and interact with the database. (Polvinen 1999: 163, 176.) In fact, there is a huge variety of different web tools, environments and frameworks available across the Web these days that support and enable the web application development (Kleppmann 2017: x). The modern applications also commonly utilize the web services provided by other web sites in their own solutions (Jamsa 2005: 38). APIs and other supporting web tools make this relatively easy, as for instance, an application can display location information by utilizing the Google Maps API.

I think I have now covered the very basic concepts around the web applications, however I must also point out another architecture model that has become widely popular next to the layered architecture model. It is so called MVC (*Model-View-Controller*) model or pattern. This architecture model as well achieves separation and independence of the system components, like the layered architecture does, which is fundamental as it allows changes to be localized i.e. components to be changed independently. (Sommerville 2011: 155, 157.)

In the MVC model the system structure is divided into three logical components of model, view and controller. In short, the model component manages the data and the associated operation, the view is in charge of how the data is presented to the user and the controller manages the user interactions and passes them to the view and model. (Sommerville 2011: 155.) Furthermore, the layered model and MVC pattern are not exclusive, meaning that they can also be used in combination, which is typically implemented so that the view and controller together form the user interface component i.e. the client tier (e.g. Rawsthorne 2011). The next figure presents the web application architecture using MVC pattern.

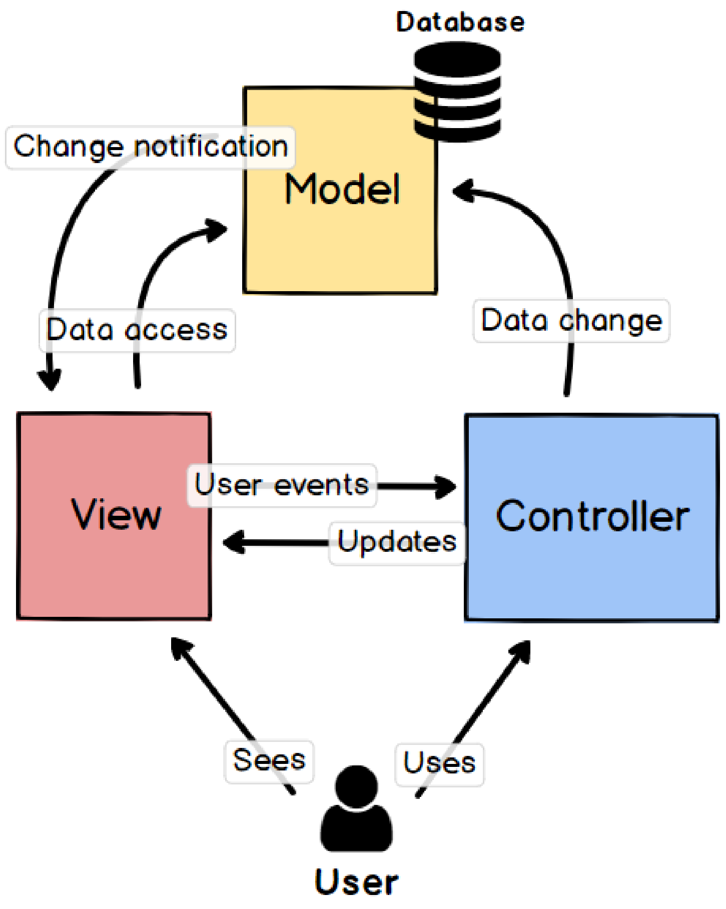


Figure 3. Web application architecture using the MVC pattern (see Sommerville 2011: 157).

2.2 Software engineering

Our modern society couldn't function without software, as everything is being computerized, as for instance nowadays most industrial, entertainment and financial systems all run with software. However, there are also software failures – exceeded budgets and deadlines, unmaintainable, unreliable and unsecure systems, you name it – and some of them may be the consequence of ignoring the software engineering methods and techniques. (Sommerville 2011: 4–5; Maciaszek 2005: 1.)

According to the IEEE (2002: 70) software engineering (SE) is the application of engineering, i.e. a systematic, disciplined and quantifiable approach, to the development, including both operation and maintenance, of software. Moreover, the software does not only refer to a single computer program, but includes also the surrounding procedures and the associated documentation (IEEE 2002: 69; Sommerville 2011: 6). Overall, the aim of SE is to support the professional software development. Typical to professional software is that it is developed for a specific business purposes and it is usually intended to be used by someone other than the developer itself. Software engineering was first discussed in a conference called “Software crisis” in 1968, as it was clear that large complex software systems would require people working together in form of projects and the use of systematic approaches would be crucial to accomplish any success. (Sommerville 2011: 5.)

There are many different types of software that also require different kinds of development approaches, and although many general issues may affect many different types of software, every software development process – the specific way of doing things – is unique and varies from case to case. Therefore, software engineering does not only give one universal engineering method to be used for every software, but it includes different approaches, techniques and other engineering disciplines that support all the aspects of software development from the requirements specification to maintaining the system as it is in use. (Sommerville 2011: 4–7, 10; Maciaszek 2005: 30.) However, it is still today argued that the substantial theory of software engineering is lacking, although it has been shaped for over fifty years (see Hall & Rapanotti 2017). Moreover, the development of the Web has naturally had a huge effect on how we develop software today (Sommerville 2011: 13, 509).

2.2.1 Software engineering activities

Software development process divides the software development into separate phases, and can be seen as the process that translates the user needs into software product, typically also including the maintenance of the final product (IEEE 2002: 70, Maciaszek 2005: 21). Although every software development process is different, there are still those fundamental software engineering activities that are applicable to all software processes, and the most effective way to produce high-quality software within the schedule and budget, is to adopt these activities. Those activities include: *software specification, software development, software validation and software evolution*. (Sommerville 2011: 6–10; Pressman & Maxim 2015: 17.)

Software specification defines the software that is to be developed – it includes the customers and engineers defining what the system is supposed to do, and whether there are any constraints existing. Software development includes typically the design and actual programming work to produce the software. Software validation, includes the software product being checked against the customer requirements, so that it matches to their wishes. And finally, the software evolution reflects the activities where the software is modified to fulfil the changing customer and market needs. (Sommerville 2011: 9.)

Of course, in practice there are multiple variation existing, and these activities are typically separated in to multiple different activities and/or overlapped, or they may be performed iteratively, and so on (Maciaszek 2005: 21; IEEE 2002: 71). The next chapter will discuss more about the different software process models that give their own guidance to the software development process.

2.2.2 Software process models

Software process models have been developed to guide the whole process of software development, and they can be seen as simplified representations of the whole software process where each model views the process from a specific perspective (Haikala et al. 1998: 25; Sommerville 2011: 29). Because each model only focuses on a particular approach, they only provide partial information about the whole process; they provide a framework, leaving out the details, being only generic models rather than definitive descriptions. These generic models can be extended, mixed together or otherwise remodeled in order to create a specific purpose software process. (Sommerville 2011: 29; Maciaszek 2005: 30.)

There are a lot of different types of generic models existing – such as the spiral model, rational unified process (RUP), model-driven architecture (MDA), extreme programming

(XP), just to mention a few (see Maciaszek 2005: 30–36; Sommerville 2011: 29–36; Pressman et al. 2015: 40–65). I will only give examples from a couple of prescriptive i.e. traditional process models, which concentrate more on the structure and order in the software development (see Pressman et al. 2015: 41): the waterfall model and the incremental development, as not all models can possibly be discussed here. However, I must say that in practice all modern software processes are invariably iterative and incremental on some level (see Maciaszek 2005: 6), and the trend in software development has clearly shifted from plan-driven ways towards more agile approaches.

The waterfall model is the most well-known process model existing – also referred as software life cycle – which suggests a systematic and sequential approach to software development. It utilizes the fundamental process activities, described earlier, and separates them into different phases, such as: *(1) requirements analysis and definition, (2) system and software design, (3) implementation and unit testing, (4) integration and system testing, and (5) operation and maintenance*. However, the phases typically vary between different interpretations. The waterfall model is an example of a plan-driven process, where in principle all the process activities should be planned and scheduled at first before starting the process. (Sommerville 2011: 29–31; Pressman et al. 2015: 42.) One basic and simplified structure of the waterfall model is presented below:

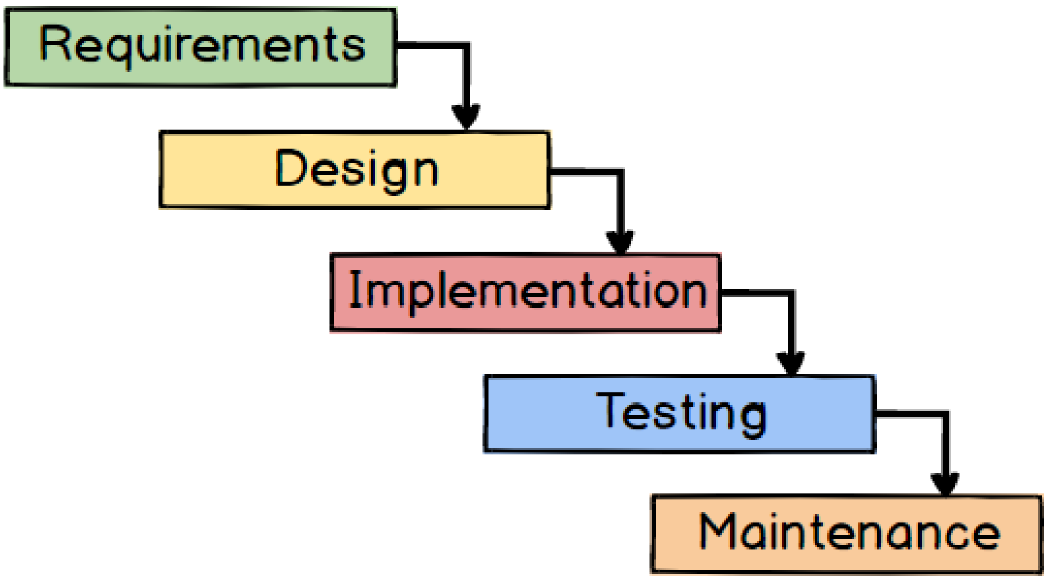


Figure 4. Example of the waterfall model (e.g. Sommerville 2011: 31).

The example contents of the different phases of the waterfall model are described below (Sommerville 2011: 31):

> *Requirements analysis and definition*: The system's requirements, including the functionality, constraints and goals are gathered, by discussing with the system users, and defined in detail in a system specification.
>
> *System and software design*: The System design involves the development of an overall system architecture, and software design identifies and describes the essential software components and their relationships.
>
> *Implementation and unit testing*: Here the software design is programmed and realized into actual software program units. Unit testing can be used here to check that every program unit meets the specification.
>
> *Integration and system testing*: The whole system is integrated together, and tested as a complete system – system testing – to compare it against the software requirements to see if the requirements are met.
>
> *Operation and maintenance*: Normally this is the longest phase, as it not only involves the system installation for the actual use, but also the maintenance of the system, including correcting the remaining errors, and improving the system as the requirements change over time.

In theory, all the process phases should be documented and the next phase should start only after the previous step is finished and approved. In practice of course, this does not always happen as real projects rarely follow the sequential process flow, and the phases are likely to overlap. For instance, during the design phase problems with the specified requirements are typically identified, and the information needs to be added to the requirements, and during the programming phase, problems with the design arise, and so on. As feedback is given from one phase to another, the made documents need to be changed, which causes rework and costs, and is therefore seen as one of the down sides of the waterfall model. (Sommerville 2011: 31; Pressman et al. 2015: 42.)

Furthermore, usually if multiple iterations are needed some parts of the process can be frozen, and the problems are left to be solved later (Sommerville 2011: 31). This problem is so called “moving target problem”, where requirements change and technologies evolve during the development process. Moreover, in classical software engineering the software product is in theory developed from the scratch, rather than reusing existing software, which may be impractical. (Schach 1999; Schach & Tomer 2000.)

The clear advantage of the waterfall model however is that the process is visible to the managers, as they can see the progress directly by comparing the development plan against the produced documents. But as referred, the down side of this is that the process is inflexible as it is more difficult to respond to the possible changes, such as to new customer requirements. That said, the waterfall model is ideal when the requirements are unlikely to change and they are well understood. Moreover, the waterfall model having a formal approach, makes it suitable for system that have high requirements for things like security and safety, because it can be clearly seen whether the system fulfils the requirements. Overall the waterfall model doesn’t really give any cost benefits compering to other models, but it is commonly used likely because it is relatively easy to manage. (Sommerville 2011: 32.)

The incremental development process model on the other hand relies on the practice where the system functionalities are divided into small portions called increments. First the initial increment of the system is created – including the most important functionality – and it is presented to the customer in order to get feedback, and then the system is evolved by adding the next functionality and again exposed to the customer. This goes on until the final version of the system is developed. (Sommerville 2011: 32–33.)

Furthermore, each increment is delivered by going through the fundamental software process activities, from the requirements to the deployment. Although in this case the activities are usually interleaved, and so the incremental development combines the elements from linear i.e. sequential and parallel process flows. (Sommerville 2011: 32–33; Pressman et al. 2015: 44).

Moreover, incremental development is often used together with iterative development, and they get easily confused. Iterative development basically refers to going through the whole system with each iteration improving it, as the incremental approach only adds something new to the previous increment, like a new functionality. (Spence & Bittner 2005.) The figure below demonstrates one simplified model of the incremental development.

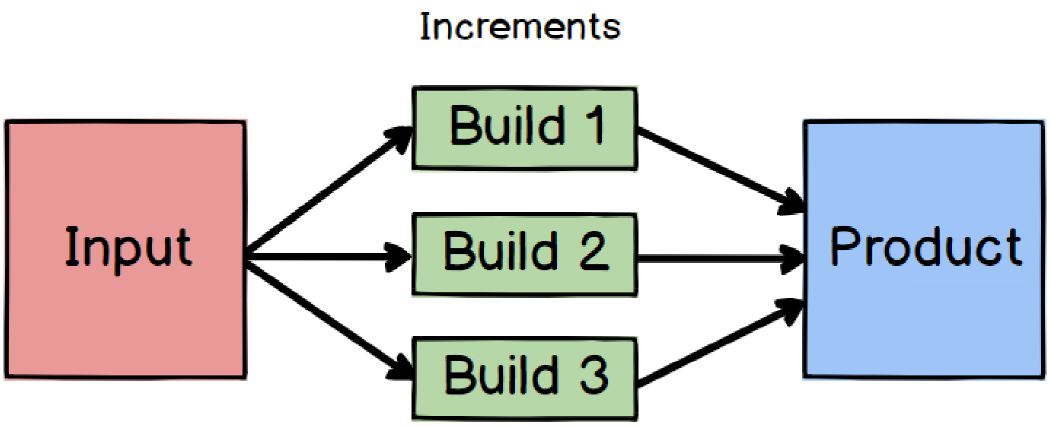


Figure 5. Example of the incremental development model (Lönnfors 2012: 16).

Incremental development is essential in agile methods, as it reflects the way people solve problems in natural bases: step by step by going back when a mistake occurs. The advantages compared to the waterfall model include that it is easier to make changes to the software while it is being developed. Also, it is easier to get customer feedback as it is easy to show the made progress to the customers, as different types of documents may not be so easy for them to understand. Moreover, it is even possible for the customers to utilize the made intermediate increments and benefit from them before all the functionality is implemented to the system, which enables in a way a more rapid delivery, compared to the waterfall model, although the final product might not be delivered any faster. (Sommerville 2011: 33.)

Overall the incremental development can be either plan-driven or agile, or a combination of some sort. In plan-driven approach the increments are to be specified in advance, as in more agile approach the increments depend on the progress and customer priorities that arise on the way. In some form, incremental development is nowadays the most common approach used in application development. However, there are still clear difficulties existing: as from the management perspective, the progress is not visible enough, because it is costly to produce good documentation for every increment, and also the system structure usually breaks down, as it is hard to control it when changes are coming in small pieces (Sommerville 2011: 34.)

These problems are more or less emphasized in agile approaches. Agile approaches are even argued to perform weak, as they are referred to use a so called lazy management method: the documentation seems to be off, projects are overrunning because they can't be properly managed, and usually no significant benefits from using these methods are achieved (e.g. Ladas 2017). The problems are of course more serious within complex, long life-time systems and with distributed software development via remote teams, which emphasizes the lack of visibility, documentation, project coordination, and control of the software structure (Sommerville 2011: 34; Ricvi, Bagheri & Gasevic 2015).

2.3 Requirements engineering

As said earlier one of the fundamental software engineering activities is software specification which is related to the requirements analysis and definition phase, and is more formally referred as requirements engineering (RE). In general, RE is a process that includes the discovering and documenting of the software requirements (Laplante 2014: 2–3; Sommerville 2011: 36, 82). It is a critical stage in the software development process, as faulty implementation typically leads to problems in the later stages of the development (Sommerville 2011: 37), and in fact, the missing and incorrect requirements are one of the biggest reasons why software development projects fail (see Paakki & Taina 2011: 6).

In order to understand RE, it is important to understand what requirements are (Laplante 2014: 3). Requirements are descriptions of what a system should do: the services that it provides but also the constraints for its operation and development (Sommerville 2011: 36, 83); it is a condition or a capability that must be met by the system or its component (IEEE 2002: 65). Requirements aim to reflect the customer needs for a specific system that has a certain purpose (Sommerville 2011: 83). In practice requirements can vary from high-level abstract statements to formal mathematical specifications, due to the different forms of representation and the fact that stakeholders have different needs (Laplante 2014: 3).

In the end, the whole RE process aims to produce an agreed requirements document that describes the system while satisfying the stakeholder requirements. Usually the requirements are presented at two levels of detail, so that both the end-users and customers but also the system developers can benefit from the document, as the first group is likely to benefit more from high-level easy-to-follow requirements and the developers need more detailed information. (Sommerville 2011: 4–5.)

The first level discussed can be referred as *user requirements* and the other one as *system requirements*. A clear separation between these two levels is recommended, because otherwise problems may arise due to the confusion, as the separation helps different types of readers to understand and communicate about the requirements. (Sommerville 2011: 83–84; Laplante 2014: 4–5.) The two levels may be described as follows (Sommerville 2011: 83, 94; Laplante 2014: 4):

1. ***User requirements*** are abstract more general statements of what the system should do: what services it should provide and what constraint there exists. They should preferably be written in natural language and may include supporting informal diagrams or other forms of representation. Overall, they should be described in a way that the non-technical persons, like the system end-users, can understand them.

2. ***System requirements*** are expanded from the user requirements, as they add more detailed descriptions of the features, functions, services and operational constraints of the system. They shall define how the system should provide the user requirements i.e. define exactly what is to be implemented, in a more structured and precise manner. System requirements are naturally derived from the analysis of the user requirements, and they are mainly intended for the system developers, but also for the system testers and maintenance persons.

The next example demonstrates how a user requirement can be expanded into several system requirements, and illustrates the distinction between them: from general to more detailed. User requirement is the main requirement as the system requirements are presented as sub-requirements. The example below is from a health care system (Sommerville 2011: 84).

1. *The system shall generate monthly management reports showing the cost of medication prescribed by each clinic during that month.*
   - *1.1. On the last working day of each month, a summary of the medicine cost and the prescribing clinic shall be generated.*
   - *1.2. The system shall automatically generate the report for printing after 17.30 on that day.*
   - *1.3. The report shall be generated for each clinic and shall list the individual medicine names, the total number of prescriptions, the number of doses, and the total cost.*
   - *1.4. If medicines are available in different dose units they should be listed separately.*
   - *1.5. Access to all cost reports shall be restricted to authorized users listed on management access control list.*

Design specification is also sometimes listed as a third level of the classification together with the user and system requirements, as the system design specification is typically derived from the system requirements (Laplanta 2014: 4).

2.3.1 Different types of requirements

The software requirements are also often classified as functional and non-functional requirements based on the type of the requirement, as earlier they were classified according to their level of detail and the different readers of the requirements (Sommerville 2011: 83–84; Laplante 2014: 4–6). These functional and non-functional requirements can be either user or system requirements. There exist also other ways to classify requirements (see Laplante 2014: 6, 10–11), but I will concentrate on these two requirement types, as they are the most common ones. These two can be described shortly as follows (Sommerville 2011:84–91; Laplante 2014: 6–10; IEEE 2002: 34; Benyon 2014: 140; Paakki et al. 2011: 28):

1. ***Functional requirements*** define what the system or its component is supposed to do; they are the requirements for the services and functions that it should offer or be able to perform. For instance, they specify: how the system should react to certain inputs, and how the system should behave in certain types of situations, what are the system outputs and exceptions, and so on. In contrast, they may also specify what the system must not do.

2. ***Non-functional requirements*** (NFRs) define the quality that the system must have; they are the terms and conditions that define how the system shall fulfill the functional requirements i.e. the functions provided by the system or its component. For instance, they may include constraints related to timing, budget, quality, development process, programming language to be used, constraints set by standards, and so on. They are generated by the system environment. The NFRs usually apply to the whole system rather than one specific functionality, and they may have an effect to the whole architecture of the system. Furthermore, for instance according to the IEEE (2002: 23, 38, 41, 56, 57) The NFRs can further be divided into *performance, design, implementation, interface* and *physical* requirements, but also other classifications exist. Moreover, it is recommended that the NFRs are written quantitatively, for instance rather than saying that the system should

be easy to use, it would be better to say that the users need to be able to use the system after four hours of training.

The distinction between the functional and non-functional requirements is rarely as simple in reality as described above. One good example for instance is that, if there is a user requirement concerning security, such as a statement to limit system access only to authorized users. This requirement appears to be a non-functional requirement, but when going in more detail, it may also generate functional requirements. In this case for instance a functional requirement would be a need to include user authentication facilities in the system. Moreover, this also shows that requirements are not independent and they affect other requirements: for instance, they create restrictions for other requirements and generate new requirements. Requirements may also specify functionality to ensure that other functionality or services are delivered properly. (Sommerville 2011: 85–87.)

2.3.2 Requirements engineering activities

In general, the RE process may involve four main activities, which are *(1) feasibility study, (2) requirements elicitation and analysis, (3) requirements specification and (4) requirements validation and verification* (Sommerville 2011: 37–38, 99). These activities cover the following: assessing whether the system is useful to the business, discovering and analyzing the requirements, representing the requirements in form of documentation, and finally checking that the requirements describe a system that the customer wants. Although in reality, the activities are more or less overlapping, and the RE process is more of an iterative than linear process, as for instance some requirements may be already documented when new requirements are generated, and so on (Sommerville 2011: 38, 99–100.) Next these activities will be discussed more in detail.

Feasibility study should take place early in the RE process. In some cases, the feasibility study is more relevant than in others, as sometimes it is not so clear whether the project creates any benefit, is cost-effective from a business point of view, or whether it is even possible to be executed in practice. The key questions that should be answered are: (1)

can the system be implemented using the currently existing technology within the schedule and budget, (2) does the system support the organization's objectives, and (3) can the system be integrated to other systems that are used. If the answer is no to even one of these questions, the project should probably not be continued any further. (Sommerville 2011: 37, 100.)

Moreover, the feasibility study should concentrate on the factors that are important for that specific project, which might for instance be an economic, technical, operational, scheduling, legal or political factor. A proper study requires evaluating these factors in order to clarify whether the project is feasible, as usually there are limited resources to complete the project. For instance, assessing the technical feasibility would be essential in a case where the system is complex and there is a need to gain understanding of the organization's ability to construct the system. (See Hoffer, George & Valacich 2014:145–157.) Moreover, the study should be relatively quick and cheap, at least in most cases (Sommerville 2011: 37).

Requirements elicitation and analysis is the activity where the requirements are discovered – *discovery can be used as a synonym for elicitation* – by observing the existing systems, and discussing with the customers and system end-users. The idea is to find out what services the system should provide, the required system performance, the existing constraint, and other things related to what the customer wants and needs. Some requirements will be more obvious than others, and therefore there are multiple different techniques to help to discover those hidden requirements. This process also includes discovering who the system users and customers even are in the first place. (Sommerville 2011: 37, 100; Laplante 2014: 11–12.) Moreover, the requirements elicitation and analysis can be divided into smaller sub-activities including *(1) stakeholder identification, (2) requirements elicitation, (3) requirements classification and organization, and (4) requirements prioritization and negotiation,* which are presented below.

Stakeholder identification can be seen as a preparing activity for the requirements elicitation (see Laplante 2014: 29), as the elicitation typically includes collecting the requirements from the stakeholders. Stakeholders consists of all the people that have stake in the software project, in other words, people that are somehow affected by the system or have influence on the system development. They can range from end-users to managers, and even include external stakeholders like regulators, who certify the acceptability of the system. Commonly the two main stakeholder groups include the customers – *typically including system end-users and system owners* – and the developers – *typically including analysts, designers, programmers, as well as the test and maintenance engineers*. Moreover, in practice the term user is commonly used to mean the customer. (Sommerville 2011: 103; Maciaszek 2005: 4.)

In the end information systems are social systems, and the success of the software project really depends on the social factors rather than technology used or something else. The skill of the developers, good management practices, and commitment of the customers are all important. (Maciaszek 2005: 4–5.) Concerning the requirements elicitation, the aim of the stakeholder identification is to avoid situations where potential stakeholders are left out, and in contrast, to make sure that the right stakeholders are included. In case where ignored stakeholders are discovered later, in the worst-case scenario when the system has already been built, changes to the requirements might be very costly. (Laplante 2014: 33.)

Different stakeholder and user groups have typically different needs and wishes for the system, and they also may need to be treated differently, for instance, children may require different elicitation techniques compared to adults. Moreover, the different groups may overlap, which also needs to be taken into account. (Laplante 2014: 50–51.) Furthermore, the stakeholder identification and analysis may cover the three activities presented below (Laplante 2014: 50). However, also activities like stakeholder prioritization may be appropriate in some cases, as sometimes different stakeholder needs and desires conflict, and the prioritization helps in solving these kinds of situations (see Laplante 2014: 41).

1. *Identifying all the stakeholders*

   One approach to identifying stakeholders is to answer a set of questions, such as: who is going to use the system, who is paying for the system, who is going to judge the system, what laws govern the system, who will be negatively affected by the system, who is involved in any part of the software life cycle, and so on (see Sommerville 2014: 33).

2. *Dividing stakeholders into classes according to their scope, interest, authorization, or other discriminating factors*

   Most of the stakeholders are likely to be included in the system end-users or owners, but some stakeholders like regulators may not even be involved in the requirements elicitation process and they need to be separated. Also, dividing the system users into smaller classes is important in order for the elicitation process to be economic. As in most cases it is not possible to take every individual into account, and therefore the different kind of users should be considered as larger entities. (Laplante 2014: 35.)

3. *Selecting a representative person or group for each stakeholder class*

   As just stated, all the users can rarely be considered as individuals, so a representative individual or small group should be selected for each of the identified classes, so that this selected person or group can be contacted during the RE process. (Laplante 2014: 35.)

Requirements elicitation is the activity of collecting information about the required system and the existing systems, and then finding and separating the user and system requirements from this gathered information. Requirements may come from different information sources. They might come from stakeholders, documentations of existing systems, specifications of similar systems, and so on. However usually the most obvious and important source of information are the system stakeholders. (Sommerville 2011: 103.)

There is a wide range of different techniques existing that can be utilized in the requirements elicitation process. For instance, the interactions with the stakeholders may include techniques like interviews and task analysis, and other techniques, such as scenarios and prototypes, may be used to help the stakeholders to understand the system better (Sommerville 2011: 103). It might typically be beneficial to use different techniques with different user classes, and as well to use multiple techniques to complement each other’s

(Laplante 2014: 49, 75). Some examples of the existing techniques are listed below (e.g. Laplante 2014: 49–50, Sommerville 2011: 104–106).

- Brainstorming
- Card Sorting
- Domain Analysis
- Group Work
- Interviews
- Introspection
- Prototyping
- Questionnaires / surveys
- Scenarios
- Task analysis
- Use Cases
- User stories.

Requirements classification and organization is the activity that takes these collected unstructured requirements, and organizes them in groups. If the previous activity was concerned of discovering the requirements, this and the next activity will concentrate on the analysis part. The most common way of grouping the requirement is to identify the subsystems from the system architecture model and to associate requirements with each subsystem. However, this obviously means that the architectural design cannot be a completely separate activity. (Sommerville 2011: 101.)

Requirements prioritization and negotiation is the activity concerned with prioritizing the requirements, and finding any problems and resolving these through negotiations, as conflicts are common when multiple stakeholders are involved. Problems with requirements in their raw form may for instance include those that are: confusing, extraneous, duplicated, conflicting, missing, and so on. In most cases it is recommended that stakeholders meet to resolve differences and agree on the compromises that need to be made. (Sommerville 2011: 101; Laplante 2014: 12, 81.)

Requirements specification is the activity of documenting the information gathered in the requirements analysis into a requirements document. This document typically includes both the user and system requirements, and in this case the document can be called a *Software (or System) Requirements Specification* (SRS). (Sommerville 2011: 37–38, 91, 94; Laplante 2014: 93.) The SRS is a formal document that describes what the system developers should implement. These documents are needed especially when the software development is outsourced or the development is distributed via remote teams. (Sommerville 2011: 91.)

There are also other alternatives for the SRS, like the possibility to present the user requirements as an introduction to a separate specification of the system requirements. Moreover, in agile development approaches it is argued that documentation is outdated while it is written, as requirements change so quickly. These approaches present other alternatives for the whole concept of documentation. (See Sommerville 2011: 91.) However, I will concentrate on the actual documentation, although for some systems agile processes might bring benefits.

The requirements document has usually multiple different types of users that may include, for instance the customers of the system, project managers, system developers, and test and maintenance engineers. Moreover, this means that also the document should communicate the requirements to all of these users, by including general descriptions for the customers, detailed description for the developers and testers, and information about the presupposed changes – i.e. the evolution – of the system for the maintenance engineers, and so on. Although in practice the realization may require compromises. (Sommerville 2011: 91–92.)

Naturally the type of the system being developed, as well as the development process and approach chosen, effects on what the document should be like: what information it should include, and in what level of detail. For instance, in safety and security critical systems more detailed requirements specification is needed as these critical factors need more detailed analysis. Also, if the system development is outsourced more precise approach is required, compared to a case where the development happens inside the organization and all the possible confusion can more easily be detected and resolved during the development process. (Sommerville 2011: 92.) Moreover, there is also an IEEE standard for the requirements document, IEEE 830, which among other things suggests possible contents for the SRS document (see IEEE 1998).

One important thing concerning the requirements is that they should not concentrate on how the system should be designed or implemented i.e. they shall not specify how the technology will meet the requirements, but only on the services it provides and the constraints that the system has. These invalid design items may for instance include requirements that specify the partitioning of the software into modules, or define any data structures to be used. The how-question shall be answered later in the development process as part of the design activity. Although in practice this cannot be completely avoided as any reuse of existing components or interoperability with other systems inevitably creates design related or architectural constraints and requirements for the new system. (Sommerville 2011: 94–95; Benyon 2014: 140; IEEE 1998: 10.)

In addition, how the requirements are documented in the SRSs may vary. Although, typically user requirements are almost always written in a natural language form, there may also exist some other forms of notation, like diagrams, tables or sketches, to support the written user requirements. Moreover, more variation typically exists in documenting the system requirements, where also natural language is appropriate, but other forms of representation, like graphical or mathematical system models, may be used. Typically, some combination of different notations is preferred. (Sommerville 2011: 95; Laplante 2014: 12.)

Moreover, there is really no documentation format better than the other, except in case of a hard-to-read, badly organized and messy document. On a general level there are three approaches to requirements specification, which include: *formal*, *informal* and *semi-formal* approaches, but as referred above, usually SRSs contain elements of at least two of them. Formal representation forms have a rigorous mathematical base, but even these formal specifications typically include elements of the others. Informal techniques on the other hand include techniques like natural language or flowcharts, and the fact is that even the most formal SRSs' typically must use natural language. Semi-formal techniques in contrast include, many of the diagrams in UML (Unified Modeling Language) and SysML (Systems Modeling Language). (Laplante 2014: 83, 94.)

Requirements validation and verification is the process of checking the requirements for realism, consistency and completeness, and determining whether the specification is correct presentation of the customer’s needs. The process typically includes activities like reviews, analysis and testing in order to ensure that the customer needs are understood and the system satisfies those needs. Validation answers the question: “Am I building the right product?” and verification answers the question: “Am I building the product right?”. During this activity any errors discovered in the SRS document are naturally corrected. (Sommerville 2011: 38, 110; Laplante 2014: 12, 108.)

3 DESCRIPTION OF THE EXISTING APPLICATION

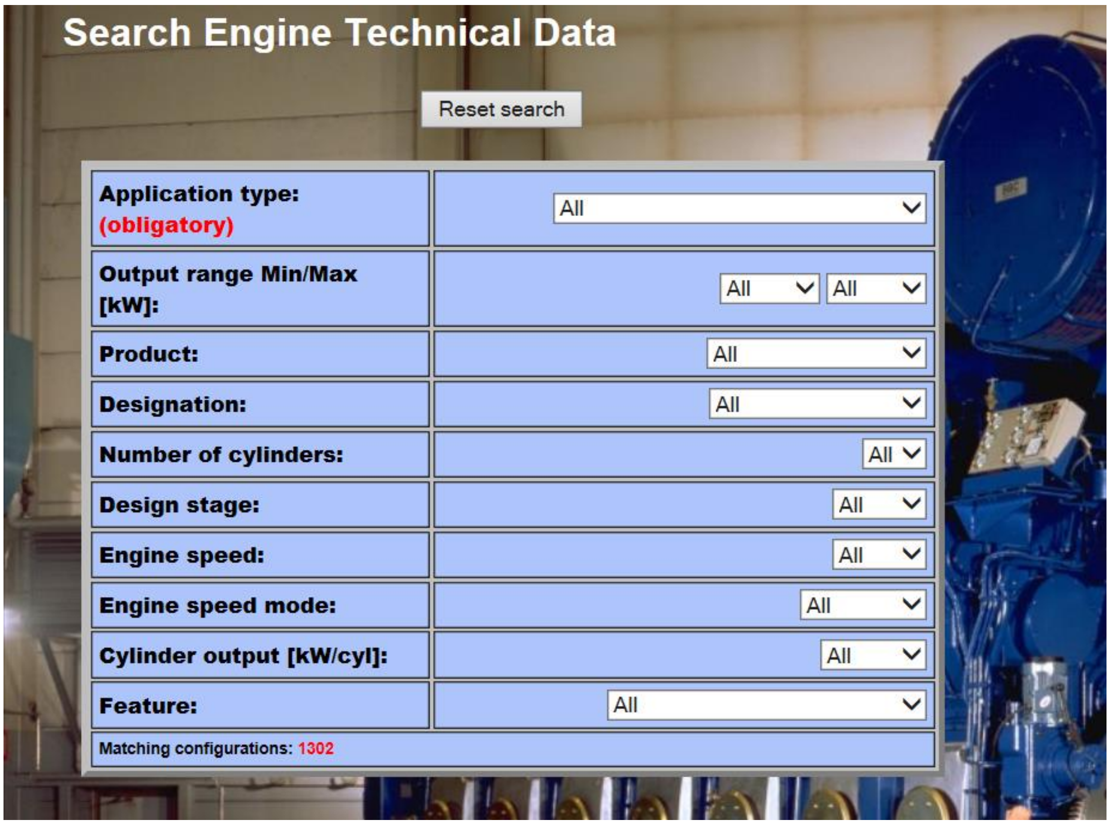
The intention of this chapter is to look more deeply into the already existing application, as there is no documentation made concerning the application that could be utilized. This inspection is seen important as the very basic logic and functionality behind the application will stay the same, as will also the database component of the application, and the underlying application type and therefore the main architectural aspects. So there will be some reuse of components and ideas from the existing application which will speed up the development process compared to starting the development from a scratch. In addition this chapter aims to provide the reasons for the emerged upgrading needs.

3.1 Overview

The application is an internal web-based business application, which primary purpose is to provide the different internal parties of the company with the engine related technical data, which resides in the Performance database. Originally the application was developed for Marine Solutions Sales department in 2001, and it has stayed mostly the same since. So the content of the application is created mainly only based on the needs of the Sales department, but also taking into account the fact that everybody in the organization has access to all the data that is available through the application. However, during these years the application has gotten feedback and improvement ideas, and also the other departments have shown interest towards the application.

In general, the existing application does no longer meet today's business requirements, as the business has evolved and created new requirements which cannot be fulfilled using the current technology of the application. Therefore, the application is not as widely utilized as it has not been found useful enough and its usage has been found too time-consuming. The potential users have instead utilized the related manuals or asked the information directly from the experts, which is naturally inefficient for everybody. Moreover, the growing number of the different company products and therefore increased information needs have emphasized the emerged upgrading needs even more, let alone the fact that also the organization has highlighted the importance of the digitalization, but as well taken more strict measures concerning the cyber security which also has created additional requirements for the system.

Overall, the application enables the users to search for engines and access specific data combinations available per an engine type, as well as to print out the results if needed. So in conclusion the basic functionality behind the application is pretty simple. In fact, from a technical perspective the application can be seen as a dynamic website rather than a web application. This is based on the fact that there is so little functionality available for the users and also because the creator's content is dominant, as the application does not enable the existing data to be modified or new data to be inserted, and there is another interface meant for this purpose external from this application. Although the difference between those two is not so clear even among the professionals, and here the system will be referred as an application. The pictures below (see figures 6, 7 & 8) represent the existing application's user interface with its main views in a natural order.

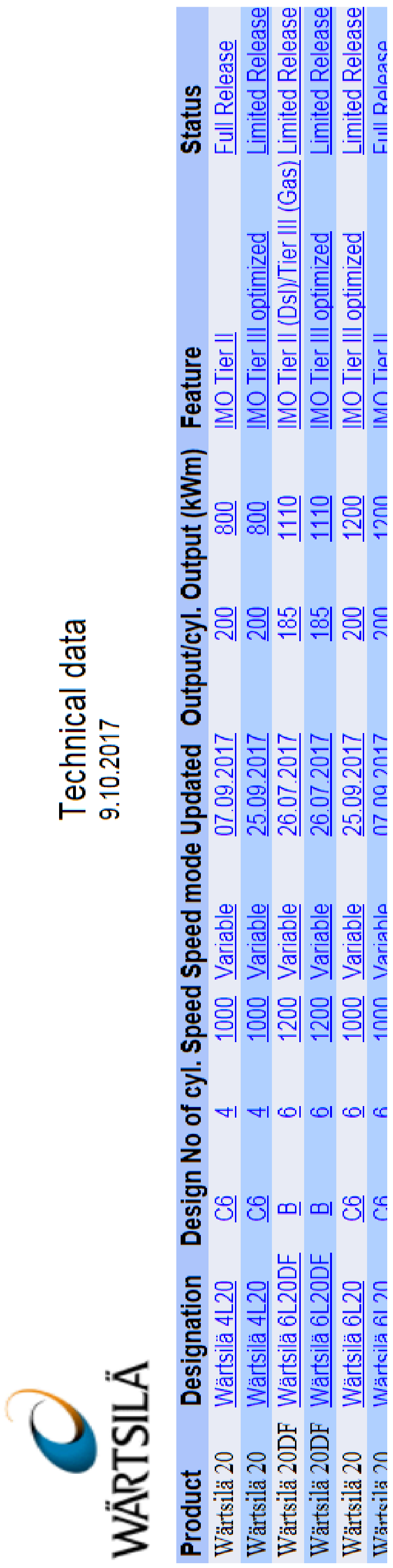


Search Engine Technical Data

Application type: (obligatory)	All
Output range Min/Max [kW]:	All All
Product:	All
Designation:	All
Number of cylinders:	All
Design stage:	All
Engine speed:	All
Engine speed mode:	All
Cylinder output [kW/cyl]:	All
Feature:	All

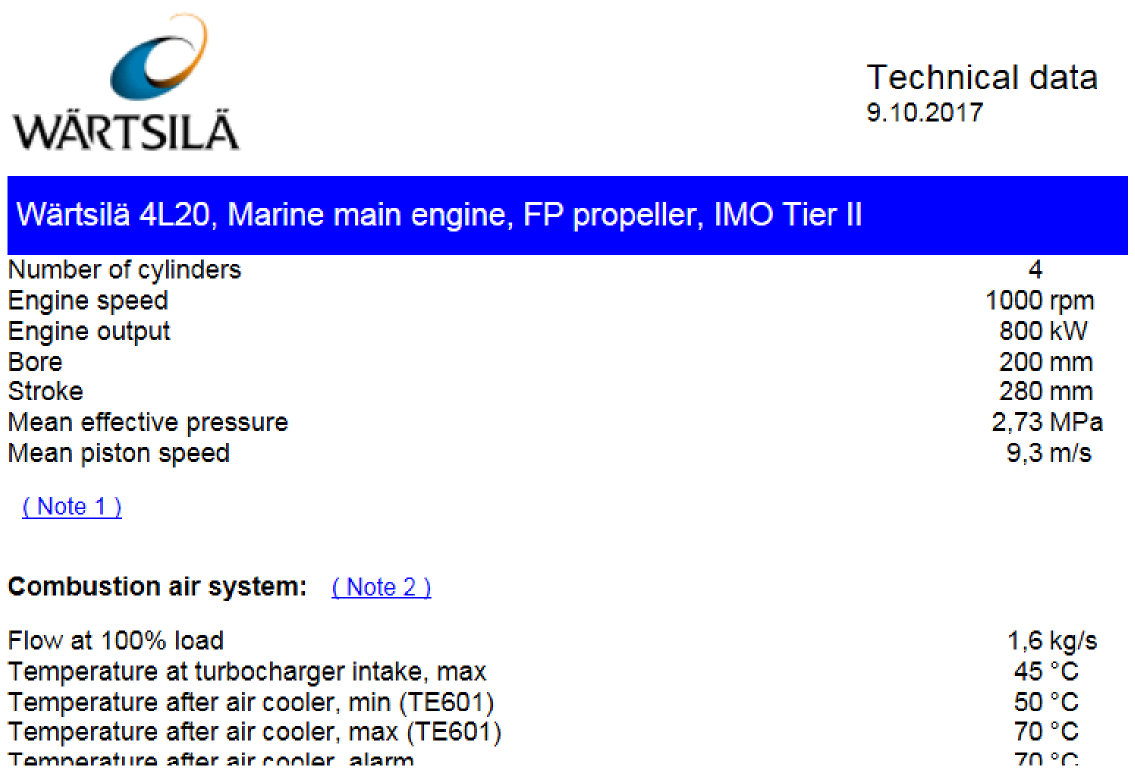
Matching configurations: 1302

Figure 6. Default i.e. search view from the existing application (9.10.2017).


Technical data
9.10.2017

Product	Designation	Design	No of cyl.	Speed	Speed mode	Updated	Output/cyl.	Output (kWm)	Feature	Status
Wärtsilä 20	Wärtsilä 4L20	C6	4	1000	Variable	07.09.2017	200	800	IMO Tier II	Full Release
Wärtsilä 20	Wärtsilä 4L20	C6	4	1000	Variable	25.09.2017	200	800	IMO Tier III optimized	Limited Release
Wärtsilä 20DF	Wärtsilä 6L20DF	B	6	1200	Variable	26.07.2017	185	1110	IMO Tier II (Dsl)/Tier III (Gas)	Limited Release
Wärtsilä 20DF	Wärtsilä 6L20DF	B	6	1200	Variable	26.07.2017	185	1110	IMO Tier III optimized	Limited Release
Wärtsilä 20	Wärtsilä 6L20	C6	6	1000	Variable	25.09.2017	200	1200	IMO Tier III optimized	Limited Release
Wärtsilä 20	Wärtsilä 6L20	C6	6	1000	Variable	07.09.2017	200	1200	IMO Tier II	Full Release

Figure 7. List of the engines matching a specific search (9.10.2017).



Technical data
9.10.2017

Wärtsilä 4L20, Marine main engine, FP propeller, IMO Tier II

Number of cylinders	4
Engine speed	1000 rpm
Engine output	800 kW
Bore	200 mm
Stroke	280 mm
Mean effective pressure	2,73 MPa
Mean piston speed	9,3 m/s

(Note 1)

Combustion air system: (Note 2)

Flow at 100% load	1,6 kg/s
Temperature at turbocharger intake, max	45 °C
Temperature after air cooler, min (TE601)	50 °C
Temperature after air cooler, max (TE601)	70 °C
Temperature after air cooler, alarm	70 °C

Figure 8. List of the data available for a particular engine (9.10.2017).

Moreover, currently not all the data in the Performance database is shown in the application, which is natural due to the fact that not everybody in the organization shall have access to all the data, as some data is more confidential. At the moment, the existing application displays, in addition to some general data, data under the flowing headings:

- Combustion air system
- Exhaust gas system
- Heat balances
- Fuel system
- Lubricating oil system
- High temperature cooling water system
- Low temperature cooling water system
- Compressed air system
- Generator data.

3.2 System architecture

The system architecture follows the three-tier architecture model, where the client tier is a so called thin client. Meaning that on the client side the user can access the application through a web browser – in this case with a device that is connected to the corporation's private network, or VPN – and no additional software is required to be installed on the user's PC. Moreover, that means that most of the application logic – developed with *Microsoft ASP* – is in the middle tier, on the web server, more precisely on *Microsoft IIS* web server environment running on Microsoft Windows Server. The application uses *Microsoft ADO* to connect with the database. In the database tier, the RDBMS used is a *Microsoft SQL Server*, which has its own database, where, among some other corporate databases, the Performance database is located. The technologies will be covered more in detail in the next chapter.

The figure below presents the application components and the interfaces in between. The middle tier application logic is the main component that needs to be redesigned, as the other components will be reused as much as possible, although some additional components to the whole system structure may need to be added. Also, as can be noticed, most of the components and techniques used are developed by Microsoft, which gives good circumstances for the integration between the components.

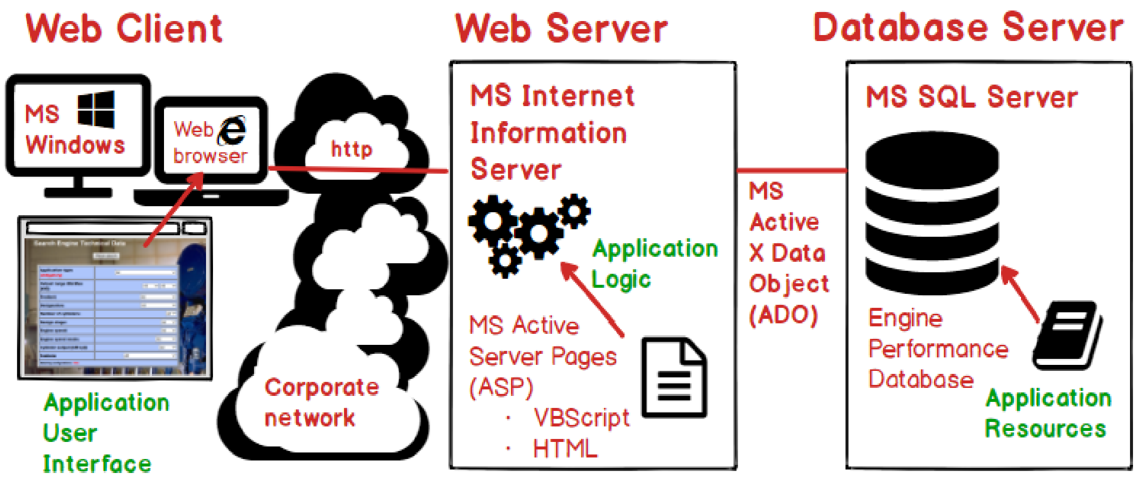


Figure 9. High-level system architecture of the application, and the used technology.

One thing that is not shown in the picture above is that the application has in fact two versions: one in the test environment and one in the production environment. This allows the possible changes to the script to be made and tested first in the test environment, before the actual implementation to the production server. However, this research will concentrate on the production version, and the test version is, more or less, a copy from the other.

3.3 Technical framework

The middle tier application logic is developed by using Microsoft ***Active Server Pages*** (ASP) that is a server side scripting environment. ASP in general is used to create and run dynamic websites as well as more interactive web applications. It allows the combining of HTML pages, scripting commands, and COM objects. (Polvinen 1999: 205; Cluts 1997.) COM objects basically enable a way for the program to perform functionalities, such as calculations, in compiled libraries, such as DLL (Polvinen 1999: 205). ASP enables scripting for IIS with the support of VBScript – *Microsoft Visual Basic Scripting Edition* – and JavaScript (Cluts 1997), in this case VBScript is used. VBScript is a light version of Microsoft Visual Basic (VB), and therefore it is usually used for short scripts, as VB can be used for bigger application, and can be used with Microsoft .NET Framework, in which case, it is referred as VB.NET (Rouse 2005; Rouse 2007b).

ASP 1.0 was released in 1996, and its follower ASP.NET, released in 2002, has then superseded the classic ASP (Wikipedia 2017a). However now the newcomer to the family is ASP.NET Core, version 1.0 was released in 2016 and 2.0 was released just now on August 2017 (Wikipedia 2017b). ASP.NET is a web application framework that was released together with .NET Framework which it runs on. ASP.NET Core is also a web framework but it uses .NET Core Framework which is today still a subset of .NET Framework – not all technologies available in .NET are available in .NET Core – but .NET Core is expected to be the future of Microsoft .NET. (Cluts 1997; Sauer 2016; Rouse 2007a; Anderson, Latham, Addie, Dykstra, Roth & Pasic 2017; Carter, Wenzel, Addie, Latham, Onderska, Pratt, Wagner, Agarwal 2016.)

So, as you can image, ASP is no longer the most modern environment to be used, although in ASP and in the newer versions the basic idea is the same, as they allow a developer to build web pages dynamically on the fly by inserting queries to databases. However, for instance the newer versions enable more scripting languages, allow to write object oriented code, and give access to more tools that come with the .NET and .NET Core Frameworks. (Cluts 1997; Sauer 2016; Rouse 2007a; Anderson et al. 2017; Carter et al. 2016.) Moreover, ASP.NET Core compared to ASP.NET among others enables even higher performance and build cross-platform on Windows, macOS and Linux, as ASP.NET is targeting Windows servers only, as is also the classic ASP (Anderson et al. 2017).

Let's look at the performance of ASP a bit further. The ASP technology is built directly on IIS, which means that when changes are made to the ASP file, the script will be automatically interpreted when the web page is loaded next time (Cluts 1997). In fact, one difference related to the performance of ASP and the newer versions, is that ASP is interpreted, as the newer are compiled (Sauer 2016). This means that when different scripting languages – server-side script and HTML – are used in ASP, the loading of a particular page requires both scripting engines to process the request, which consumes more time and memory. In ASP.NET and the Core version, these inefficiencies are eliminated so that the pages are always compiled in .Net classes, and when the page is accessed, it is provided to the client by executing the compiled code. (Bean Software 2017.) The fact that the classic ASP is not compiled, at least partly, explains why the existing TDS application is so slow, but of course the way the script is written and the whole architecture matters as well.

As said earlier, Microsoft ***Internet Information Services*** – also referred as Internet Information Server – (IIS) running on Microsoft Windows Server is the web server environment used, which is only natural because ASP is a feature of IIS. In general IIS is a flexible general-purpose web server software that runs on Windows systems, accepts requests coming from clients and returns the responses. In this case the information flow happens across the corporate intranet, normally with HTTP protocol. Today ASP.NET Core is the newest framework that IIS works with. IIS enables developers to use integrated tools, such as Microsoft Visual Studio IDE, for creating web content, like web applications. IIS has also evolved over the years with Microsoft Windows, and new features and functionality has become available. (Rouse, Bigelow, Dodge, Lehto & Weiner 2017.)

The application uses Microsoft ***ActiveX Data Objects*** (ADO) with ***SQL Server Native Client*** (SNAC) ***OLE DB provider*** to connect with the SQL Server. This can be seen from the application logic, as the ADO connection object is used and the database provider is specified in the ASP script. The script for creating the connection to the database, leaving out the actual location and authentication parameters, is presented below:

```
<%

Set TechConn = Server.CreateObject("ADODB.Connection")

strConn = "Provider = SQLNCLI10; Server = ServerName;
Database = DatabaseName; Uid = UserName; Pwd = Pass-
word;"

TechConn.open strConn

%>
```

ADO is universal data-access technology that aims to provide data access regardless of the scripting language or data source used, that way eliminating the need to convert existing data to another format in order to access it (Roff 2001: 3). Actually, Microsoft has developed a whole series of different technologies to access data, together referred as Microsoft Data Access Components (MDAC), that’s main technologies include ADO.NET, ADO, OLE DB, ODBC and RDS (Roff 2001: 3–4). Microsoft ADO.NET – with the .NET Framework – presents the next generation of ADO (Roff 2001: xi).

The used ADO solution consists of a specific set of Microsoft COM objects wrapped around ***Object Linking and Embedding Database*** (OLE DB) technology (Roff 2001: ix). The term SNAC is used to refer to all ODBC and OLE DB drivers – programs installed on workstation Control Panel – for SQL Server, as the drivers are specific to the particular DBMS used (Hubbard & Guyer 2017; Milener & Guyer 2017). The used SNAC is automatically installed with the SQL Server (Guyer & Mansfield 2016). OLE DB provider –

set of libraries used to communicate with the data source – provides the functionality to access all kinds of data sources: in this case SQL Server relational database, but also for instance to Excel spreadsheets (Roff 2001: ix). ADO can be seen as an application-level interface to OLE DB, and respectively OLE DB can be seen as an API to variety of different data sources (Roff 2001: 7–8).

Microsoft ***SQL Server*** is the RDBMS used, although when the existing TDS application was first implemented it was Microsoft Access, which presents the previous generation of the Microsoft SQL Server. Like other RDBMS software SQL Server is built on top of SQL language that is commonly used to manage databases and create queries for the data they contain. Microsoft has released multiple versions of the SQL Server, and the number of integrated management and analytics tools and functionalities has grown within the versions. (Rouse, Hughes & Stedman 2017.) The specific SQL Server used here is SQL Server 2014, and the newest version is SQL Server 2017 that was actually just released in October 2017, when this thesis was started.

3.4 Application logic

The very basic middle tier logic behind the application is likely to stay the same in the improved application, although the execution may differ. Next let's look at a simplified example scenario of how the application works: how the middle tier application logic communicates with the client and the database in a general level. Of course, the application is really a lot more complicated than this, but this chapter aims to provide a simplified version of its logic.

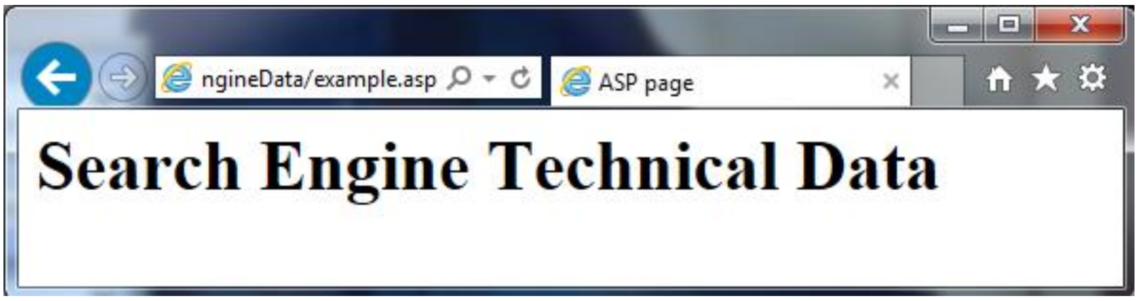
I will use an application type in this example, which is one data field available in the database. The user chooses the application type from a drop-down list in the ***Default.asp*** page. After choosing the application type, the user presses the "Search engine!" button to search for the engines matching this search criterion. A list with all the matching engines opens on a page named ***EngineDataRange.asp***. User chooses one engine, presses the link and it opens up a page named ***EngineDataById.asp***, which shows the detailed data available for the specific engine that is retrieved from the database. This is the scenario in general level, and those three files are the most relevant files in this application and therefore in this example.

Furthermore, in reality the *EngineDataById.asp* calls functions from the file named *Functions.asp* to formalize and display the different data sections and to do any of the calculations necessary, but I have skipped this to simplify the example and added the necessary code to the *EngineDataById.asp*. Also, the connection to the database is created from a different file named *Connection.asp*, which content I already discussed in the earlier chapter. There is also a separate file named *EngineDataPrint.asp* used to create a printer-friendly version from the *EngineDataById.asp,* which will not be discussed further.

The main logic behind the three most relevant files will be presented here, but I will discuss the code structure and content first more in general. One can notice from the presented scripts that some parts of the scripts are written inside the tags of <% and %>. This means that this script is executed on the server side, and is in this case written in VBScript language. There is also script without these tags, and that is HTML, inside <html> tags. In contrast to VBScript, HTML is rendered on the client side. It forms the structure of the page that is visible for the user on the browser. The basic HTML script structure is presented in the next script example with a piece of ASP included, as the ASP is typically included in an HTML document. This means that the page itself is sent to the client in an HTML format, after the server has read and executed the needed tasks in ASP. *Response.Write* is a basic statement in ASP that is used to display text to the client. (e.g. W3Schools 2018a, 2018b.)

```
<html>
<head>
     <title>ASP page</title>
</head>
<body>
     <%
     Response.Write "<h1>Search Engine Technical
     Data </h1>"
     %>
</body>
</html>
```

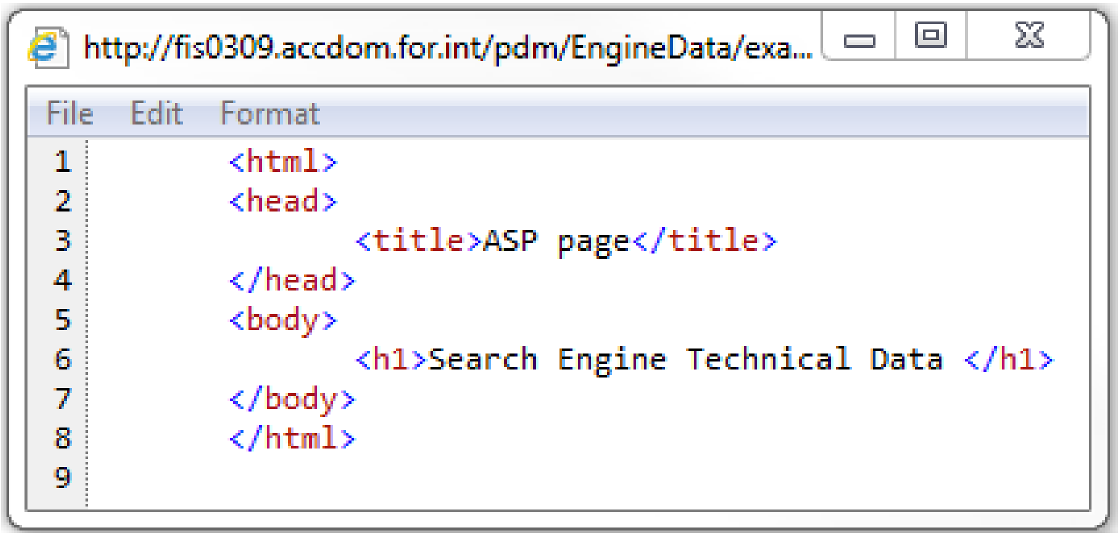
This document would look as follows, displayed on a browser:



Search Engine Technical Data

Figure 10. The example script displayed on the web browser (file name: example.asp).

However, if you look at the source view from the browser, the ASP script is not shown. This is due to the fact that only the server can see the ASP script, and the client only sees the HTML result (see W3Schools 2018a). Here is what the source view looks on the browser:



```
<html>
<head>
        <title>ASP page</title>
</head>
<body>
        <h1>Search Engine Technical Data </h1>
</body>
</html>
```

Figure 11. Source view from the example script on the browser.

Now the basic structure of an ASP file – text file with the extension .asp – should be about clear. As the application is interactive, it needs input from the user of the web browser. One of the most common ways to get input is with HTML forms, which are also used in this application. A method attribute specifies the HTTP method used to submit the data from the form in the client to the server. The most commonly used HTTP request methods are POST and GET. Probably the most obvious difference between these two is that POST method does not display the data in the page address field, as in GET it is visible. That is why POST is referred as the safer option. GET works in a way that it requests the data from a server with a query string that is added to the URI, as POST in contrast submits i.e. sends the data to be processed to the server inside the body of the HTTP request. (W3Schools 2018c, 2018d & 2018e.)

As this is a simplified example, the example file: Default.asp, is only going to have a heading, drop-down list to choose the application type (either Marine main engine, FP propeller; or Marine main engine, CP propeller), and a submit button. See the two pictures below which present the browser view for the example Default.asp.

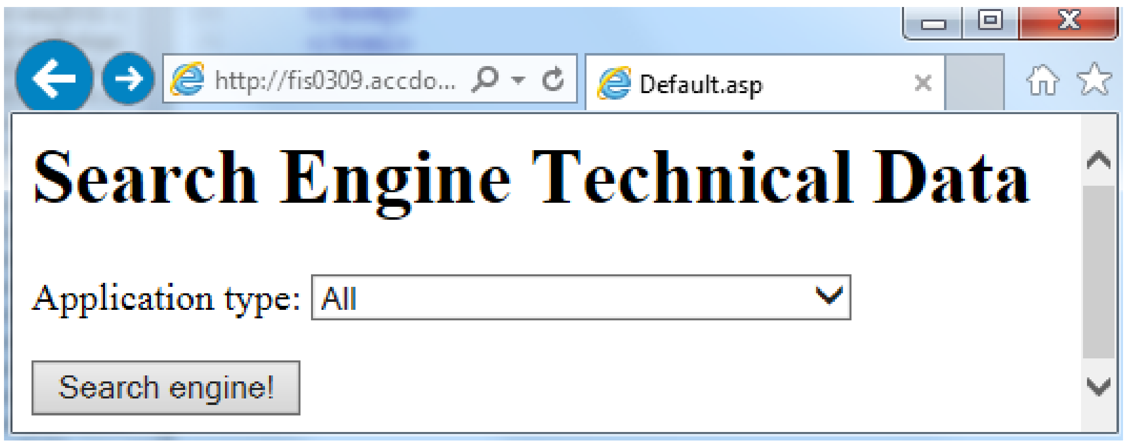


Figure 12. Simplified Default.asp file.

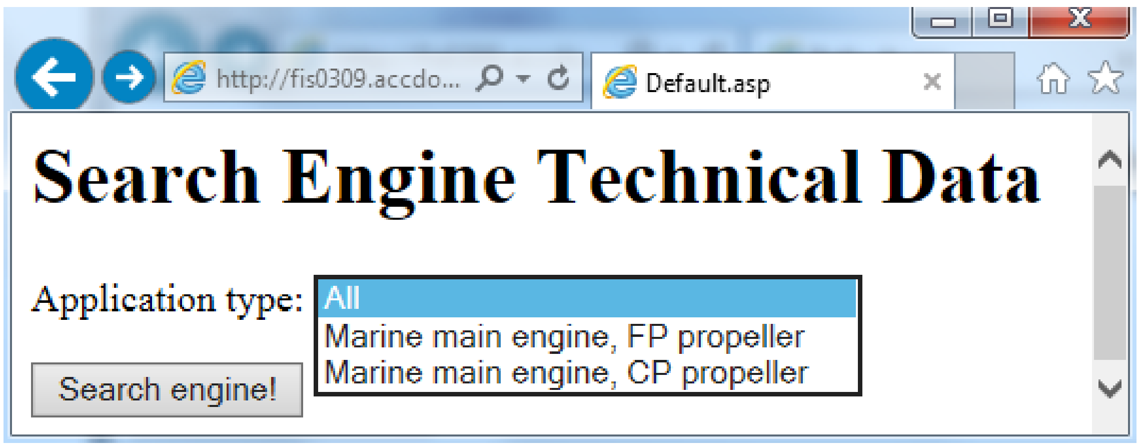
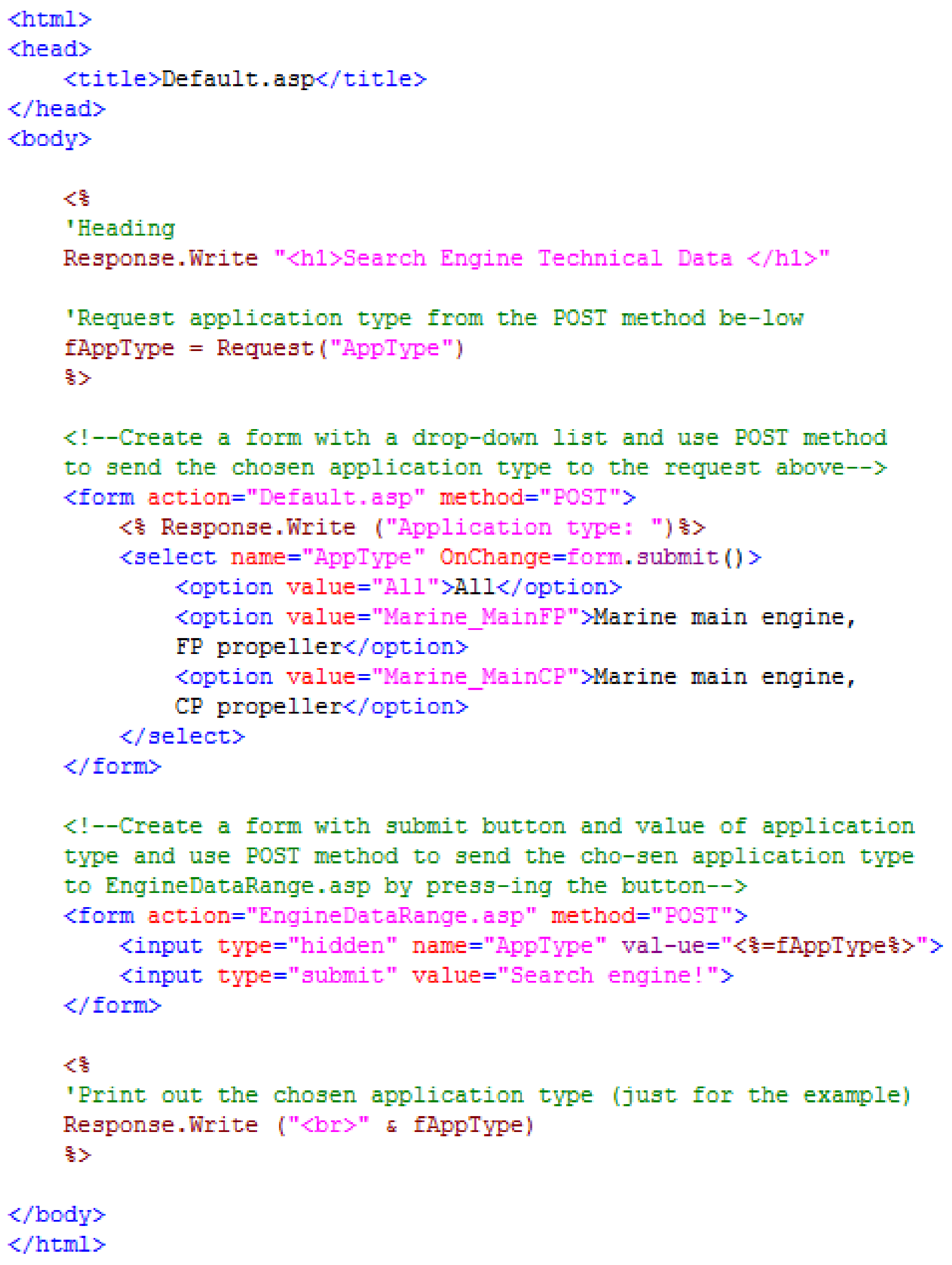

http://fis0309.accdo...
Default.asp
Search Engine Technical Data
Application type:
All
Marine main engine, FP propeller
Marine main engine, CP propeller
Search engine!

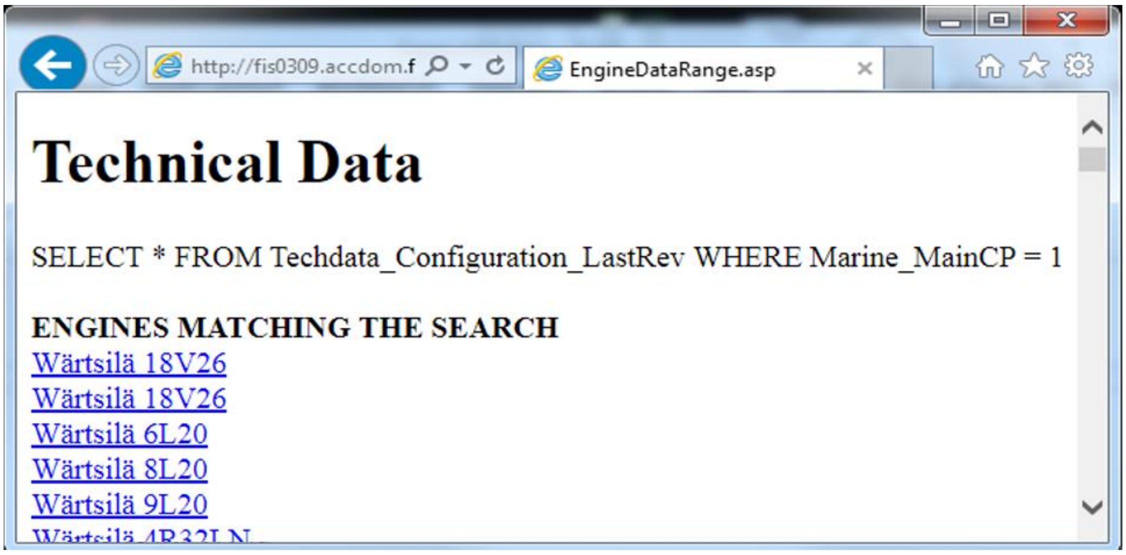
Figure 13. Simplified Default.asp file with drop-down list's values showing.

The Default.asp includes two HTML forms – inside of <form> tags – using the method POST. The first one is the drop-down list that sends the chosen application type value to be processed on the file itself, and it is requested in the beginning of the script and stored in a variable called *fAppType*. The other form includes the submit button, and sends the application type value to the EngineDataRange.asp, which requests it and retrieves the value in the beginning of the script. The following script presents the example Default.asp.



```
<html>
<head>
    <title>Default.asp</title>
</head>
<body>

    <%
    'Heading
    Response.Write "<h1>Search Engine Technical Data </h1>"

    'Request application type from the POST method be-low
    fAppType = Request("AppType")
    %>

    <!--Create a form with a drop-down list and use POST method
    to send the chosen application type to the request above-->
    <form action="Default.asp" method="POST">
        <% Response.Write ("Application type: ")%>
        <select name="AppType" OnChange=form.submit()>
            <option value="All">All</option>
            <option value="Marine_MainFP">Marine main engine,
            FP propeller</option>
            <option value="Marine_MainCP">Marine main engine,
            CP propeller</option>
        </select>
    </form>

    <!--Create a form with submit button and value of application
    type and use POST method to send the cho-sen application type
    to EngineDataRange.asp by press-ing the button-->
    <form action="EngineDataRange.asp" method="POST">
        <input type="hidden" name="AppType" val-ue="<%=fAppType%>">
        <input type="submit" value="Search engine!">
    </form>

    <%
    'Print out the chosen application type (just for the example)
    Response.Write ("<br>" & fAppType)
    %>

</body>
</html>
```

Now if the user chooses the application type to be *Marine main engine, CP propeller*, and presses the submit button, the browser will show the EngineDataRange.asp page. For that to happen the server needs to retrieve some data from the database, and SQL query is needed for that. In the next picture the simplified example of the EngineDataRange.asp browser view is presented, showing the engines matching the search criterion. Also, the performed SQL query is presented just to get an idea of the query, although it is just for this example.



http://fis0309.accdom.f

EngineDataRange.asp

Technical Data

SELECT * FROM Techdata_Configuration_LastRev WHERE Marine_MainCP = 1

ENGINES MATCHING THE SEARCH

Wärtsilä 18V26
Wärtsilä 18V26
Wärtsilä 6L20
Wärtsilä 8L20
Wärtsilä 9L20
Wärtsilä 4R32LN

Figure 14. Simplified EngineDataRange.asp presenting the SQL query.

The chosen application type is included to the query as criteria in a form "Marine_MainCP = 1". As in the database, the column is named as Marine_MainCP, and the value is either zero or one, and this specific query retrieves all the engines from the database that have the value one in that column. Also in the database, there is this view called TechData_Configuration_LastRev that is a collection of linked tables and it holds the latest data revision, and the data is retrieved from there. The specific view from the SQL Server is presented below. Also in the picture after that picture the same query is executed directly in the SQL Server. Of course, all the available values i.e. columns cannot be fitted to the picture, but I have tried to show some values related to this example scenario.

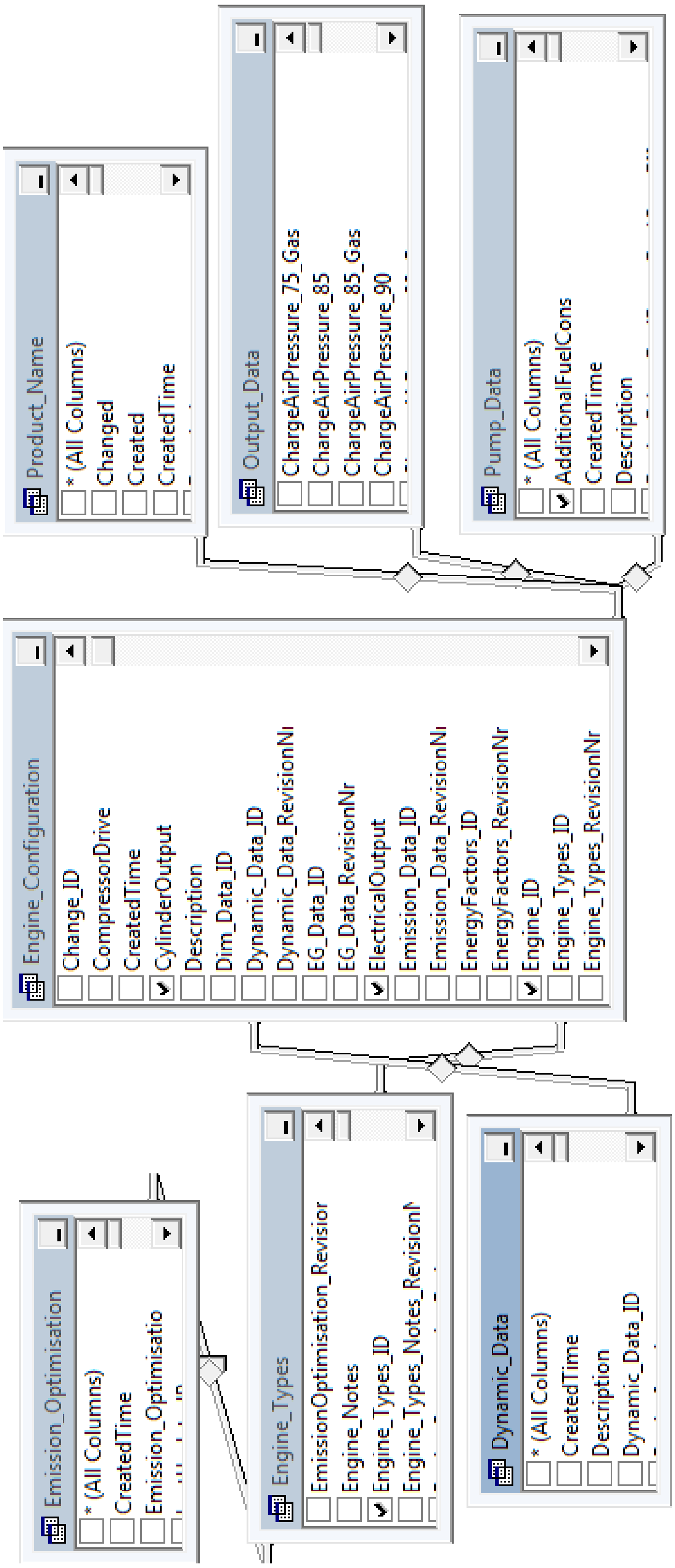
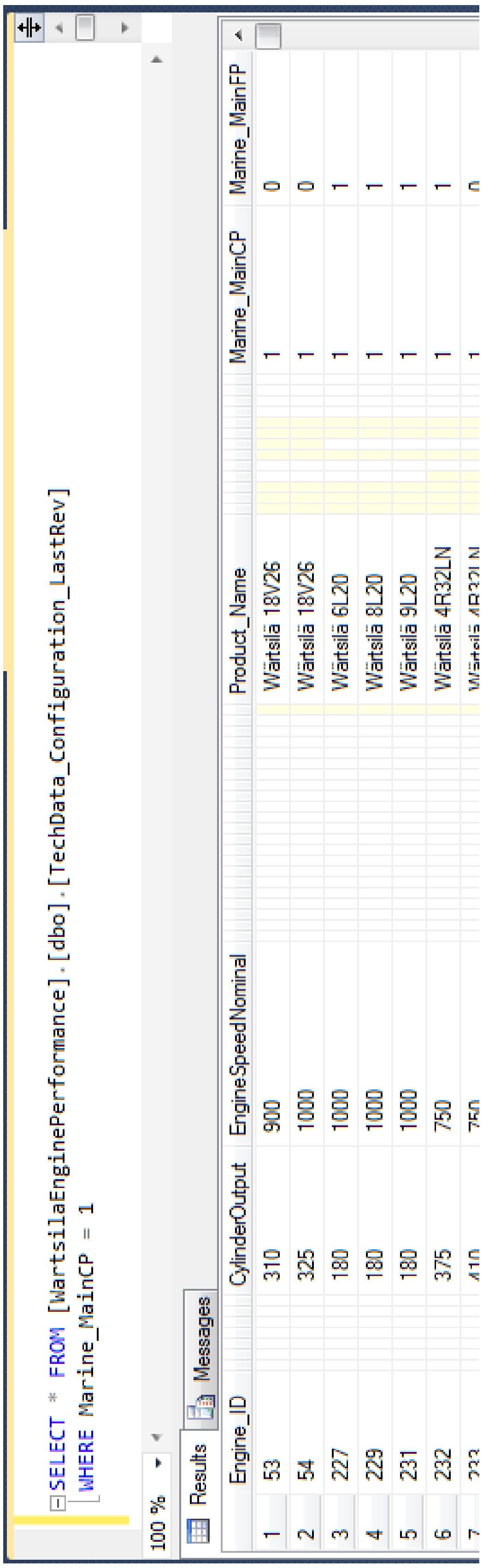


Figure 15. TechData_Configuration_LastRev view from the SQL Server (24.10.2017).

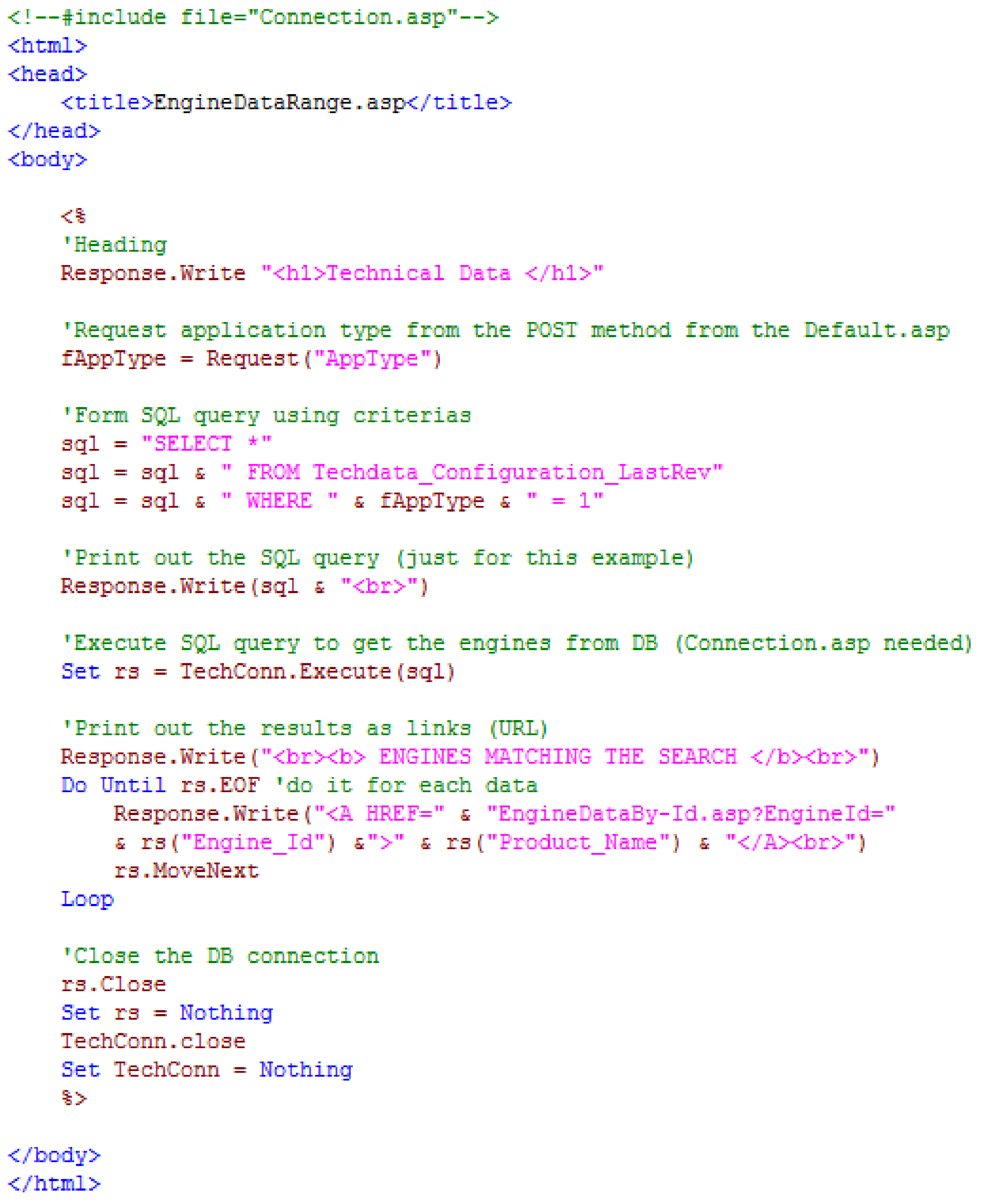


```
SELECT * FROM [WartsilaEnginePerformance].[dbo].[TechData_Configuration_LastRev]
WHERE Marine_MainCP = 1
```

	Engine_ID	CylinderOutput	EngineSpeedNominal	Product_Name	Marine_MainCP	Marine_MainFP
1	53	310	900	Wärtsilä 18V26	1	0
2	54	325	1000	Wärtsilä 18V26	1	0
3	227	180	1000	Wärtsilä 6L20	1	1
4	229	180	1000	Wärtsilä 8L20	1	1
5	231	180	1000	Wärtsilä 9L20	1	1
6	232	375	750	Wärtsilä 4R32LN	1	1
7	233	410	750	Wärtsilä 4R32LN	1	0

Figure 16. The example query executed directly in the SQL Server (24.10.2017).

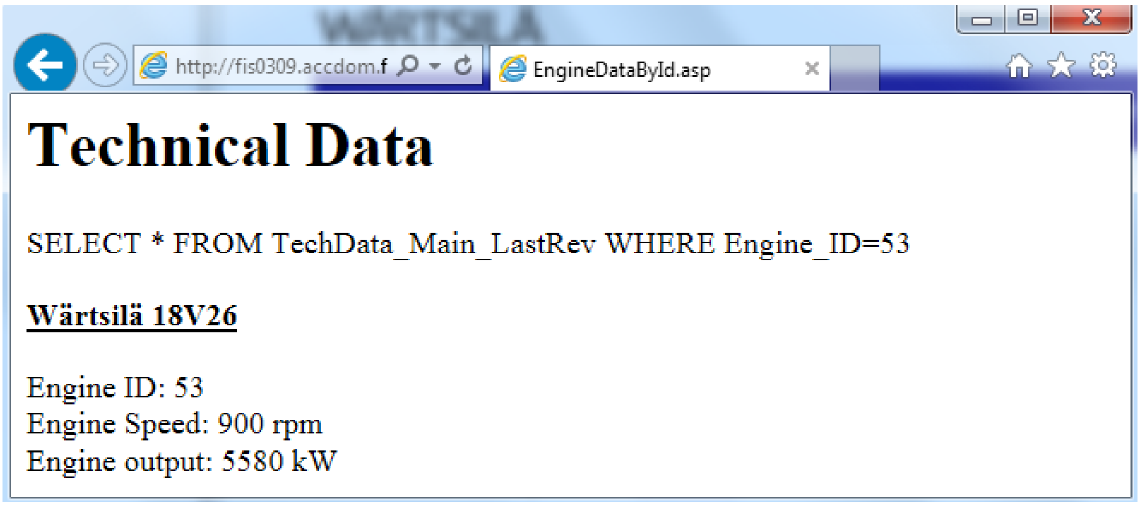
The example EngineDataRange.asp file is presented below.



```
<!--#include file="Connection.asp"-->
<html>
<head>
    <title>EngineDataRange.asp</title>
</head>
<body>

    <%
    'Heading
    Response.Write "<h1>Technical Data </h1>"

    'Request application type from the POST method from the Default.asp
    fAppType = Request("AppType")

    'Form SQL query using criterias
    sql = "SELECT *"
    sql = sql & " FROM Techdata_Configuration_LastRev"
    sql = sql & " WHERE " & fAppType & " = 1"

    'Print out the SQL query (just for this example)
    Response.Write(sql & "<br>")

    'Execute SQL query to get the engines from DB (Connection.asp needed)
    Set rs = TechConn.Execute(sql)

    'Print out the results as links (URL)
    Response.Write("<br><b> ENGINES MATCHING THE SEARCH </b><br>")
    Do Until rs.EOF 'do it for each data
        Response.Write("<A HREF=" & "EngineDataBy-Id.asp?EngineId="
        & rs("Engine_Id") &">" & rs("Product_Name") & "</A><br>")
        rs.MoveNext
    Loop

    'Close the DB connection
    rs.Close
    Set rs = Nothing
    TechConn.close
    Set TechConn = Nothing
    %>

</body>
</html>
```

In the EngineDataRange.asp file, the SQL query is first formed and then executed utilizing the Connection.asp file. Then the results are printed to the user using hyperlinks that use GET request to request the next page i.e. the EngineDataById.asp file by inserting the specific engine’s id information in the URL. In the beginning of the EngineData-ById.asp file the value of engine’s id is requested and stored to a variable called *fEngineId*. In the end, the database connection is closed.

Now when the user presses one of the hyperlinks, he or she gets to the EngineData-ById.asp page. The simplified example browser view is presented below. Also in here the SQL query is printed out just for this example. The query criterion holds the specific engine’s id information which is used to retrieve that specific engine’s data from the database. In this example, a couple of example values are printed out to the user as a result.



Technical Data

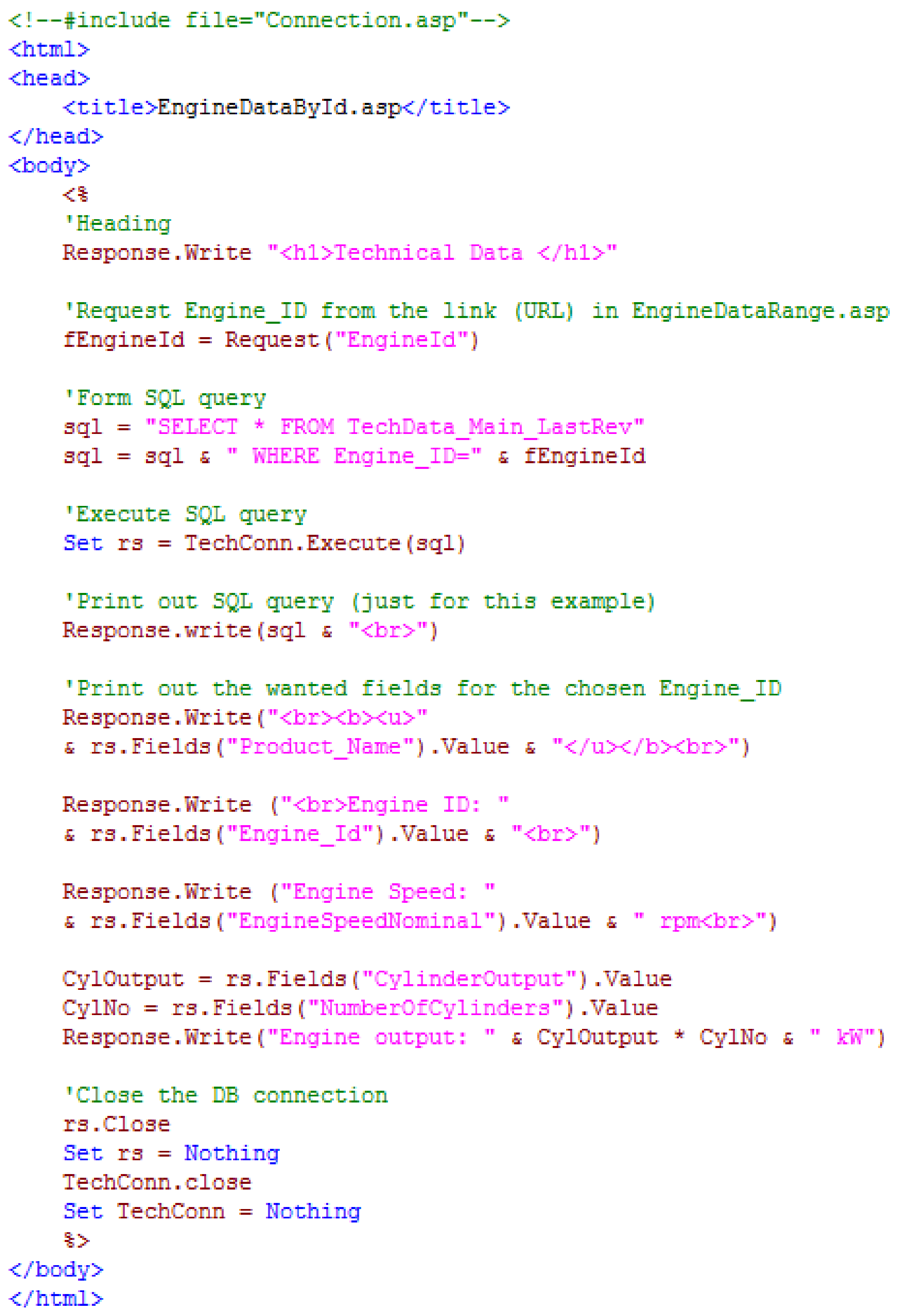
SELECT * FROM TechData_Main_LastRev WHERE Engine_ID=53

Wärtsilä 18V26

Engine ID: 53
Engine Speed: 900 rpm
Engine output: 5580 kW

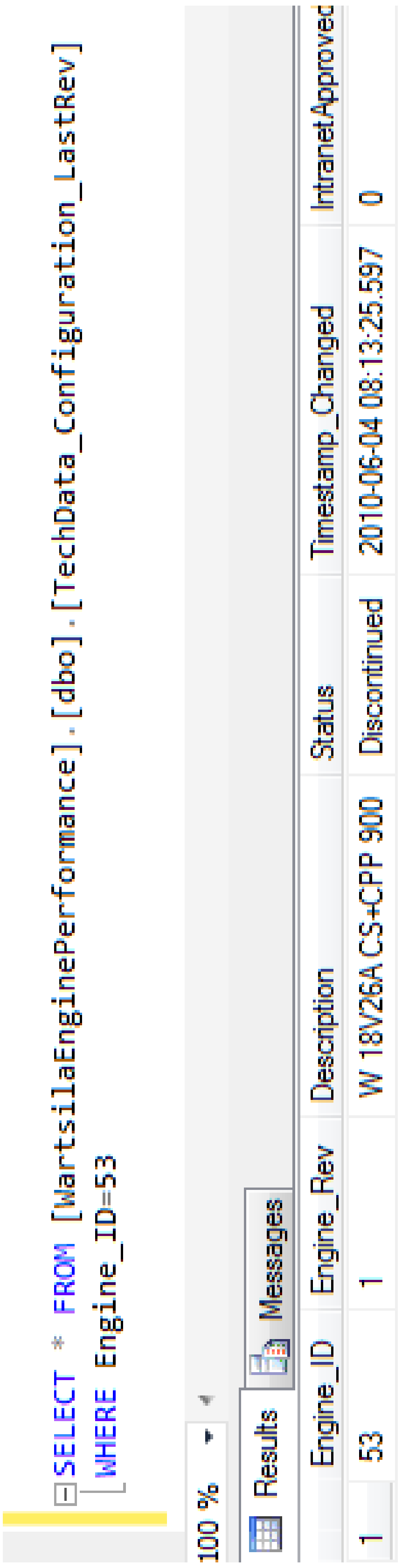
Figure 17. Simplified EngineDataById.asp with the SQL query.

The example EngineDataById.asp file is presented next. The basic working principle is the same as in the previous file, as first the query is formed then executed and then the wanted values are printed out to the user. There is just no links, but only text, and only one row needs to be retrieved from the database, so no loop is needed to print out the data.



```
<!--#include file="Connection.asp"-->
<html>
<head>
    <title>EngineDataById.asp</title>
</head>
<body>
    <%
    'Heading
    Response.Write "<h1>Technical Data </h1>"

    'Request Engine_ID from the link (URL) in EngineDataRange.asp
    fEngineId = Request("EngineId")

    'Form SQL query
    sql = "SELECT * FROM TechData_Main_LastRev"
    sql = sql & " WHERE Engine_ID=" & fEngineId

    'Execute SQL query
    Set rs = TechConn.Execute(sql)

    'Print out SQL query (just for this example)
    Response.write(sql & "<br>")

    'Print out the wanted fields for the chosen Engine_ID
    Response.Write("<br><b><u>"
    & rs.Fields("Product_Name").Value & "</u></b><br>")

    Response.Write ("<br>Engine ID: "
    & rs.Fields("Engine_Id").Value & "<br>")

    Response.Write ("Engine Speed: "
    & rs.Fields("EngineSpeedNominal").Value & " rpm<br>")

    CylOutput = rs.Fields("CylinderOutput").Value
    CylNo = rs.Fields("NumberOfCylinders").Value
    Response.Write("Engine output: " & CylOutput * CylNo & " kW")

    'Close the DB connection
    rs.Close
    Set rs = Nothing
    TechConn.close
    Set TechConn = Nothing
    %>
</body>
</html>
```

The last picture below presents the same query executed directly in the SQL Server. Again, not all the columns fit to the picture.



```
SELECT * FROM [WartsilaEnginePerformance].[dbo].[TechData_Configuration_LastRev]
WHERE Engine_ID=53
```

	Engine_ID	Engine_Rev	Description	Status	Timestamp_Changed	IntranetApprove
1	53	1	W 18V26A CS+CPP 900	Discontinued	2010-06-04 08:13:25.597	0

Figure 18. Example query executed directly in the SQL Server (24.10.2017).

4 RESEARCH DESIGN

As stated earlier in this thesis, every software development project is unique, so there is really not a one single right way of doing things. This particular development project will follow the linear process flow of the waterfall model. This was a natural choice as the model divides the overall project into multiple phases and this research covers only the first phase of the requirements engineering and the results will work as an input for the coming phases. However, naturally some feedback from the following phases may be later given to the documentation produced in this phase, so some overlapping is to be expected.

Other things that supports the usage of the waterfall model as the underlying framework includes that the different phases are also divided between different and distributed teams or persons on the way of the whole project. In addition, the model provides good circumstances management wise as well, and it is seen more important to do the project properly than rush into a working program, as the existing application is still working. Moreover, although the high-level project structure follows the waterfall model, the activities involved may be more or less interleaved, iterative or incremental, and as said before the RE process is typically more iterative than linear.

The whole process of requirements engineering aims to produce an agreed software requirements specification, which describes the system while satisfying the stakeholder requirements. Next, I will give more detailed information about the activities, methods and tools that will be used to manage and carry out the actual execution of this phase. In other words, this chapter will represent the research plan.

The requirements engineering process will for the most part follow the process flow and activities presented earlier in the requirements engineering theory, leaving out the feasibility study as a separate phase, as at this point it is clear that the system is needed and can be developed, although assessing feasibility at the requirement level may be relevant. The RE process will include the following set of activities:

1. *stakeholder identification*
2. *requirements elicitation*
3. *requirements analysis and agreements*
4. *requirements specification*
5. *requirements validation.*

In practice the activities will be interleaved, as naturally feedback is given from one phase to another: for instance, missing stakeholders may be discovered during the elicitation, missing requirements may be discovered during analysis, and so on.

Stakeholder identification in general aims to identify the project stakeholders. In this case the most important stakeholders left to be identified are the application end-users that are going to be involved in the requirements elicitation. The other relevant stakeholders that are already recognized include the system owners that are from the PI team itself, and are responsible for managing the whole development project, maintaining the existing and the new system-to-be, as well as the Performance database that is the main data source for this application.

The system owners are closely involved in the RE activity, as they have the required domain knowledge, and they also have the administrative role in the application. Moreover, at this point the final developers of the application are not yet agreed on, but are likely to be from inside the organization. I myself will work as the requirements engineer, being in charge of all the RE activities and producing of all the related materials and required documentation.

Furthermore, as the application is essentially intended for the internal business use of the company, those end-users that need to be identified are naturally from inside the organization. In fact, as a starting point everybody in the organization shall have access to the most general data and services provided by the application. However, it is not possible or even appropriate to consult everybody in the organization, so there is a need to determine who are the right persons to be involved in the elicitation activity.

The identification of those end-users will happen by negotiating together with the system owners, as they have the domain knowledge and best understanding to answer the questions of who is going to use the system, who could benefit from the system, as well as who is possibly otherwise affected by the system. Furthermore, as the intention is also to recognize new potential users for the application, suggestions from others are also welcome.

Basically, the stakeholder identification activity will involve dividing the system users – everybody inside the organization – into smaller classes i.e. user groups according to their discriminant factors. In this case the user groups shall be formed based on their access roles i.e. authorization in the application. The organizational structure will naturally help in recognizing these different user profiles, as typically users in the same work area need access to the same data and services provided by the application. After, the most important and relevant user groups are recognized, a representative person or small group shall be selected from each class to be contacted during the RE process.

In practice, the system owners shall first give their suggestions for the possible user groups and contact persons. After that, those selected persons, and their managers, will be contacted to get feedback for those suggestions, they will also naturally be informed about the whole project and its objectives. At this point, if the contact persons feel that they are not the correct persons for the job, or they notice that some user group is missing, or something like that, they can give their comments for improvements. It is also to be expected that the user groups and contact persons will evolve during the RE process, for instance in the elicitation activity.

Requirements elicitation is the activity that aims to uncover what the customer wants and needs. In this research this activity includes discovering information about the new required system, but also requires the domain knowledge about the other already existing systems in the organization as well as the organizational standards, as the same practices may need to be followed as with the other systems. Furthermore, also the knowledge about the already existing system will be utilized, keeping in mind the reuse of the existing database as well as potential partial reuse of other components and ideas from the system.

As said earlier there are multiple different requirements elicitation techniques existing to choose from. This research will use two complementing methods to discover the requirements from the identified stakeholders: semi-structured interviews – *used to interact with the stakeholders* – in combination with early prototyping – *used to help the stakeholders to understand the system better*. Next, I will discuss the methods more in detail, and explain how they will be utilized in this research.

Interviews belong to the traditional methods of requirements engineering, and it is actually the most commonly used elicitation technique (Sommerville 2011: 104; Maciaszek 2005:50). Basically, there are two main types of interviews: *structured* i.e. formal interviews and *unstructured* i.e. informal interviews. The obvious difference between the two is that structured interviews are prepared in advance and they have a clear agenda and a set of pre-defined questions, as unstructured interviews are more like informal meetings that aim to encourage the stakeholders to speak their mind and that will lead to the requirements. Interviews can also naturally be a mixture of both, typically referred as *semi-structured* interviews, as rarely completely informal meetings work, as there has to be something to start the conversation and keep it in the subject (Sommerville 2011: 104; Maciaszek 2005:51.)

That is why I will refer the interviews in this research as conversational or semi-structured interviews, as there will be a clear agenda, but otherwise the nature of the interviews will be more conversational and there will be no pre-defined questions (see Laplante 2014: 59). Conversational interviews typically aim to relaxed conversation, and that is one reason why they were found suitable for this research considering the target corporate’s culture (see Laplante 2014: 59).

In general, effective interviews are open-minded, and really intend to listen what the stakeholders have to say and not make any own pre-consumptions. They also encourage the interviewee to the discussion with some other ways than just saying “tell me what you want”. This is where we come to the advantages of the prototyping, as working together with and discussing about a prototype system, is likely to encourage stakeholders as it is easier to discuss in defined context than just in general terms. Moreover, the interviewing on its own is likely to miss some essential information, and therefore it is generally recommended to be used together with other requirements elicitation techniques. (Sommerville 2011: 104.)

Prototyping is more modern elicitation technique compared to the interviews. A prototype is an initial version of the software system, aiming to visualize the system. Prototypes can actually be used in multiple different phases of software development, however in this case the prototyping is used as an elicitation technique in the RE process, in this activity of elicitation the prototype will be used to discover the requirements, but it will as well be used in the other phases of the requirements analysis, specification and validation to become. (Sommerville 2011: 45; Maciaszek 2005: 54.) More in detail, in the requirements elicitation the created prototype will be used to imbody typical usage scenarios in the application in order to give context to the interview, generate conversation and to keep the conversation on the right track (see Benyon 2014: 144).

Overall, prototyping is an effective way of eliciting requirements that would be otherwise harder to obtain from the stakeholders, as prototypes tend to bring up unanticipated aspect of the system. They also enable people to see how well the system supports their work, stimulates them to get new ideas for requirements and find strengths and weaknesses in the software. Prototypes typically show things in a different light e.g. a function described in a specification may seem that it is very well explained and correct, but when it is visualized in a prototype with other proposed requirements the stakeholders may see that their requirement was incomplete or incorrect, and the specification may be then modified. In other words, the prototyping is seen to reduce misinterpretations. (Sommerville 2011: 45; Maciaszek 2005: 54.) Also, according to the IEEE (1998: 9), prototypes are good in providing quick feedback, as customers in general are more likely to react to a prototype than a written requirement.

Typically, a prototype is a quickly made model of the software solution that presents the graphical user interface (GUI) of the system and somehow simulates the system behavior. Sometimes prototyping can be referred as rapid or early prototyping, and the idea behind that is that the prototyping enables stakeholders to experiment with the system early in the development process, as it is developed relatively fast. (Sommerville 2011: 45; Maciaszek 2005: 54; Laplante 2014: 59.) This is also the case in this research, as the prototype is intended to be a quickly made model of the user interface that can easily be modified as more input is gotten. This leads us the fact that the prototype is developed iteratively based on the feedback gotten from the held interviews. In fact, iterative development is one essential characteristics of prototyping (see Sommerville 2011: 45).

Moreover, prototypes can be either working or non-working models. Working models in case of a software system typically means a working code, but it may also be a simulation or something else. Non-working models on the other hand, may include storyboards and mock-ups from the UI. In case of a working code prototype, the prototype may be an evolutionary prototype, which means that the code is designed to be reused and evolved into the final software product. This is typically the case in an agile software development. In contrast to evolutionary i.e. non-throwaway prototype, there is also a throw-away prototype, which is designed to be thrown away when it is not needed anymore, and it is not intended to evolve to be the final product. (Laplante 2014: 62; Maciaszek 2005: 54.)

In this research a throw-away prototype will be used, and the prototype will be a so-called paper prototype consisting of mock-ups from the UI. However, it is going to be a working model, as distinct from the previous in a way that there will be links from one mock-up to another, so that the mock-up prototype can easily be used to demonstrate the different usage scenarios. However, the prototype is not actually doing anything, as it is just a paper sketch of some sort, and it is not intended to be as detailed as the real system, but rather aims to visualize the most relevant parts of the UI. Moreover, the prototype will be developed using a tool called *Balsamiq Mockups* that enables an easy creation for these types of prototypes.

So, as I have now presented the techniques used in the elicitation, I will continue with how the actual elicitation activity will be handled. The initial prototype from the system is naturally first created. This is done based on the already existing application, as well as with the comments from the system owners, as they have already gotten feedback from the existing system during its lifetime so they naturally have some understanding of the existing problems. However, they do not have all the required information to improve the application as they don’t know what it is like to use the system in some other organizational units, and that is why also others are needed to be involved in the elicitation to give their own domain knowledge.

As the initial prototype is ready, the interview agenda will be planned and the meetings will be scheduled. The agenda with its main points will involve telling and discussing about the project in general and demonstrating and discussing the already existing application as well as the prototype with the typical usage scenarios in order to discover the requirements. Naturally, also the usage scenario will need to be prepared before the meetings. The pre-defined agenda will also include some information about the basic already identified requirements.

Furthermore, there will be separate meetings held for the different contact persons and groups, and the prototype will be constantly evolved based on the feedback from these interviews. In this case only one round with the interviews is likely to be enough, but also more iterations are possible if an appropriate solution is not received. I, as the requirements engineer, will be in charge of leading the interviews: presenting the materials as well as generating conversation and keeping it on track. Also, in addition to the representative contact persons, at least one representative system owner shall be involved in the interviews as well to give the required domain knowledge.

During the interviews I will also take notes, and aim to reflect back to conform that the things said are understood correctly. Moreover, the meetings will also be recorded, which makes it easier to analyze the interviews later on and helps with the burden of making the notes. (See Sommerville 2011: 145.) The final notes made from the meetings – i.e. requirements in their raw form – will also be sent to the representative participants soon after the meeting to get the possible comments and corrections. The representatives are also to be encouraged to share any requirements that might arise afterwards. Moreover, first and foremost all the meetings possible are to be held face-to-face, but as we are talking about a global organization and application, also skype meetings are likely to be required, which obviously is not as effective way to keep these kind of interviews, but is necessary under the circumstances.

Requirements analysis and agreements will involve organizing and prioritizing the raw requirements, as well as finding and resolving any problems existing. First, the requirements shall be divided into functional and non-functional requirements and inside of those two categories they shall further be organized based on some recognized patterns around which to group the requirements, which also brings up the relationships among them (see IEEE 1996: 18). Moreover, also any duplicates shall be extracted.

Furthermore, the organized list of the raw requirements will be reviewed with the system owners, possibly also involving other people with special expertise if necessary, in order to correct any problems: confusing, extraneous, conflicting or missing requirements. In this case the end-user representatives will not be involved in the negotiations as that would require a lot more time and other resources, and as the system owners are responsible for the system they want to be the ones to make the final decisions. They also have the best overall understanding on what the application should include, and what can be done in the limits of the existing database. The prototype will also play a great role in those reviews as it helps to identify those problems.

Further on, the negotiations will also include discussion and conclusion on the prioritization of the requirements. Decisions about the relative priority of the requirements is important since there are typically limited resources to fulfill all of the requirements, at least in the first version of the application (see Benyon 2014: 140). In this research the so called *MoSCoW rules* will be used in the prioritization. The word MoSCoW is an acronym derived from its prioritization categories, which are as follows (Benyon 2014: 140):

- ***Must have*** – which includes the fundamental requirements without which the system would be useless or unworkable.
- ***Should have*** – which includes the requirements that would be essential if more time were available, but the system is useful and usable also without them
- ***Could have*** – which includes the requirements with lesser importance, which can be easily left out of the current development
- ***Want to have but Won't have this time around*** – which can wait for later development i.e. represent the future improvements.

Requirements specification activity, also referred as requirements representation or modeling (see Laplante 2014: 12), involves converting the information gathered in the analysis into a requirements specification. In this research I will refer the produced requirements specification as an SRS, and it will by most part be a natural language document including the organized and prioritized functional and non-functional requirements, but also including the produced prototype to provide a visual representation for those written requirements. In fact, the use of prototypes as part of the SRS has widely increased (see Benyon 2014: 139–140).

The requirements and the whole SRS document will be stored and realized in an application lifecycle management software called *Polarion ALM*, which is currently used inside the organization and can therefore be utilized throughout the whole development project and product life cycle, as the tool enables the requirements to be changed relatively easy, as modifiability is one important characteristic of a good SRS (see Laplante 2014: 94, IEEE 1998: 8).

Overall, the requirements shall be presented in a way that the document can be used to communicate the requirements with the different stakeholders: with the customers as well as with the developers, also including the testers and the maintenance engineers further in the development process as the project proceeds. Therefore, the final SRS shall include both the user and the system requirements, as well as the prototype to give the visual form of representation. Furthermore, the requirements shall be represented in clear language and formulated so that it will also be possible to test whether the final system fulfills the requirements (see Benyon 2014: 140), meaning for instance that the non-functional requirements are presented in a quantifiable manner, so that the requirements are verifiable (see IEEE 1998: 7). Although in reality the final acceptability of the system can only be tested when the system is integrated in its actual environment.

Also, an important thing to remember is that the requirements shall not intend to specify how the technology will meet the requirements, although in practice there will inevitably be some design and implementation details included. Furthermore, for example the IEEE (1998: 4) lists important characteristics of a good SRS that are to be considered. These include that the SRS shall be: *correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable* and *traceable,* which shall be pursued.

Requirements validation is the last activity in the RE process involving the requirements validation and verification. This phase will include reviews with the system owners to complete the final checks to ensure that the overall SRS document is complete, i.e. that it includes all the significant requirements and that the requirements are realistic, consistent and complete. The underlying aim of this activity is to ensure that the described system really satisfies the customer needs and is the right product to be implemented. The prototype will again serve to support these reviews. In the end, the activity aims to correct all the errors that were still left in the SRS, and finally getting the needed approvals for the document.

5 REQUIREMENTS SPECIFICATION

After a number of different RE activities, the requirements engineering phase is now complete, and this chapter presents the results of this phase. As the ultimate intention of this research was to produce the requirements specification i.e. the SRS document, this chapter will include the main results that are included in this technical documentation intended for the internal use of the company. In other words, this chapter presents all the significant requirements for the improved TDS web application; it describes the system-to-be, but does not describe how the product will be received, although some design details are also inevitably included.

More precisely, this chapter includes descriptions of the different user profiles needed for the application, the produced UI mockup prototype that is kept as part of the SRS to provide visualization and enable better communication of the requirements, as well as the main functional and non-functional requirements in a written form in order to specify the application in detail. In the end, the discovered requirements are the important results of this phase, and are needed as an input for the other development activities to become.

Overall, the RE process followed the predefined research design. In practice, altogether ten interviews were held with the system end-users and approximately thirty-five end-user representatives participated in those meeting, also reviews with the system owners where kept regularly, at least once a week, during the whole RE process, also additional meetings with some experts were held. As a result, the process produced twenty-six non-functional requirements, and twelve high-level functional user requirements further on specified with 128 system requirements, which are presented in this paper. However, in reality there are a bit more requirements as some were left out form this paper as they were classified as confidential.

Furthermore, although almost all the significant requirements are included in this paper, the actual SRS document naturally also includes additional information about the requirements that are not described here, as well as some details that were found confidential, which are therefore presented in this paper only as more vague statements. Moreover, naturally some changes to these requirements are to be expected later in the development process: latest in the maintenance phase.

5.1 Application user profiles

At the moment, the users of the application consist of the company internal employees, as the application is currently only meant for the internal business use of the organization. However, in the future also the company's joint ventures shall potentially have access to the application. Although this possibility is taken into account, they are not included in the RE activities.

The different user groups identified in the stakeholder identification match the user profiles of the system which are also strongly connected to the underlying organizational structure. Furthermore, the exact representatives involved in the RE process can be found from the detailed technical documentation, as well as additional sub-groups that were utilized in the elicitation activity. In conclusion, the users of the application are to be divided at least in the following user profiles that are presented below, and one user of the application can naturally belong to one or more of these profiles.

1. *General user*
2. *Marine, Sales*
3. *Marine, Project Management*
4. *Marine, Product Management*
5. *Technology & EPC*
6. *Factory*
7. *Services*
8. *Energy Solutions*
9. *Administrator*
10. *Joint Venture.*

These different profiles are required, as the application holds data content, which access shall be restricted only to certain user profiles. Especially in the future, as the data range available through the application will likely become wider, the different user profiles are needed in authorizing the data. From the technical viewpoint these user profiles shall be mainly associated with the already existing Active Directory user groups. Moreover, as this paper will not include detailed information about the access to the specific data content, only two different user profiles are separated for the system based on the main application functionalities: the *typical end-user* (1 – 8, 10), and the *administrator* (9).

5.2 Prototype representation

Before presenting the requirements in a written form, screenshots from the produced UI mock-up prototype will be used to demonstrate the behavior of the system. Overall, during the RE process, the prototype not only played a great role in the requirements elicitation, analysis, validation, and in producing the written form of requirements, but it is also kept as part of the SRS documentation to assist in communicating the requirements between the different stakeholders as the project continues.

More in detail, the prototype as part of the SRS aims to quickly provide a clear mental picture of the system, and to ensure a mutual understanding of the requirements (see Heisler, Tsai & Ramamoorthy 1989: 348). The prototype should as well make it easier for the different readers of the document to understand and give context for the written requirements. Overall, the prototype is important part of the SRS as it gives a clear expression of the system, which is found essential for the success of any software development process (see Heisler et al. 1989: 348). Moreover, according to the IEEE (1998: 9) an SRS that is based on a prototype tends to undergo less changes during the development, which is naturally something that should be pursued.

In this research, the prototype aims to visualize the most relevant functionalities and concepts of the application, whereas the written form of requirements aims to provide more detailed description of the system-to-be. Although some characteristics of the system, like screen or report formats, may be directly derived from the prototype (see IEEE 1998: 9), as the prototype naturally contains some design details that are not defined by the written form of requirements.

Furthermore, the actual prototype contains links between the different layouts that were found important especially in the elicitation phase to easily demonstrate the different usage scenarios in the prototype system. In this paper, I have only included screenshots from the prototype, leaving out some details and layouts with lesser importance, as well as any of the sensitive and confidential data content. The full version of the prototype is naturally included in the company internal documentation. The diagram below presents a simple navigation map through the prototype i.e. through the system-to-be.

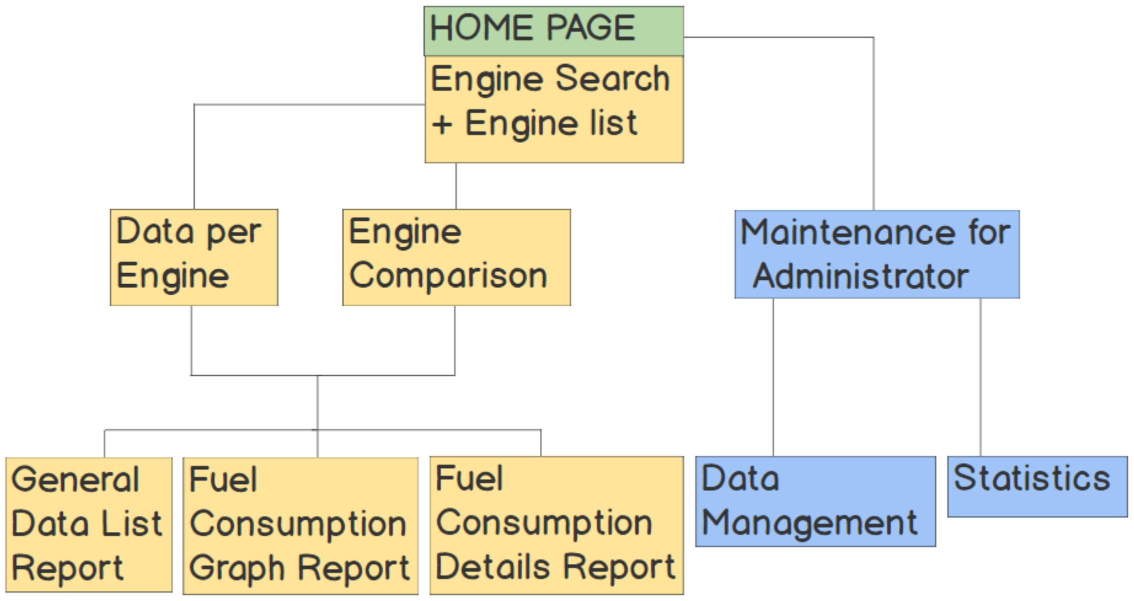

HOME PAGE
Engine Search + Engine list
Data per Engine
Engine Comparison
Maintenance for Administrator
General Data List Report
Fuel Consumption Graph Report
Fuel Consumption Details Report
Data Management
Statistics

Figure 19. Navigation map through the application.

The navigation map above demonstrates how the users move through the application. The typical end-user would search engines, and then either view data per one engine or add multiple engines to comparison and view the data per multiple engines enabling comparison. As a result, the different data and graph reports shall be available for the user. The administrator on the other hand would use this same application user interface to manage the data available in the application, as well as to view statistics related to the usage of the application. I start by presenting the application prototype first through the eyes of the typical user, and then from the administrator's perspective.

The next picture demonstrates the engine search layout, which is also the home page of the application.

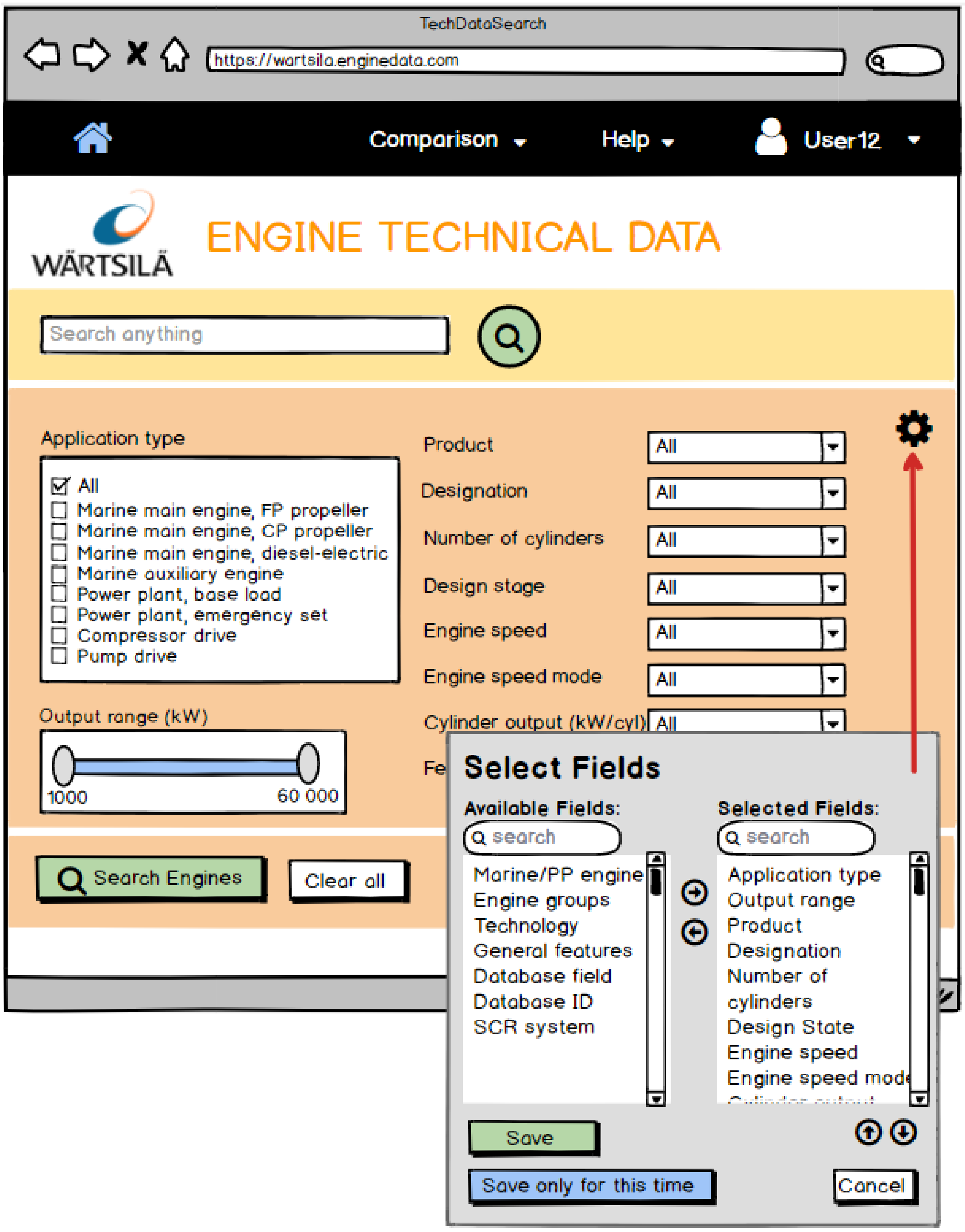
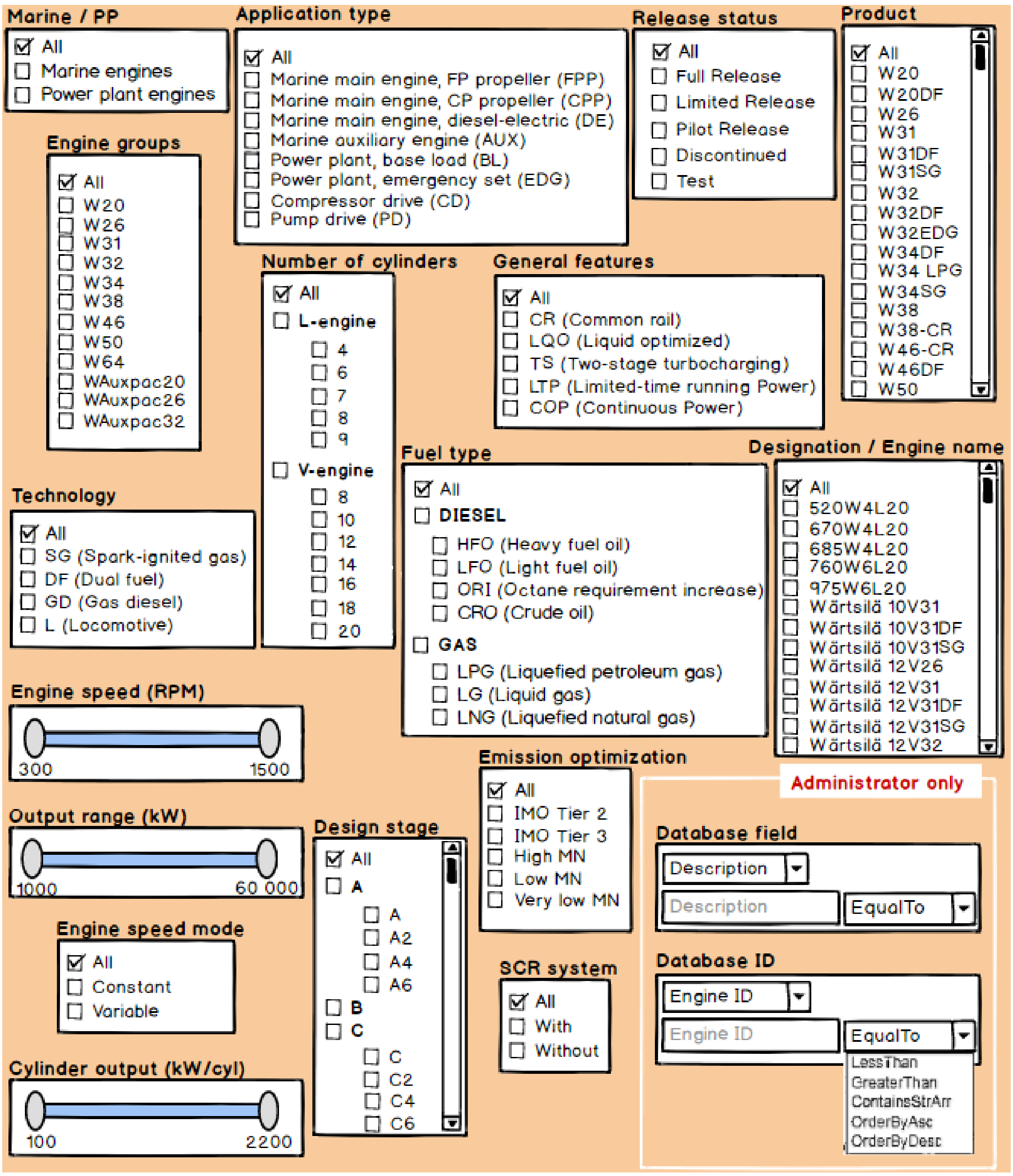


Figure 20. Demonstration of the engine search layout, which also works as the home page for the application.

The basic functionality for this layout enables users to *(1) search engines based on different criteria*, and to *(2) modify the search view i.e. determine the visible search criteria*.

This layout above includes some example search alternatives, but the next picture will give more detailed view on the preferred search criteria.



Marine / PP
- ☑ All
- ☐ Marine engines
- ☐ Power plant engines

Application type
- ☑ All
- ☐ Marine main engine, FP propeller (FPP)
- ☐ Marine main engine, CP propeller (CPP)
- ☐ Marine main engine, diesel-electric (DE)
- ☐ Marine auxiliary engine (AUX)
- ☐ Power plant, base load (BL)
- ☐ Power plant, emergency set (EDG)
- ☐ Compressor drive (CD)
- ☐ Pump drive (PD)

Release status
- ☑ All
- ☐ Full Release
- ☐ Limited Release
- ☐ Pilot Release
- ☐ Discontinued
- ☐ Test

Product
- ☑ All
- ☐ W20
- ☐ W20DF
- ☐ W26
- ☐ W31
- ☐ W31DF
- ☐ W31SG
- ☐ W32
- ☐ W32DF
- ☐ W32EDG
- ☐ W34DF
- ☐ W34 LPG
- ☐ W34SG
- ☐ W38
- ☐ W38-CR
- ☐ W46-CR
- ☐ W46DF
- ☐ W50

Engine groups
- ☑ All
- ☐ W20
- ☐ W26
- ☐ W31
- ☐ W32
- ☐ W34
- ☐ W38
- ☐ W46
- ☐ W50
- ☐ W64
- ☐ WAuxpac20
- ☐ WAuxpac26
- ☐ WAuxpac32

Number of cylinders
- ☑ All
- ☐ L-engine
  - ☐ 4
  - ☐ 6
  - ☐ 7
  - ☐ 8
  - ☐ 9
- ☐ V-engine
  - ☐ 8
  - ☐ 10
  - ☐ 12
  - ☐ 14
  - ☐ 16
  - ☐ 18
  - ☐ 20

General features
- ☑ All
- ☐ CR (Common rail)
- ☐ LQO (Liquid optimized)
- ☐ TS (Two-stage turbocharging)
- ☐ LTP (Limited-time running Power)
- ☐ COP (Continuous Power)

Technology
- ☑ All
- ☐ SG (Spark-ignited gas)
- ☐ DF (Dual fuel)
- ☐ GD (Gas diesel)
- ☐ L (Locomotive)

Fuel type
- ☑ All
- ☐ DIESEL
  - ☐ HFO (Heavy fuel oil)
  - ☐ LFO (Light fuel oil)
  - ☐ ORI (Octane requirement increase)
  - ☐ CRO (Crude oil)
- ☐ GAS
  - ☐ LPG (Liquefied petroleum gas)
  - ☐ LG (Liquid gas)
  - ☐ LNG (Liquefied natural gas)

Designation / Engine name
- ☑ All
- ☐ 520W4L20
- ☐ 670W4L20
- ☐ 685W4L20
- ☐ 760W6L20
- ☐ 975W6L20
- ☐ Wärtsilä 10V31
- ☐ Wärtsilä 10V31DF
- ☐ Wärtsilä 10V31SG
- ☐ Wärtsilä 12V26
- ☐ Wärtsilä 12V31
- ☐ Wärtsilä 12V31DF
- ☐ Wärtsilä 12V31SG
- ☐ Wärtsilä 12V32

Engine speed (RPM)
300 – 1500

Output range (kW)
1000 – 60 000

Engine speed mode
- ☑ All
- ☐ Constant
- ☐ Variable

Cylinder output (kW/cyl)
100 – 2200

Design stage
- ☑ All
- ☐ A
  - ☐ A
  - ☐ A2
  - ☐ A4
  - ☐ A6
- ☐ B
- ☐ C
  - ☐ C
  - ☐ C2
  - ☐ C4
  - ☐ C6

Emission optimization
- ☑ All
- ☐ IMO Tier 2
- ☐ IMO Tier 3
- ☐ High MN
- ☐ Low MN
- ☐ Very low MN

SCR system
- ☑ All
- ☐ With
- ☐ Without

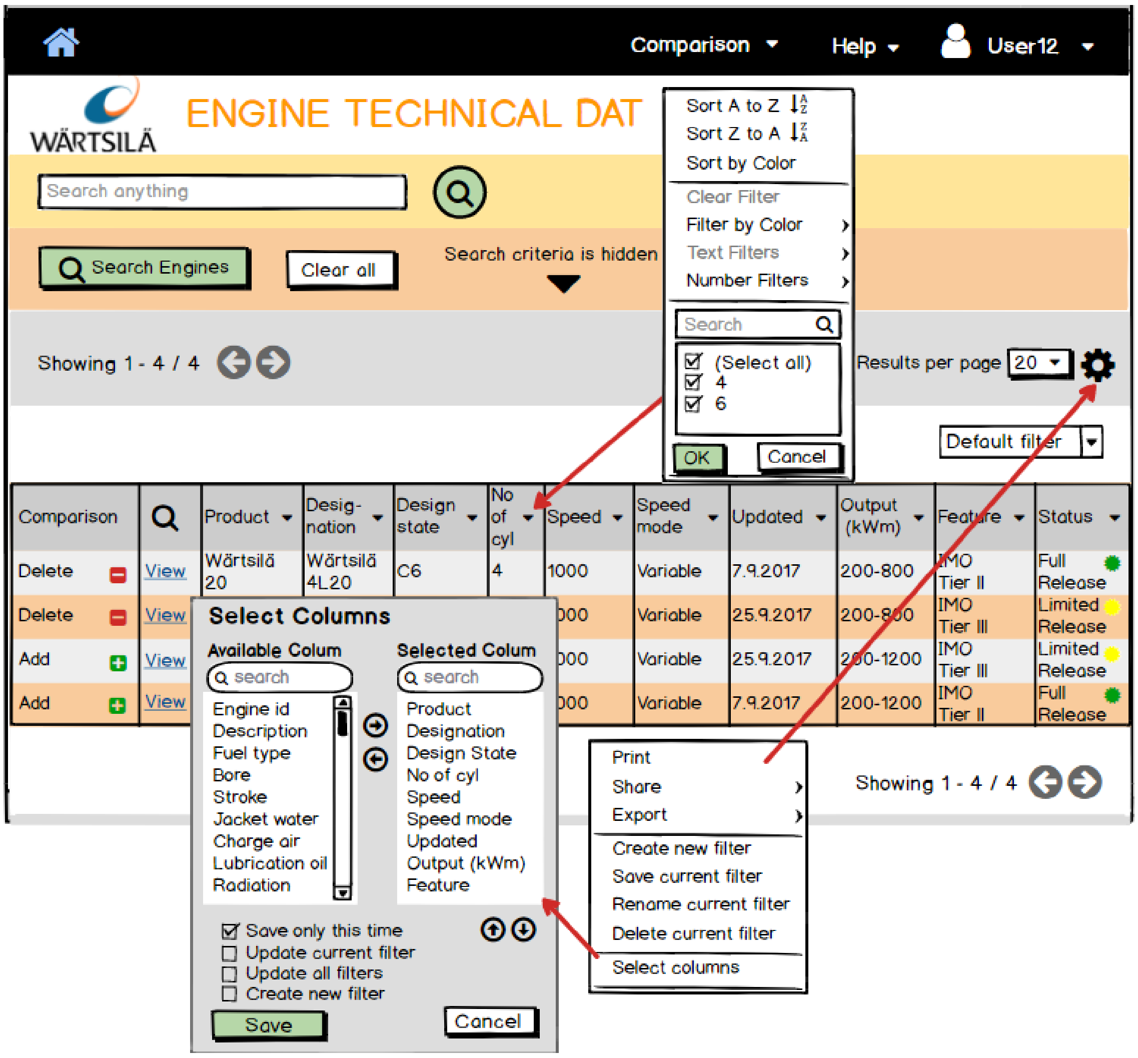
Administrator only

Database field
Description ▾
Description | EqualTo ▾

Database ID
Engine ID ▾
Engine ID | EqualTo ▾

Figure 21. Demonstration of the preferred search criteria.

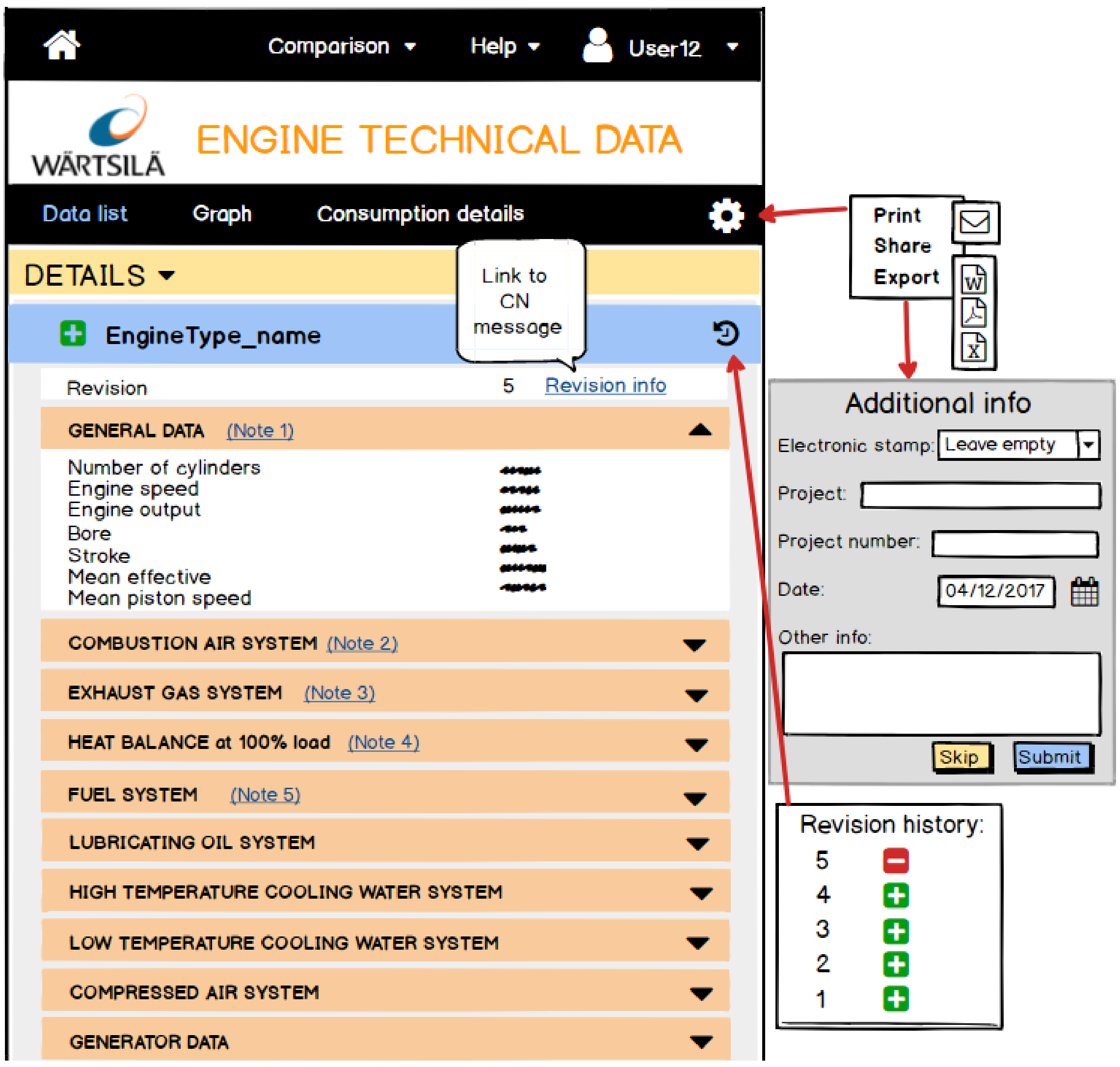
After the user has entered the search criteria and completed the search, a list of the engines matching the search shall be presented. The next picture demonstrates this list.



Comparison		Product	Desig-nation	Design state	No of cyl	Speed	Speed mode	Updated	Output (kWm)	Feature	Status
Delete	View	Wärtsilä 20	Wärtsilä 4L20	C6	4	1000	Variable	7.9.2017	200-800	IMO Tier II	Full Release
Delete	View					000	Variable	25.9.2017	200-800	IMO Tier III	Limited Release
Add	View					000	Variable	25.9.2017	200-1200	IMO Tier III	Limited Release
Add	View					000	Variable	7.9.2017	200-1200	IMO Tier II	Full Release

Figure 22. Demonstration of the list of engines matching the search criteria.

The demonstration includes presenting the available engines in a table form. The most important functionalities provided by the engine list include: *(1) view the available data per engine, (2) add or delete engine from comparison, (3) determine the visible table columns i.e. the data that is visible from the engines, (4) filter, sort and search based on the different table column values*, (5) *print, share or export the engine list*, and *(6) create and save own filters*. The next picture presents the ***general data list report*** for an engine, which will open as the user chooses to view the data available for a specific engine.


Comparison ▾ Help ▾ User12 ▾

WÄRTSILÄ

ENGINE TECHNICAL DATA

Data list | Graph | Consumption details

DETAILS ▾

Link to CN message

EngineType_name

Revision 5 Revision info

GENERAL DATA (Note 1)

Number of cylinders
Engine speed
Engine output
Bore
Stroke
Mean effective
Mean piston speed

COMBUSTION AIR SYSTEM (Note 2)

EXHAUST GAS SYSTEM (Note 3)

HEAT BALANCE at 100% load (Note 4)

FUEL SYSTEM (Note 5)

LUBRICATING OIL SYSTEM

HIGH TEMPERATURE COOLING WATER SYSTEM

LOW TEMPERATURE COOLING WATER SYSTEM

COMPRESSED AIR SYSTEM

GENERATOR DATA

Print
Share
Export

Additional info

Electronic stamp: Leave empty
Project:
Project number:
Date: 04/12/2017
Other info:

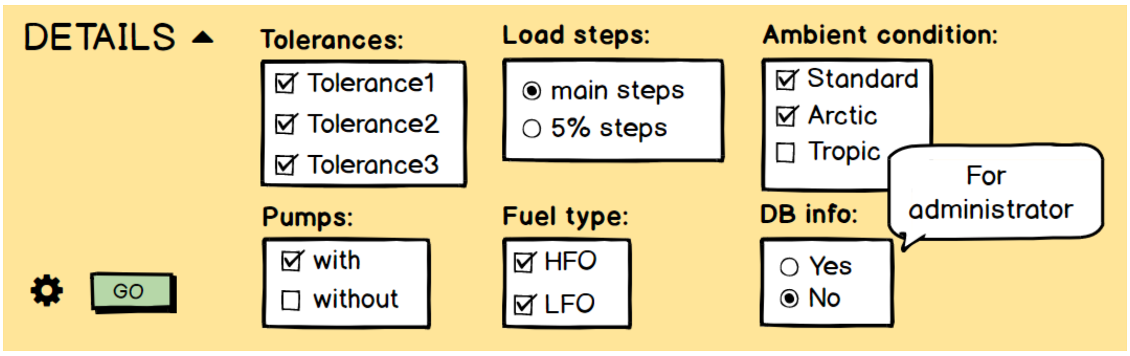
Skip Submit

Revision history:

5	⊟
4	⊞
3	⊞
2	⊞
1	⊞

Figure 23. Demonstration of the general data list report.

The general data list report view enables users to perform the following main activities: *(1) fetch and compare different data revisions, (2) display the detailed revision documentation, also referred as CN (Change Notice) message, (3) print, share and export the available data, (4) add to / delete from comparison,* and *(5) tailor the general data list report (in the picture referred as details).* Tailoring of the report is demonstrated in the picture below.



Tolerances: [✓ Tolerance1] [✓ Tolerance2] [✓ Tolerance3]

Load steps: [✓ main steps] 5% steps

Ambient condition: [✓ Standard] [✓ Arctic] Tropic

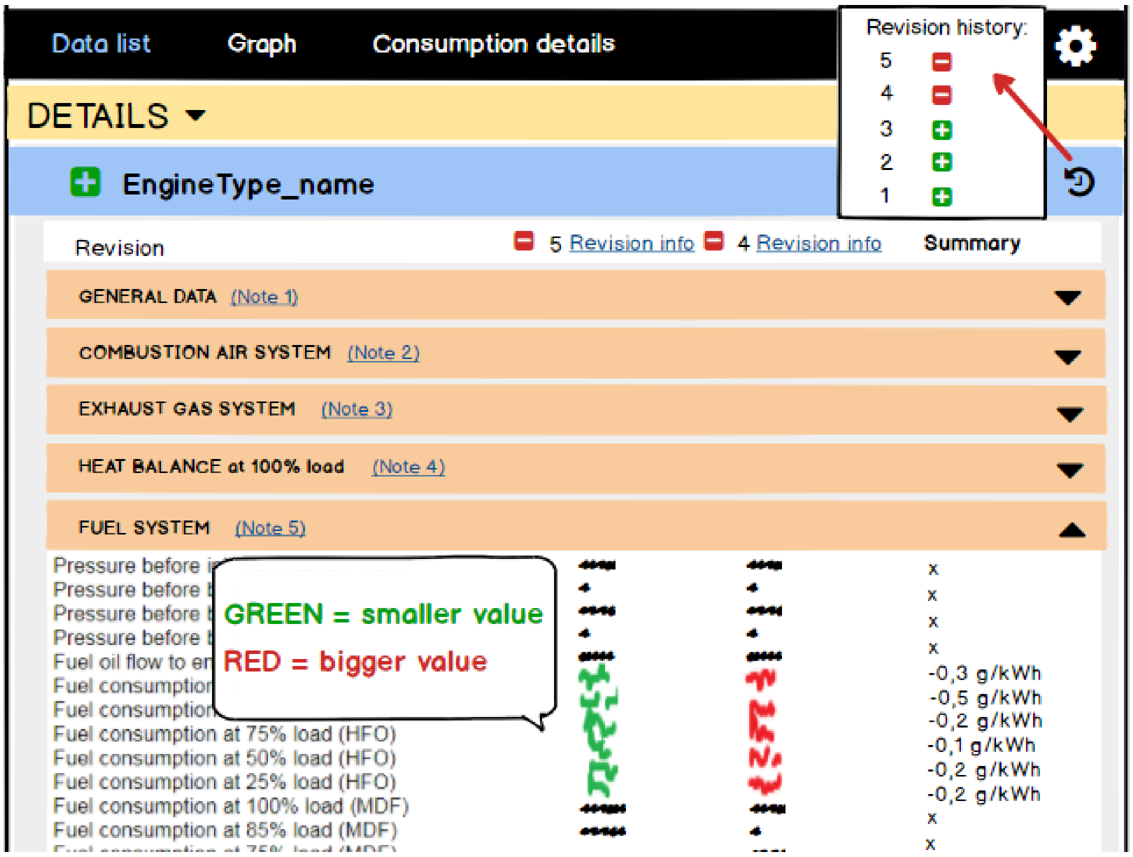
Pumps: [✓ with] without

Fuel type: [✓ HFO] [✓ LFO]

DB info: Yes [✓ No]

Figure 24. Demonstration of the options available to tailor the data list.

The basic idea behind the tailoring options is that the user can restrict the values that are visible in the produced report that may be further on shared with the company’s customers or with colleges, etc. To continue, the next picture demonstrates the comparison of the different data revisions, as users shall be enabled to fetch earlier data revisions and perform quick comparison of the changes between those revisions.



Revision history:

5	🟥
4	🟥
3	🟩
2	🟩
1	🟩

GENERAL DATA (Note 1)

COMBUSTION AIR SYSTEM (Note 2)

EXHAUST GAS SYSTEM (Note 3)

HEAT BALANCE at 100% load (Note 4)

FUEL SYSTEM (Note 5)

Revision	🟥 5 Revision info	🟥 4 Revision info	Summary
Pressure before i	[illegible]	[illegible]	x
Pressure before	[illegible]	[illegible]	x
Pressure before	[illegible]	[illegible]	x
Pressure before	[illegible]	[illegible]	x
Fuel oil flow to en	[illegible]	[illegible]	x
Fuel consumption	[illegible]	[illegible]	-0,3 g/kWh
Fuel consumption	[illegible]	[illegible]	-0,5 g/kWh
Fuel consumption at 75% load (HFO)	[illegible]	[illegible]	-0,2 g/kWh
Fuel consumption at 50% load (HFO)	[illegible]	[illegible]	-0,1 g/kWh
Fuel consumption at 25% load (HFO)	[illegible]	[illegible]	-0,2 g/kWh
Fuel consumption at 100% load (MDF)	[illegible]	[illegible]	-0,2 g/kWh
Fuel consumption at 85% load (MDF)	[illegible]	[illegible]	x
[illegible]		[illegible]	x

Figure 25. Demonstration of the revision comparison.

Furthermore, there shall also exist other reports than the general data list report. The next picture demonstrates the ***fuel consumption graph report***.

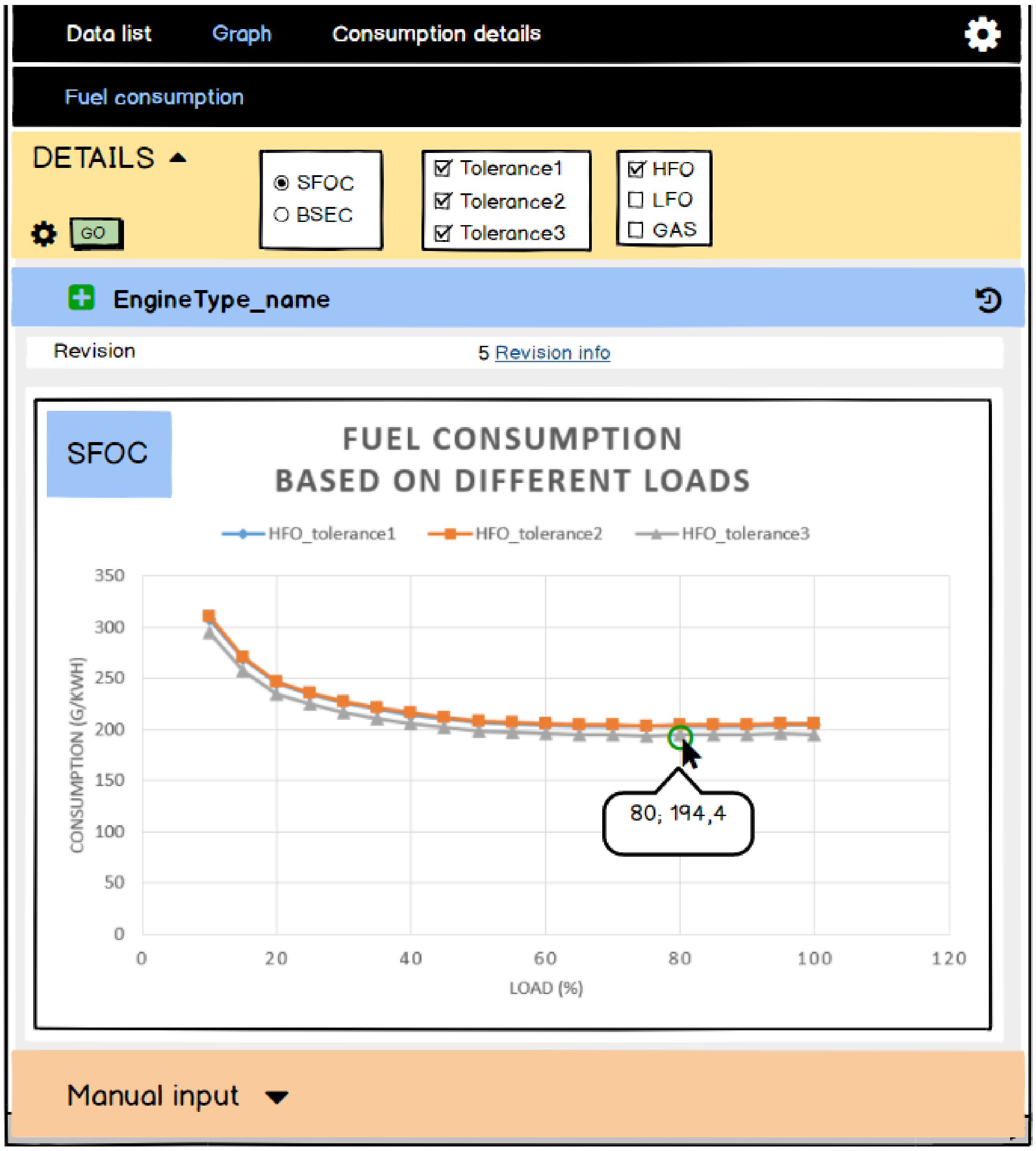


Figure 26. Demonstration of the fuel consumption graph report.

This view contains one new functionality compared to the general data list report: *a manual input*. This manual input can be used to easily add any fuel consumption graph to the same system of coordinates to enable quick comparison between those graphs included.

Moreover, the manual input shall also include an option to *save the added graphs for later use*. The next picture demonstrates the manual input option.

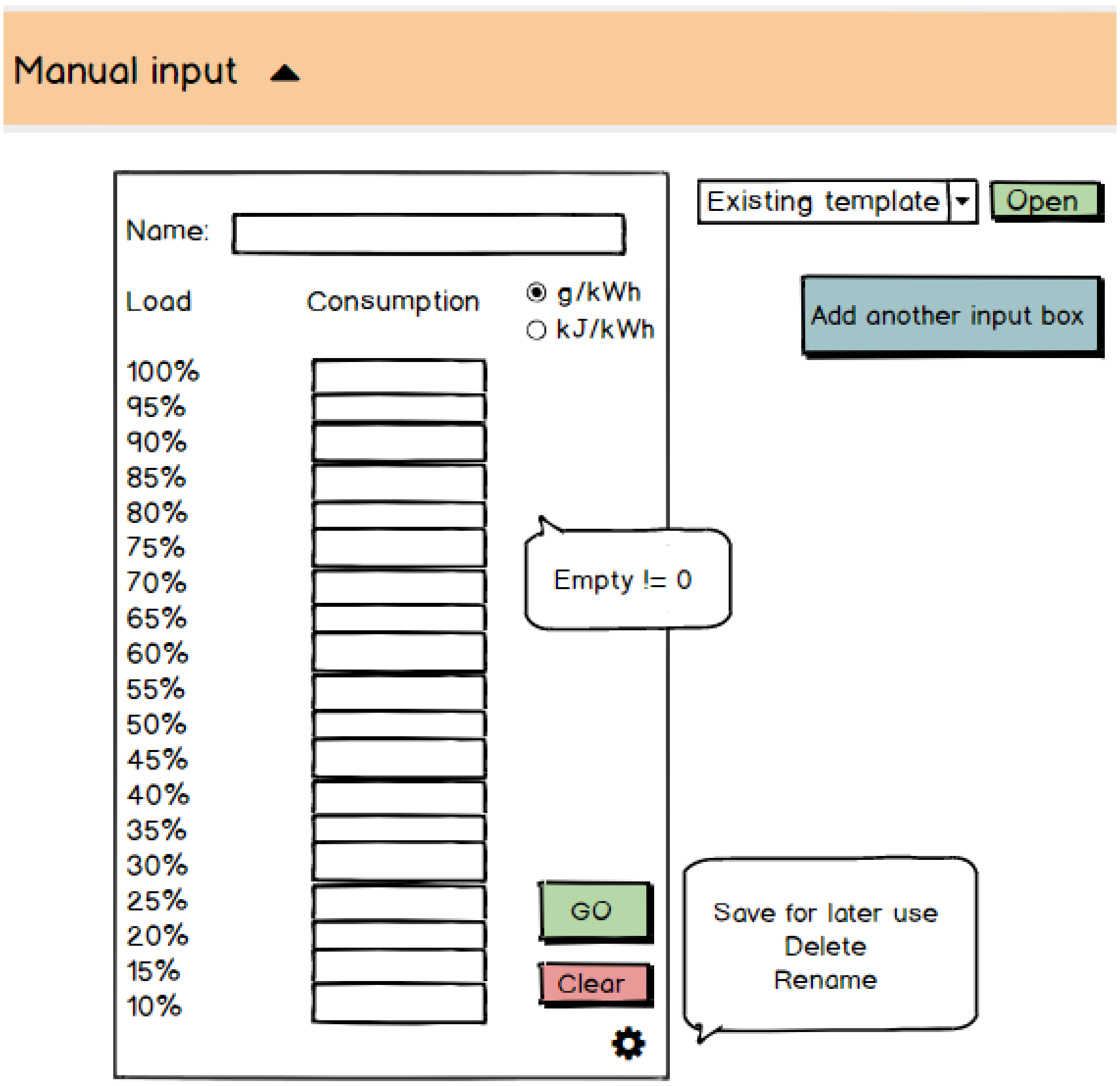
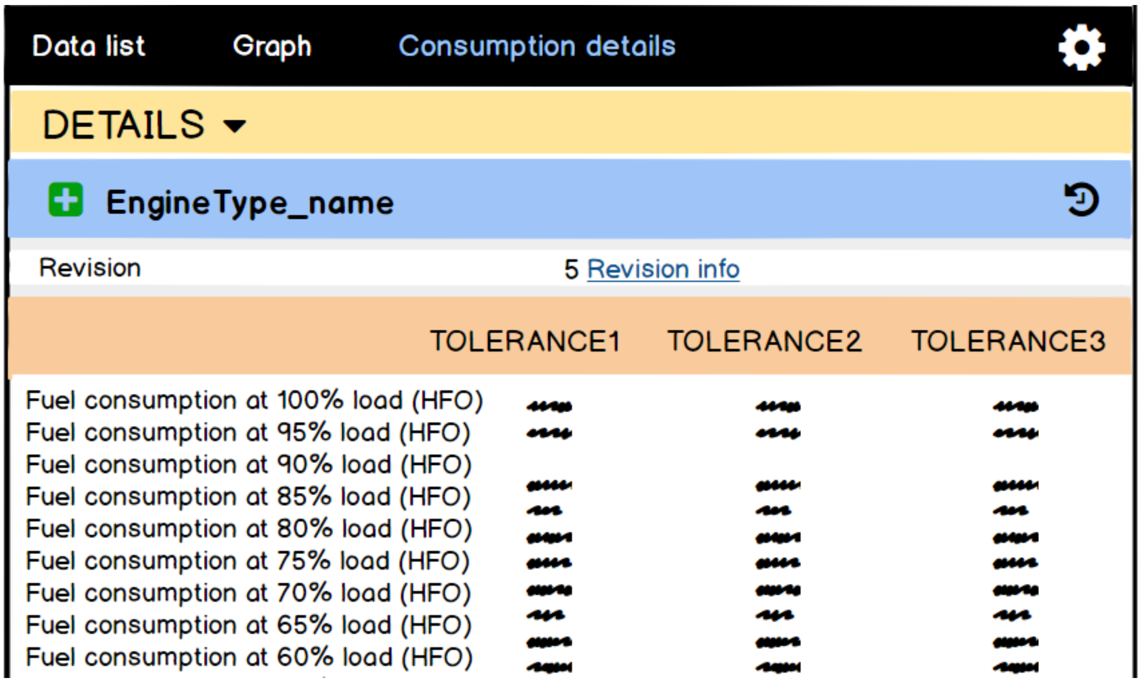


Figure 27. Demonstration of the manual input for the fuel consumption graphs.

One more different report that exists in this prototype is the ***consumption details report***. This view shall present the fuel consumption values with different tolerances side by side, mainly to easy the communication between the management and the sales functions, as well as to provide the values for the factory side's test run purposes. The next picture demonstrates the consumption details view.



DETAILS

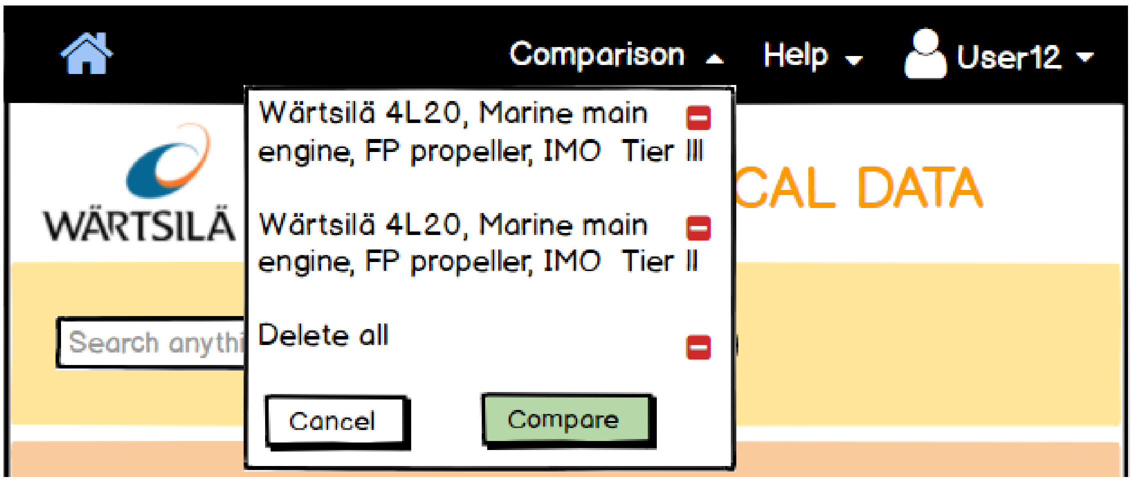
EngineType_name

Revision 5 Revision info

	TOLERANCE1	TOLERANCE2	TOLERANCE3
Fuel consumption at 100% load (HFO)	[illegible]	[illegible]	[illegible]
Fuel consumption at 95% load (HFO)	[illegible]	[illegible]	[illegible]
Fuel consumption at 90% load (HFO)			
Fuel consumption at 85% load (HFO)	[illegible]	[illegible]	[illegible]
Fuel consumption at 80% load (HFO)	[illegible]	[illegible]	[illegible]
Fuel consumption at 75% load (HFO)	[illegible]	[illegible]	[illegible]
Fuel consumption at 70% load (HFO)	[illegible]	[illegible]	[illegible]
Fuel consumption at 65% load (HFO)	[illegible]	[illegible]	[illegible]
Fuel consumption at 60% load (HFO)	[illegible]	[illegible]	[illegible]

Figure 28. Demonstration of the consumption details view.

Now the basic functionality and views behind the user's choice to view the data per one engine has been presented. Now in addition, the typical user of the application shall also be able to compare different engines: in the navigation map referred as ***engine comparison***. The prototype has so far already demonstrated couple of ways to add engines to comparison: from the list of engines and from the different report views. The next picture demonstrates the access to the comparison view.



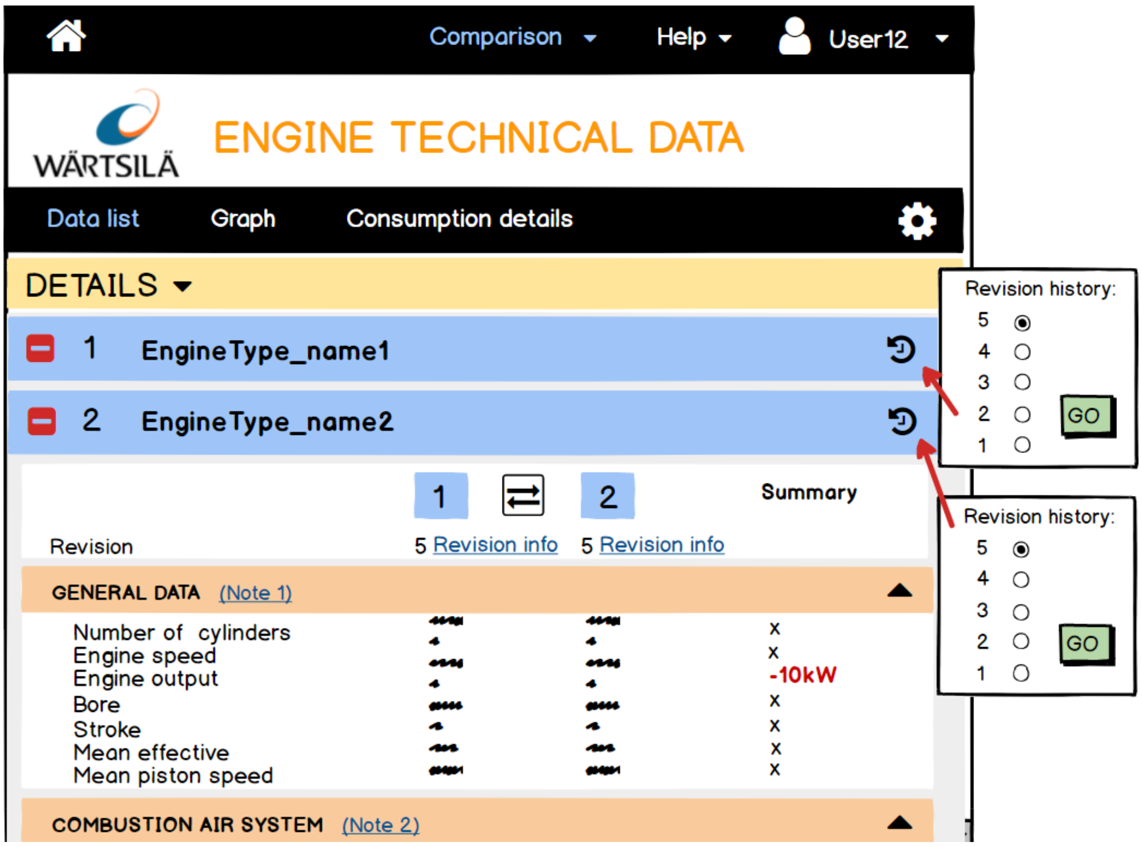
Wärtsilä 4L20, Marine main engine, FP propeller, IMO Tier III

Wärtsilä 4L20, Marine main engine, FP propeller, IMO Tier II

Delete all

Figure 29. Demonstration of the access to engine comparison.

This drop-down window enables the users to *(1) view what is currently in the comparison*, as well as to quickly *(2) delete engines from the comparison*. The same data and graph reports and the related functionalities as in the data per engine, shall in similar way be available in the engine comparison. However, small variations may exist and as it is a comparison the values concerned with the engines shall be presented side by side including the possible summary row, and in case of the graph report the compared engines shall be presented in the same systems of coordinates. The next three pictures below will demonstrate the same views as before when comparing two engines in the engine comparison.



	1	2	Summary
Revision	5 Revision info	5 Revision info	
GENERAL DATA (Note 1)			
Number of cylinders			x
Engine speed			x
Engine output			-10kW
Bore			x
Stroke			x
Mean effective			x
Mean piston speed			x
COMBUSTION AIR SYSTEM (Note 2)			

Figure 30. Demonstration of the general data list report in the engine comparison.

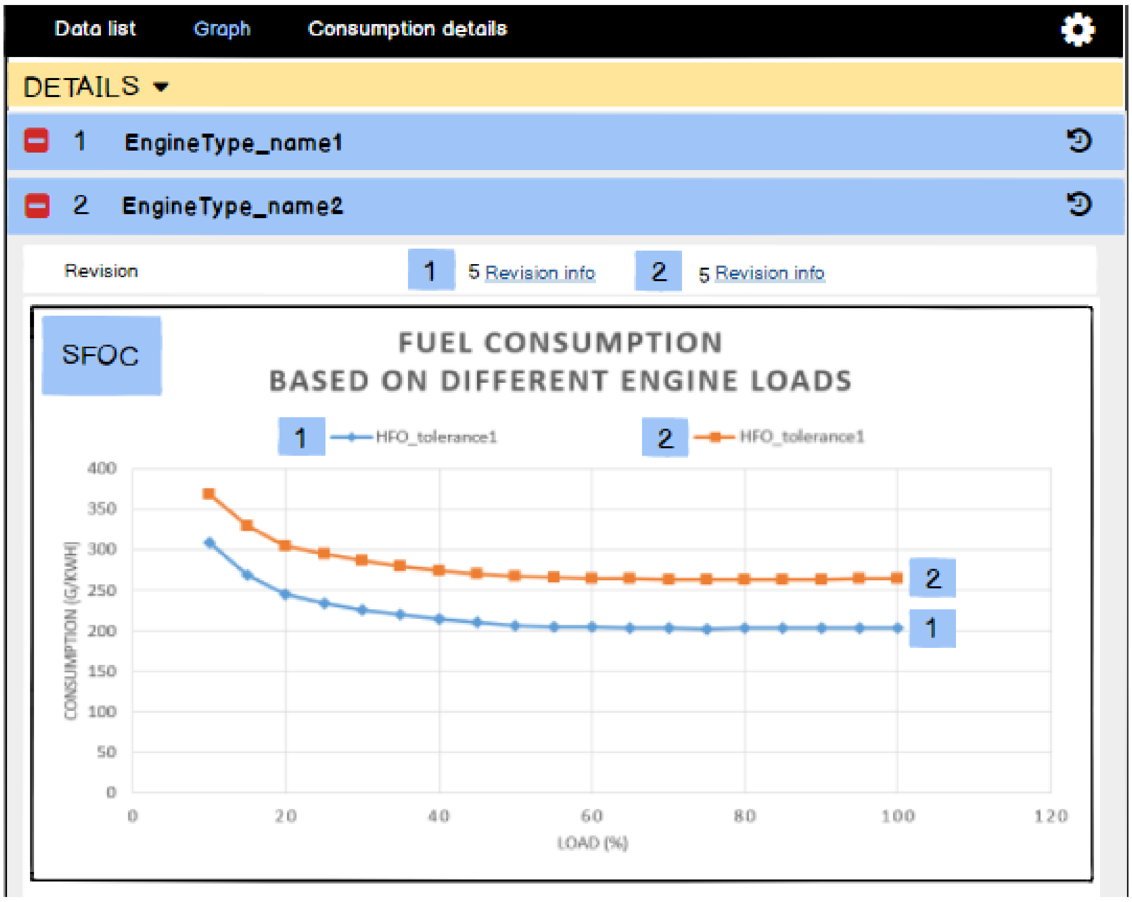
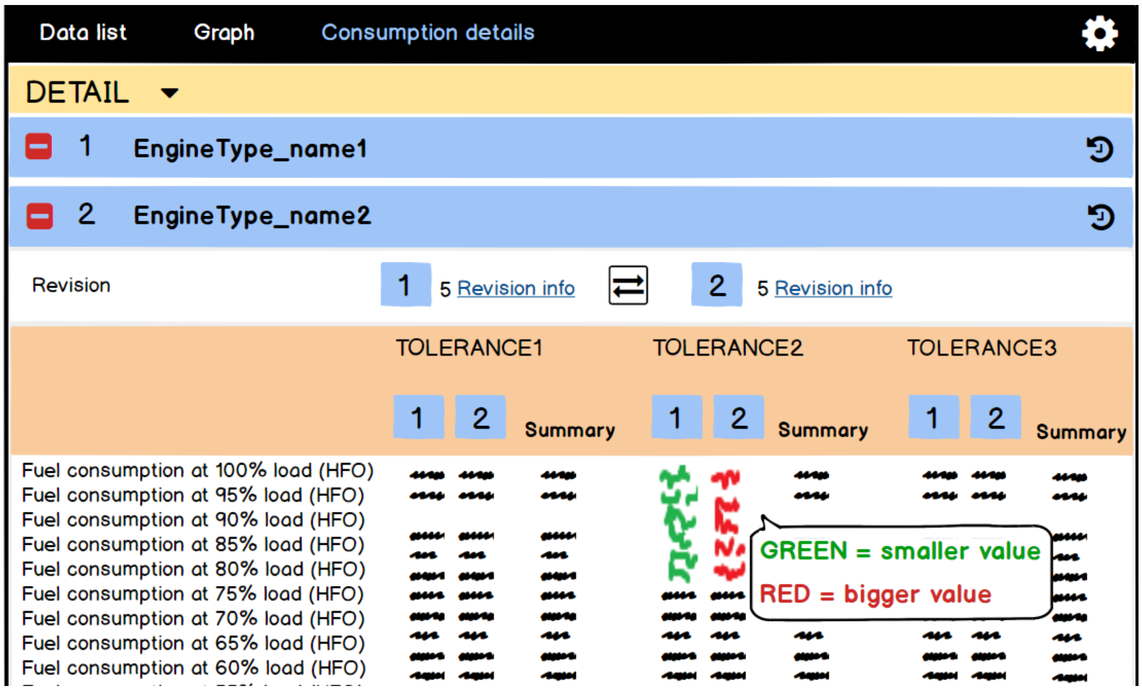


Figure 31. Demonstration of the graph view in the engine comparison.



	TOLERANCE1			TOLERANCE2			TOLERANCE3		
	1	2	Summary	1	2	Summary	1	2	Summary
Fuel consumption at 100% load (HFO)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Fuel consumption at 95% load (HFO)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Fuel consumption at 90% load (HFO)				[illegible]	[illegible]				
Fuel consumption at 85% load (HFO)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]				[illegible]
Fuel consumption at 80% load (HFO)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]				[illegible]
Fuel consumption at 75% load (HFO)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]				[illegible]
Fuel consumption at 70% load (HFO)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]				[illegible]
Fuel consumption at 65% load (HFO)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Fuel consumption at 60% load (HFO)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Figure 32. Demonstration of the graph view in the engine comparison.

Now I have gone through the most important functionalities and views for a typical user, and will move on to the administrator role. The administrator shall have access to the ***data management*** as well as to the ***statistics*** related to the usage of the application. The data management shall enable the administrator to manage *(1) what data is available through the application*, as well as to *(2) determine the user profiles that have access to that data, or the different reports in general*. The statistics side on the other hand shall include *(1) information related to the user activity in the system*, as well as *(2) information related to the possible system interruptions*. I have not demonstrated the statistics side in the prototype, but will next demonstrate the data management side with the general data list report. The next two last pictures demonstrate how an administrator can create a new data section (figure 33) and define the authorized user profiles for each data field (figure 34).

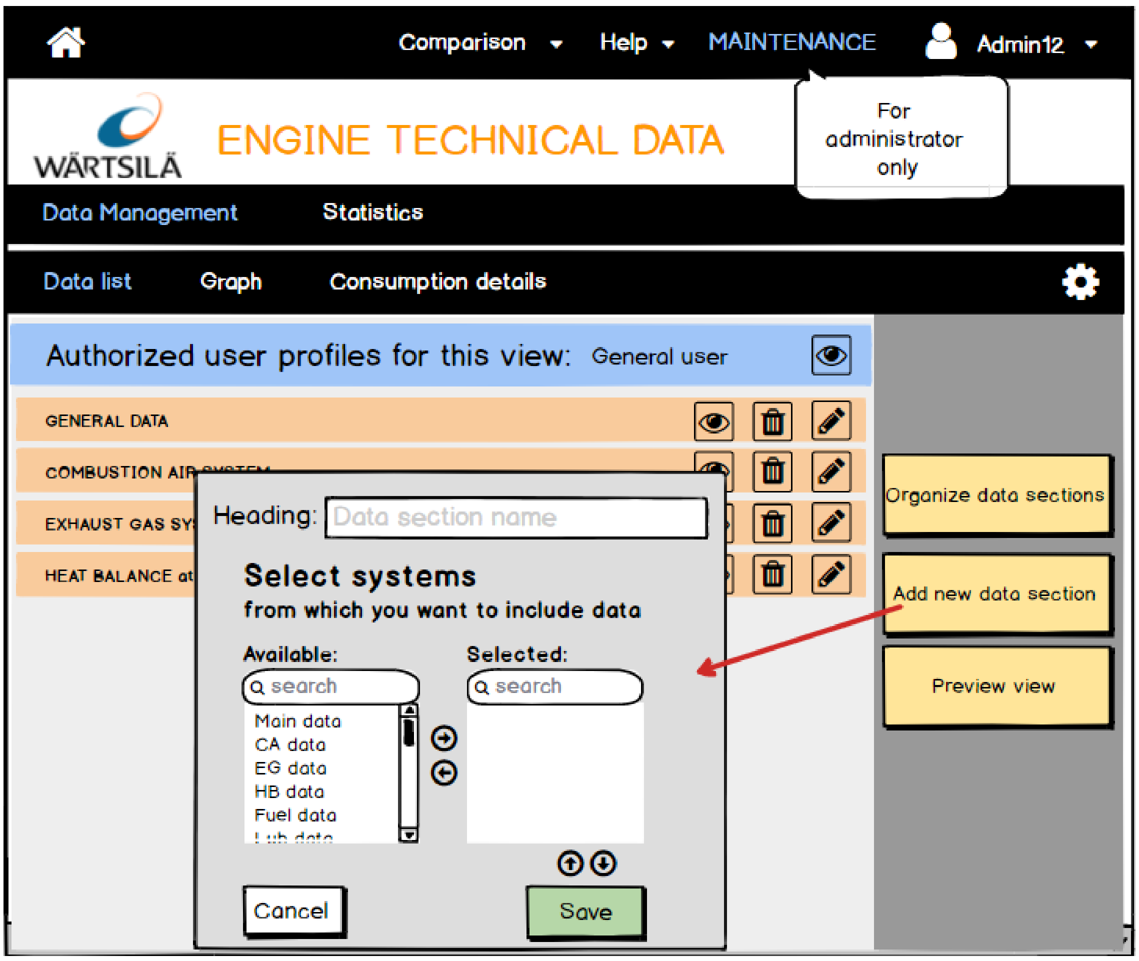
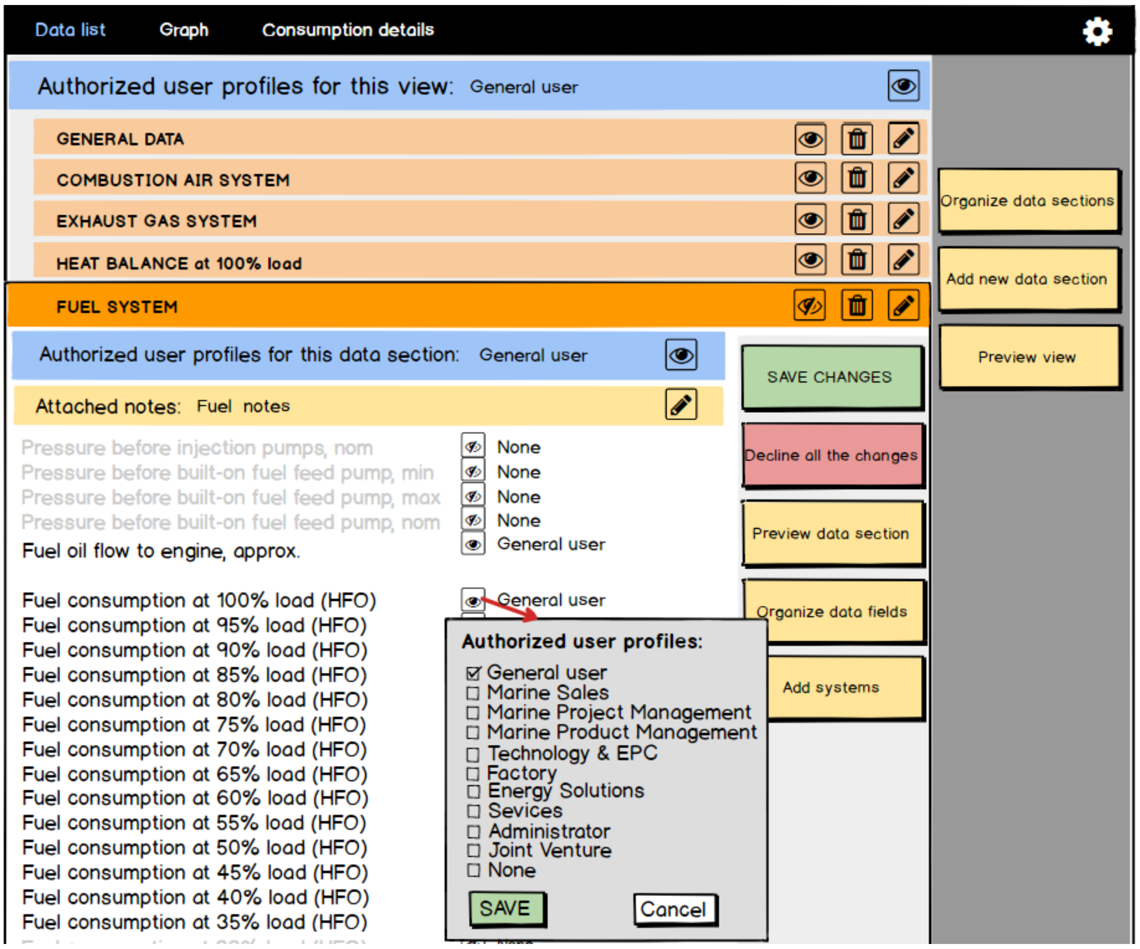


Figure 33. Demonstration of the data management, where the administrator can add a new data sections to the general data list report.


Data list Graph Consumption details

Authorized user profiles for this view: General user

GENERAL DATA

COMBUSTION AIR SYSTEM

EXHAUST GAS SYSTEM

HEAT BALANCE at 100% load

FUEL SYSTEM

Authorized user profiles for this data section: General user

Attached notes: Fuel notes

Pressure before injection pumps, nom	None
Pressure before built-on fuel feed pump, min	None
Pressure before built-on fuel feed pump, max	None
Pressure before built-on fuel feed pump, nom	None
Fuel oil flow to engine, approx.	General user
Fuel consumption at 100% load (HFO)	General user
Fuel consumption at 95% load (HFO)	
Fuel consumption at 90% load (HFO)	
Fuel consumption at 85% load (HFO)	
Fuel consumption at 80% load (HFO)	
Fuel consumption at 75% load (HFO)	
Fuel consumption at 70% load (HFO)	
Fuel consumption at 65% load (HFO)	
Fuel consumption at 60% load (HFO)	
Fuel consumption at 55% load (HFO)	
Fuel consumption at 50% load (HFO)	
Fuel consumption at 45% load (HFO)	
Fuel consumption at 40% load (HFO)	
Fuel consumption at 35% load (HFO)	

Authorized user profiles:

- ☑ General user
- ☐ Marine Sales
- ☐ Marine Project Management
- ☐ Marine Product Management
- ☐ Technology & EPC
- ☐ Factory
- ☐ Energy Solutions
- ☐ Sevices
- ☐ Administrator
- ☐ Joint Venture
- ☐ None

Figure 34. Demonstration of the data management, where the administrator can define the authorized user profiles for any particular data field in the general data list report.

5.3 Functional requirements

Functional requirements define what the software is supposed to do, i.e. what are the functionalities it shall offer. The prototype representation already gave a good visualization of the most relevant system functionalities and data content. Furthermore, with the help of the prototype the different functional requirements for the application where written in a natural language form, which gives more detailed descriptions for the requirements.

The list of the functional requirements is structured in a way that the user requirements are described as the main requirements, and the more detailed system requirements are presented as the sub-requirements beneath those general level user requirements. In some cases, the system requirements are even further divided in to lower-level sub-requirements to bring up more detailed dependences between those requirements. Basically, the difference and intention of the classification to user and system requirements is to provide information for the different readers of the document, as the user requirements are mainly intended for the customer stakeholders of the application and the system requirements for the developers.

The written representation of the functional requirements also includes information about the user profiles in the level of detail where the typical user and the administrator are separated. In this paper, if the functionality is only intended for the administrator it is clearly stated in the requirement description and heading part, and otherwise the requirement concerns the typical user. Moreover, also information about the priority of the requirements is included, where the MoSCoW rules are used to classify the requirements. The prioritization is presented below, including the used color coding. Moreover, the full SRS document, as well includes other relevant information about the requirements that improves the traceability of the SRS document, such as the information about where the requirement has come from and what is the motivation to fulfill the requirement.

Table 1. Prioritization used for the requirements.

Must have	*Critical / High priority*
Should have	*Medium priority*
Could have	*Low priority*
Want to have	*For later development*

Furthermore, the prioritization of the requirements shall be interpreted so that if the requirement is a sub-requirement for another requirement, it means that if the higher-level requirement is completed then the priority of the sub-requirement is the following. This makes it possible to describe the dependencies between the requirements so that the higher-level requirement may for instance be a should have, but its sub-requirement can still be a must have, if the higher-level requirement is fulfilled.

For this chapter, I have only included the very much shortened list of the functional requirements, only including the user requirements to provide the overall picture of the system, leaving out the more detailed system requirements. This is as the full list of all the functional requirements is relatively long to be included here. These functional user requirements are presented in the table below, and the full list of the functional requirements can be found from the appendixes (appendix 1).

Table 2. Functional requirements only including the user requirements.

REQUIREMENT	PRIORITY
1. Search engines Users shall be able to search engines easily based on their own preferred search criteria.	Must have
2. Search results: engine list Users shall be able to browse the results from the search, and still continue to limit the results based on their needs in order to find what they were looking for.	Must have
3. General data list report Generate a ready-made data report showing all the necessary technical data for an engine fast and easy, so that this can even be done in front of a customer. Also enable customization of the report.	Must have
4. Fuel consumption graph report Generate ready-made graph report visualizing an engine's fuel consumption. And enable users to insert competitor engine's fuel consumption values to the graphs as well to enable quick comparison.	Must have
5. Fuel consumption details report Generate a ready-made data report that enables users to quickly view fuel consumption values with different tolerances: average, single and 5 % sales tolerance, for an engine.	Must have
6. Engine comparison Enable users to compare different engines, based on the information available in the generated reports.	Must have
7. Revision history Enable users to backtrack the happened changes between different data revisions, also including the reasoning behind the changes as well as the information about who has requested the change.	Must have

8. Export, share & print out content User shall be able to export, share and print out any relevant content produced by the application.	Must have
9. Data management for administrator The data content in the application, as well as the access rights for that content shall be managed through the application by the administrator.	Must have
10. Statistics for administrator The application shall generate different statistic for the administrator, related to the usage of the system.	Should have
11. Help & contact information The application shall contain basic information about who to contact in case of problems or to give feedback.	Must have
12. Information exchange with other internal systems The application shall exchange information with other company internal systems in order to access data, and to provide content to other systems.	Must have

5.4 Non-functional requirements

The non-functional requirements (NFRs) are concerned with the quality that the system must have; i.e. they are the terms and constraints that define how the software shall fulfill the previously described functional requirements. Therefore, NFRs play an important role in the acceptability of the final system (see Benyon 2014: 140). Moreover, when it comes to the NFRs, extra attention is payed to the fact that they are written quantitively, i.e. if necessary they shall be associated with different metrics, in order for the requirements to be measurable and that way verifiable.

In practice, the NFRs were discovered by analyzing the stakeholder goals: analyzing the different challenges and issues the stakeholders were having with the already existing system, as well as the concerns related to the system-to-be. Also, the constraints created by the application runtime environment were examined, and the best practices and other trends in the application domain were observed. Furthermore, the requirements must naturally also comply with the company internal standards and policies, which were also taken into account. Moreover, as typically NFRs are hard to detect, the process of discovering the NFRs, also involved leveraging from the existing templates for NFRs: for example, Paradkar (2017) covers critical NFRs applicable to different IT systems, in his book called: *Mastering Non-functional Requirements*, which was utilized. (see Paradkar 2017; Paakki et al. 2011: 6, 28.)

The discovered NFRs were categorized based on the found discriminant aspects between them. These used categories are presented below.

Table 3. Categories used to classify the non-functional requirements.

1. Maintainability
2. Performance
3. Scalability & Capacity
4. Availability
5. Recovery
6. Security
7. Manageability
8. Interoperability
9. Usability

For this paper I have included the following information about the discovered NFRs: description of the NFR including the target value, the category and the priority. The prioritization uses the same MoSCoW classification that was used with the functional requirements, however the prioritization of the NFRs is independent from the prioritization of the functional requirements because the abstraction is a bit different.

Additional information about NFRs can be found from the full SRS document, such as the related motivation to complete those requirements. For this chapter I have again compressed the representation of the requirements, this time leaving out the related target values. The list of the non-functional requirements with the target value information can be found from the appendixes (appendix 2). The table below presents the compressed list of the NFRs.

Table 4. Non-functional requirements without the target value information.

CLASSIFICATION	REQUIREMENT	PRIORITY
1. Maintainability	*1.1. Supported technology*	Must have
	1.2. Architectural standards	Should have
	1.3. Coding standards	Should have
	1.4. Proper documentation	Should have
2. Performance	*2.1. Response time*	Should have
	2.2. Processing time	Should have
	2.3. Querying time (read only)	Should have
	2.4. Total number of users	Should have
3. Scalability & Capacity	*3.1. Number of concurrent users*	Should have
4. Availability	*4.1. Hours of operation*	Should have
	4.2. Geographic operation	Should have
5. Recovery	*5.1. Mean time to recovery*	Should have
	5.2. Backups	Should have
6. Security	*6.1. Authentication*	Must have
	6.2. Authorization	Must have
	6.3. Wärtsilä network	Must have
	6.4. Organizational compliance	Must have
	6.5. Layered architecture	Must have
7. Manageability	*7.1. Logging & tracking*	Should have
	7.2. Alerts	Could have
8. Interoperability	*8.1. Compatibility with other systems*	Should have
9. Usability	*9.1. Ease of use*	Should have
	9.2. GUI standards	Should have
	9.3. Browser support	Should have
	9.4. Delivery mediums	Should have
	9.5. SSO (Single sign-on)	Should have

6 DISCUSSION AND CONCLUSIONS

Overall this research is part of a software development project that aims to improve an existing web-based business application: and as the needed improvements go beyond the maintenance of the existing system a completely new system is required. The research included the first phase of this development project, covering the requirements engineering activities, and as a result providing a requirements specification for the new application. This SRS document answers the underlying main research question that was to define the functional and non-functional requirements for the system-to-be. This document is necessary in order for the software development to proceed, as it is needed as an input for the coming development activities.

The produced SRS holds the requirements in an organized and prioritized written natural language form, but also provides visualization for those requirements with the assistance of an UI mockup prototype representation. Although, RE is one of the most critical tasks for the success of the software development, giving a base for the project, it is not an easy task to arrive at a complete SRS that holds all the significant requirements, and also fulfills a number of other important quality attributes. In this case for example, the structure of the produced SRS aims to enable the document to be easily evolved during the development, as latest in the maintenance phase the requirements for the application are likely to change. It also aims to provide the requirements in a verifiable way so that in the testing activities it can easily be measured whether the system complies with the requirements.

One of the key things that made this RE process successful was the utilization of prototyping. As missing requirements are one of the biggest reasons why software development projects fail, the prototyping really helped to brought up even the hidden requirements and prevented any misinterpretation of the requirements as it visualized how the different requirements work together in the solution. In addition, it was found that people were more excited to give feedback on the prototype than on the written form of the requirements. In general level, it is also seen that SRSs that are based on a prototype typically are more successful in way that they require less changes during the development.

Moreover, the prototype as part of the SRS document really helps to communicate those requirements to the different stakeholders which is desirable also as the software production continues. However, the written requirements are also important because they specify the requirements in more detail, as the prototype essentially intents to visualize the most relevant concepts and behavior of the system but leaves out some details. Although, in contrast the prototype provides some information that is missing from the written requirements, such as more detailed data content, formats and structure of the different layouts.

If there would have been more time and other resources available to complete this research, the research could have involved more detailed evaluation of the prototype, for instance in form of a walkthrough as part of the requirements validation. It would have also been possible to carry out another iteration round with the requirements elicitation activity, but it was not found necessary as the end-user representatives where in general already happy with the solution, and to carry out another would have required a lot more resources, and people naturally are busy with their own work tasks. Also, it would have been better to keep all the related interviews as face-to-face to get better feedback.

In addition, one of the research questions was to define the appropriate technologies and tools to be used in the implementation of the software. This is relevant as these decisions naturally also effect the following activities in the development process, and are important decision that need to be made. The non-functional requirements specified in the SRS define some constraints that effect these choices, as the application shall be accessed through a web browser, the used technologies shall be supported also in the long-term, and the software shall be compatible with the other needed systems.

Considering the current circumstances, Microsoft's environment seems like the natural choice to continue with also in the future, as this is also used with the existing application which now only needs an upgrade to the up-to-date technologies. That said the .NET Framework or the more modern cross-platform solution of .NET Core, would both work as a natural and good choice for the new application. With the .NET, the web development framework would be the ASP.NET or ASP.NET Core. Further on these technologies support different programming languages, naturally including the HTML which can be used in combination with Visual Basic or C#, or even be assisted with JavaScript and CSS. Visual Studio IDE would in this case be the development tool to be used. Also, the Entity Framework (EF) or EF Core could be utilized as a data access technology which could easy the implementation compared to the typical ADO.NET.

The libraries to be used, let alone the other technology details, are not discussed here. Overall, this environment would easily provide the easy communication with the other Microsoft systems that the application shall interact with. Moreover, the .NET is fully supported and it certainly seems like it will be a good choice also in the long run as it has become popular and it clearly has constantly been evolved to the right direction. This environment is also used and supported by the organization, and it has the needed features for the application to succeed, as it has a rich collection of different functionalities.

One research question was also concerned with giving recommendation for the continuation of this development project. In general, the project shall continue to follow the sequential process flow of the waterfall model as a base, moving next to the design activities, followed by the implementation, testing and maintenance phases. This is as the development is distributed via remote teams or team members where the waterfall model supports the project management wise. However, it is naturally acceptable for those activities to overlap, and I would recommend reviews to be held regularly as well as other testing activities to be started early, as failures in the system tend to grow and become costly if detected late. In addition, all the functionalities defined in the SRS are not even intended to be in the first version of the application, so the development shall be that way incremental, meaning that the software may be taken into use before it is totally complete.

Furthermore, also other development targets inside the organization were emphasized during this development project. These mainly involve the related technical database, which currently does not support the realization of all the discovered requirements. These improvements are important in order for the target application to be successful also in the future as the business evolves: new requirements emerge and the amount of information grows, but they will as well support the correspondence between the other tools and manuals that also retrieve data from the same database.

REFERENCES

Anderson, R., L. Latham, S. Addie, T. Dykstra, D. Roth & A. Pasic (2017). *Choose between ASP.NET and ASP.NET Core.* Guide by Microsoft Corporation [online document]. [17.10.2017] Available online: https://docs.microsoft.com/en-us/aspnet/core/choose-aspnet-framework.

Bean Software (2017). *Classic ASP vs. ASP.NET* [online document]. [6.10.2017] Available online: http://www.beansoftware.com/ASP.NET-Tutorials/Classic-ASP-vs-AS.NET.aspx. Tutorial by Bean Software Oy.

Benyon, David (2014). *Designing Interactive Systems: A comprehensive guide to HCI, UX and interaction design.* Third edition. Pearson Education Limited. United Kingdom: Edinburgh Gate. ISBN: 978-1-4479-2011-3.

Block, G., P. Cibraro, P. Felix, H. Dierking & D. Miller (2014). *Designing evolvable web APIs with ASP.NET.* O'Reilly 2014. First edition. United States of America. ISBN: 978-1-449-33771-1.

Carter, P., M. Wenzel, S. Addie, L. Latham, P. Onderka, T. Pratt, B. Wagner & V. V. Agarwal (2016). *Choosing between .NET Core and .NET Framework for server apps.* Guide by Microsoft Corporation [online document]. [17.10.2017] Available online: https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server.

Cluts, Nancy W. (1997). *An ASP You Can Grasp: The ABCs of Active Server Pages*. Technical articles. Guide by Microsoft Corporation [online document]. [6.10.2017] Available online: https://msdn.microsoft.com/en-us/library/ms972317.aspx?f=255&MSPPError=-2147217396.

Guyer, Craig & Cody Mansfield (2016). *Installing SQL Server Native Client*. Guide by Microsoft Corporation [online document]. [11.10.2017] Available online: https://docs.microsoft.com/en-us/sql/relational-databases/native-client/applications/installing-sql-server-native-client.

Haikala, Ilkka & Jukka Märijärvi (1998). *Ohjelmistotuotanto*. Helsinki: Suomen atk-kustannus 1998. ISBN: 951-762-696-7.

Hall, Jon G. & Lucia Rapanotti (2017). *A design theory for software engineering*. Information and Software Technology: Vol. 87, July 2017, pp. 46–61. ISSN: 0950-5849.

Heisler, K. G., W. T. Tsai & C. V. Ramamoorthy (1989). *Integrating the role of requirements specification into the process of prototyping: the protospec*. Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences; Software Track, Year: 1989, Volume: 2, Pages: 348 – 357. Conference Location: Kailua-Kona, HI, USA, USA. Date of Conference: 3-6 Jan. 1989. Publisher: IEEE. ISBN: 0-8186-1912-0.

Henderson, Cal (2006). *Building Scalable Web Sites.* O'Reilly Media, Inc. First edition. United States of America. ISBN: 0-596-10235-6.

Hoffer, Jeffrey A., Joey F. George & Joseph S. Valacich (2014). *Modern systems analysis and design.* Pearson cop. 2014. 7 ed., International ed. ISBN: 9780273787099.

Hubbard, Jennifer & Craig Guyer (2017). *Using ADO with SQL Server Native Client*. Guide by Microsoft Corporation [online document]. [11.10.2017] Available online: https://docs.microsoft.com/en-us/sql/relational-databases/native-client/applications/using-ado-with-sql-server-native-client.

IEEE (2002). *IEEE Standard Glossary of Software Engineering Terminology.* American National Standard (ANSI). IEEE Std 610.12-1990(R2002). The institute of Electrical and Electronics Engineers, Inc. 345 East 47th Street, New York, NY 10017-2394, USA. ISBN: 0-7381-0391-8, SS13748.

IEEE (1998). *IEEE Recommended Practice for Software Requirements Specification*. In IEEE Software Engineering Standards Collection. Los Alamitos, Ca.: IEEE Computer Society Press. ISBN: 0-7381-0332-2.

IEEE (1996). *Guide for Developing System Requirements Specification*. Software Engineering Committee of the IEEE Computer Society. The Institute of Electrical and Electronics Engineers, Inc. USA: New York. ISBN 1-55937-716-X

Jamsa, Kris (2005). *.NET Web Services Solutions.* Wiley. ISBN: 9780782141726.

Kleppmann, Martin (2017). *Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems.* Beijing: O'Reilly 2017. First edition. ISBN: 978-1-449-37332-0.

Ladas, Corey (2017). *Tom Gilb's "evolutionary delivery," a great improvement over its successors.* Essay by Lean Software Engineering [online document]. [13.10.2017] Available online: http://leansoftwareengineering.com/2007/12/20/tom-gilbs-evolutionary-delivery-a-great-improvement-over-its-successors/.

Laplante, Phillip A. (2014). *Requirements engineering for software and systems*. CRC/Taylor & Francis cop. 2014. 2nd edition. ISBN: 978-1-4665-6081-9.

Laudon, K. C. & C. G. Traver (2016). *E-commerce: business, technology, society*. Harlow, Essex: Pearson, Twelfth edition, global edition. ISBN: 978-1-292-10996-1.

Lönnfors, Sebastian (2012). *Theoretical and practical Requirements Engineering.* Degree Thesis: Information Technology [online document]. [19.1.2018] Available online: https://www.theseus.fi/bitstream/handle/10024/45134/Lonnfors_Sebastian.pdf?sequence=1.

Maciaszek, Leszek A. (2005). *Requirements analysis and system design*. New York: Pearson/Addison Wesley 2005. ISBN: 0-321-20464-6.

Milener, Gene & Craig Guyer (2017). *Microsoft-Supplied ODBC Drivers.* Guide by Microsoft Corporation [online document]. [11.10.2017] Available online: https://docs.microsoft.com/en-us/sql/odbc/microsoft/microsoft-supplied-odbc-drivers.

Paakki, Juha & Juha Taina (2011). *Ohjelmistojen vaatimusmäärittely*. Helsingin yliopisto. Tietojenkäsittelytieteen laitos [online document]. [16.2.2018] Available online: https://www.cs.helsinki.fi/u/paakki/Vaatimus-11-Luentokalvot-1.pdf. Teaching material.

Paradkar, Sameer (2017). *Mastering Non-functional Requirements*. Packt Publishing. ISBN: 9781788299237.

Polvinen, Timo (1999). *Tietokannat käytännön työssä: tietokantojen peruskirja*. Jyväskylä: Teknolit 1999. ISBN: 952-5159-91-4.

Pressman, Roger S. & Bruce R. Maxim (2015). *Software Engineering: a practitioner's approach*. 8th edition. McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121. ISBN: 978-0-07-802212-8.

Rawsthorne, Peter (2011). *MVC in a three-tier architecture*. Essay by Critical technology [online document]. [5.3.2018] Available online: http://criticaltechnology.blogspot.fi/2011/09/mvc-in-three-tier-architecture.html.

Roff, Jason T. (2001). *ADO: ActiveX Data Objects*. O'Reilly & Associates, Inc. United States of America. First Edition. ISBN: 1-56592-415-0.

Rouse M., A. Hughes & C. Stedman (2017). *Microsoft SQL Server*. Microsoft Ignite 2017 conference coverage. Encyclopedia material by TechTarget [online document]. [7.10.2017] Available online: http://searchsqlserver.techtarget.com/definition/SQL-Server.

Rouse, M., S. J. Bigelow, K. Dodge, B. Lehto & M. Weiner (2017). *Internet Information Services (IIS)*. Encyclopedia material by TechTarget [online document]. [7.10.2017] Available online: http://searchwindowsserver.techtarget.com/definition/IIS.

Rouse, Margaret (2005). *VBScript*. Encyclopedia material by TechTarget [online document]. [17.10.2017] Available online: http://searchenterprisedesktop.techtarget.com/definition/VBScript.

Rouse, Margaret (2007a). *ASP.NET (ASP+).* Encyclopedia material by TechTarget [online document]. [6.10.2017] Available online: http://searchwindevelopment.techtarget.com/definition/ASPNET.

Rouse, Margaret (2007b). *Visual Basic .NET (VB.NET or VB .NET)*. Encyclopedia material by TechTarget [online document]. [17.10.2017] Available online: http://searchwindevelopment.techtarget.com/definition/Visual-Basic-NET.

Sauer, Daniel C. (2016). *ASP Classic Vs ASP.Net*. Guide by Microsoft Corporation [online document]. [6.10.2017] Available online: https://blogs.msdn.microsoft.com/codedevelopment/2016/07/12/425/.

Schach, Stephen R. & Amir Tomer (2000). *Development/Maintenance/Reuse: Software Evolution in Product Lines.* In: Donohoe P. (eds) Software Product Lines. The Springer International Series in Engineering and Computer Science, vol 576. Springer, Boston, MA. Experiences and Research Directions, Proceedings of the First International Conference, SPLC 1, Denver, Colorado, USA, August 28-31, 2000. ISBN: 978-1-4613-6949-3.

Schach, Stephen R. (1999). *Classical and Object-Oriented Software Engineering with UML and C++*. 4th Edition. McGraw-Hill, New York, U.S.A., 1998. ISBN: 0–07–290168–3.

Sommerville, Ian (2011). *Software Engineering*. 9th edition. United States of America: Pearson. 773 p. ISBN: 978-0-13-705346-9.

Spence, Ian & Kurt Bittner (2005). *What is iterative development?* Essay by IBM developerWorks [online document]. [13.10.2017] Available online: https://www.ibm.com/developerworks/rational/library/may05/bittner-spence/.

Wikipedia (2017a). *Active Server Pages* [online document]. [6.10.2017] Available online: https://en.wikipedia.org/wiki/Active_Server_Pages. Encyclopedia material.

Wikipedia (2017b). *ASP.NET Core* [online document]. [17.10.2017] Available online: https://en.wikipedia.org/wiki/ASP.NET_Core. Encyclopedia material.

Williams, Hugh E. & David Lane (2002). *Web database applications with PHP and MySQL*. O'Reilly 2002. First edition. United States of America. ISBN: 0-596-00041-3.

W3Schools (2018a). *ASP Syntax*. ASP Classic. The world’s largest web developer site [online document]. [5.3.2018] Available online: https://www.w3schools.com/asp/asp_syntax.asp. Tutorial.

W3Schools (2018b). *HTML5 Tutorial.* The world’s largest web developer site [online document]. [5.3.2018] Available online: https://www.w3schools.com/html/. Tutorial.

W3Schools (2018c). *HTML Forms.* The world’s largest web developer site [online document]. [5.3.2018] Available online: https://www.w3schools.com/html/html_forms.asp. Tutorial.

W3Schools (2018d). *HTML <form> method Attribute*. HTML Tags. The world’s largest web developer site [online document]. [5.3.2018] Available online: https://www.w3schools.com/tags/att_form_method.asp. Tutorial.

W3Schools (2018e). *HTTP Methods: GET vs. POST*. HTML Reference. The world’s largest web developer site [online document]. [5.3.2018] Available online: https://www.w3schools.com/tags/ref_httpmethods.asp. Tutorial.

APPENDIXES

APPENDIX 1: Functional requirements.

REQUIREMENT	PRIORITY
1. Search engines Users shall be able to search engines easily based on their own preferred search criteria.	Must have
1.1. Users shall be able to search engines based on different search criteria and their combination.	Must have
1.1.1. Limit the search results based on engine groups.	Must have
1.1.2. Limit the search results based on product.	Could have
1.1.3. Limit the search results based on engine name (i.e. designation).	Could have
1.1.4. Limit the search results based on number of cylinders.	Must have
1.2.5. Limit the search results based on L-engine / V-engine.	Could have
1.1.6. Limit the search results based on general features.	Must have
1.1.7. Limit the search results based on technology.	Must have
1.1.8. Limit the search results based on application type.	Must have
1.1.9. Limit the search results based on fuel type.	Must have
1.1.10. Limit the search results based on Diesel / GAS.	Could have
1.1.11. Limit the search results based on output range.	Must have
1.1.12. Limit the search results based on cylinder output.	Could have
1.1.13. Limit the search results based on engine speed.	Must have
1.1.14. Limit the search results based on engine speed mode.	Must have
1.1.15. Limit the search results based on emission optimization.	Must have
1.1.16. Limit the search results based on SCR system (with/without).	Must have
1.1.17. Limit the search results based on release status.	Must have
1.1.18. Limit the search results based on design stage.	Must have
1.1.19. Limit the search results based on Marine / PowerPlant engine.	Should have
1.1.20. Administrator shall limit the search results based on database ID.	Must have
1.1.21. Administrator shall limit the search results based on database field.	Must have
1.2. Users shall be able to determine which of the available search criteria is visible in the search window, and save the criteria as a personal default.	Should have
1.3. No obligatory fields in search.	Should have
1.4. Users shall be able to see what search options are under the different search criteria.	Must have

1.5. The search shall be dynamic i.e. only the available search options are visible under the search criteria, and the already chosen search criteria shall limit the other available search criteria.	Must have
1.6. Users shall be able to choose multiple different search options under the same search criterion (e.g. search for engines with engine group W31 or W32, in one search).	Must have

2. Search results: engine list Users shall be able to browse the results from the search, and still continue to limit the results based on their needs in order to find what they were looking for.	Must have
2.1. The engines matching the used search criteria (i.e. results) shall be displayed after the search is complete.	Must have
2.2. The results shall be displayed in a structure (e.g. table) that provides some information (e.g. in columns) about the engines.	Must have
2.3. Users shall be able to change the default information available from the engines, and determine which information is visible.	Should have
2.3.1. Users shall be able to save the selected information (e.g. columns).	Must have
2.3.2. Users shall be able to save multiple different information alternatives (e.g. filters).	Should have
2.4. Users shall be able to filter the engine list based on the available information.	Should have
2.5. Users shall be able to sort the engine list based on the available information.	Should have
2.6. Users shall be able to search the available information for a specific value.	Could have
2.7. Users shall be able to determine how many results for the search are displayed on one page.	Could have
2.8. There shall be information about how many engines are matching the search criteria i.e. how many engines are in the engine list.	Must have
2.9. The engine list shall include easy access to the different reports available for the engines (e.g. view button).	Must have

3. General data list report Generate a ready-made ***data report*** showing all the necessary technical data for an engine fast and easy, so that this can even be done in front of a customer. Also enable customization of the report.	Must have
3.1. Generate a report with all the relevant technical data available for a specific engine.	Must have
3.2. The report shall include *notes* that are related to the specific data contents being displayed.	Must have
3.3. Users shall be able to select for which loads the report contains data: main steps, 5 % steps (e.g. check box).	Should have
3.4. Users shall be able to select whether the report contains data with or without the effect of pumps, or displays both values (e.g. check box).	Should have

3.5. Users shall be able to select for which available fuel types the report contains data: HFO, LFO, GAS (e.g. check box).	Should have
3.6. Users shall be able to select for which available fuel consumption tolerances the report contains data (e.g. check box).	Should have
3.7. Users shall be able to select for which ambient conditions the report contains data: standard (ISO), arctic, tropic (e.g. check box).	Want to have
3.8. The administrator shall be able to view the report with database Id information: both main and sub Ids.	Should have
3.9. The administrator shall be able to view the report with database field information.	Could have
3.10. Users shall be able to change to view information from another available design stage for the engine.	Could have

4. Fuel consumption graph report Generate a ready-made ***graph report*** visualizing an engine's fuel consumption. And enable users also to insert fuel consumption values to the graphs manually to enable quick comparison.	Must have
4.1. Generate graphs with fuel consumption values based on different loads, for a specific engine.	Must have
4.1.1. Display SFOC (diesel: g/kWh) and BSEC (gas: kJ/kWh) values in different systems of coordinates.	Must have
4.1.2. Display SFOC graphs for both HFO and LFO diesel fuels.	Must have
4.1.3. Display different diesel fuels (HFO, LFO) in a same system of coordinates at the same time.	Must have
4.1.4. Display graphs for fuel consumption values with different tolerances.	Must have
4.1.5. Display the wanted fuel consumption values with different tolerances, in a same system of coordinates at the same time (e.g. check box).	Could have
4.1.6. Enable users to easily see the exact values in the different parts of the graph. (e.g. when user puts mouse on top of the graph the detailed value on that spot pops up.)	Could have
4.2. Enable manual input for fuel consumption values to the same systems of coordinates with the already existing values.	Should have
4.2.1. The manual input shall be implemented so that the graph is drawn based on the given fuel consumption values for different loads, but it shall not be predefined on what loads the values shall be given.	Must have
4.2.2. Users shall be able to add multiple graphs to the same system of coordinates.	Should have
4.2.3. Users shall be able to save the inserted graphs for later use.	Should have

5. Fuel consumption details report Generate a ready-made ***data report*** that enables users to quickly view fuel consumption values with different tolerances, for an engine.	Must have
5.1. Generate a report that includes fuel consumption values with different tolerances, displayed side by side, for a specific engine.	Must have
5.2. Users shall be able to select for which available fuel types the report contains data: HFO, LFO, GAS (e.g. check box).	Should have
5.3. Users shall be able to select for which loads the report contains data: main steps, 5 % steps. (e.g. check box).	Should have
5.4. Users shall be able to select for which available fuel consumption tolerances the report contains data (e.g. check box).	Should have
5.5. The administrator shall be able to view the report with database id information: both main and sub ids.	Should have
5.6. The administrator shall be able to view the report with database field information.	Could have

6. Engine comparison Enable users to compare different engines, based on the information available in the generated reports.	Must have
6.1. Users shall be able to add multiple engines to the comparison.	Must have
6.2. Users shall be able to add an engine to the comparison directly from the engine list as well as when browsing the available reports for the engine.	Must have
6.3. The engines shall only be deleted from the engine comparison, when the user chooses to delete them.	Must have
6.4. There shall be option to delete a specific engine from the engine comparison or delete all at the same time.	Must have
6.5. Users shall easily see which engines are currently in the comparison.	Must have
6.6. Users shall be able to access the engine comparison anywhere from the application: not depending on what they are currently browsing.	Must have
6.7. Users shall be able to select which of the engines added to the comparison are currently being displayed in the reports (e.g. check box).	Should have
6.8. If there is some data that is not available for all the engines that are in the comparison, the data still needs to be displayed for the engines that it is available for.	Must have
6.9. The engine comparison shall include the same content as the reports that are as well generated for one engine, but so that the reports contain the data from all the engines that are in the comparison.	Must have
6.10. In the ***data reports*** , the data available from the compared engines shall be displayed side by side, and it shall be visible where the differences are.	Must have
6.10.1. Display a summary row when there are two engines in the comparison.	Should have

6.10.2. Display a range row when there are more than two engines in the comparison.	Should have
6.10.3. When there are differences between the compared engine values, display which is the biggest (e.g. in red) and which is the smallest (e.g. in green) value.	Should have
6.10.4 Users shall be able change the order in which the engines are displayed in.	Should have
6.11. In the ***graph report*** , the compared engines shall be displayed in the same systems of coordinates.	Must have

7. Revision history Enable users to backtrack the happened changes between different data revisions, also including the reasoning behind the changes as well as the information about who has requested the change.	Must have
7.1. The application shall display which data revisions are at the moment being displayed. (By default, the latest data revision is shown.)	Must have
7.2. Together with the information about which revisions are currently displayed, there shall be a link to the CN (Change Notice) message for more detailed information on that specific engine's data revision changes.	Must have
7.3. If there is some data that is not available for all the revisions that are displayed, the data still needs to be displayed for the revisions that it is available for.	Must have
7.4. In the ***data reports*** , users shall be able to fetch different data revisions of the data, and display them side by side, and it shall be visible where the differences are.	Must have
7.4.1. Display a summary row when two different revisions are displayed.	Should have
7.4.2. Display a range row when there are more than two revisions are displayed.	Should have
7.4.3. When there are differences between the revisions, display which is the biggest (e.g. in red) and which is the smallest (e.g. in green) value.	Should have
7.4.4. Users shall be able to change the order in which the revisions are being displayed in.	Should have
7.5. In the ***graph report*** , users shall be able to fetch different data revision of the data to the same systems of coordinates.	Could have
7.6. In the ***engine comparison*** , users shall be able to fetch the different data revisions for the engines that are being compared.	Could have

8. Export, share & print out content User shall be able to export, share and print out any relevant content produced by the application.	Must have
8.1. The application shall enable exporting of the content to Excel, Word and PDF.	Must have
8.2. The application shall enable sharing the content with others on Outlook.	Must have
8.3. The application shall enable printing out the content.	Must have
8.4. Users shall be able to print, share and export all the reports generated by the application (also including the reports with engine comparison as well as with different revision information).	Must have
8.5. Users shall be able to determine which of the reports or which parts of the reports they want to include to the printed, shared and exported documents.	Should have
8.6. Users shall be able to add additional info to the reports, as optional information.	Must have
8.6.1. There shall be option to determine an electronic stamp, project name, project number and date (by default the current date).	Must have
8.6.2. There shall be a free text field.	Should have
8.6.3. It shall be clearly stated in the option to add additional info that it is optional and does not actually connect the data to any project.	Must have
8.7. Users shall be able to perform these activities (print, share, export) to the engine list (i.e. search results).	Should have
8.8. Every document that is shared, exported or printed out from the application, shall automatically include information about what this data is.	Must have

9. Data management for administrator The data content in the application, as well as the access rights for that content shall be managed through the application by the administrator.	Must have
9.1. Admin shall be able to determine the user profiles that have access to the different reports generated by the application.	Should have
9.2. Admin shall be able to determine the user profiles that have access to the certain data fields that are visible in the ***data reports*** .	Must have
9.3. Admin shall be able to determine which data is visible in the ***data reports*** .	Must have
9.3.1. Admin shall be able to create new (+ delete) data sections to reports.	Must have
9.3.2. Admin shall be able to add (+ delete) data fields under different data sections.	Must have
9.3.3. Adding of the data fields shall happen so that the admin can first select the system and/or sub-system where the preferred data field is located, and select the correct data field from there.	Should have
9.3.4. Admin shall be able to determine the order of the different data sections.	Should have

9.3.5. Admin shall be able to determine the order of the data fields inside the data sections.	Should have
9.3.6. Admin shall be able to name the different data sections.	Must have
9.3.7. The names of the data fields, as well as the units, shall be retrieved directly from the databases. If they do not exist admin shall define them.	Must have
9.3.8. Admin shall be able to delete data sections and data fields.	Must have
9.4. Admin shall be able to preview what the general data list report looks like in the eyes of the different user profiles, before saving the changes.	Could have
9.5. The made changes shall be visible to the users only after the admin determines so, and makes the changes visible to the users.	Should have
9.6. Admin shall be able to determine which search criteria is available in the data search.	Should have
9.7. Admin shall be able to determine to which user profiles the different search criteria is available for.	Should have
9.8. Admin shall be able to create completely new reports to the application.	Want to have
9.9. Admin shall be able to determine for which engines the different user profiles have access to. This shall happen based on the release status information of the engine.	Should have

10. Statistics for administrator The application shall generate different statistic for the administrator, related to the usage of the system.	Should have
10.1. Admin shall be able to view the authentication and authorization log.	Should have
10.2. The application shall generate reports about the user group and individual user activity, for the admin.	Should have
10.3. Admin shall be able to view all the alerts and the related information about the application interruptions and failures.	Should have

11. Help & contact information The application shall contain basic information about who to contact in case of problems or to give feedback.	Must have

12. Information exchange with other internal systems The application shall exchange information with other company internal systems in order to access data, and to provide content to other systems.	Must have
12.1. The application shall not be used in modifying any of the data in the data sources: only intended as a read-only display.	Must have
12.2. QMS to access the application content: the produced reports.	Must have
12.3. QDMS to access the application content: the produced reports.	Want to have
12.4. QDMS DB to be used as one source of data.	Want to have
12.5. Performance DB shall be used as the main source of data.	Must have
12.6. Competitor DB(s) shall be used as source of data in the future.	Want to have
12.7. Company Active Directory DB to be used as user data identification source to manage authentication and authorization.	Must have

APPENDIX 2. Non-functional requirements.

CLASSIFICATION	REQUIREMENT	PRIORITY
1. Maintainability	***1.1. Supported technology*** The application shall be implemented using a technology framework that is fully supported and updated.	Must have
	1.2. Architectural standards Accepted standards and design patterns shall be used in the application architecture.	Shoud have
	1.3. Coding standards The code shall be built modularly, allowing separate system elements to change independently.	Shoud have
	1.4. Proper documentation Proper documentation shall support the maintainability.	Shoud have
2. Performance	***2.1. Response time*** The server shall respond to a simple client request in less than 1 sec x 90 % of the time.	Should have
	2.2. Processing time Interaction with the server that require more processing, such as login, shall take less than 5 sec x 90 % of the time.	Should have
	2.3. Querying time (read only) Querying the database(s) shall take up to 2 sec x 90 % of the time.	Should have
	2.4. Total number of users The system shall handle 200 users altogether.	Should have
3. Scalability & Capacity	***3.1. Number of concurrent users*** The system shall handle 50 concurrent users during peak hours.	Shoud have
4. Availability	***4.1. Hours of operation*** The system should always be available during the typical working hours, and any maintenance where the system is offline shall be done outside of these times.	Shoud have
	4.2. Geographic operation The geographic location shall not restrict the availability of the system.	Shoud have
5. Recovery	***5.1. Mean time to recovery*** Time available to get the application back online after failure: 2 days.	Should have
	5.2. Backups The application shall be responsible to taking backups that may be restored to working state.	Should have

6. Security	***6.1. Authentication*** Users must be authenticated to access the system, which must be done against the organization's Active Directory (AD) database.	Must have
	6.2. Authorization Access to all data shall be limited to authorized users, which must be done against the organization's Active Directory (AD) database.	Must have
	6.3. Wärtsilä network Access only over the company intranet and extranet.	Must have
	6.4. Organizational compliance Compliance with the organization's security standards and policies.	Must have
	6.5. Layered architecture A layered structure for the system architecture shall be used with the most critical assets protected in the innermost layers.	Must have
7. Manageability	***7.1. Logging & tracking*** All authentication and authorization events shall be logged.	Should have
	7.2. Alerts Alerts required when the system suffers from interruption or failure.	Could have
8. Interoperability	***8.1. Compatibility with other systems*** Ability to communicate with the other internal and external applications and systems that the application needs to interact with in order to exhange information.	Should have
9. Usability	***9.1. Ease of use*** The typical system end-user shall be able to use the application without any training, max 30 min training session for special needs (e.g. administrator).	Should have
	9.2. GUI standards Best practices and principles for GUI design shall be used.	Should have
	9.3. Browser support The application shall be accessed over the web, and support multiple different browsers.	Should have
	9.4. Delivery mediums The application shall support the following delivery mediums: PCs, mobile phones.	Should have
	9.5. SSO (Single sign-on) SSO shall be used, so that the users only need to log in once at their workstation to access the application.	Should have