UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

AUTOMATION AND INFORMATION TECHNOLOGY

Jimi Kallio

UPDATING A POWER PLANT SIMULATION SOFTWARE TO A NEW PLATFORM

Master's thesis in Technology for the degree of Master of Science in Technology

Vaasa 23.04.2020

Instructor Teemu Mäenpää, Kristian Nyqvist

CONTENT REGISTER

SYMBOLS AND ABBREVIATIONS

ABC	Architect Business Cycle
APEX	Internal tool for power plant pricing
API	Application Programming Interface
Aux load	Auxiliary consumer load
CHP	Combined Heat and Power
DSRM	Design Science Research Method
ELM	Extreme Learning Machine
GIE	Gas Infrastructure Europe
Hz	Hertz (1/s)
GPRS	Gas Pressure Reduction Station
GW	Gigawatt
IDE	Integrated Development Environment
kV	kilo Volt
MN	Methane Number
NOx	Nitrogen oxides (NO and NO2)
PoC	Proof of Concept
ppm	Parts Per Million
SCR	Selective Catalytic Reactor
SG	Spark ignited Gas
temp	Temperature
UI	User Interface
UML	Unified Modeling Language
VBA	Visual Basic for Applications

UNIVERSITY OF VAASA
Faculty of technology
Author: Jimi Kallio
Topic of the thesis: Updating a Power Plant Simulation Software to a new Platform
Supervisor: Teemu Mäenpää
Instructor: Kristian Nyqvist, Wärtsilä
Degree program: Master of Science in Technology
Option: Automation and Information Technology
Year of entering the university: 2012
Year of completing the thesis: 2020 **Pages:** 78

ABSTRACT

This thesis is a two-part project in which the first part is mapping the current state of a power plant simulation program and the second part is a proof of concept for new platform and architecture for the program. The basics of software architecture and power plant simulation is established. The first goal for this thesis is to get a clearer understanding and documentation for the current state of the program. The second goal is to make an architectural plan of the next version of the program.

Software architecture is the plan on which a program is built upon. Software architecture helps developers understand the structure of the program. This is very important considering the possibility of multiple developers and future updates. A software without a good architectural plan is not a good program. Power plant simulation programs are no different. Power plant simulation programs usually has large amounts of code behind them, and a good architectural structure is key to successfully understand, operate, and develop the program.

The result of this thesis is a plan for the next version of the program. The next version follows a new architectural structure. The new architecture is considered as a proof of concept for the eventual final new version and platform of the program. The new architecture has some basic functionalities from the old program architecture as well as some new features. New user feedback and suggestions for improvement are considered for future versions and prototypes for the program. There are three feasible options on how to continue the development of the simulation program. First option is to remain with current platform and develop the program as before. The second option is to remain with the current platform but updating the architecture according to the new architectural model. The final option is to redevelop the program using a new platform and developing the architecture as planned using existing features and functions.

KEYWORDS: Software architecture, Power plant, Simulation, Prototyping, Updating, User experience.

VAASAN YLIOPISTO
Tekninen tiedekunta
Tekijä: Jimi Kallio
Työn otsikko: Voimalaitos simulointiohjelmiston päivittäminen uudelle alustalle
Työn valvoja: Teemu Mäenpää
Työn ohjaaja: Kristian Nyqvist, Wärtsilä
Tutkinto: Tekniikan maisteri
Oppiane: Automaatio ja tietotekniikka
Opintojen aloitusvuosi: 2012
Työn valmistumisvuosi: 2020 **Sivumäärä:** 78

TIIVISTELMÄ

Tämä tutkielma on kaksiosainen projekti, josta ensimmäinen osa on voimalaitoksen simulointiohjelman nykytilan kartoitus ja toinen osa on konseptin todistus ohjelmiston uutta alustaa varten. Ohjelmistoarkkitehtuurin ja voimalaitos simuloinnin perusteet ovat määritelty. Tutkielman tavoite on pääasiassa luoda selkeämpi käsitys ja dokumentointi simulointiohjelman nykytilasta. Toinen tavoite on luoda suunnitelma ohjelmiston seuraavasta arkkitehtuurista.

Ohjelmistoarkkitehtuuri on suunnitelma, johon ohjelma perustuu. Ohjelmistoarkkitehtuuri auttaa kehittäjiä ymmärtämään ohjelman rakennetta. Tämä on erittäin tärkeää, kun otetaan huomioon, että ohjelmistolla voi olla useita kehittäjiä ja tulevaisuudessa saatetaan tehdä päivityksiä ohjelmistoon. Ohjelmisto ilman hyvää arkkitehtuurista suunnitelmaa ei ole hyvä ohjelmisto. Voimalaitos simulointiohjelmat eivät eroa tästä. Voimalaitos simulointiohjelmissa on yleensä suuria määriä koodia niiden taustalla, ja hyvä arkkitehtuurirakenne on avainasemassa ohjelman onnistuneeseen ymmärtämiseen, käyttämiseen ja kehittämiseen.

Tämän tutkielman tuloksena on suunnitelma ohjelmiston seuraavaa versiota varten. Uusi arkkitehtuuri voidaan nähdä konseptin todistuksena lopullista uutta versiota varten. Arkkitehtuurissa on huomioitu muutama perusominaisuus vanhasta versiosta ja muutama uusi ominaisuus. Uusia käyttäjäkokemuksia ja parannusehdotuksia otetaan huomioon ohjelmiston tulevaisuuden versioita ja prototyyppejä varten. Löytyi kolme toteutettavaa optiota simulointiohjelmiston jatkoa varten. Ensimmäinen optio on jatkaa tämänhetkisellä alustalla ja jatkaa ohjelmiston kehittämistä kuten ennen. Toinen optio on jatkaa tämänhetkisellä alustalla mutta päivittää ohjelmisto arkkitehtuuri uuteen arkkitehtuuri malliin. Viimeinen optio on rakentaa ohjelmisto uusiksi uudelle alustalle ja päivittää arkkitehtuuri suunnitelman mukaisesti käyttäen jo olemassa olevia toimintoja ja funktioita.

AVAINSANAT: Ohjelmistoarkkitehtuuri, voimalaitos, simulointi, prototyypitys, päivittäminen, käyttäjäkokemus.

1 INTRODUCTION

This thesis is a part of a larger project of creating a new program to replace an old Microsoft Excel based program that simulates power plants. The thesis is done for Wärtsilä by commission and under a thesis agreement. The instructor for this thesis is Kristian Nyqvist.

When planning for new power plants, Wärtsilä calculates as many scenarios and values as possible based on the customer needs and variables. There can be hundreds of different options to calculate before finding a suitable option that satisfies the customer budget and needs. Wärtsilä has developed a tool, PerfPro, which takes user input of variables and calculates almost all aspects from emissions to heat recovery of a power plant.

This tool, PerfPro, was developed first in the late 80's by Kim Jansson and has gone through multiple generations of versions to include new technologies and features. The program architecture is pretty messy and has extremely much code in it. This makes the program slow and outdated. Also, documentation is limited and outdated. More about the need for the project in subchapter 1.1.

This thesis is divided into three parts, first there will be theoretic part with software architecture, existing software, and research methods. The second part is for the documentation on the current version of PerfPro. And finally, there will be the part for speculation, suggestions and architectural vision for the next generation of PerfPro.

1.1 Need for project

The old program is a Microsoft Excel based program with a large amount of VBA code and other files that the program communicates with. The other files are mainly in C++ and C#. The first version of the program is from the late 1980's made by Kim Jansson. Of the versions still available, the oldest one is from the year 1999. There has been plenty of updates and versions of the program released every year since. The latest release is

from the beginning of 2019. The program's VBA part basically has no documentation, the binaries have some documentation, but is out of date. There is an undocumented software architecture plan that can be seen when reading through the mostly uncommented code. (Nyqvist 2017)

First part of this project is to map the current state and the flow of control and data in the program. When the current state is documented, a new platform is discussed. The choice of the platform is based on interviews with the users and developer's preferences and knowledge. When the platform is decided, a new architectural vision and plan for implementation is made.

The current state of the program in Excel format is outdated. Additionally, the lack of documentation and size of the program makes it difficult for new users and new developers to understand and use the program. The documenting and mapping of the current state is a priority of this project. The architectural vision for the next generation will be more like a proof of concept.

In addition to the industry needs for this project, there is a more scientific aspect. The industry needs focus more on the specific case of PerfPro, while the scientific point of view focuses on the research methods and existing technologies, as well as choices of platform in specific cases. For example, the choice of programming language, technology stack and the reasoning for the choice can be considered as a scientific aspect.

There are numerous examples of good architectural reconstruction leading to better performance and maintenance for software. One example of this is the academic study "A software maintenance process architecture" by S. M. Brown, N. Wild and J. D. Carlin. In this study the authors examined architectures of software with maintenance as a primary focus point. More streamlined architectures generally had a more positive impact on software maintenance. (Brown, Wild & Carlin 1996)

1.2 Goals

The first goal for this thesis is to map out the current state of the program, behavior and features are described in more detail. By user interviews we get the information what the program is used for, and if there are features that are not in use. Also, from these interviews we will find out if the users want or need new features for the program.

The second goal of this thesis is to make a plan for next generation of the program. This includes making an architectural vision of the new program and making documentation for the plan and a roadmap. Finally, there will be suggestions about the future of the simulation program.

1.2.1 Software mapping

Software mapping is a part of the software architecture and is a visualization of the program construct. There are many different standards and methods for software mapping. One popular mapping format is unified modeling language, or UML. UML is a good way of visualizing an object-oriented program. With a UML it is easy for anyone to get a basic understanding of the program and can more easily understand the concept and use of the program. (IEEE 1471-2000)

1.2.2 Proof of concept

Proof of concept, PoC in short, is a demonstration that a concept, theory or technology has a feasible and potential real-world solution. The goal is to prove practical potential. A PoC in software engineering can be a prototype of a larger project to show that the project is worth investing in. A PoC is usually made for just that reason, to save money and time, and to not invest too much in a project that is not a good solution. More about software prototyping in next sub-chapter (Techopedia Inc. 2019)

1.2.3 Software prototype

A software prototype is similar to any prototype in any other industry. It is a chance for the manufacturer to see the idea of a product way before it is complete. This gives the chance for change in an early stage of the development. A prototype gives the opportunity to evaluate the product to ensure everyone is on the same page and the program is doing what is intended. (Stanley 2019)

For software prototyping there are usually four steps. These steps are the following:

- Identify initial requirements.
- Develop initial prototype
- Review
- Revise

The first step is all about making sure that the developer has an idea of what the software is supposed to do. The publisher, or stakeholders, identifies the user group and makes sure that all the user requirements are known to the developer. The second step is for the developer to create a prototype that meets the requirements from step one. The prototype does not have to be a complete product. Step three happens when a prototype is presented to the stakeholders. This is a chance for the developer and publisher to discuss changes at an early stage of the software development. In later stages, and with more complete prototypes, the end user has a chance to test the product. This is called beta testing. The final step is to make revision to the prototype. These revisions are based on feedback from the publisher and end users. (Stanley 2019)

1.3 Research question

The primary research question for this thesis will be: does the program need an update to a new platform, if so, to which platform? There are many options for new platform for the program. The employees that develop and administrate the current software are skilled programmers and can basically work with any platform. It is more of a question about which platforms are feasible for the new features that are planned for the next generation of the program.

1.4 Structure and layout

The thesis is divided into seven chapters. The first and last chapters are the standard introduction and conclusions chapters. In the introduction chapter there is mentioned the need for this project and the research question, as well as the goals and objectives. The conclusions chapter is not a summary of the thesis, but rather an open discussion of the results and findings related to this thesis.

The second, third and fourth chapters are more of an academic approach to the subject. First there is a chapter called software architecture. In this chapter we will discuss the concept of software architectures, as well as design patterns and software mapping. The third chapter is called power plant simulation software. In this chapter there will be general discussion about power plant simulation and which kind of tools and knowledge we need to accurately simulate power plants. The fourth chapter is for research methods. We will go through the approach to research in academic works and the research environment. Also, data gathering as part of this thesis is established.

The fifth chapter will look at the current situation of the power plant simulation program. This program and the amount of code is rather large, about 400 000 lines of code in total, and basically has no documentation on it. User experience will be the main source for mapping and documenting the current state of the program.

In the Sixth chapter there is the plan for the next generation of the program. Its architectural vision and speculation on new platforms. Discussions with the developers will lead to decisions for the updated program. New software architecture plans can be found here.

2 SOFTWARE ARCHITECTURE

This chapter will go through the basics of software architecture. First there will be general concepts and methods in software architecture. After that there will be more specific aspects of software architecture such as visualization, UML, module hierarchy and design patterns.

2.1 General

Software architecture, like any other architecture is from the mind of the architect and is similar to design. Architectural vison and design have a lot in common. Software architecture usually reflects on the company or organization which has developed the software. A good architecture is the basis of a functioning program and is a very important part of the development life cycle. (Bass, Clements & Kazman 1998: 21-25)

A basic program is constructed based on one thing, requirements. When the programmer knows the requirements, he or she can construct a working program based on the requirements. An example of a simple calculating program and the requirements can be to input two numbers and what to do with them, add, subtract, multiply or divide, and getting the result as an output. In this example there are many different approaches to making a functioning program that fulfills the requirements. These different approaches can be taught of as the architectural visions of the developer. All in all, the solution and implementation determine the architecture of the program. (Bass et al. 1998: 3-12)

In larger projects the possibility of different implementations and architectures is increased exponentially. If we give two programmers the exact same requirements, it is almost certain that the two programs differ architecturally from each other. This is even more probable if the number of requirements is larger. This proves that requirements and architecture is not the same thing. (Bass et al. 1998: 3-12)

Nowadays software development is very much business driven projects. Architect business cycles, or ABC, is the term used when developing and implementing software for a company or organization. The list of requirements and architectural visons come from many different sources in an ABC. In organizations many different stakeholders have different views that the architect must follow. If these stakeholders have differing views about the final product, it can cause big problems for the developer. Examples of stakeholders can be: management, marketing, end users, maintenance, and customers. (Bass et al. 1998: 12-17)

One of the most important stakeholders is the maintenance stakeholder. If the developer is not the same person that maintains the program, it can be extremely hard for the maintainer to know what is going on without a good architecture. (Koskimies & Mikkonen 2005: 20-22)

In short software architecture can be summarized with the following quote from the book: Software Architecture in Practice, by Len Bass (Bass et al. 1998: 6), "*The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.*"

In Philippe B. Kruchten's article: "*The 4+1 View Model of Architecture*" Kruchten describes the 4+1 architectural visualization and documentation of software architecture. The 4 main views are: Logical-, Process-, Development- and Physical Views. The final (+1) view is a use case view of the software. (Kruchten 1995: 43)

The logical view focuses on functional requirements, what the system should provide to the users. The designer divides the components into key abstractions. The abstractions usually are represented by objects or object classes, their variables and relationship between each other. A class diagram or UML diagram is a good visualization of the logical view. More about UML in sub-chapter 2.3. (Kruchten 1995: 43-44)

For the process view, we concentrate on the non-functional requirements. These requirements can be for example: performance and system availability. Process view also specifies which threads of control executes which operation of each class from the logical view. The process view at the highest- and simplest level can be a set of individual logical networks of communicating processes distributed across hardware resources. (Kruchten 1995: 44-45)

With development view we will look at dividing the program into software modules. These modules are smaller program libraries or subsystems. These subsystems are organized into a hierarchy of layers, each with a well-defined interface to the other layers. In this part we will determine the internal requirements, such as software management and programming languages. (Kruchten 1995: 45-46)

The physical view is for non-functional requirements. System availability, reliability, scalability, and performance are just some of these non-functional requirements. Components within the system should be mapped in the logical-, process and development view. These components are now represented as various nodes in the physical view. Mapping of these nodes should be very flexible, and this view should have minimal impact on the source code. (Kruchten 1995: 46-47)

2.2 Visualizing the architecture

Visualizing the architecture of a program is a good way to explain the architecture of a program to the stakeholders. A flowchart is always a good addition to the documentation of a program. A flowchart is a simple visualization of the proses that happens when running a program. An example of a flowchart for a calculator program is shown in figure 1 below.

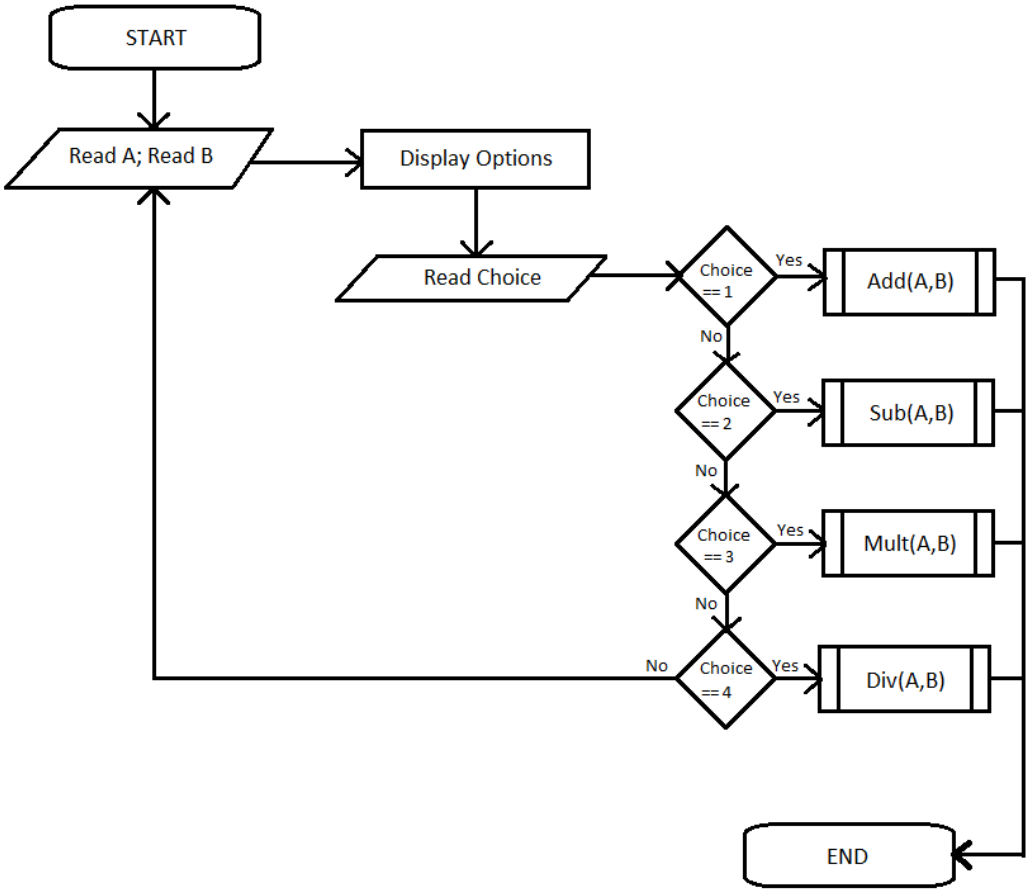


Figure 1. Flowchart of a calculator program.

A flowchart must always have a start and end tile. In general, flowcharts consist of inputs tiles, output tiles and choice tiles. Read tiles are demonstrated with the tilted rectangles and are implemented in the code with a read function. Straight rectangle tiles are representing things happening within the program. This can be printing variables or text for the user or making calculations with the variables among other things. The tilted squares are for choices for the program. This is usually implemented with if-statements. The flowchart chooses the route to follow based on if the statement inside the squares is true or false. Rectangles with double-lined short sides are representing sub-functions within the program. These can be opened up in their own flowcharts. It is possible for the flowcharts path to come back to a previous tile, if this is the case, the code usually has a loop to implement this option. (Lucidchart 2019)

In the example flowchart in figure 1, we have an implementation of a calculator program. After the start tile, we get to the input tile, in which we get the two numbers, A and B, that we want to operate with. After getting the values, we print the list of choices to the user. Then we get the choice from the user and test if the choice is 1, 2, 3, 4 or something else. If the choice is something between one and four, we do the corresponding function and end the program. If the choice is something else, we go back to the tile where we ask for the numbers. This means that our program will need a loop that end when the user has chosen a valid operation for the variables.

2.3 UML

Unified modeling language, or UML, is another good way to visualize the architecture of a program. UML is used mainly in object-oriented programming. Object-oriented programming consists of objects that behave in a certain way within the program. An UML maps out the objects and the relationship between the objects. In figure 2 there is an example of a UML diagram.

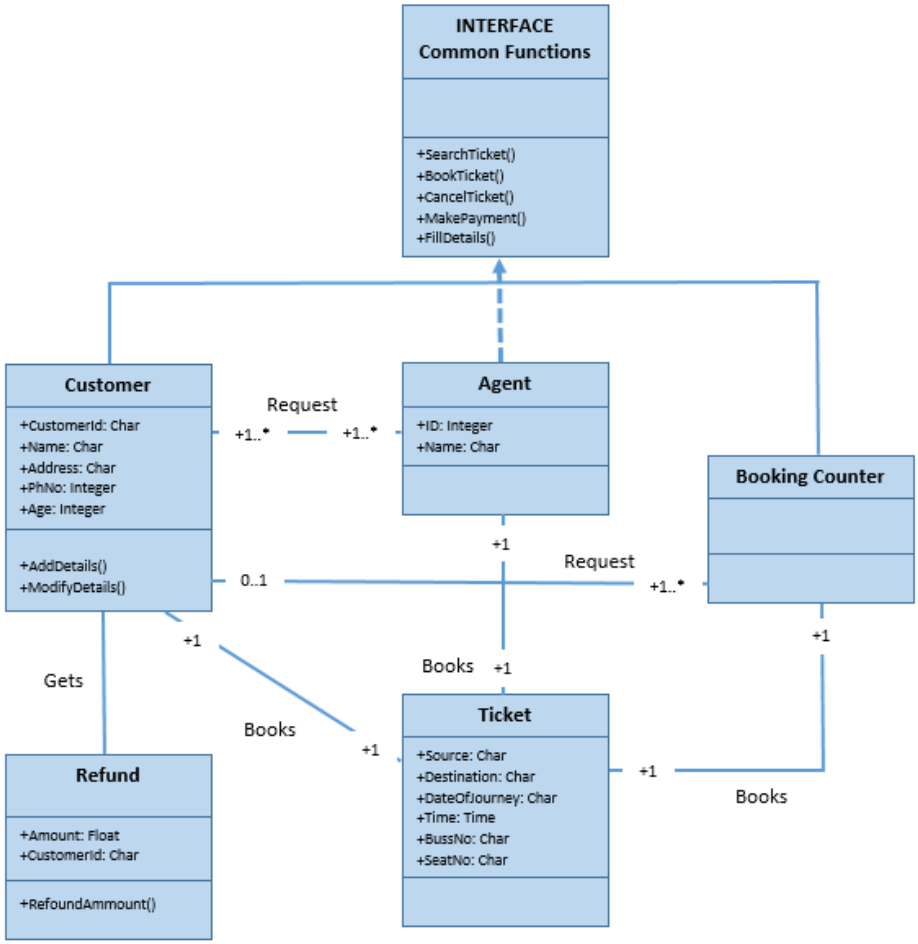


Figure 2. UML diagram of a ticket booking service program. (Tutorialspoint 2019)

In this example we have a UML diagram of a ticket booking service. The program uses objects to classify customers, the ticket, refunds, among other objects. Every object has a set of variables and procedures. Also, the relationship between the objects is mentioned. (Tutorialspoint 2019)

2.4 Software model hierarchy

A software model hierarchy is an important part of software architecture in programs with multiple layers of files and functions within the whole project. Usually the top level is represented by the human-computer interaction interface, and the bottom layer is directly in touch with the databases. Layer hierarchy can range from a simple list to a pyramidic model with multiple layers simultaneously at a specific point.

A visualisation of these layered software architectures can be found in figure 3 below from Herberto Graca's article "*Layered Architecture*". (Graca 2017)

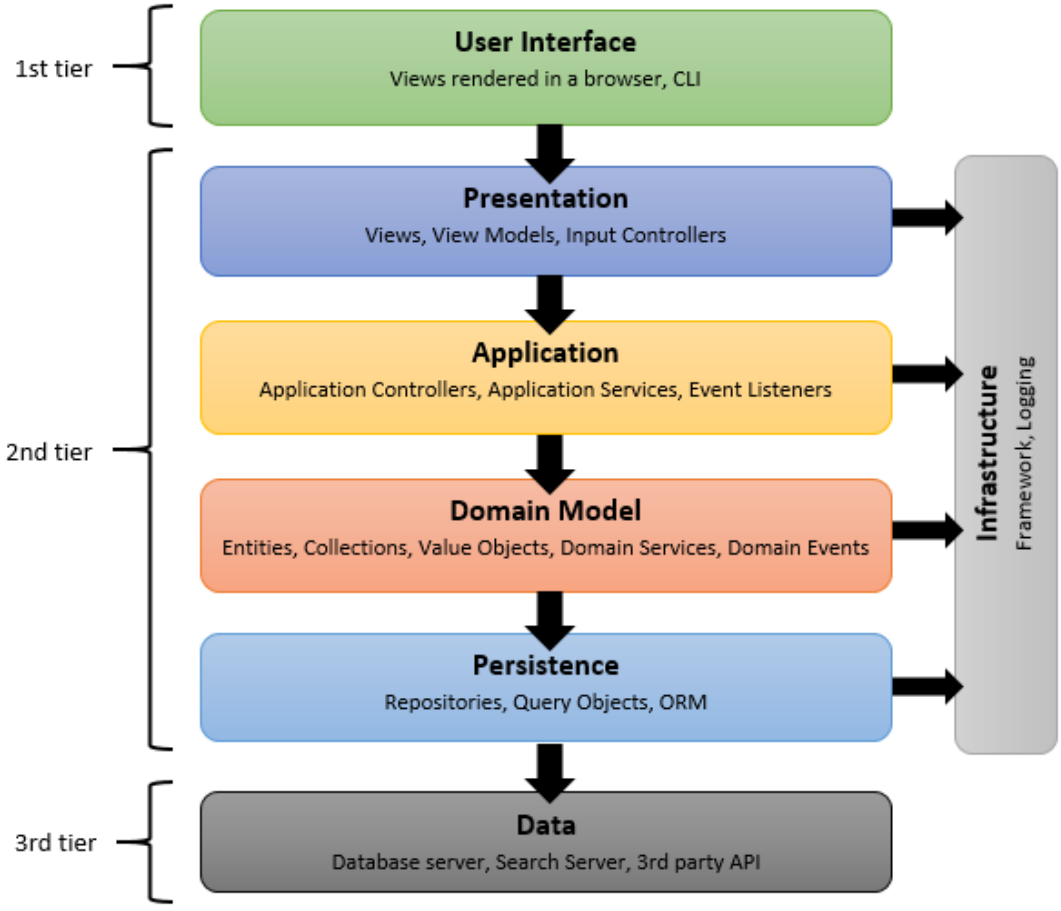


Figure 3. Layered architecture hierarchy. (Graca 2017)

The model in figure 3 is called an n-tier hierarchy model. In this particular example, there are 3 tiers. The first tier is in direct contact with the user and the third is with the database, servers and third-party API's. Everything in tier two is in contact with the infrastructure. (Graca 2017)

A more complex layer architecture includes relationships between different components. Also, these components can be parallel within a tier or layer. An example of a more complex hierarchy is visualized in figure 4 below.

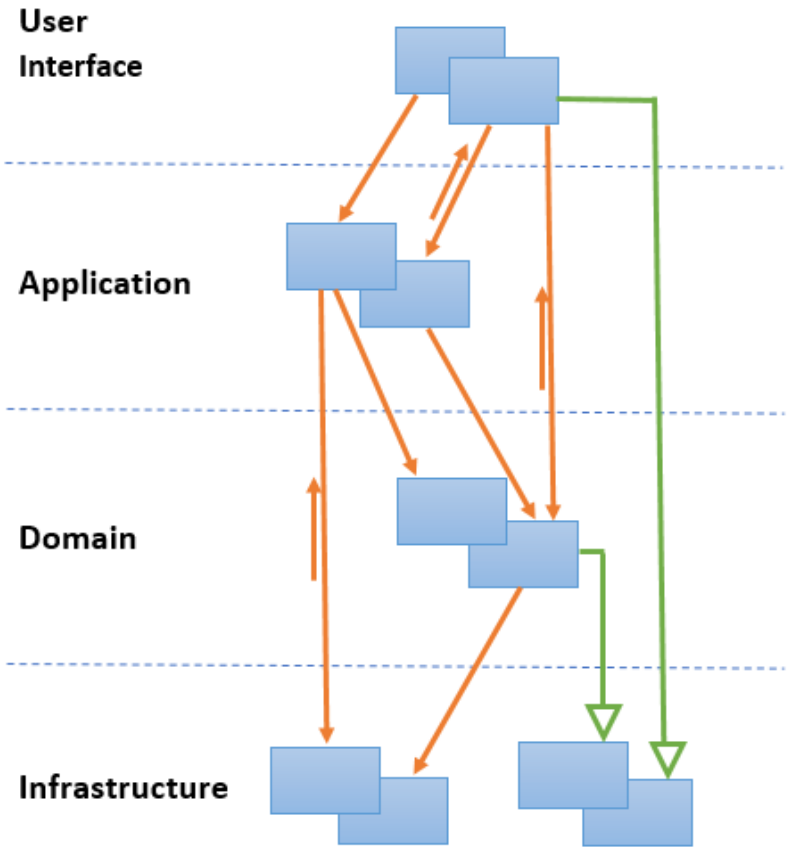


Figure 4. Layer hierarchy with relationships. (Evans 2003: 68)

This figure does not have named components, but instead visualize the different dependencies between the components. These components are however, divided into different layers: user interface, application, domain, and infrastructure. (Graca 2017)

2.5 Design patterns

Designing software with a pattern is hard, especially in object-oriented software. The design should be specific enough for the requirements of the software but also simple enough to solve future problems and requirements. A reusable and flexible design is almost impossible to perfect. Good programmers can trough experience create good designs that meet the requirements but also are highly flexible. (Gamma, Helm, Johnson & Vlissides 1995: 1-4)

Experienced programmers will not create unique code from scratch for every new problem they face. Instead they reuse some solutions that has worked for them or others in the past. By reusing previously working methods and functions, software becomes flexible, elegant, and reusable. This will result in software design patterns. (Gamma et al. 1995:1-4)

We can narrow down different design patterns into four essential elements. These elements are:

- 1. Pattern name.
- 2. The problem.
- 3. The solution.
- 4. The consequences.

The pattern name describes the design problem, solution, and consequences in a couple words. Naming patterns increase our vocabulary and helps us document design patterns. Naming patterns also helps us think about design patterns as abstract objects and visualizing trade-offs and such. The problem part describes when we apply the pattern. It explains our problem and the context it is in. Usually the problem includes a list of requirements and conditions that must be met to apply the pattern. The solution includes the

elements of the design, the relationships responsibilities, and collaborations. The solution is not a concreate design for any specific problem, but rather a template that can be applied in different situations. The consequences is the part of results and trade-offs when applying the pattern. Consequences are usually not a major part when describing design decisions, but they are critical when evaluating design alternatives for understanding upsides and downsides when applying a pattern. Examples of trade-offs can be space and time trade-offs. A larger, more complex program can calculate results faster but takes up more space than a slower, more simple solution. (Gamma et al. 1995:1-4)

3 POWER PLANT SIMULATION SOFTWARE

This chapter will be about power plant simulation software in general. This chapter will answer these questions: What is important when simulating power plants? Are there similar programs available? Furthermore, in the last subchapter we will find out about researches in software updating into new platforms.

3.1 Basics

What is a power plant simulation program? There are two kinds of simulation programs: static and dynamic. A static simulation program takes variables as inputs and gives outputs as results based on calculations with the inputs. Dynamic programs include time as a factor. Dynamic simulation programs show at each time point what is going on in the simulation.

As with any simulation software, the more input variables, the more accurate the simulation. Power plant simulation is no different. The variables in power plant simulation can be divided into two categories, internal and external variables. The internal variables are about the variables in the source of the power, in this thesis it is the engines. The type of engine and fuel used in running the engine are some examples of internal variables in engine power plants. The external variables are variables that come from the surroundings. For example, the altitude and temperature have an impact on the power plants outputs.

The more outputs a simulation program can calculate the better. The most relevant output is the energy that the power plant generates. But by adding more features that calculate different outputs, the better the software. For example, emissions and cost of project are also relevant information.

With a dynamic power plant simulation program, we can simulate the total process from startup to ramp down and analyze the data at any given time point. The dynamic simulation program gives us more data but requires much more programming power to operate. Even though dynamic programs give us more data to analyze, a static program can give more accurate outputs compared to real power plants. Both static and dynamic simulation programs have their pros and cons.

3.2 Other similar programs

There already are multiple power plant simulation software available. One of these is called Ebsilon, by Steag. Steag is a German company that offers support and knowledge into the energy markets and power plant solutions. Steag has its own software, Ebsilon, which works similarly to PerfPro and is a power plant simulation software. Ebsilon concentrates more on the thermodynamic processes within a power plant. Ebsilon introduces specific parameters for the model and calculates optimized solutions. Ebsilon also simulates the effects of component degradation, different load cases and changes in the environmental conditions. (Steag 2019)

Ebsilon is already in use for some applications in Wärtsilä energy solutions. This could be a solution for the problems with PerfPro, but not an especially feasible one. PerfPro is after all such a specific simulation program for Wärtsilä engines that a total jump to Ebsilon would not work in the short-term. However, some components of Ebsilon could be used alongside with next generation of PerfPro. Especially the thermodynamic cycles, such as heat recovery.

There are several more power plant simulation software for free on the internet. Most of them are very complex and have a lot of features and options for simulating. All are grate options for general simulation. However, PerfPro is still the best option for Wärtsilä. PerfPro has all the necessary data about Wärtsilä specific engines, which the simulation software online does not have.

There are some academic works about power plant simulation software in databases like IEEE and ACM. These have much information on the basics of power plant simulation, and some go into more specific topics like machine learning and artificial intelligence as aids for the simulation.

For example, in Rui Xu and WeiZhong Yan's work "Continuous Modeling of Power Plant Performance with Regularized Extreme Learning Machine" we can learn about extreme learning machines, or ELM, and the implementation to an online ensemble regression approach to dynamic power plant simulation. The ELM will automatically adjust the models to respond to changes for example in the environment, either gradually or abrupt. This simulation method had a drawback with algorithm performance not being stable due to the random nature of ELM. The solution was to implement the regularized ELM as base model within the online ensemble framework. This will lead to more stable performance of algorithms, tested with multiple real-life power plant data sets with mean average less than the accepted one percentage. (Xu & Yan 2019: 1-8)

Another interesting example of an academic work involving power plant simulation is: "A machine-learning approach to speed-up simulation towards the design of optimum operating profiles of power plants" by Erik Rosado-Tamariz, Miguel Zuniga-Garcia, Gulliermo Santamaria-Bonfil and Rafael Batres. In this publication the focus is on a machine learning algorithm as basis for a power plant optimization program. This machine-learning approach requires much less computing power in a few iterations making the rigorous simulation model into a surrogate model. This will drastically improve the simulation time. However, also this method requires much data about the components and is a rather complex system. (Rosado-Tamariz, Zuniga-Garcia, Santamaria-Bonfil & Batres 2019: 194-199)

In addition to academic works of power plant simulation software, there are some academic works based on architectural reconstruction and platform update. One such academic work is "Architecture reconstruction and evaluation of blockchain open source platform" by Jungho Kim, Sungwon Kang, Hwi Ahn, Changsup Keum and Chan-Gun

Lee. In this academic work the authors use open source blockchain platforms to reconstruct any architectural model. They use mainly two different open source methods: Hyperledger and Ethereum, to do the reconstructing. These two methods use blockchain technology, which is a chain of linear code that is available for anyone on the network to use. This is built into blocks of code segments arranged in a sequence (eToro 2017). The result of the evaluation was that Hyperledger is more modifiable and has better performance, while Ethereum was better in dealing with security. (Kim, Kang, Ahn, Keum & Lee 2018: 185-186)

These topics are interesting and could be applied to a future generation of PerfPro but is not in the scope of this thesis and the next generation of PerfPro. The research in this field is limited and focuses on too specific areas and not the concept of architectural reconstruction and platform update as a whole. In addition, PerfPro as a power plant simulation program is too specific to Wärtsilä engines that previous research cannot be directly applied to PerfPro.

4 RESEARCH METHOD

In this chapter we will go through approaches for research methods in academic researches. The approach is the design science approach. After the approach we will go through the research environment and data gathering in more detail. Finally, there is a subchapter for research process overview, where there will be reflections on this thesis with the theory of design science research methods.

4.1 Approach

Research questions are the basis of research methods. Research questions come from field problems instead of knowledge problems. Solution-oriented knowledge is the key to solving field problems. Solution-oriented knowledge links interventions or systems to outcomes. Pragmatic validity is the justification for research questions. (Van Aken & Romme 2009: 5-12)

Research for projects can be divided into three cycles: relevance-, design- and rigor cycle. In figure 5 below these three cycles are visualized with environment, design, and knowledge bases.

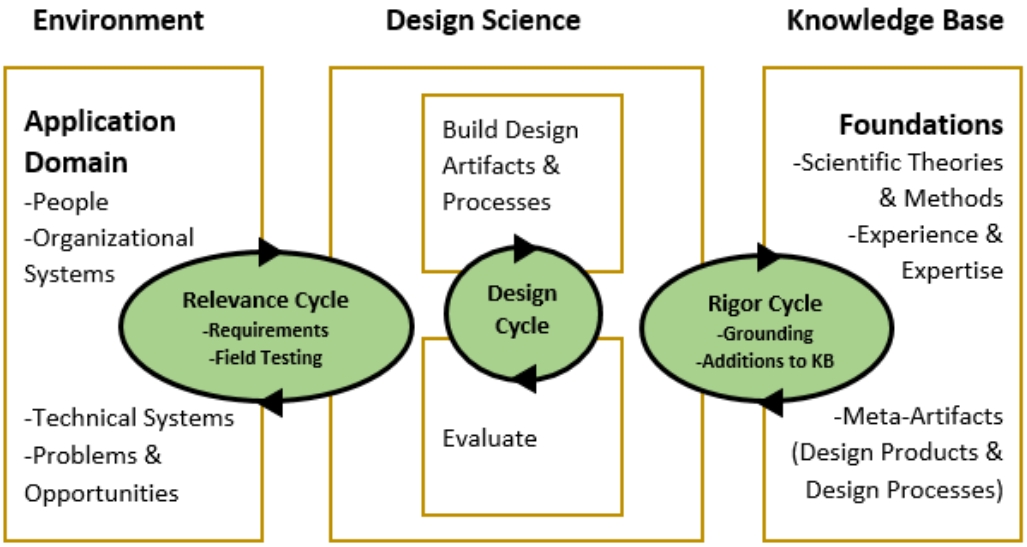


Figure 5. Research cycles. (Hevner & Gregor 2013: 337-355)

The relevance cycle interacts with both the environment and the design science. The rigor cycle interacts with both design science and the knowledge base. The design cycle is only within design science. The relevance cycle focuses on the requirements and field testing. These are substantial parts within the environment and communication to the design. The rigor cycle focuses on grounding, meaning that the research is based on academic theories and that the project uses these methods correctly, and additions to knowledge. The design cycle is a cycle between designing and developing artifacts and evaluating them.

The purpose of research in science is to produce a theory and test it. A theory is a means of representation, a construct, and a statement of relationships within a specific scope. A theory has a specific casual explanation, testable hypothesis, and a perspective statement. (Van Aken et al. 2009: 5-12)

In figure 6 below there is a visualization of a design science research method.

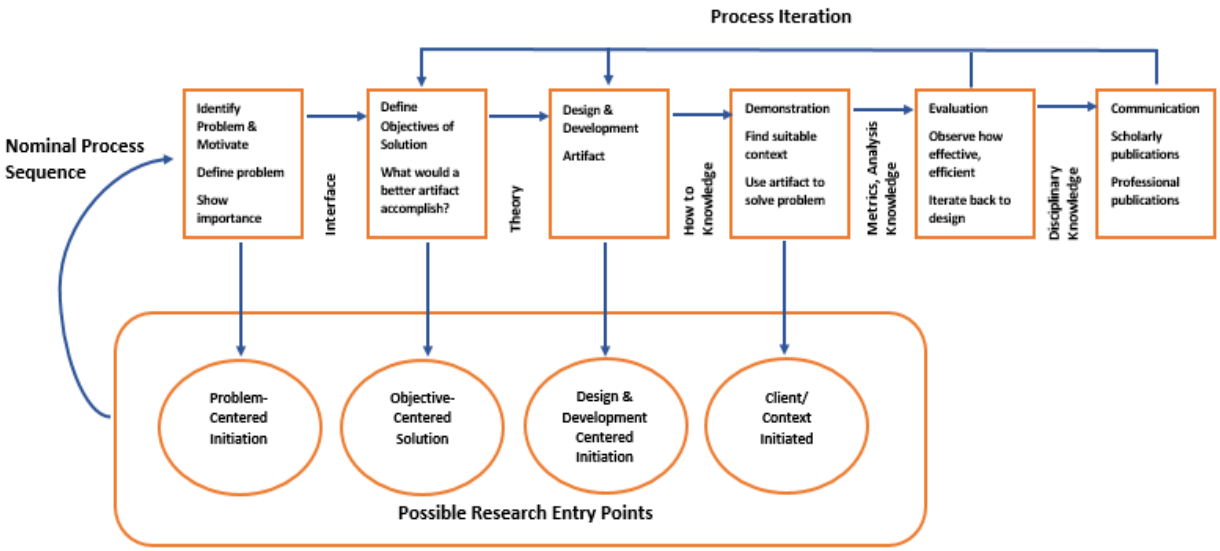


Figure 6. Design science research method. (Peffers, Tuunanen, Rothenberger & Chatterjee 2007: 11)

Here in figure 6 we can see the chain of processes in the design science research method. The nominal process sequence core is as follows: identify problem, define objectives of solution, design and development, demonstration, evaluation, communication. From the first four phases there are different possible research entry points. Their research entry points are differently centered depending on the phase of the nominal process sequence. Evaluation and communication are in an interaction with previous phases.

Problem identification and motivation defines the research problem and is justification for the value of the solution. There are two major resources required for this phase. These are: knowledge of the state of the problem and the importance of a solution. Defining the objectives for a solution implies a solution from the problem definition and what is possible. For this part knowledge of the state of problems and current situation is important. (Peffers et al. 2007: 44)

Design and development phases are basically creating artifacts. These artifacts can be constructs, models, methods, or new properties of technical, social, or informational resources. A design research artifact is any designed object in which some research contribution is included or embedded into the design. Knowledge of theory that can be included

in the solution is important for this phase. Demonstration of the artifact is the next phase. The demonstration is mainly for showing that the artifact can solve one or more aspects of the problem. This demonstration can be a simulation, case study, proof of concept or other activity. Knowledge of how to use the created artifact is the main knowledge of this phase. (Peffers et al. 2007: 44)

The evaluation phase is observing and measuring how well the artifact supports the solution to the problem. This includes comparing objectives of a solution to an actual observed result from the demonstration phase. Analysis techniques and knowledge of relevant metrics is important. Evaluation could include any empirical evidence or logical proof. After this phase, the developer decides if the solution is appropriate. If the solution is good, then we can go to the last phase, communication, and if not, we go back to the design and development phase. Communication is the last phase of the design science research method. Communicating the problem and its importance, the artifacts utility and novelty, and the rigor of its design are all major parts of this phase. Communicating to relevant audiences is part of the process. Communication has a requirement of knowledge of the disciplinary culture. (Peffers et al. 2007: 44)

Problems that can be solved with the design science approach vary from minor improvements to a software design to completely new software that the environment demands. An example would be a missing feature to a program that the environment requires. As for the research method, here we start with figuring out the requirements from the environment. This can be done by interviewing people from the environment that set the requirements. Then we study the problem and figure out if this sort of problem has been solved before. The artifact, in this case a new feature will be developed based on the interviews and research into the problem. (Hevner et al. 2013: 337-355)

4.2 Research environment

With research environment we mean the environment in the application domain. This is visualized in in Hevner’s Research Cycles for information systems research framework

in figure 7 below. The research environment is shown as the box to the left. The environment includes the people, organizational systems, and technical systems.

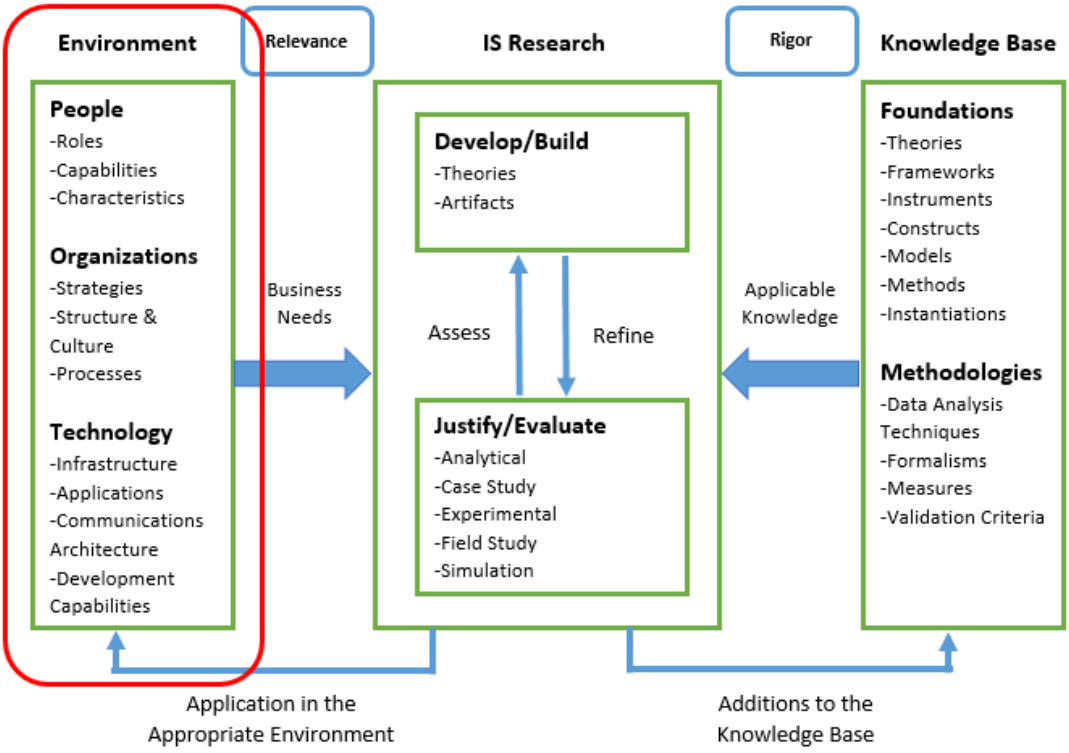


Figure 7. Information systems research framework. (Hevner, March, Park & Ram 2004: 197)

The environment includes people, organizations, and technology. Under people there is highlighted the roles, capabilities and characteristics of the people or persons involved in the project background. For example, when developing a tool within a company, the end users in the company is considered the people of the research environment.

The second part of the research environment is the organizations. The organizations account for strategies, structure, culture, and processes. The organization usually can be interpreted as the whole of the people associated with the project environment. The organization is more of a concept that has its role in the research environment.

The final part is for technology. Technology in this context include: infrastructure, applications, communications, architectures, and developments capabilities. Basically, technology in research environment is the tools used by the people and organization to fulfill the business needs of the process.

The environment determines the business needs for the process. These business needs become requirements for the process. When we do research for solving problems or creating artefacts, we turn to the environment for the details of the problems. Also, improvements and suggestions come from the research environment.

4.3 Data gathering and analysis

Data gathering is an essential part of the science research method. Data should be gathered from reliable sources or experts from the environment we discussed in the previous sub-chapter. The data gathered, is the basis for most aspects of the process, from requirements to developing a solution.

Academic works is a good place to start the data gathering. Known theories and previous researches gives a good basis for any research paper. In academic works and research papers the source is the most important variable. A reliable source is important when getting data from academic works.

If no previous academic works are available about the subject the paper is written on, data gathering needs a more hands on approach. Data gathering can then be done by surveys and polls. Surveys and polls are essentially asking for the data from the environment. By making a survey, we can approximate the whole environments opinions and data. For most cases we need to use statistical calculus to figure out if the approximated data is valid in the whole population. (Bhat 2019)

For this thesis, data gathering is an essential part of the project. There are some academic works that helps to generate a knowledge base, but most of the information comes from

user and developer experience. To most efficiently gather the data from the users we need a survey. The questions asked in this survey are based on previous knowledge of user needs and wishes. With as many as possible replies, we can approximate the opinions, needs and demand of the whole environment for PerfPro. Previous researches and similar programs are available, but PerfPro is way too detailed of a program developed for Wärtsilä needs that the other similar programs do not help much with the knowledge base for this project.

4.4 Research process overview

This sub-chapter is more of a comparison between the design science research method theory and actual industry needs for this thesis. This comparison will mainly be from Peffers design science research method compared to Wärtsilä as industry and environment. (Peffers et al. 2007: 11)

Wärtsilä is a corporation founded in 1839 in Finland. Wärtsiläs main product is engines (diesel, gas, and dual fuel), and engine technologies for marine- and energy markets. Wärtsilä emphasizes environmental and economic solutions for its customers. Wärtsilä is a global company and operates in over 200 locations and in more than 70 countries worldwide. (Wärtsilä 2019)

Energy solutions strive towards a 100% renewable energy future. As of 2019, energy solutions have a total power plant capacity of 70 GW and has installed power plants in 177 different countries. Energy solutions offer flexibility and reliability when it comes to power plants. Wärtsilä has achieved a global leading role in the power plant market. There are multiple different fuels that the power plants operate with. These fuels include: diesel, liquefied natural gas, heavy fuel oil, light fuel oil, natural gas, and liquefied petroleum gas, among others. (Wärtsilä 2019)

For this thesis, Wärtsilä is considered the environment. More specifically, every employee that uses the PerfPro software in its current state. These employees are mainly

working in the energy solutions business line of Wärtsilä. PerfPro is used in sales process to determine performance figures for potential power plants based on engine and site preferences. The sales department sets and calculates the variables accordingly to customer demands. With this information PerfPro can generate a guarantee page that is used to guarantee different performance specific details within the power plant. This covers the people and organization parts of the environment in the information systems research framework in figure 7. For the technology part, we consider the software itself to be represented here. Also, different communicational tools are within this framework. Also, other similar tools and software that communicate with PerfPro is found here.

The business needs come mainly from the users. There have been some complaints about the current state of PerfPro and the research question has been to figure out feasible solutions for the future of PerfPro. In order to fully understand the program, there must be some sort of documentation about PerfPro, with use cases and architectural overviews. In addition to the users, the developers are in a key role in determining the feasibility of the user needs and preferences.

Data gathering will come from interviews with the developers and some documents about the current state of the software. For the business needs the data will come from a survey set to all Wärtsilä employees that use PerfPro in some capacity. The results of the survey will generate the requirements and direction for future versions of PerfPro. The architectural vison for the next generation will come from discussions with the developers. The developer’s input is crucial in determining whether a solution is feasible or not.

Below is a summary table of design science research methods listed by Peffers processes, and the corresponding phases for this project.

Table 1. Comparison DSRM and project.

Peffers DSRM	**Wärtsilä PerfPro project**
Identify problem.	PerfPro should be able to run faster and Excel is an outdated UI platform. Survey users to get new requirements.
Define objectives of solution.	Find feasible continuity for PerfPro. New platform could eliminate problems with PerfPro. Add needed features and simplify program.
Design and development.	Plan for new architectural vision. Prototype possible solutions. Continue development on best concept.
Demonstration.	Prototype review by users and developers.
Evaluation.	Testing period with sales and performance experts.
Communication.	Distribution to end users and performance experts via Wärtsilä internal networks.

The first step in Peffers design science research method is to identify the problem. The problems with PerfPro has been known before this project. To get the specific problems and as many opinions as possible we need the survey. The survey is to be sent to all Wärtsilä employees that potentially use PerfPro, via email lists. There is no email list specifically for PerfPro users, but we know in which lists there are some PerfPro users.

The second step is to define objectives of solution. The main objective is to figure out a feasible continuity for PerfPro. The continuity which fixes most of the problems identified in the previous phase is chosen. This can for example be a totally new program with same features as current PerfPro or basically the same Excel based program with minor fixes to improve functionality and performance.

Next is the design and development phase. This phase includes planning of new architectural model and listing of requirements. Also, the final continuity is decided in this phase. Prototyping of initial possible solutions is done here. Prototypes give directive to continue development in a feasible direction.

The final three steps, demonstration, evaluation, and communication are not in the scope of this thesis. These three steps can be made in the future if this thesis provides a feasible option for the continuity of PerfPro lifecycle. Demonstration phase can be taught of as the prototyping cycle. In the prototyping cycle, we develop partially a concept for the eventual program and validate it with the end users and other developers. The feedback we get from each prototype guides us to better the next prototype and eventually perfect the program. Evaluation phase consists of testing of the final version of PerfPro. This should be done by parts of the end users at a time because the calculation of power plant variables is a crucial part of Wärtsilä business cycle and cannot be totally stopped at any time. The final phase is the communication phase. Updates and new versions of PerfPro are communicated to the end users via Wärtsilä internal networks and platforms. The start page of current PerfPro has some notifications about the availability of new updates and versions. This is a good way of communication and delivery of the new PerfPro version to all PerfPro users.

5 ANALYSIS OF CURRENT STATE OF PERFPRO AND SURVEY

This part is for the power plant simulation program in use today. Documentation, software architecture and software mapping are found here. Also, user experience and improvement suggestions are listed in subchapter 5.2. Data for these user experiences and improvement suggestions were gathered through a survey sent to all current PerfPro users. The point with having a use case as well as flowchart with the survey in this chapter is to compare the current state of PerPro with the user experiences with the software. This will generate a baseline for PerfPro and the eventual requirements for future generations. After analyzing the current state of PerfPro, there will be a plan for the next generation of the program and speculation on the new platform and architecture in the next chapter.

5.1 Analysis of PerfPro

PerfPro is the name of the power plant simulation software that this thesis focuses on. PerfPro is a Microsoft Excel based calculating program for Wärtsilä power plants. By feeding in engine specifications and site conditions, the program calculates derating graphs, flowcharts and guarantees of energy outputs, emissions, and efficiency among other interesting variables. (Jansson 2019)

PerfPro communicates with many different libraries. Among these libraries is a library called Tlib. Tlib is a Wärtsilä made library that calculates chemical and physical standards for PerfPro. Another separate file calculates the methane number. More about methane number later. All the different libraries and files communicate with one another and is the basis of PerfPro. (Jansson 2019)

The first version of Perf is from 1989, made by Kim Jansson. It has been in Excel format since the beginning. With new versions of Excel, Perf has had to be updated. Perf was the predecessor to PerfPro. The first version on PerfPro is from 2010. The newest version of PerfPro is from 2019. (Nyqvist 2017)

The amount of code in the different files that PerfPro consists of is rather large. The file "UnsavedPerfPro" works as the user interface file. This file is an Excel macro-enabled file with about 17 000 lines of VBA code. The file "PerfCode1", 23 500 lines of code, is basically a translator file in VBA code that communicates with the other programs and functions in different programming languages. "Perf2", the calculation engine, consists of functions that calculates different values based on physical standards and guidelines. It consists of over 30 000 lines of C++ code. The database "PerfEngine_Data" consists of 550 different engines and all the differences between them. "PerfProcess" is a database for cooling systems. It consists of about 50 different cooling solutions. (Nyqvist 2017)

5.1.1 Use case

In this sub-chapter we will go through the program from the user's perspective. First when opening the program, a starting page will appear. A screenshot of this start page is visualized in figure 8 below.

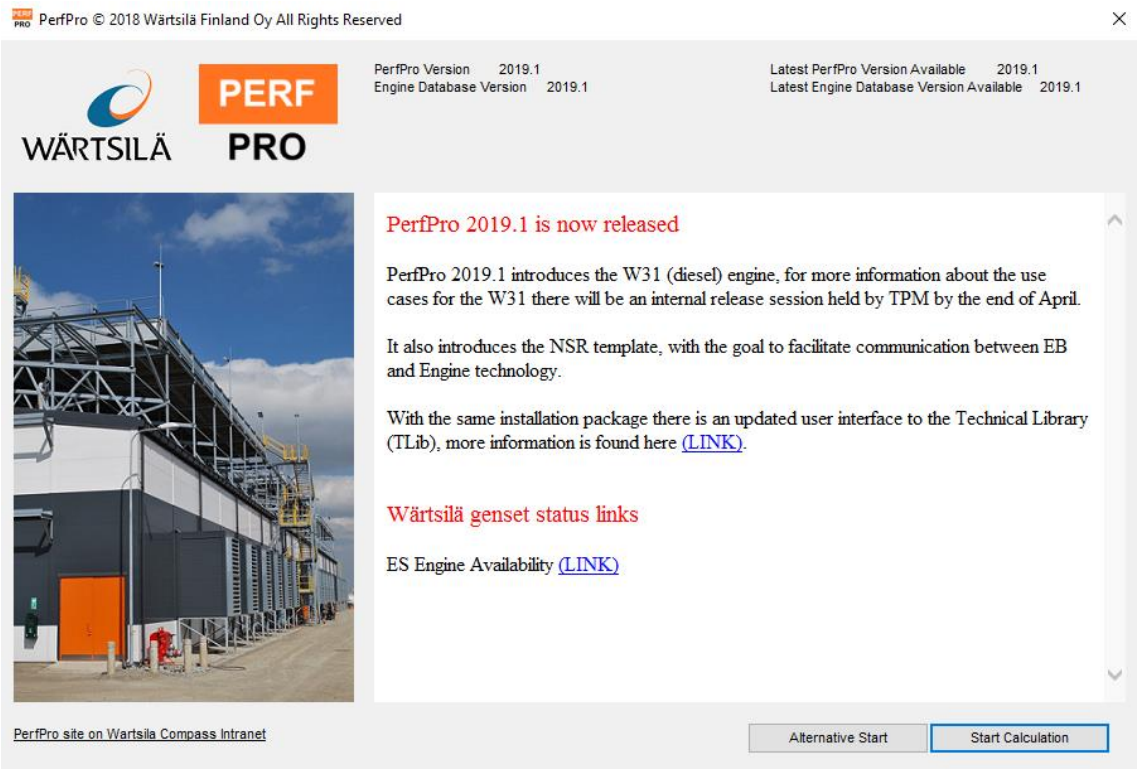


Figure 8. PerfPro start page.

The main reason for the start page is to choose normal- or alternate start. Usually the user selects the normal start, but if the user wants to access the VBA codes or has an older version of the program that has for example problems with the current operating system, the user can select alternate start. From the start page you can also find which version of Perf you are running, as well as the latest news regarding the software. This is also the interface to make updates to databases. There is also a link to the intranet page with more information. By clicking “Start Calculation” the program opens the excel file “Unsaved PerfPro”.

The file “Unsaved PerfPro” can be considered as the user interface. In this file the user sets all the inputs and after calculations can see the outputs. This interface consists of 3 different main pages: 1) design, 2) site conditions and 3) performance. Design is for engine specifications; site conditions is for site specific conditions and performance is for the calculated results. Design page opens first. At first the page seems empty, but many rows and columns have been hidden. In figure 9 we can see how the page looks when first opened.

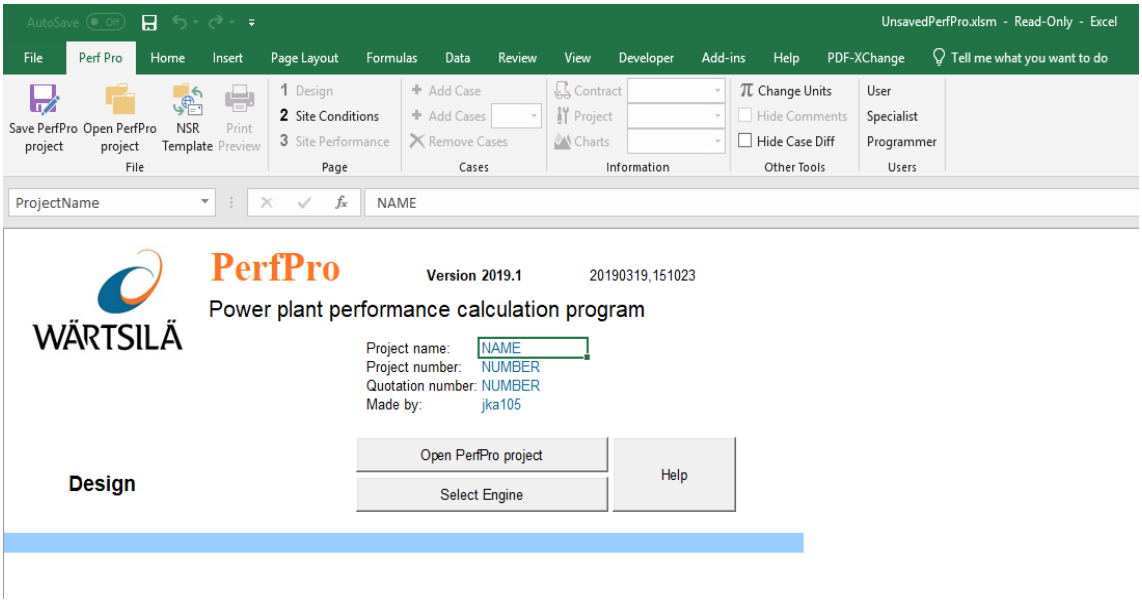


Figure 9. Unsaved PerfPro.

The custom ribbon on the top of the page has many shortcuts and common functions on it. Most of them can be applied in the program itself. The one function that is not found

anywhere else is the User/Specialist/Programmer functions. These functions allow the user to switch between access levels to hidden values and pages by knowing the passwords. A “Specialist” can for example change limitations of any variable. Furthermore, a “Programmer” can and change functionality within the program.

Here we see two options, “Select Engine” and “Open PerfPro Project”, and a help button. “Open PerfPro Project” is used to load a previously saved project so that all the different variables and calculations are loaded up correctly. The save files are macro free and cannot be opened without using this load function. The user should know witch version of Perf is he or she is using. This is because a save file from an older version can have problems to open in newer Perf’s. This is a usual challenge in software development. Also, only one version of Perf can be installed at the same time on a computer.

To make a new Perf, the user first enters information to the cells in blue text. Project name, -number and quotation number helps to keep track of different projects. The “Made by” field is automatically filled with the Wärtsilä account name of the user. To actually start the calculations the first step is to select the engine.

By pressing the “Select Engine” button a new window will open. In this window the user will select the specific engine to be used in the calculations. Engine specifications include: groups, types, cylinder amounts, frequency, fuel, and voltage. An example of a specific engine is in the figure, figure 10, below. The engine is: W20V31SG, with 50 Hz, high MN, temp < 55 °C, NOx 36 – 90 ppm, 11 kV.

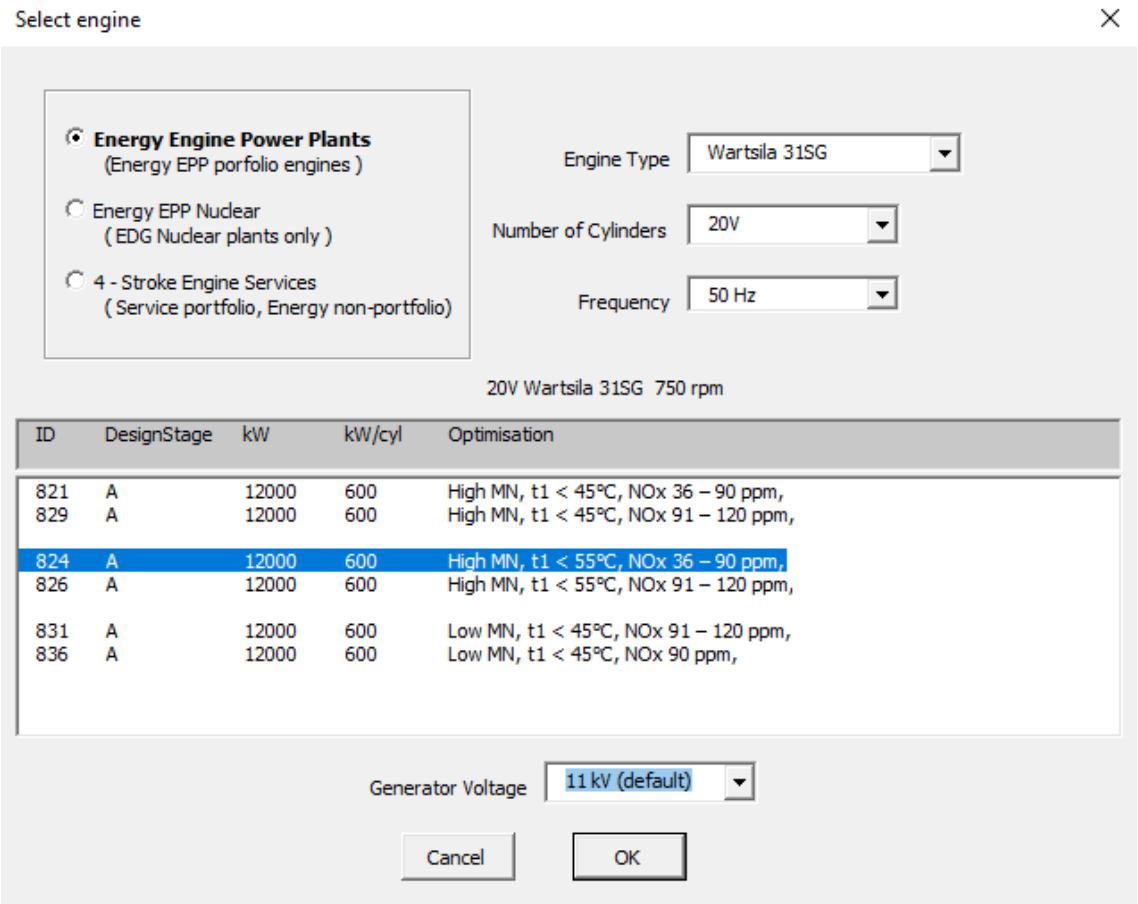


Figure 10. Select engine page.

In this particular example, the engine group, type, and cylinder amount are specified in the name, W20V31SG. 31SG stands for the group: Wärtsilä 31-cm boar, SG is the engine type: gas engine, and V20 stands for 20 cylinders. 50 Hz is the frequency of the grid that the power plant will be attached to and so the generated frequency. High MN (Methane Number) is for the gas used as fuel in the engine. Temperature less than 55 °C is for the ambient temperature on the site. NOx 36 – 90 ppm means that the emission from the engine has a nitric oxide (NO) or nitrogen dioxide (NO_2) amount between 36 and 90 parts per one million parts, in the exhaust gas. Finally, the amount of 11 kV is for the generator voltage.

After selecting a specific engine, hidden information and variables become visible. Now the user can do the changes to other specifications. Cells in blue are modifiable for the user. If the user is in specialist or programmer mode, everything is modifiable. Most of

the values can be changed from metric units to other units, for example imperial units, if needed. There is also a button in the custom ribbon to change all units at the same time.

There is a default fuel composition added after selecting the engine. In gas fuel engines the composition is by default in mol-percentage. Other percentage options are in mass- or volume percentage. The default gas is: 95% methane (CH4), 2% ethane (C2H6), 3% propane (C3H8) and 0,001% i-butane (i-C4H10). This composition is default because it results in methane number 80. More about methane number later. In gas fuels there are multiple components that can be used. Most of the components are hydrocarbons, but also carbon oxides, hydrogen, oxygen, and helium among other components can be added.

When calculating with gases as the fuel, Perf will automatically calculate the methane number. This number is an indication of the knocking effect caused by the gas. The methane number can basically be anything but usually it is between 40 and 120. The methane number is calculated in a separate file. Methane number is the resistance of fuel gases to engine knock. Methane number is a reference number comparison of a gas composition with pure methane resulting in a methane number of 100 and pure hydrogen results in a methane number of 0. Methane numbers below 80 usually reduces gas efficiency as well as increases emissions or create problems within the engine. (GIE 2012)

On the design page there are also many other variables that can be changed, some examples are: number of engines, pressures of air and gas, altitudes, power factors, minimum and maximum temperatures, and humidity. All of the variables have intervals of acceptable values, so that the program would work. To change these intervals the user must change to specialist or programmer view. After setting these variables, the next step is to select a cooling method and process. By clicking the button to select cooling method and process at the bottom of the design page, a new window will open. This window is shown in figure 11 below.

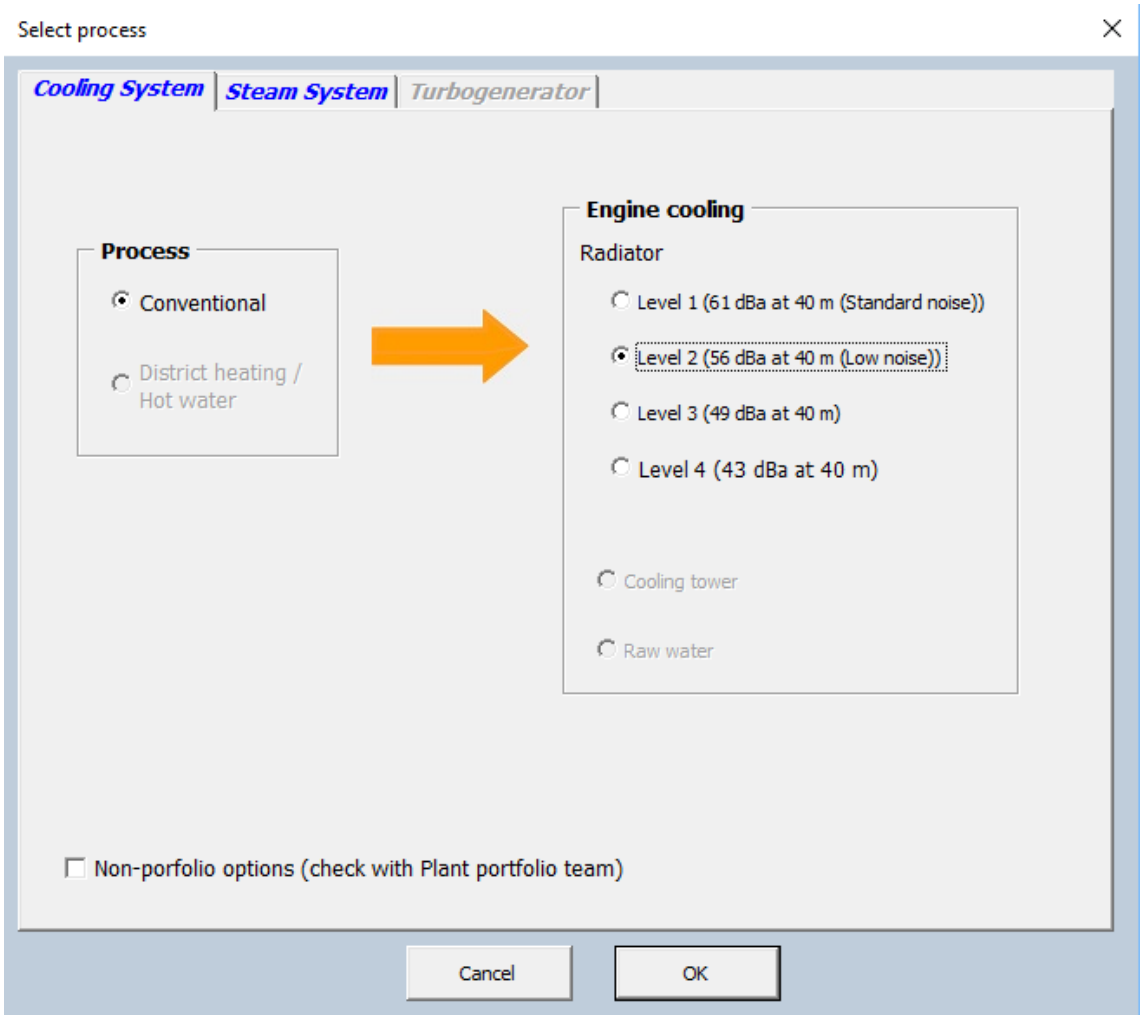


Figure 11. Selection of cooling system.

The first step is to determine if we will use radiator, cooling tower or raw water for cooling of engine. The conventional option is to use a radiator and then select the level of the radiator based on the noise the level generates. In this particular example we have set the engine same as in previous example, and that particular engine does not have option for cooling tower or raw water. For heat recovery purposes, the user switches to the steam system page at the top. In the figure below there is a picture of the steam system page.

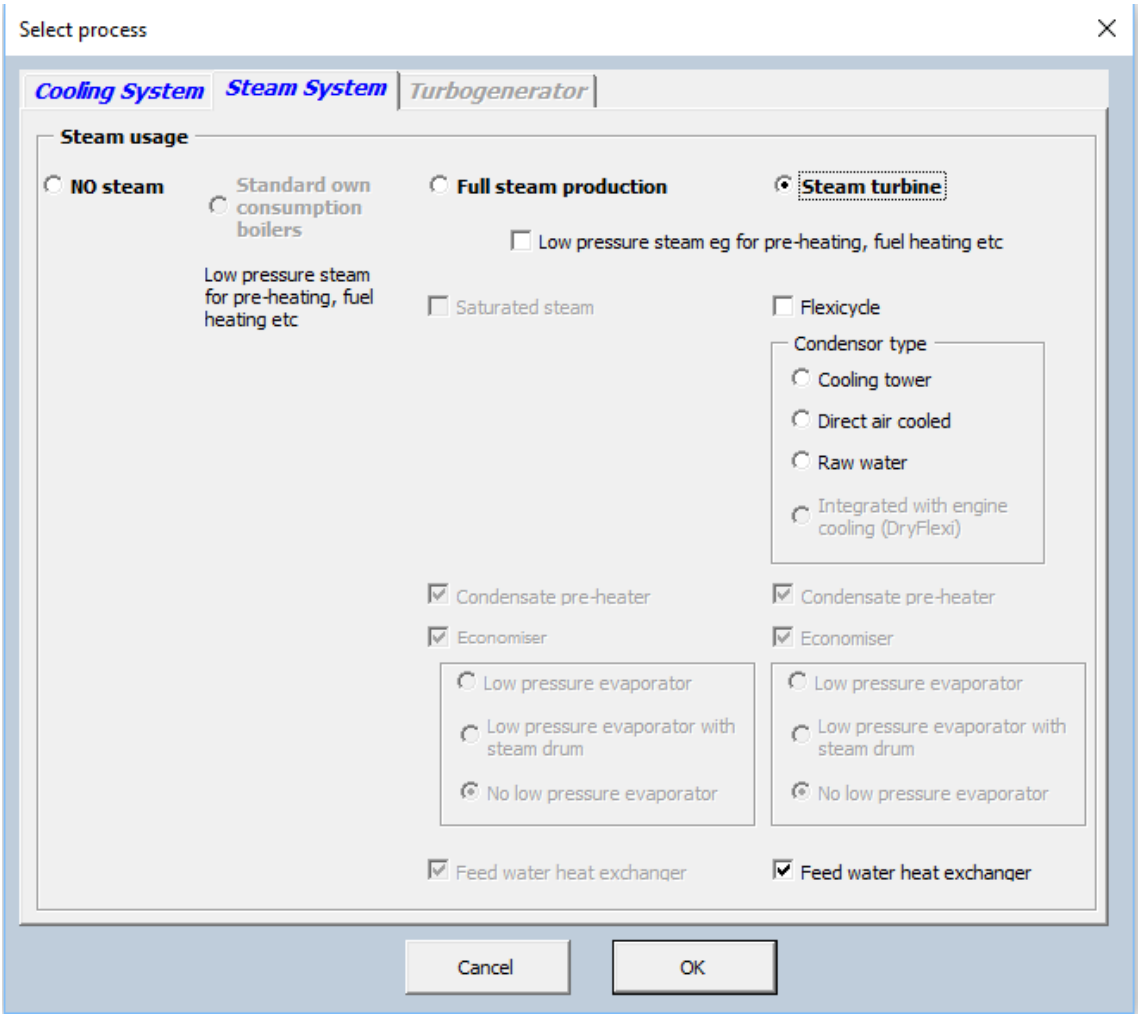


Figure 12. Steam system for heat recovery.

Here we can see the different options for heat recovery using steam systems. The options are to use a boiler, full steam production or a steam turbine. When the user has set all the cooling and heat recovery options, he or she clicks ok and then a new window with the cooling circuits appear.

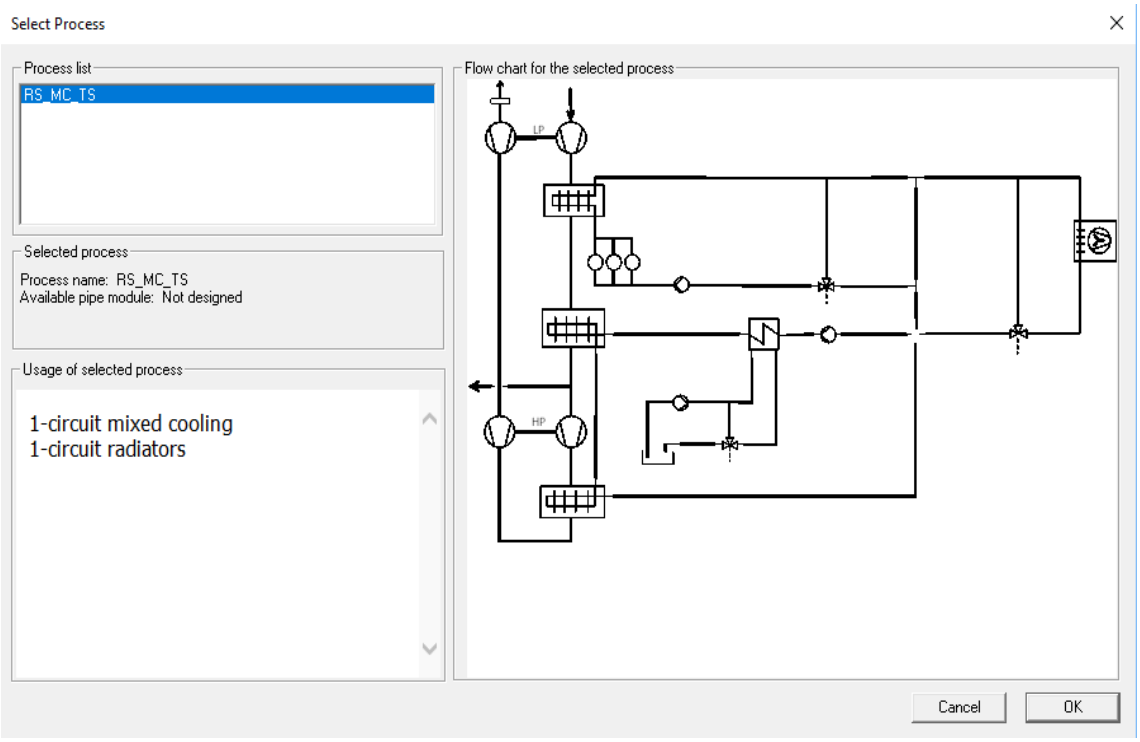
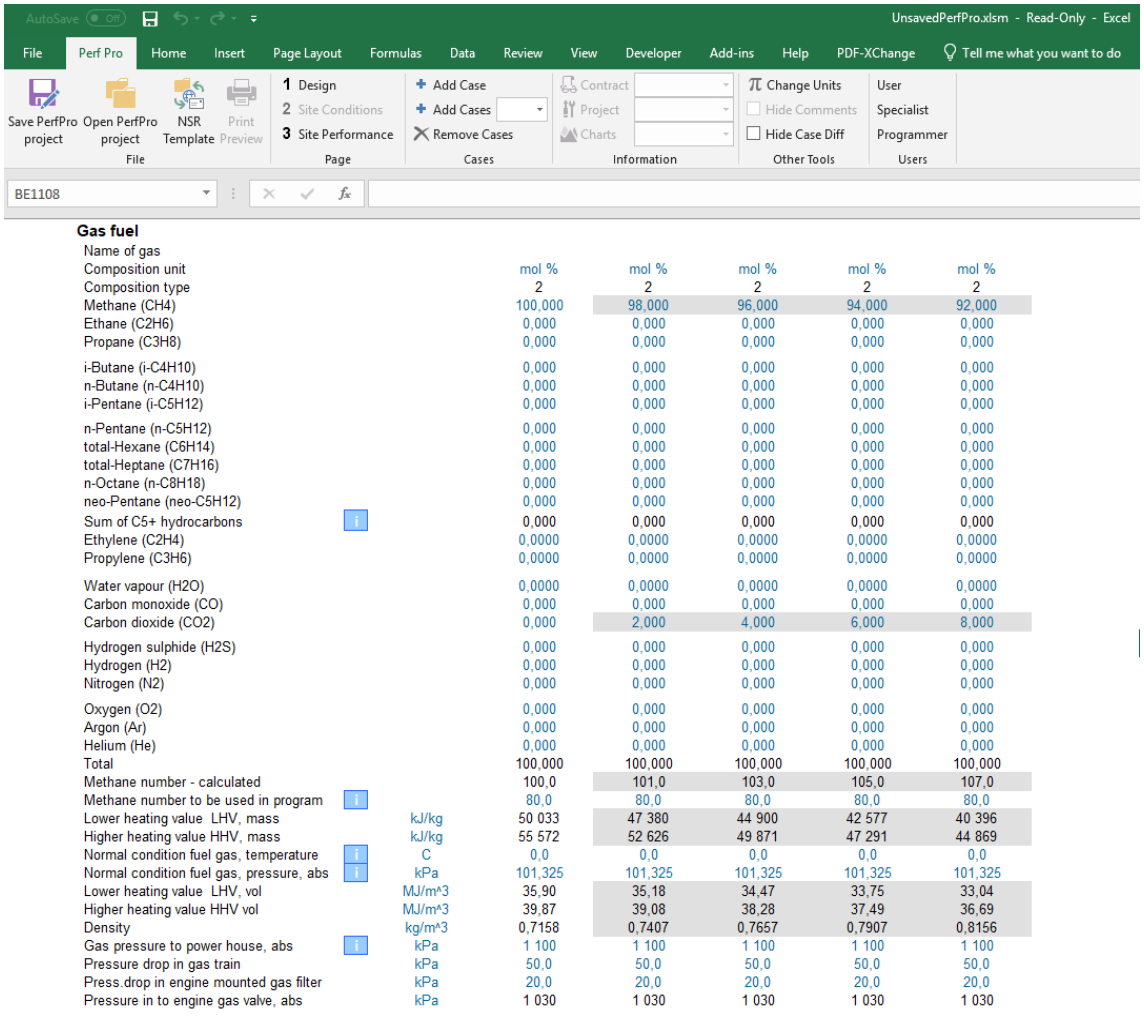


Figure 13. Cooling circuit.

In this particular example there is only one option for the radiator, a 1-circuit mixed cooling system with radiator. Sometimes there is an option for 2-circuit cooling, however, this is not very common.

When the cooling method is set, the user can choose to optimize SG engine. This is an option only for gas engines and by clicking it the user can change some variables. Now when everything is set on this page, the user can advance to the site conditions page, either by clicking the button at the bottom of the design page or the button in the custom ribbon.

Now we have proceeded to the site conditions view. On site conditions page the main feature is to add cases. Adding cases copies the variables and calculations from the selected case and places this new case next to the original. The user can add more than one case. Cases are added from the custom ribbon, one or multiple at a time. After this the user can change something in the new case and compare everything to the original case. An example of this is in figure 14 below.



Gas fuel

Name of gas							
Composition unit			mol %	mol %	mol %	mol %	mol %
Composition type			2	2	2	2	2
Methane (CH4)			100,000	98,000	96,000	94,000	92,000
Ethane (C2H6)			0,000	0,000	0,000	0,000	0,000
Propane (C3H8)			0,000	0,000	0,000	0,000	0,000
i-Butane (i-C4H10)			0,000	0,000	0,000	0,000	0,000
n-Butane (n-C4H10)			0,000	0,000	0,000	0,000	0,000
i-Pentane (i-C5H12)			0,000	0,000	0,000	0,000	0,000
n-Pentane (n-C5H12)			0,000	0,000	0,000	0,000	0,000
total-Hexane (C6H14)			0,000	0,000	0,000	0,000	0,000
total-Heptane (C7H16)			0,000	0,000	0,000	0,000	0,000
n-Octane (n-C8H18)			0,000	0,000	0,000	0,000	0,000
neo-Pentane (neo-C5H12)			0,000	0,000	0,000	0,000	0,000
Sum of C5+ hydrocarbons	i		0,000	0,000	0,000	0,000	0,000
Ethylene (C2H4)			0,0000	0,0000	0,0000	0,0000	0,0000
Propylene (C3H6)			0,0000	0,0000	0,0000	0,0000	0,0000
Water vapour (H2O)			0,0000	0,0000	0,0000	0,0000	0,0000
Carbon monoxide (CO)			0,000	0,000	0,000	0,000	0,000
Carbon dioxide (CO2)			0,000	2,000	4,000	6,000	8,000
Hydrogen sulphide (H2S)			0,000	0,000	0,000	0,000	0,000
Hydrogen (H2)			0,000	0,000	0,000	0,000	0,000
Nitrogen (N2)			0,000	0,000	0,000	0,000	0,000
Oxygen (O2)			0,000	0,000	0,000	0,000	0,000
Argon (Ar)			0,000	0,000	0,000	0,000	0,000
Helium (He)			0,000	0,000	0,000	0,000	0,000
Total			100,000	100,000	100,000	100,000	100,000
Methane number - calculated			100,0	101,0	103,0	105,0	107,0
Methane number to be used in program	i		80,0	80,0	80,0	80,0	80,0
Lower heating value LHV, mass		kJ/kg	50 033	47 380	44 900	42 577	40 396
Higher heating value HHV, mass		kJ/kg	55 572	52 626	49 871	47 291	44 869
Normal condition fuel gas, temperature	i	C	0,0	0,0	0,0	0,0	0,0
Normal condition fuel gas, pressure, abs	i	kPa	101,325	101,325	101,325	101,325	101,325
Lower heating value LHV, vol		MJ/m^3	35,90	35,18	34,47	33,75	33,04
Higher heating value HHV vol		MJ/m^3	39,87	39,08	38,28	37,49	36,69
Density		kg/m^3	0,7158	0,7407	0,7657	0,7907	0,8156
Gas pressure to power house, abs	i	kPa	1 100	1 100	1 100	1 100	1 100
Pressure drop in gas train		kPa	50,0	50,0	50,0	50,0	50,0
Press.drop in engine mounted gas filter		kPa	20,0	20,0	20,0	20,0	20,0
Pressure in to engine gas valve, abs		kPa	1 030	1 030	1 030	1 030	1 030

Figure 14. Gas composition change.

In this example we added 4 new cases and changed the gas from 100% methane to 92% methane and 8% carbon dioxide in steps of two percentage units. Everything with a grey background indicates changes to the previous case. We can see that many values changed in the different cases quickly by just looking at the cells with a grey background.

The most common case difference is ambient conditions and engine load. Engine load can vary from 100 and down. An engine is seldom used at 100 % effectiveness all the time, sometimes the power plant is used to less than the maximum, this is where we use engine load at smaller values than 100. The sites own consumption is also added here. Own consumption can be for example lights, fans, or cooling. After all the cases are set,

we click on the site performance button, either at the bottom of the page, or from the custom ribbon.

In the site performance page, we basically cannot change anything. All the cases are listed here, and all the variables and calculations are summarized here. If we want to make changes, we go back to either the design page or site conditions page. Units can still be changed. From here we can view flowcharts, guarantee pages and derating graphs.

In flow charts view, we can toggle through the cases and see a circuit of the system with temperatures, flows and other values. A picture of this can be seen in figure 15.

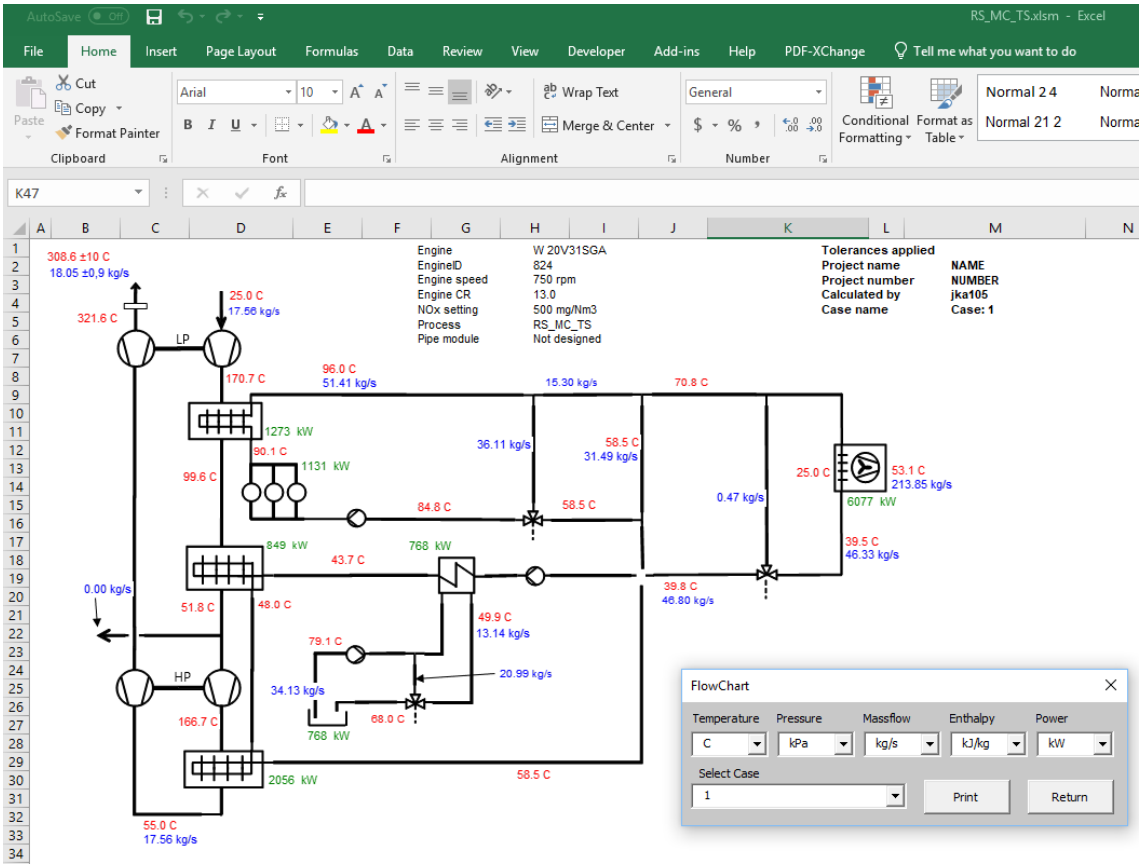
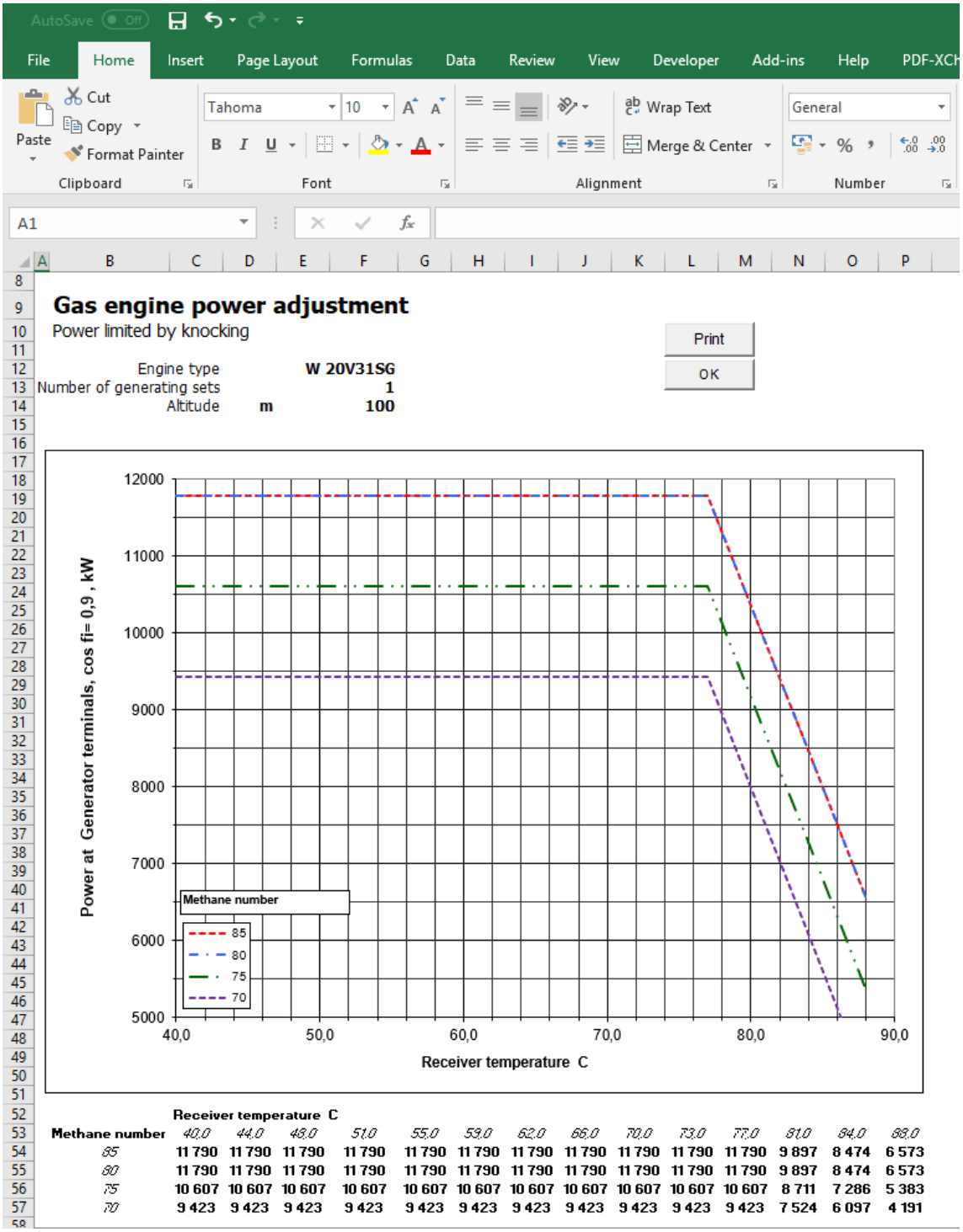


Figure 15. Flowchart.

Derating graphs show different derating options. Knocking caused by the methane number can lead to derating and can be shown as a graph. A picture of this is in figure 16 below.



Receiver temperature C														
Methane number	40,0	44,0	48,0	51,0	55,0	59,0	62,0	66,0	70,0	73,0	77,0	81,0	84,0	88,0
85	11 790	11 790	11 790	11 790	11 790	11 790	11 790	11 790	11 790	11 790	11 790	9 897	8 474	6 573
80	11 790	11 790	11 790	11 790	11 790	11 790	11 790	11 790	11 790	11 790	11 790	9 897	8 474	6 573
75	10 607	10 607	10 607	10 607	10 607	10 607	10 607	10 607	10 607	10 607	10 607	8 711	7 286	5 383
70	9 423	9 423	9 423	9 423	9 423	9 423	9 423	9 423	9 423	9 423	9 423	7 524	6 097	4 191

Figure 16. Derating caused by knocking.

Other derating graphs are gas derating and ambient derating. Also, on this page we can see the heat rate graph. In every graph there is a print option. This print option constructs

a new page with the current graph and other valuable information placed nicely so that printing the page will look good for the customer. This is included in the contract.

The guarantee page is the most important page of the program. When generating it we choose which case is the guarantee case. This page can be printed to a nice format to give to the customer and is a reference document of what Wärtsilä guarantees given all the different variables given earlier in the program. The guarantee page is a summary of the most important values and calculations regarding the project.

5.1.2 PerfPro program flow chart

A simplified flowchart of the PerfPro software is modelled in figure 17 below. Detailed descriptions will follow.

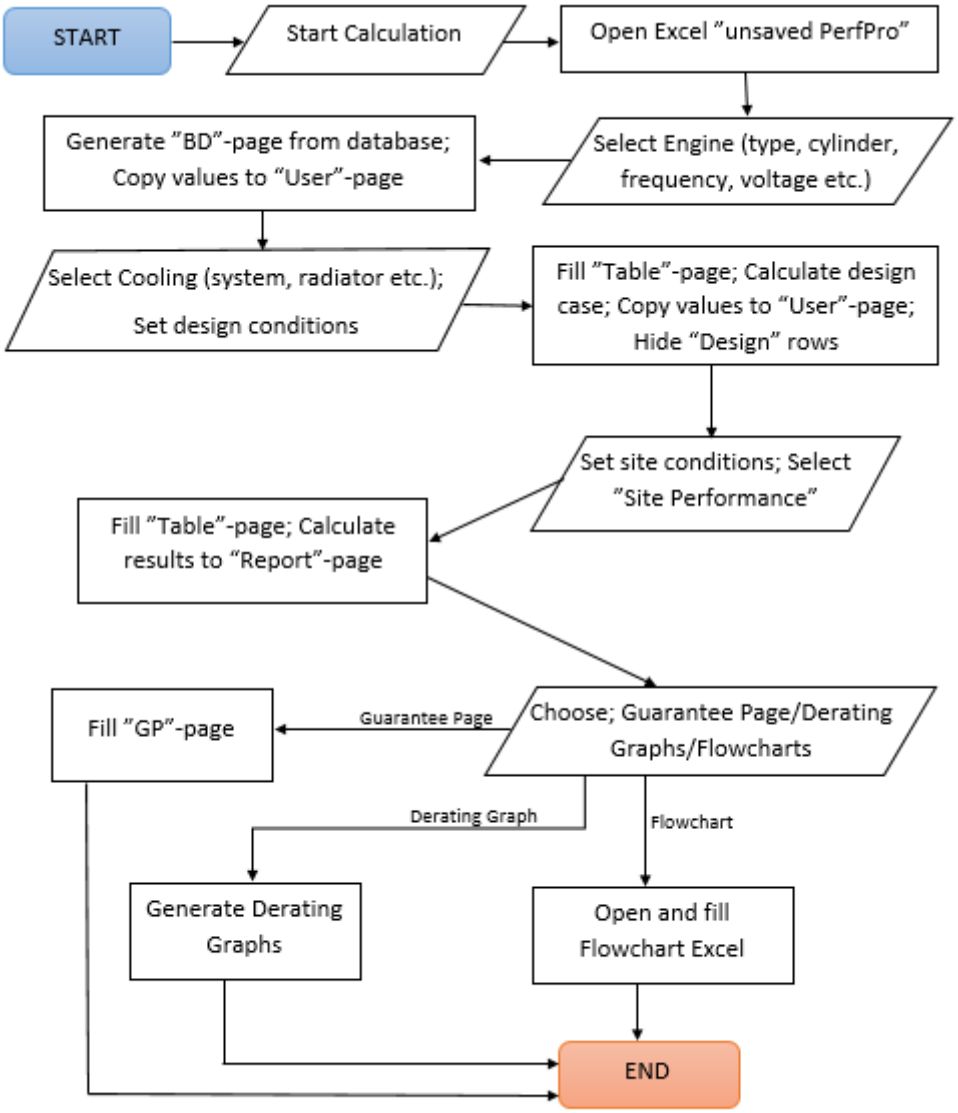


Figure 17. PerfPro software flowchart.

This description of the flowchart is pretty streamlined compared to the actual program. At any point it is possible to either save or exit the program. Additionally, there is an option to go back a stage or two at any point to change something. By changing something, most of the parameters will stay in the values set by the user. However, if changing for example the engine type, most of the parameters and values will have to be inserted again because the values does not work with the other engine type.

Basically, this flowchart descripts the different steps the user makes in the use case chapter 5.1.1. There are more detailed options available such as going to specialist or programmer view to change or view different options. In addition to the use case similarities, this flowchart also shows what the program does after each variable or group of variables is inserted. This gives information about where the different calculations are done.

5.2 PerfPro user survey

User input was implemented through a survey. The platform for the survey was Google Forms. Google Forms is an easy way to make a survey and by sharing the link, users can easily access and answer the questions. Also, the summary page with percentages and lists of answers is very clear and to the point. Another feature that is good with Google Forms, is that it is possible to look at any specific submission and link the answers together.

The questions asked are listed below:

- How often do you use PerfPro?
- Business unit
- Line of work
- General opinion on PerfPro
- How often are you uncertain about your inputs in PerfPro?
- How often are you uncertain about your calculation results in PerfPro?
- Which would you rather use: mobile device or laptop?

- Wish for: online vs. offline
- Wish for: easy to use, light version vs. high flexibility, heavy version
- Opinion on Excel as user interface
- What works well?
- What does not work well?
- Your name (optional)
- Interest to test future versions of PerfPro
- Open feedback about PerfPro

The three first questions are about the person that takes the survey. We get general knowledge about in which departments PerfPro is used and to which extent. The three next questions give the overall picture of the state of PerfPro. A general grade is given and after that we want to know how often the user is uncertain about the inputs or outputs. The three next questions give us feedback on user preferences in the future. We cannot just ask the users if they want a mobile version or if they want to be able to use PerfPro both offline and online. Instead we gave options on different scales for example offline versus online. The three next questions are all open questions where the survey taker can freely write about their experiences with the program. The two next questions are linked together. If the user is interested in testing future versions of the program, they must enter their name for us to be able to find said person for testing. The last question is for any comments regarding the program, if the user still has something on their mind.

There is no designated email list for the users of PerfPro. Instead this survey was sent to different email groups where there are at least some users. In total the survey link was sent to about 600 employees. In the email it was mentioned to ignore the survey if you

never heard of PerfPro. Out of the 600 employees it is approximated that about 200 of them should have worked with PerfPro. The survey was up for 2 weeks and got 80 answers. A reminder was sent after one week. No question was mandatory, so all questions did not have 80 answers.

For the results, they are divided into multiple choice and open questions. First a summary of the multiple choice regarding the three first questions will be displayed in table 2. After that there will be a table (Table 3) for uncertainty with inputs and outputs. The final part of multiple-choice questions with answer levels from 1 to 5 will be next. For the open questions, there is a compilation of positive and negative comments separately.

Table 2. Survey question results 1-3.

Question 1	**Daily**	**Weekly**	**Monthly**	**Rarely**	**Never**	**Total**
How often do you use PerfPro?	14 (17,7%)	25 (31,6%)	22 (27,8%)	17 (21,5%)	1 (1,3%)	79
Question 2	**Energy**	**Marine**	**Service**	**Other**		**Total**
Business unit.	73 (92,4%)	4 (5,1%)	2 (2,5%)	0 (0,0%)		79
Question 3	**Sales**	**Project**	**Technical Support**	**Other**		**Total**
Line of work	38 (48,1%)	12 (15,2%)	20 (25,3%)	9 (11,4%)		79

From these questions we get a general idea about the users. In which departments the program is used. Mainly in energy sector and within energy, in sales. Also, the first question about frequency of use could have a weight on the answers.

Table 3. Survey question results 5 & 6.

Question	**Never**	**Sometimes**	**Often**	**Total**
5. How often are you uncertain about your inputs in PerfPro?	14 (19,4%)	53 (73,6%)	5 (6,9%)	72
6. How often are you uncertain about your calculation results in PerfPro?	23 (31,1%)	45 (60,8%)	6 (8,1%)	74

If we use a value of 1 for never, 0 for sometimes and -1 for often, the average for uncertainty about inputs would be about 0,125 and average for uncertainty about output would be about 0,230. Both values are fairly close to the neutral 0, but on the positive side. Additionally, output certainty is a bit higher than input certainty.

Table 4. Survey question results 4 & 7-9.

Question	**1**	**2**	**3**	**4**	**5**	**Total**
4. General opinion on PerfPro	0 (0,0%)	3 (3,8%)	17 (21,5%)	52 (65,8%)	7 (8,9%)	79
7. Mobile devices (1) vs. Laptop (5)	0 (0,0%)	0 (0,0%)	6 (7,6%)	7 (8,9%)	66 (83,5%)	79
8. Online (1) vs. Offline (5)	16 (21,6%)	6 (8,1%)	17 (23,0%)	14 (18,9%)	21 (28,4%)	74
9. Light version (1) vs. Heavy version (5)	9 (12,2%)	12 (16,2%)	13 (17,6%)	14 (18,9%)	26 (35,1%)	74

Average for general opinion is 3,797. That grade is a fairly good one. Before making this survey, the expected grade was 3. The actual grade was a bit higher, but not over a good grade of 4. The mobile vs. laptop question was the most one-sided question. The question

had an average grade of 4,759. Online vs. offline and light- vs. heavy version questions were close to a neutral grade. Online vs. offline had an average of 3,243, and light vs. heavy average was 3,486. Both of these questions were so close to a neutral average, that they do not have much weight on future versions of PerfPro. However, the answers were slightly towards offline and heavy versions, respectively.

As for the open questions, there are compilations about every question in the paragraphs below. The open questions are:

- 10. Opinion on Excel as a user interface
- 11. What works well?
- 12. What does not work well?
- 15. Open feedback about PerfPro.

Additionally, there was an open field to input the survey participants name and willingness to test future versions of PerfPro. These names are not relevant to this thesis. However, 45/74 (60,8 %) were willing to test and 29/74 (39,2 %) was not.

Opinion on Excel as user interface was pretty neutral. If grading positive comments with a grade of 4 or 5, negative with 1 or 2 and neutral with 3, we get an average of 2,887. Some highlights are listed below.

- Well known software base
- Easy access
- Enables utilization and movement to other Excel files
- Excel updates can change functionality

- Slow
- Messy
- Crashes
- Infects other sessions
- Requires restart after sleep mode

Excel as a user interface can be summarized as easy to learn and easy to access. Every computer in this company has the Microsoft Office package installed automatically and most, if not all, employees in this company has used Microsoft Excel before. However, when working with large files and programs, like PerfPro, Excel tends to get very slow and occasionally crashes. Especially when working with multiple sessions open at the same time. The last three points in the list above should be classified as bugs. These problems should be targeted for continuous development and should not be understood as known and accepted features.

The next two open questions were quite similar. What works well, and what does not work well? There were 51 comments on what works well, and 50 comments on what does not work well. But when reading the comments, many were the same or did not bring up new points. Therefore, as positive comments, there are 15 and 21 negative comments. In the positive section there was more general opinions that for example everything works well enough, but in the negative part the points were more specific. Some highlights for the parts that works well are listed below.

- Engine part
- Site specific estimates fast and easy
- Cooperation with PerfPro team excellent

- Many cases simultaneously
- Ability to simulate almost any condition
- Pre-selected parameters
- Clear indications of what you can change and which not
- Quick calculations
- Standard engines
- Heat rate calculations
- Copying values to other Excel files
- No need to call helpdesk for installing the software
- Changing input data
- Works offline
- Cooling systems

Most of these are pretty self-explanatory. Many of the comments were suggesting that the program works well, fast and is easy to operate when doing standard engine and accessory calculations. A big plus was also that the users find the cooling systems and heat rate calculations to work well. Next there is a list of the comments in what does not work well.

- Selecting correct engine can be hard
- Getting back from unit selection

- Derating curves
- Slow & crashes
- Need to jump between different sheets
- CHP default settings
- Aux load selection
- Automatic updates
- APEX and Perf should be more in-line
- Slow start
- Generating performance figures
- Printing and sorting
- Heat recovery
- General information, guidelines
- Too much work to change engine type
- Comparing different engines
- Saving and using old projects
- SCR consumption missing
- GPRS and heating consumption missing

- Trace heating consumption missing
- Calculation parasitic load

These comments are more specific in what the users found that does not work well. A few comments are about general use and that it should be easier to e.g. jump between different sheets or that something is unclear. Other comments were about some feature missing or a calculation gives a different value than in other tools. The last four points about missing features are about continuous development and are not related to a new platform. Finally, there is a list of general comments or open feedback that the users gave.

- Ability to create and save custom gas compositions
- Connection with other tools, APEX, CRM
- Platform change can not affect functionality and process
- High cost and risk if outsourcing development
- Performance experts could have extended version with more parameters, fine tuning
- Separate sheet for project general data, conclusions, submitted info…
- 2 engine comparison with same site conditions, in perf or separate tool
- Heat calculations must be added
- Delayed implementation of new engine types
- Feedback to be collected more often
- Scripting to automate case running

- Light version with more graphs and tables straight out of perf
- All perf data to be stored online for easy access
- Multi-unit plant level efficiency curves for gross and net efficiency, saw tooth curves
- Why some values are limited?

From open feedback there were many comments about communication. Feedback from users could be collected more regularly and the roadmap and visons should be viewable for all. Another comment that people made often was the wish for multiple engine comparison in case part in PerfPro. Also, more cooperation with the power plant prizing tool APEX was wished. The point about delayed implementation of new engine types is due to the lag between sales release and availability of actual information. We cannot release an engine to PerfPro without knowing the facts first. As for the point about some values being limited, there are some values that are physically limiting for an engine to operate. For example, fuel composition and moisture in the air can make it impossible to run an engine.

All in all, the survey got a fairly good amount of answers and a general vison of the user's perspective was achieved. The open questions highlighted which parts the users find good and where improvements must be made. Also, some features that are missing from PerfPro were mentioned. Something concrete from this survey was the total lack of wish for mobile version of PerfPro. The mobile version has been taught of in the past, but now it can be forgotten at least from the next few versions. The requirements for the eventual next generation of PerfPro are going to be based on the opinions and data gathered from this survey. More about the requirements in sub-chapter 6.1.

6 PLAN FOR NEXT PERFPRO

This chapter can be considered as the artefact from the design scientific research method. The artefact will be a plan for next generation of PerfPro. This includes the requirements and the architectural vision. The requirements are mostly based on the survey results and opinions of the developers. The architectural vision will be based on developer preferences and knowledge. In the end of this chapter, there will be a sub-chapter for platform speculation. Here we can find the suggestions and feasible options for the next generation platform.

6.1 Requirements

The most important requirement is continuity. The updating of PerfPro cannot interfere with way of working for sales and sales support. While under development, PerfPro must work in some capacity for sales in order for Wärtsilä to be able to make new contracts with customers. Also, the next version of PerfPro must be similar enough to the old version's user interface so that the end users can learn the new user interface in reasonable time.

As for other requirements, based on the survey results, we do not need a mobile version. A mobile version has been a concept to implement in the past, but in the survey results everyone preferred the standard laptop version. In the questions online or offline and light or heavy versions, the results were much closer. However, the scale tipped towards offline and heavy versions. For purposes of this project, the next version will be a mainly offline version, with some online aspects. Also, there will be only one heavy version instead of a light and heavy version separately.

Other requirements are based on the survey results from chapter 5.2. These requirements are mostly suggestions and remarks about functionality and features that should be included in PerfPro. There requirements are listed below.

- Selecting correct engine to be made easier
- Add SCR consumption
- Add GPRS and heating consumption
- Add trace heating consumption
- APEX and Perf more in-line
- Improve printing and sorting
- Improve error messages
- Comparing different engines feature
- Backward compatibility
- Ability to create and save custom gas compositions
- Ability for perf data to be stored online for easy access

In addition to these requirements, all the main features from current PerfPro to be included in next generation of PerfPro. Also, the problems with PerfPro crashing, not working with other sessions open and the need for restart after sleep mode should be investigated and if possible fixed.

6.2 Architecture

The second most important requirement, and reason for this project, is the update of the architecture. The current architecture can be summarized in figure 18 below.

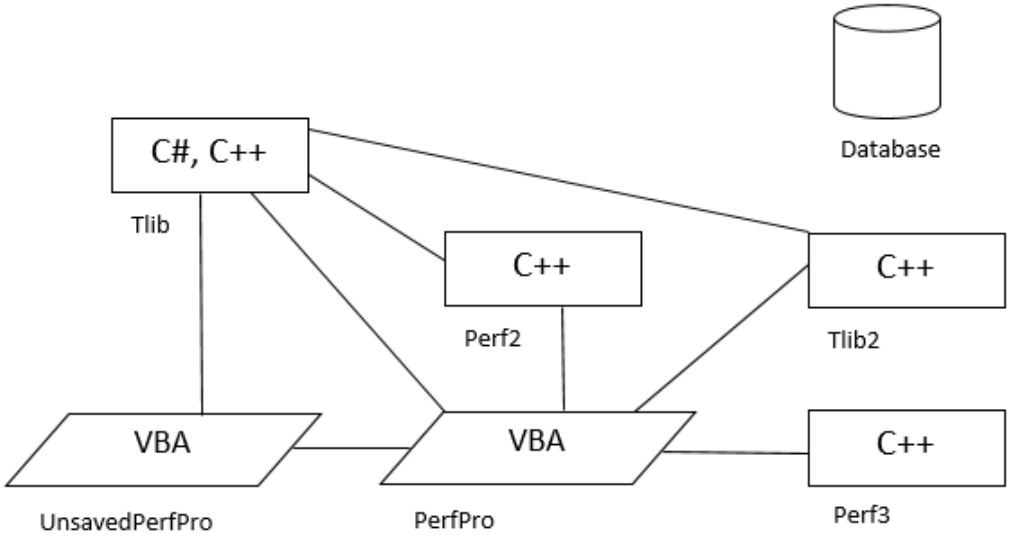


Figure 18. Current architectural model.

This model has the Excel parts marked as parallelograms. Unsaved PerfPro is the representation of the user interface and PerfPro is the representation of most of the VBA code. PerfPro is in direct connection to Perf2, Perf3, Tlib and Tlib2. Perf3 is an interface to the other files. Tlib2 and Perf2 contain the data and functions for the engines and cooling systems. Tlib is the file that contains all the basic functions. All the information is stored in databases. UnsavedPerfPro, PerfPro, Perf2 and Perf3 reads data from the database. This architectural model has become too complex and slow and needs to be streamlined.

The old architecture is simplified in the picture above and in reality, it has much more small connections and jumps between the different blocks. Also, within the blocks there are several connections that all take time to go through when the program is calculating values for the user. This makes it extremely hard for new developers to figure out the program and find specific functions and features. All in all, the architecture is messy and slows down the calculation process unnecessarily much.

One major problem with the current architecture is that the user interface, UnsavedPerf-Pro, is dependent of Tlib and the database. This solution has worked in the past when the functions and features were limited. In the current state, there are too many functions and

features in PerfPro, and unnecessary jumps between files for simple calculations. The most simple calculations could be moved into either UnsavedPerfPro or PerfPro files and converted into VBA functions. This could help performance issues and make the calculations smoother.

There are steps that can be made to improve current PerfPro, and assist the eventual transition to the new architecture. Documenting and listing different features and knowing where in the code they can be found is a good place to start. Also, adding the new requirements in some capacity will further help the transition. Combining files to minimize jumps between the different files could be done. However, this would be eventually done in the next generation, so it would be doing the work twice.

One example of assisting the transition is to future proof the user interface. Future proofing the UI means that we make an UI that is compatible even for future versions. There are five steps to consider when designing a future proof UI. First, we need to ignore the current design trends. The focus should be on the end-users instead of design trends and fads. In some cases, the current trend can be beneficial for the future of the UI, but most of the time focusing on customer needs is much more beneficial. Secondly, maintaining end-users as top priority is crucial. Elimination the learning curve usually is better than a complex UI. Thirdly, flexibility and adaptability. By designing our UI to be flexible and adaptable, we basically future proof the UI. If something radical happens, in our case of PerfPro an engine is discontinued and a new model is developed, the UI must be able to adapt to the changes. Fourthly, we can optimize connections and flow. A future proofed UI always follows strict principles in flow and connections. We want our interface to be easy to use, fast and logical. Finally, attention to detail. Attention to detail will improve any and all design. End-users will not notice especially good UI, instead anything that does not work will be noticed immediately. Attention to detail is the best way to avoid errors in the UI. By following these five steps, we have a good chance our UI will stand the test of time. (Falk 2015)

A streamlined architectural model for the new generation of PerfPro is visualized in figure 19 below.

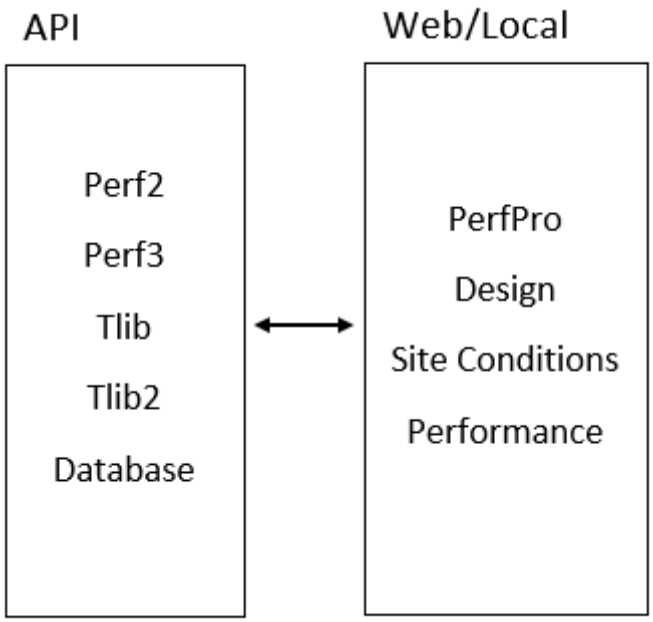


Figure 19. New architectural model.

This streamlined model has all 4 files combined with the database to form an application programming interface. The other side of this model is named Web/Local. This is for the new user interface. The new user interface platform can with this model be either online or offline. The similar PerfPro and all three stages of input/output data is listed here. All calculations and data are in direct connection to the API, where all the information and functions are stored.

The physical location of the API is still under consideration. The logical option for the API location would be online. There are several pros and cons with an online API. An online API would not need physical space on the end-user's computer and updating the API would be more convenient. As a downside, the end-user would always have to be connected to the internet when calculating performance numbers with PerfPro. The API model would make it easier to communicate with other Wärtsilä programs, such as APEX.

Stateless API could be a good option. Stateless API means that the server does not store the state of the client's sessions on the server side. Every request from the client to the

server contains all the necessary information needed to understand the request. The session state is then completely on the client side. A stateless API helps by deploying the users to multiple servers. All servers can handle any requests because of the session related dependency is no longer in play. Statelessness APIs reduces complexness by removing state synchronizing logic. Statelessness also makes it easier to cache. (Fielding & Taylor 2000)

This new architecture improves performance by being faster. The program will no longer have to jump between files to calculate and copy data. Also, this new architecture will help future developers update the code by being easy to locate the different parts of the code. Debugging and development of future functions and features will also be easier in a more streamlined architecture.

As for the requirements, this new architecture will in theory eliminate some problems with previous versions of PerfPro. For example, problems with PerfPro when other sessions are open and problems after recovering from sleep mode should be eliminated with the new architecture. The API model was a logical continuation for PerfPro. The API model is fairly flexible and can have an online or partially online solution. Also, it is much easier for the developers in the future to find and fix bugs that may occur.

6.3 Platform speculation

We now know the requirements and architectural model for the next generation of PerfPro. The final step in the plan for the new software is deciding on a platform. Deciding on a platform in this context means that we need to decide a path to take for the future of PerfPro. There are countless options to take in choosing a platform. Here we have narrowed the options down to three feasible options to consider for the future and continuity of PerfPro. These options are:

- 1. Continue using Microsoft Excel as user interface and continue to use C# and C++ files such as Tlib as calculation data.

- 2. Continue with Microsoft Excel and rework the data calculation files into VBA files within the program.

- 3. Remake PerfPro user interface to an .exe C# program. Utilizing already existing C# calculation files such as Tlib.

These three options are presented in the following subchapters.

6.3.1 Option one

The first option is considered as the “change nothing and continue way of working” – option. This is in practise the way PerfPro is used, developed, and updated currently. New requirements and needs are currently updated into the program by adding new features or fixing existing ones into the Excel user interface or C# and C++ files. Advantages with this option is that we keep the familiar user interface and way of working for the users. This is also the cheapest option in terms of development cost of new systems and programs. As for the disadvantages, this option probably does not fix most of the problems with PerfPro. The constant jumping from VBA to C# and C++ code in the calculations make the program slow and unreliable when it comes to crashes and such.

Initially nothing will change except for some major bugs found in the survey will be fixed. But when time goes by and new features will be added, the development of PerfPro should strive for the API model. This option would start as its own option but eventually evolve into option two. The difference is that option one is a slower transition to the next generation while option two would basically build a new program from the start and when ready, PerfPro would have its next generation. To visualize the architecture, we start from the current situation in figure 18, and step by step strive to develop the program to the model in figure 19.

6.3.2 Option two

The second option also benefits from the user interface staying in Excel. This option requires more work than the first because we need to rewrite all the calculation files into VBA code and include them in the same PerfPro file. Even if we restructure the program to fully be within one Excel and VBA file, there is a chance that this will not fix all the problems with PerfPro. This will streamline the program but because of the number of features and code, PerfPro can still be slow, messy, and clumsy in Excel.

This option requires a lot of work for the rewriting and reconstructing of the code. This can be done quicker or slower depending on how much resources we want working on this. Less resources are more like option one, with slowly striving for a new program entirely within VBA and Excel. A faster approach is to dedicate more resources to constructing the new program as soon as possible. This will most likely hinder the development and maintenance of the old program that will be in use while the new program is constructed.

6.3.3 Option three

The final option requires the most work. For this option, we need to rework the entire user interface to a new platform. The most feasible platform would be to create an .exe program in C# with Visual Studio programming environment. This is basically starting from scratch and requires much work. However, we can reuse some of the existing calculation files and databases. The biggest challenge is to make the user interface as familiar as possible for the end users so that they can start using this new PerfPro as quickly as possible. This option will fix most of the problems with PerfPro but is the most challenging to implement. Some features, such as copying data from Excel PerfPro directly to other spreadsheets, will be hard to implement in a natural way for the end users. This is possible to implement but requires some new methods that the end users must learn in order to use.

The reason for choosing C# as the programming language is that all the current developers of PerfPro are familiar with C# and Visual Studio as IDE. The logical choice came down to either C# or C++ because most of the current functions and features in the different Perf and Tlib files are written in either C# or C++. Visual Studio has a grate programming environment for C# with graphical elements supported. This is the main reason for choosing C# as the new programming language.

This program would be in three parts. First there is the use interface, The UI should visually be as close as possible to the current state of PerfPro UI in Excel. It does not need to be in a spreadsheet format, but the different inputs and outputs should be in the same order and phases. The second part is the calculation engine. This would be part of the API, and in difference to the current state, in one file. This seems challenging at first, but because most of the code for calculations already exists, it is a matter of combining all these features and functions. The final part is the database. The database for engines already exists and is in a good state. The only thing to do is to communicate with the database to get access to the variables and values from the database. Below is figure 20, that is a simplistic visualization of these parts.

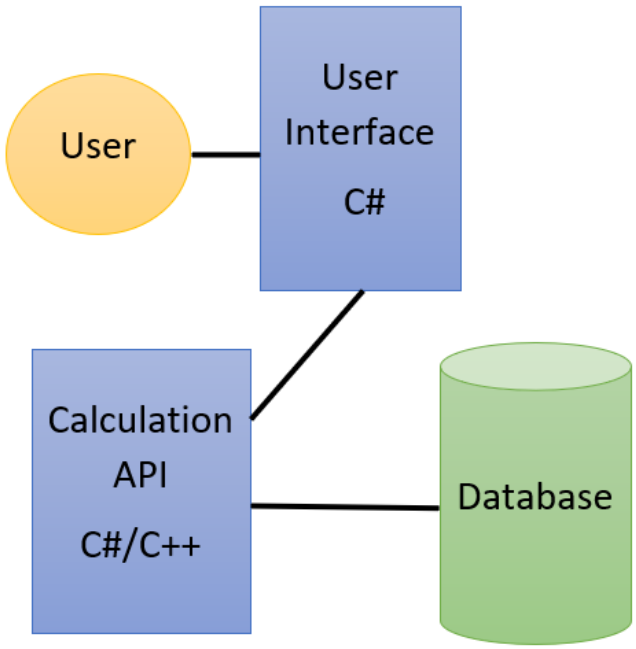


Figure 20. Simplistic visualization of new software.

The UI would be in its own entity while the calculation files would be constructed of the different files within the API. This calculation engine is in direct contact with the database. The API and database would be online for easy maintenance and updates, while the UI would be a downloadable package. For offline work there should be an option to download the API so that it is possible to make calculations, but the program will then possibly not be up to date.

All options are in some capacity feasible. None will eliminate all problems with PerfPro, but all will fix some problems. The options are roughly in order from least work to most work and least fixes to most fixes. Every option has its advantages and disadvantages. It is up to the developers to figure out which option is the best and eventual continuity for PerfPro.

7 CONCLUSIONS AND DISCUSSION

Software architecture is an important part of software development. Documenting and mapping programs help new users and developers to understand the features and intended purpose of a program. Documenting and mapping software are a time-consuming task but is rewarding if done correctly. Having a good plan in the development stage brings clarity and stability to program development.

Unfortunately, the power plant simulation program had next to no documentation and architectural mapping done to it as of the start of this thesis. A part of this thesis was to make this documentation and map the current state of the program. The program is quite large, and many features of the program is not used anymore. This is the reason why the documentation and mapping were done based on user experience instead of reading through the code.

The user experience data was collected by a user survey. The survey was sent through email groups, which contained employees known to work with PerfPro. The survey was up for about two weeks just after summer vacations. Around 40% of all PerfPro users responded to the survey. The first significant data we got was a general opinion on PerfPro. This grade had an average of about 3,8 on a scale from 1 to 5. The survey had direct questions of preference between mobile versus personal computer, online versus offline and light versus heavy version. The only clear opinion was that personal computer is much more desirable than a mobile version, so the idea of a mobile PerfPro was put on hold for now. Online versus offline and light versus heavy version was much closer, but the scale tipped a bit towards offline and heavy versions.

The rest of the survey had options to write open answers to different things like Excel as UI and what works well and what does not. These open answers gave much needed feedback from the users and we found many things that should be fixed in the next generations. These open answers were the basis of the requirements for next generation. We also redesigned the software architectural model to an API model to eliminate some problems related with slowness and crashing with the old model.

Next, we figured out possible solutions for next generation. This thesis found five different paths for the continuity of PerfPro. The first two are not very feasible options as the first is to use an existing simulation program as substitute for next PerfPro, and the second option is to outsource the development of the next generation of PerfPro. The first option falls short because PerfPro needs too much Wärtsilä specific data to operate properly. The second option is not feasible because of the cost of outsourcing such a large project. Also, by outsourcing, Wärtsilä gives out crucial data to outside developers.

The next three solutions are the feasible ones. First there is the continued use of Excel as UI and continuing development as before. This is the easiest solution and has the lowest impact on the problems. By continuing the way of work, we have currently we can fix some problems with PerfPro but probably not all problems. The second feasible option is to redevelop an Excel based UI with the new architectural model and try to fix the problems from scratch. We can still reuse some code from the different Perf and Tlib files, but much work is needed for this option. The final option is to use a completely new platform for the UI. This could be done by developing an .exe program in either C# or C++ and reuse most of the features and functions from the various existing Perf and Tlib files.

The future of this project is to start making prototypes of the feasible options. The initial prototypes are the most important because based on these prototypes we choose the path that the next version of PerfPro will follow. When we have decided on an option, we can do even more prototypes of that option and again choose the path to follow. We do as many prototypes as possible until we have a solution that we feel confident to develop the final product.

This thesis starts with the theoretical part. Everything from software architecture in general to software patterns and research methods. This thesis also includes a use case for current PerfPro that can be used to teach new users the basics of PerfPro. In addition, we figured out that there are similar power plant simulation programs already existing but because Wärtsilä engines are in a pivotal role in PerfPro, we cannot use these existing simulation programs as the next generation.

The survey got us concrete data to work with. The user experiences with PerfPro as a tool for the end users had much new information we needed to figure out the needs, requirements and wishes for the future and continuity of PerfPro.

The research done in this thesis had an impact on the research needs. The use case documentation can be used when learning about the current state of PerfPro. Also, the survey was done for the first time in a while and we learned about errors and bugs in the current state, as well as comments and wishes for future versions. As for the eventual next generation, the process is still ongoing. We are fairly certain that a platform update to a new platform is required. The prototypes of the different options will make it clear which direction PerfPro will have in the future.

As stated earlier in chapter 3, there has not been very much similar research done about this subject. Power plant simulation in general is a very broad subject and all previous research concentrate on a specific aspect or method, for example machine learning. There was some research done about platform switching. Much inspiration was taken from these kinds of previous researches and the choice of API model and C# programming language had an impact from the earlier research.

The results of this thesis do not have a great impact on the academic side. Some aspects of surveying software users and the decisions of questions can be used in future research. Also, the choice of use case in documentation instead of blatant code analysis is a different aspect to software documenting. In practice, this thesis has more use. Wärtsilä can use the use case as documentation for PerfPro and use it for future research and to introduce the program to new users. The survey resulted in much needed user feedback from many different aspects of users. And finally, the suggestions on the next generation of PerfPro can have great impact. We need to wait and see what the prototypes potentially can improve in PerfPro.

As conclusion, this thesis had much planned aspects to cover that got scrapped when we realized how big this thesis already had gotten. For example, there was a plan to include the first prototypes as a part of this thesis. The documentation of current PerfPro, survey

and plan for next PerfPro were all the parts of the bigger project that was included in the end. All in all, the future of PerfPro is still up in the air and hopefully this thesis has had some impact in clearing the continuity of PerfPro.

BIBLIOGRAPHY

Bass, L., Clements, P., Kazman R. (1998). *Software Architecture in Practice. 7. Edition*. Boston (MA): Addison Wesley Longman, Inc. ISBN 0-201-19930-0

Bhat, A. (2019). *Survey Data Collection: Definition, Methods with Examples and Analysis.* Web-article [online] Available: https://www.questionpro.com/blog/survey-data-collection/

Brown, S. M., Wild, N., Carlin, J. D. (1996). *A software maintenance process architecture.* Publication in Proceedings of 9th Conference on Software Engineering Education.

eToro (2017). *eToro's quick and simple guide to Blockchain.* [online] Available: https://www.etoro.com/blog/trading-essentials/etoros-quick-and-simple-guide-to-blockchain/

Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software. 1st Edition.* Boston (MA): Addison Wesley Longman, Inc. ISBN 978-0321125217

Falk, C. (2015). *Future-Proof Your UI Design Using These 5 Rules.* Web-article. [online] Available: https://www.altia.com/2015/07/31/future-proof-your-ui-design-using-these-5-rules/

Fielding, R., Taylor, R. (2000). *Architectural styles and the design of network-based software architectures* (Vol. 7). Irvine: University of California, Irvine.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design Patterns, Elements of Reusable Object-Oriented Software. 1. Edition.* Boston (MA): Addison Wesley Pearson Education. ISBN: 0-201-63361-2

GIE (2012). *GIE Position Paper on impact of including Methane Number in the European Standard for Natural Gas.* Gas Infrastructure Europe Research Paper. Ref. 12GIE127 November 2012

Graca, H. (2017). *Layered Architecture.* Web-article. [online] Available: https://herbertograca.com/2017/08/03/layered-architecture/

Hevner A. R., March S. T., Park J., Ram S. (2004). *Design Science in Information Systems Research.* Management Information Systems Quarterly Publication.

Hevner. A. R., Gregor, S. (2013). *Positioning and Presenting Design Science Research for Maximum Impact.* Management Information Systems Quarterly Publication.

IEEE (2000). *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. [Standard] IEEE 1471-2000

Jansson, K. (2019). *PerfPro.* Wärtsilä internal documentation on PerfPro.

Kim, J., Kang, S., Ahn, H., Keum, C., Lee, C. (2018). *Architecture reconstruction and evaluation of blockchain open source platform.* Publication in ICSE ’18: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, pages 185-186.

Koskimies, K., Mikkonen, T. (2005). *Ohjemistoarkkitehtuurit*. *1. Edition.* Helsinki (FIN): Talentum Media Oy. ISBN 952-14-0862-6

Kruchten, P. B. (1995). *The 4+1 View Model of Architecture.* Scientific article. [online] Available: https://www.researchgate.net/profile/Philippe_Kruchten/publication/238381956_A_41_View_Model_of_Software_Architecture/links/570ec85208aee76b9dadff07/A-4-1-View-Model-of-Software-Architecture.pdf

Lucidchart (2019). *What is a Flowchart?* [online] Available: https://www.lucidchart.com/pages/what-is-a-flowchart-tutorial

Nyqvist, K. (2017). *PerfPro Short Intro*. Wärtsilä internal documentation on PerfPro.

Peffers, K., Tuunanen, T., Rothenberger, M. A., Chatterjee, S. (2007). *A Design Science Research Methodology for Information Systems Research.* Publication in Journal of Management Information Systems, Volume 24 issue 3, Winter 2007-8, pages 45-78.

Rosado-Tamariz, E., Zuniga-Garcia, M., Santamaria-Bonfil, G., Batres, R. (2019). *A machine-learning approach to speed-up simulation towards the design of optimum operating profiles of power plans.* Publication in IEEA ’19: Proceedings of the 8th International Conference on Informatics, Environment, Energy and Applications, pages: 194-199.

Stanley, L. (2019). *What is Software Prototyping? – Definition, Models & Tools*. [Web-article] Available: https://study.com/academy/lesson/what-is-software-prototyping-definition-models-tools.html

Steag (2019). *Ebsilon Professional, The Planning Tool for the Power Plant Process*. [online] Avalable: https://www.steag-systemtechnologies.com/uploads/pics/Brochure_EBSILON_The_planning_tool_for_the_power_plant_process_eng_01.pdf

Techopedia Inc. (2019). *Proof of Concept (PoC)*. [Definition] Available: https://www.techopedia.com/definition/4066/proof-of-concept-poc

Tutorialspoint (2019). *Design Patterns*. [online] Available: https://www.tutorialspoint.com/design_pattern/

Tutorialspoint (2019). *UML – Use Case Diagrams*. [online] Available: https://www.tutorialspoint.com/uml/uml_use_case_diagram.htm

Van Aken, J. E., Romme, G. (2009). *Reinventing the Future: Adding Design Science to the Repertoire of Organization and Management Studies.* Publication in Journal of Organization Management.

Wärtsilä (2019). Company homepage. [online] Available: http://www.wartsila.com/about

Xu, R., Yan, W. (2019). *Continuous Modeling of Power Plant Performance with Regularizes Extreme Learning Machine.* Publication in 2019 International Joint Conference on Neural Networks, Budapest, Hungary, pages 1-8.