



FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

SolCelSim – A Comsol App for Charge Transport in a Multilayer Solar Cell

Master's in Informatics Engineering

Internship Report

João André Vieira

E-Mail: Uc2013136370@student.uc.pt

Adviser: Bruno Cabral

Date: 01/07/2019

Abstract

Scientific Simulation software can often be daunting, even for experts of the associated scientific area, as they usually have complex interfaces with a very large number of features. At the same time, some ideas cannot be realized by someone without programming knowledge, and require a lot of repetitive and menial work.

More specialized applications can be used to make the process of using scientific simulation software simpler, while also allowing the use of more complex features with minimal work.

This project consists of the development of an application for Comsol Multiphysics. It focuses on the simulation of a multi-layer PEC solar cell used for hydrogen production. Features that researchers need in this kind of simulation are made available, while Comsol features that aren't useful for this scenario are excluded. The result is a user-friendly application that can be used by any researcher working on this area of physics.

Keywords

Comsol Multiphysics

Application Builder

Multilayer PEC Solar Cell

Hydrogen Production

Contents

List of Figures.....	9
List of Tables.....	10
List of Terms and Acronyms	12
1. Introduction.....	14
2. Related Work.....	16
2.1 Similar Software	16
2.2 Framework.....	22
2.2.1 Comsol Multiphysics	22
2.2.2. Comsol Application Builder.....	22
2.3 Model.....	24
3. Methodology and Planning.....	25
3.1 Methodology.....	25
3.2 Work Plan.....	26
3.3 Development Schedule	31
3.4 Risk Assessment.....	36
4 Requirements and Architecture	38
4.1 Functional Requirements	38
4.2 Non-Functional Requirements	40
4.2.1. Usability.....	40
4.2.2. Extensibility	40
4.2.3. Maintainability.....	40
4.2.4. Accuracy	40
4.3 Architecture	41
5. Interaction Design	47
5.1 Interface Design.....	47
5.2 Principles	47
5.3 Design Walkthrough.....	47
5.4 Use Case Diagram.....	48
5.5 Mockups	49
6. Final Interface.....	54

7. Testing and Quality Assurance	59
7.1 Alpha Testing	59
7.2 Acceptance Testing	59
7.3 Accuracy Testing/Boundary Value Testing	60
7.4 Exploratory Testing.....	60
7.5 Negative Testing.....	61
7.6 Usability Testing	62
7.7 Others	64
8. Challenges.....	65
Conclusion	67
Future Work.....	67
Final Comments	67
Bibliography.....	69
Attachment 1 – Comsol Explanation	70
Attachment 2 – Solar Panel Software	77
Attachment 3 – Functional Requirements	82
1.1. Layer Stack	82
1.2. Layer Parameters.....	83
1.3. Recombination Model.....	84
1.4. Boundary Conditions	85
1.5. Meshing.....	85
1.6. Computation.....	85
1.7. Experimental Data.....	88
1.8. Parameter Fitting.....	88
1.9. Global Parameters.....	89
1.10. Model Save and Load.....	89
1.11. Report	89
1.12. Heterointerfaces.....	90
Attachment 4 – Server Architecture	91

List of Figures

Figure 1 - GPVDM.....	18
Figure 2 – PC1D	18
Figure 3 – SCAPS	19
Figure 4 – AFORS-HET	20
Figure 5 - Trello	26
Figure 6 - Initial Work Plan.....	27
Figure 7 - End of First Semester Work Plan	28
Figure 8 - End of Second Semester Schedule.....	29
Figure 9 – Initial Development Plan.....	33
Figure 10 – Final Development Plan	34
Figure 11 – Sprint 0 Burndown	35
Figure 12 – Remaining Sprints Burndown.....	35
Figure 13 – General Architecture.....	41
Figure 14 – Diagram with Software Components	43
Figure 15 – Application Flow Diagram.....	45
Figure 16 – Application Flow Diagram (Solver)	46
Figure 17 – System Use Case Diagram.....	48
Figure 18 – Main Menu.....	49
Figure 19 – Layer Parameters.....	50
Figure 20 – Recombination Type.....	50
Figure 21 – Recombination Parameters	51
Figure 22 – Solve	51
Figure 23 – IV Inputs.....	52
Figure 24 – Results	53
Figure 25 – Method and Termination	53
Figure 26 – Main Menu.....	54
Figure 27 – Results Screen.....	55
Figure 28 – Contact Properties.....	56
Figure 29 – Method and Termination	56

Figure 30 – IV Solver Options	57
Figure 31 – Parameter Sweep Options	57
Figure 32 – Layer Parameters.....	58
Figure 33 – Model Builder.....	70
Figure 34 – Application Builder.....	71
Figure 35 – Layers.....	72
Figure 36 – Add Layer	73
Figure 37 – Getting two parameters	73
Figure 38 – Duplicating a default layer.....	73
Figure 39 – Adding Distance Value	74
Figure 40 – Setting Endpoints	74
Figure 41 – Adding New Doping Model.....	75
Figure 42 – Domain	75
Figure 43 – Build.....	76
Figure 44 – Updating Global Parameters.....	76
Figure 45 - PV Analyser.....	78
Figure 46 - PV PanelSim.....	78
Figure 47 – SES.....	79
Figure 48 - System Advisor Model.....	80
Figure 49 – Solar Pro IV Curve Graph	81
Figure 50 - Solar Cell Layers	82
Figure 51 – Connections from User to Server	91

List of Tables

Table 1 – Related Work Comparison	21
Table 2 – End of First Semester Work Plan.....	28
Table 3 – End of Second Semester Work Plan.....	30
Table 4 – Risk Assessment.....	36
Table 5 - Functional Requirements.....	39
Table 6 – Accuracy Testing Example	60

Table 7 – Negative Testing Example.....	61
Table 8 – Task Time to Complete per User	63
Table 9 – Questionnaire Results Rankings	63
Table 10 – Related Work Comparison.....	80
Table 11 - Experimental Data.....	88

List of Terms and Acronyms

a-Si	Amorphous Silicon
CIGS	Copper Indium Gallium Selenide
CV	Capacitance-Voltage
GUI	Graphical User Interface
IPCE	Incident Photon to Current Efficiency
IV	Current-Voltage
PEC	Photoelectrochemical
SPICE	Simulation Program with Integrated Circuit Emphasis
TE	Thermal Equilibrium

Agile	Software Development approach that incentivizes flexibility and continuous improvement
Auger Recombination	Non-radiative process involving three carriers ¹
Boundary-Value Analysis	Software testing technique that tries to represent boundary values in a range
Burndown Chart	Graph showing ratio between work left to do and time left
Control-Flow Testing	Testing strategy that carefully selects a set of test paths ²
Direct Recombination	Recombination through emission or absorption of a photon
Electrolysis	Chemical decomposition using electrical current
Impact Ionization	Process in which energetic charge carriers lose energy to create others
Nyquist Plot	Parametric plot used to assess stability of a system
Ohmic Contact	Junction between conductors with linear IV curve
Parameter Fitting	Estimation of parameters using sample data
Photovoltaic Panels	Panels designed to convert light into electricity
Quasi-Fermi Level	Population of electrons when displaced from equilibrium
Schottky Barrier	Electron barrier formed at metal-semiconductor junction

¹ <https://www.sciencedirect.com/topics/chemistry/auger-recombination>

² <https://www.cs.drexel.edu/~spiros/teaching/CS576/slides/2.control-testing.pdf>

Scrum	Agile methodology that divides work into iterations called “sprints”
Thermionic Emission	Thermally induces flow of charge carriers
Trap-assisted	Recombination when an electron falls into a certain energy level ³
Usability Heuristics	Commonly recognized usability principles

³ <http://ecee.colorado.edu/~bart/book/recomb.htm>

1. Introduction

This report will detail the work done for the course of Internship/Dissertation at the University of Coimbra, within the Intelligent Systems specialization of the Master's in Informatics Engineering.

With growing concerns regarding global warming and the unsustainability of fossil fuels, research regarding cheaper and more efficient use of renewable energy has been intensifying. However, performing physical experiments is expensive, due to the need to acquire proper material and equipment, making computer simulations especially important. Not only do they allow researchers to quickly test and compare different parameters, obtaining detailed results every time; they are also able to perform automatic parameter fitting.

Currently, researchers rely on generic computing software like Wolfram Mathematica and Matlab. While these programs are very powerful, demand for more specialized software that offers a deeper focus on a smaller number of features is growing.

One of these specializations is the production of hydrogen using solar energy. The application developed during this internship is aimed at filling the demand for that type of simulation, specifically one that uses multi-layer Photoelectrochemical (PEC) solar cells. The objective was to create a powerful desktop application that could fulfil the needs of any researcher in the area, without requiring any programming knowledge.

During this internship, the Comsol Multiphysics software was used. This is a physics simulation program developed by Comsol Inc. in Sweden during 1986 that has been frequently updated since. Application Builder, a feature of Comsol Multiphysics, was the framework used. It allows for a quick implementation of standard Comsol Multiphysics features, while also facilitating the creation of new features using the JAVA programming language.

This project was developed at the University of Žilina, at the satellite location Inštitút Aurela Stodolu in Liptovský Mikulás, Slovakia. It was done as part of the Erasmus+ Internship Program.

It comes in the sequence of research regarding PEC solar cells done by Dr. Peter Cendula, who served as the client for this project, at Zurich University of Applied Sciences; and work done on Comsol by Matúš Vaňko, at the University of Žilina. The former research focused on the more theoretical aspects of the use of PEC solar cells for hydrogen production. The latter work is more practical, exploring the creation of a GUI that allows users to simulate this situation.

Current software used for simulating solar cells often requires a thorough understanding of the application itself, on top of the necessary knowledge about solar cells. The purpose of this project was to eliminate that barrier, making researcher's jobs easier while still providing a very powerful specialized simulation tool.

This internship was very valuable as an Informatics Engineering internship. Although learning about aspects of Scientific Simulation was important, the most relevant part of this project was the knowledge that could be gained related to Software Development. The necessity of writing good code, with proper documentation and testing, allowed for the opportunity to apply knowledge gained through the Informatics Engineering course, while also learning new things about Software Development such as, for example, good UI design.

This report details every aspect of the creation of this app that is considered relevant. The *Related Work* section shows the research done about previous projects within the field and other related simulation apps.

In *Methodology and Planning*, information about how the development process was organized is laid out, with the schedules for both the complete work and the development phase; along with the chosen software development methodology and the reasons for that choice. The *Requirements and Architecture* sections offer information used during the development and testing phases of the project.

The following sections offer a more concrete look at the ideas behind the design of the application. *Interaction Design* and *Final Interface* explain the choices made regarding how the user is able to interact with the application, and how the application looks. Finally, some space is reserved for final remarks.

2. Related Work

This chapter serves not only to look at and analyse existing software in the area of solar cell simulation, but also to explain in detail what the Comsol framework consists of in the scope of this project. The initial model that this project is based on is also examined.

2.1 Similar Software

Environmental concerns have resulted in a more urgent need for research regarding renewable energies. One of this decade's developments have been in PEC (photoelectrochemical) water splitting. This concept consists of producing hydrogen from water using photoelectrochemical materials and sunlight. Individual molecules of hydrogen and oxygen are separated through this process. Usually, panel systems similar to photovoltaic panels are used. With continued research, production costs are being lowered and efficiency is being improved continuously⁴.

Recent research on photovoltaic-electrolysis has been motivated mostly by the necessity to reduce the cost of this type of hydrogen production [Jia et al., 2016]. Currently, the cost of fossil fuels is still generally lower; but with a decrease in material costs and an increase in production efficiency, this scenario could be inverted in the future. Solar-to-Hydrogen (STH) is a way to measure the efficiency of the system, with recent work showing up to 30% efficiency. These results show a huge improvement (that hasn't been surpassed since) over the 18% STH efficiency achieved earlier [Ager et al., 2015] the 22% achieved in Bonke et al, or the 24.4% in Nakamura et al.

Solar energy is defended as being the “main energy system of the future”, with “power intermittency, grid flexibility, and surplus electricity” being the main obstacles [Delgado E et al., 2018].

In the research part of this project, it was also important to look at a previous paper by the client for this project [Cendula et al., 2014]. It defends the use of computer software for simulation of PEC solar cells, over the previous method of jumping directly from sketching into physical experiments.

When designing systems and devices for solar energy conversion, some guidelines are necessary. In Dumortier et al., guidelines that optimize “technical, economic, sustainability, and operating time constraints” have been designed with the help of a simulation platform created by the research team. With this simulation platform, it was verified that the devices that optimize efficiency, cost and energy demands are “device types utilizing high irradiation concentration, as well as expensive photoabsorbers and electrocatalysts”. This paper also alerts about the importance of replacing components with a frequency appropriate for the device being used (too early means higher costs, too late means loss in efficiency due to degradation).

The last few years have also seen the release of new solar cell simulation software⁵. One of these examples is General-purpose Photovoltaic Device Model (GPVDM⁶), designed to simulate, for example, organic solar cells, crystalline silicon solar cells, a-Si solar cells and

⁴ <https://www.energy.gov/eere/fuelcells/hydrogen-production-photoelectrochemical-water-splitting>

⁵ <https://www.linkedin.com/pulse/7-most-popular-solar-pv-design-simulation-software-eslam-allam/>

⁶ <https://www.gpvd.com/>

Copper indium gallium selenide (CIGS) solar cells. Much like the project described in this report, ease of use was also a focus during the development.

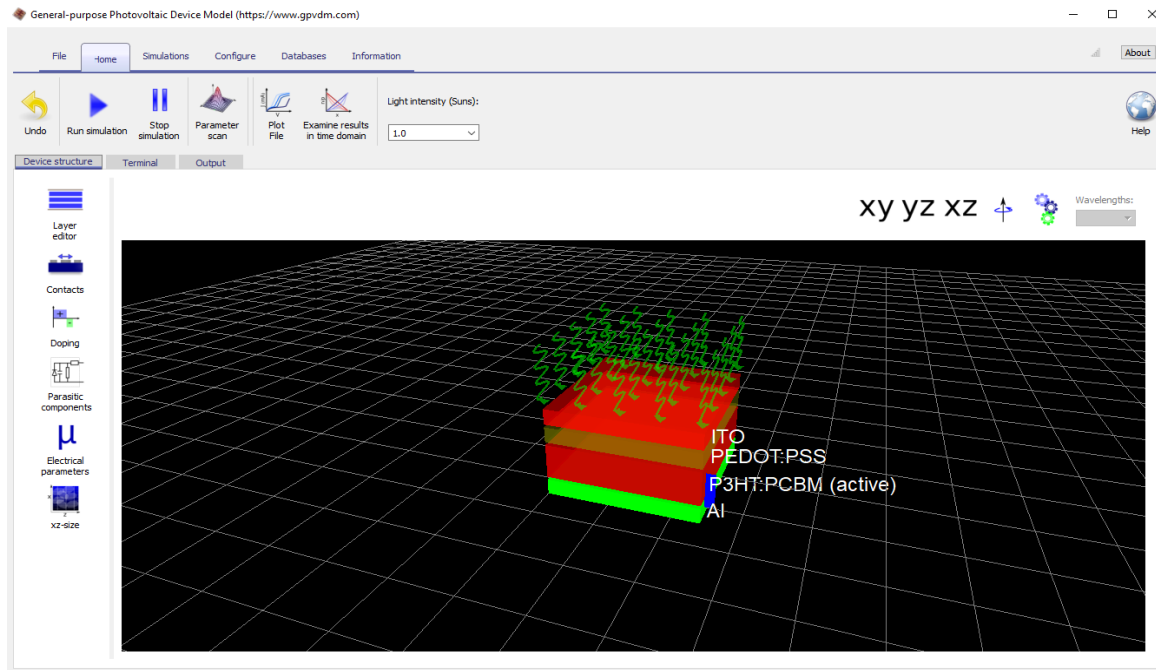


Figure 1 - GPVDM

PC1D⁷ is one of the more commonly used solar cell modelling programs. This program was developed in Australia, at the University of Sydney, in 1982. It can be used to simulate the performance of new devices and is available for free, along with its source code. Its graphical user interface allows users to begin modelling quickly and easily. The user can change a variety of component parameters, including material type, thickness and doping type. As parameters are changed, the one-dimensional schematic for the device changes accordingly, allowing the user to directly see the effect of his changes.

This software is currently the industry standard for solar cell simulation.⁸

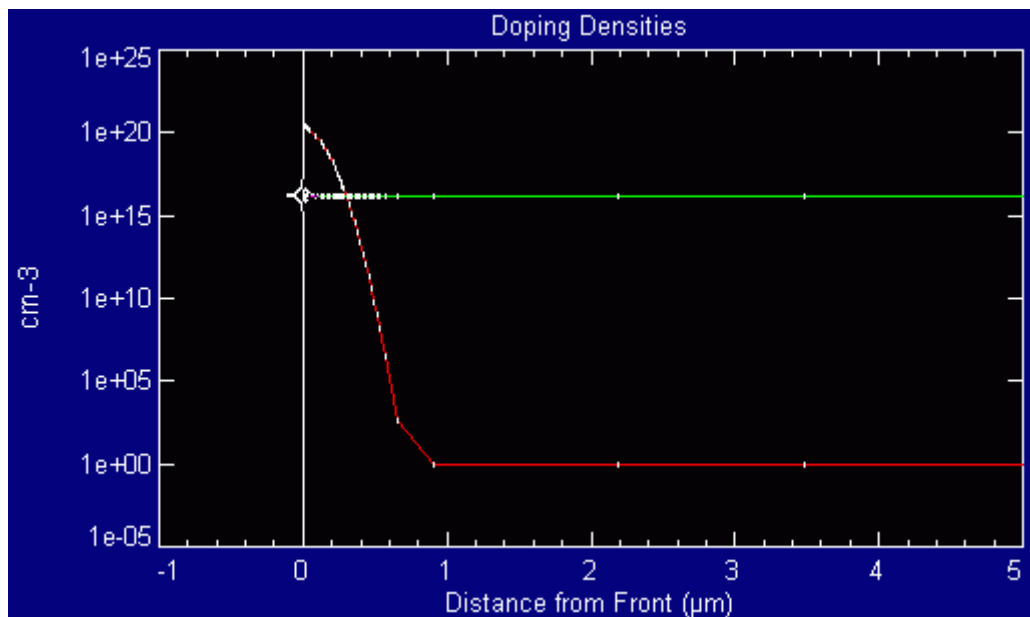


Figure 2 – PC1D

⁷ www.pv.unsw.edu.au/info-about/our-school/products-services/pc1d

⁸ <http://pv-i2e.mit.edu/>

SCAPS⁹ is another one-dimensional simulator, created at the University of Gent. It can display up to seven semiconductor layers, three different recombination mechanisms and multiple contact functions, among other features. It features a script language that facilitates initializing the program with the right internal variables. Like the previous example, designing a simple and user-friendly interface was also a priority in the development of this software.

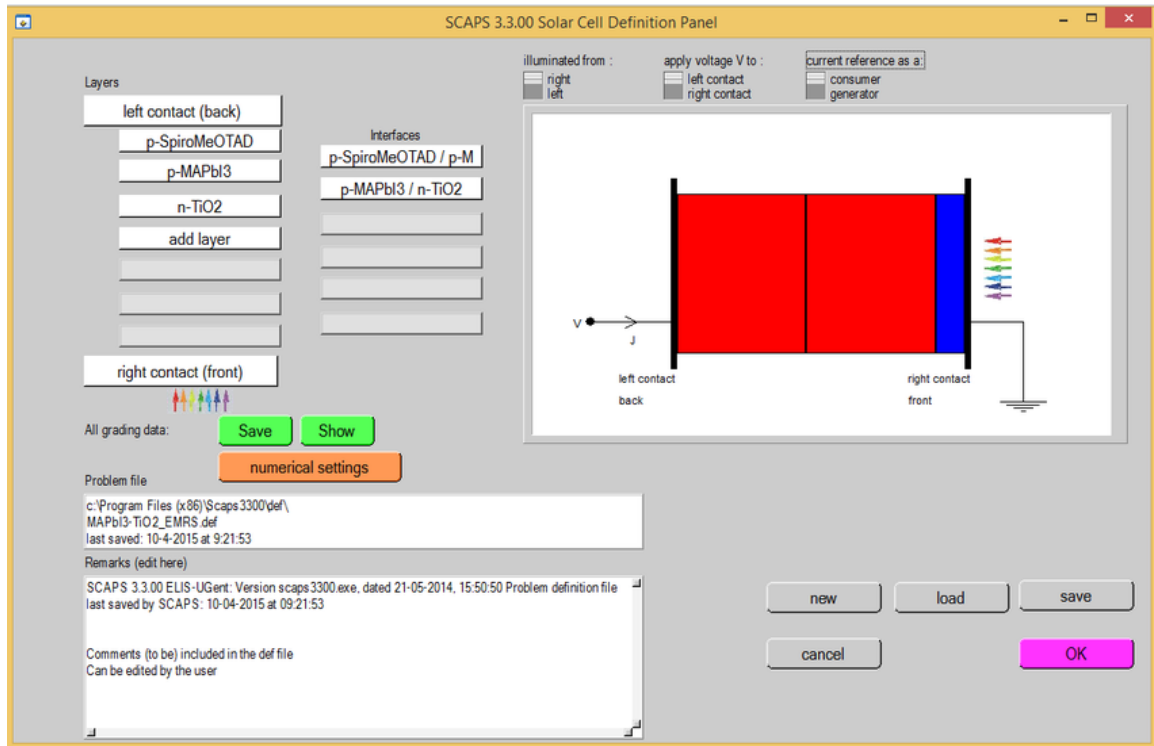


Figure 3 – SCAPS

AFORS-HET¹⁰ is a public simulation tool that can be used to simulate multiple layers, with the ability to pick a variety of boundary conditions. Different characterisation techniques such as IV (current-voltage), impedance and capacitance have also been implemented. The user interface allows users to compare different generated simulations.

⁹ <http://scaps.elis.ugent.be/SCAPSinstallatie.html>

¹⁰ https://www.helmholtz-berlin.de/forschung/oe/ee/si-pv/projekte/asicsi/afors-het/index_en.html

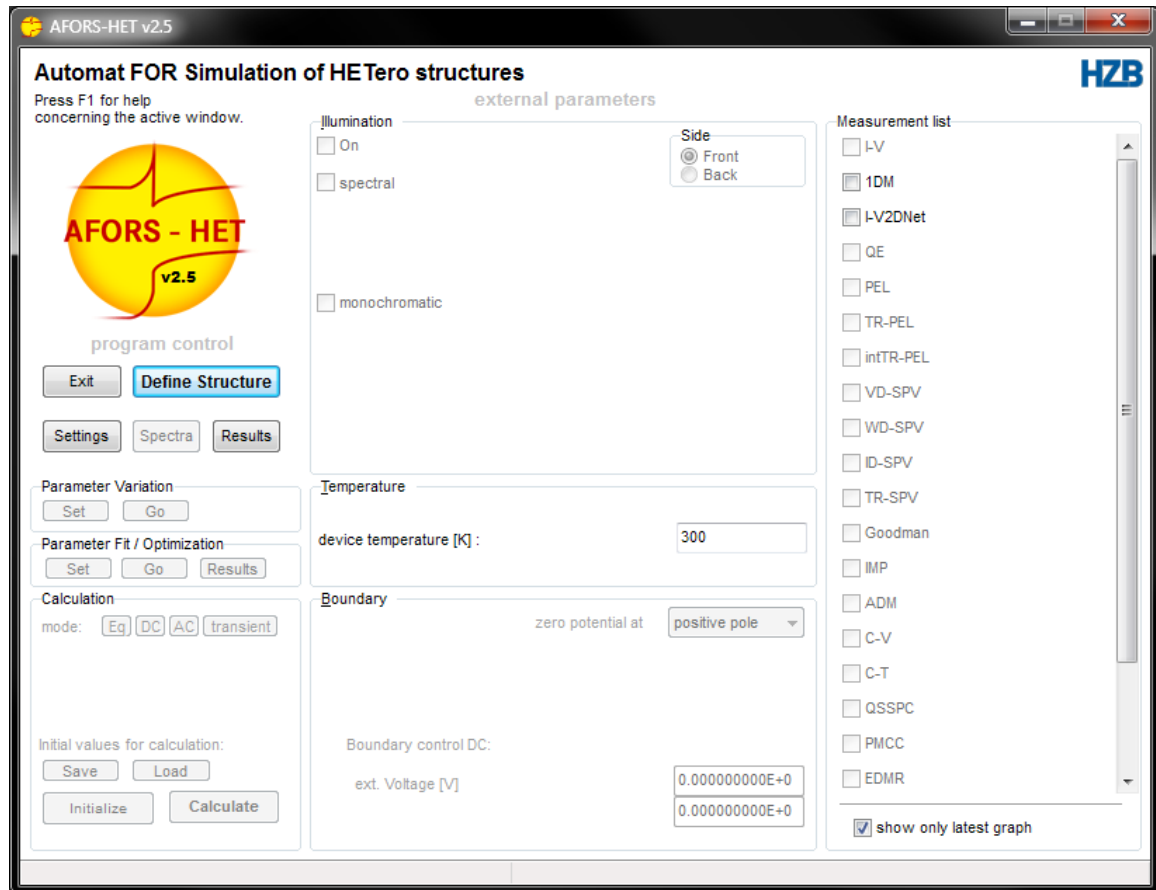


Figure 4 – AFORS-HET

The reason why the software developed during this internship should stand out over the applications listed in this section, is the focus on both PEC and PV solar cells, instead of just PV solar cells. PEC cells have the ability to perform photon conversion and electrochemical reaction on the same device¹¹. They are, however, more recent, and still not able to be commercialized. As such, the need to be able to simulate them is particularly important. Furthermore, researchers from any University with a Comsol licence will be able to use the application.

¹¹<https://dspace.library.uu.nl/bitstream/handle/1874/362927/Master%20thesis%20PEC%20versus%20PV-E%20a%20future%20potential%20comparison%20-%20W.A.%20de%20Jong%20-%20Energy%20Science%20-%202023-2-2018.pdf?sequence=2&isAllowed=y>

Table 1 shows a comparison between the simulation software examined in this section. The last line shows the features of this project, for the sake of comparison:

Software Name	Price	Graphics	Multi-Layer	Recombination Types
GPVDM	Free	3D	Yes	Two
PC1D	Free	1D	No	Two
SCAPS	Free	1D	Yes	Three
AFORS-HET	Free	1D	Yes	Three
SolCelSim	Free	1D	Yes	Five

Table 1 – Related Work Comparison

From making this analysis, a few ideas were taken. For example, the GUI design for this project took inspiration from GPVDM, as its general layout fit into the objective of this project. The position and size of the Layer Plot was one of the ideas that was used for the user interface, along with the interface for layer, contact and parameter editing. Like the observed examples, this application was made available for free.

While most of the examples allow the simulation of multiple layers, most are limited in number. For example, SCAPS allows at most 7 layers at the same time. In comparison, our application can support up to 20 layers at a time.

One of the objectives of this project was to allow users to have more possibilities regarding how they customize the simulated solar cell. The higher number of recombination type options is an example of this.

The way the user interacts with all these applications is similar: different parameters and settings can be selected and, according to those inputs, the user can see a graphical representation of the simulated system, along with the appropriate plots. The status of the simulation and various inputs can also be saved and loaded so that the user can continue interacting with the same simulation later. SolCelSim maintains this general idea and workflow.

Some examples of solar panel software were planned to be included in this section but were removed due to differing too much from the purpose of this project. Information about these examples can be found in the **“Attachment 2 - Solar Panel Software”** section of this report’s attachments.

2.2 Framework

2.2.1 Comsol Multiphysics

Comsol Multiphysics is a multi-purpose simulation software that allows the modelling of a large number of devices and processes in the areas of physics research and engineering. Different modules and add-ons can be added to complement the use of the software and allow access to new features. This is especially useful when working on more specialized areas of engineering and research that Comsol does not support natively.

When working with Comsol, the user's workflow is the standard one in physics modelling: defining the geometry of the model, the properties of the parts that constitute it, and changing values related to physics phenomena. In the end, the user obtains an accurate simulation of the generated model, along with information in the form of reports and graphical visualization.

Comsol offers the user a high degree of creativity, allowing them to define their own equations and expressions, and taking into consideration different operating conditions and physical effects. Since simulation software is often restrictive in what it allows users to specify, Comsol has an advantage in this regard compared to its alternatives. When a user makes a change to the geometry of a model, the associated physics settings are affected, and vice-versa.

Some additional features Comsol supports includes meshing of the model (both manual and automatic), different types of studies/analysis (both time-dependent and stationary), parametric sweeps and optimization studies.

2.2.2. Comsol Application Builder

In version 5.0, Comsol Multiphysics implemented a new feature called the Application Builder. This was the tool used to develop the project detailed in this report.

The Application Builder is a separate module from the Model Builder, where developers can create an entirely new application, based on the previously created model. It gives the developer access to numerous parameters within the model, tools to create a GUI, and the possibility of creating code that uses the Comsol API, allowing the developer a large degree of control over the application.

The Application Builder is divided into four parts:

Forms - Where the creation and editing of the user interface is allowed. Within each form, different types of elements such as input boxes and buttons can be added. In addition, a form can be contained within another form as a sub-form.

Declarations – Where the developer can set global variables and temporary files to be used by both the forms and the methods.

Methods – Where the developer can write Java code that controls how the application behaves, with access to the entirety of the Comsol API. The method editor also includes a few built-in methods that allow interaction with the user interface, such as enabling and disabling certain menu items and toolbars.

Library – Where specific files such as external images can be stored.

Every action the developer writes in the application builder can be linked to the model. For example, a button can be created to increase the size of the geometry, and the result of those changes can be seen as soon as that button is clicked.

The Application Builder was created to allow experts to make their model available to people within their organization. This way, people from different engineering disciplines, for example, can make changes to the simulation without having to understand the underlying structure of the model file.

A typical workflow in physics simulation involves a large number of steps, from creating a geometry to post-processing the results. In addition, if any changes are made, previous steps need to be redone. The goal of the Application Builder is to simplify this entire process.¹²

Comsol Multiphysics also comes with twenty example apps to give developers an idea of what's possible with the Application Builder.

Usually, the Application Builder is used for simple demonstrations of an underlying model: the simulation of the model is shown, and the user can alter a few parameters and display a few different graphics.

However, the objective of the SolCelSim project is more ambitious than that: it involves having a fully featured solar cell simulation software that allows the user to change every value that is relevant to solar cell simulation; while automatizing certain processes like the creation of multiple layers and the parameter sweep that would usually require the user to take multiple steps to achieve.

The addition and removal of layers is a good example of how useful SolCelSim can be for researchers. In order to remove a layer from the model, the user would usually have to go through a long process of manually deleting multiple semiconductor properties from the model, changing domains of others, duplicating some and then changing their positions in the model, changing a few global parameters, etc. SolCelSim simplifies this long and tedious process into a single button press. By simply pressing the “Remove Layer” button in the application, SolCelSim automatically handles the entire sequence in the backend and the user immediately sees the results of this change in the simulation. This way, the user can focus on making the changes they want without wasting time with such repetitive sequences. While the removal of layers was specifically given as a practical example, the application also simplifies all the other features that users working with Solar Cell simulation, from altering meshing options to performing parameter sweep.

With the creation of this application, users within research fields in physics that want to simulate solar cells can do it in a significantly more efficient and productive way. In addition, people who would generally avoid Comsol Multiphysics due to the learning process associated (and thus would stick to older software they are already familiar with) can have access to all the necessary Comsol Multiphysics features in a simplified user interface that takes less time to get used to.

Another advantage is that new employees and students can be easily introduced to the physics concepts present in the model, without the learning curve needed to solve differential equations.

¹² <https://www.comsol.com/showcase/application-builder>

By using SolCelSim, the simulation of physics models becomes limited to only Solar Cells, but the process of simulating a multi-layer solar cell becomes significantly streamlined, more efficient, and should result in an increase in productivity within the research group.

2.3 Model

Each Comsol Application Builder application is associated with a model that it interacts with according to user input. For this project, a base “baseline.mph” file was used as the underlying model, and changes were made over time, as deemed necessary.

At first, this model included 3 layers: Cu₂O, AzO and ZnO. Each layer included a number of parameters and was associated to a one-dimensional geometry and multiple semiconductor properties. Material properties, global parameters and an interpolation file were also previously defined. A few results tables for IV and TE studies were already computed and plotted.

Over time, multiple changes were made to the underlying model. They were done both to improve the quality of the “default” model the user of SolCelSim will interact with, and to facilitate specific interactions with the Application Builder. For example, a few global parameters were added specifically to give information about the status of the model, which the application uses during certain methods. Some different tables, studies, semiconductor properties and absorption coefficients were also added.

While it's important that the default underlying model is good, the application was fully designed to deal with changes to the model without breaking. This means that even if the user changes every parameter and adds/removes multiple layers, to the point where the underlying model is completely different from the default, SolCelSim will still function without any issues.

3. Methodology and Planning

This section contains information regarding the Software Development Methodologies used throughout this project, along with the initial and final schedules for the project. Finally, there is also information about how risk assessment was performed.

3.1 Methodology

For this project, an agile methodology inspired by SCRUM was adapted to better fit individual work¹³. In this case, the same person does the work of the Scrum Master, Product Owner, and developer. While there couldn't be team meetings (because the project was made by a single developer), there was still frequent contact with the client and daily personal analysis of the work done in the previous day.

Development was set to be divided into 1-month long sprints. Besides the project backlog, which listed every task that should be accomplished during development, there were also sprint backlogs. Each sprint started with a planning phase, where the list of requirements was re-evaluated, and the objectives for that sprint were set. At the end of each sprint, there was a period for evaluation of the flaws and successes of the previous sprint, a comparison between the work done and the initial plan, with a Burndown chart to help this analysis. Following the implementation, there was also a period of testing in every sprint.

Although the first sprint lasted one month, work for that sprint was finished significantly earlier than expected. As a result, future sprints were changed to last two weeks instead, while maintaining the same amount of work. This proved to be the right decision, as it allowed for a more accurate assessment of the time required for each sprint and less time wasted. While an alternative option of continuing with one-month sprints but doubling each sprint's workload was also considered, a personal decision to go with two-week sprints was made, in part to help adapt to the incoming new requirements.

After the development phase, there was a month that consisted of frequent iterations with the client, identifying possible new requirements and testing previously developed features. For this phase, this version of the SCRUM methodology would not have been as useful. As such, a more iterative approach with short iterations cycles was adopted. In practice, this meant making some changes to the application, allowing the client to identify things he would like to see changed, and repeating this process in short cycles of a few days to slightly over a week.

The website Trello was used to help organize different tasks into a Scrum Board. Within Trello, Tasks can be moved from the backlog to the development and testing sections and marked as complete. Every sprint, a new Scrum Board was created, even if not every task in the previous one was completed. Some feature add-ons for this website were used for, as an example, easily creating Burndown charts.

¹³ <https://www.lucidchart.com/blog/what-is-agile-scrum-methodology>

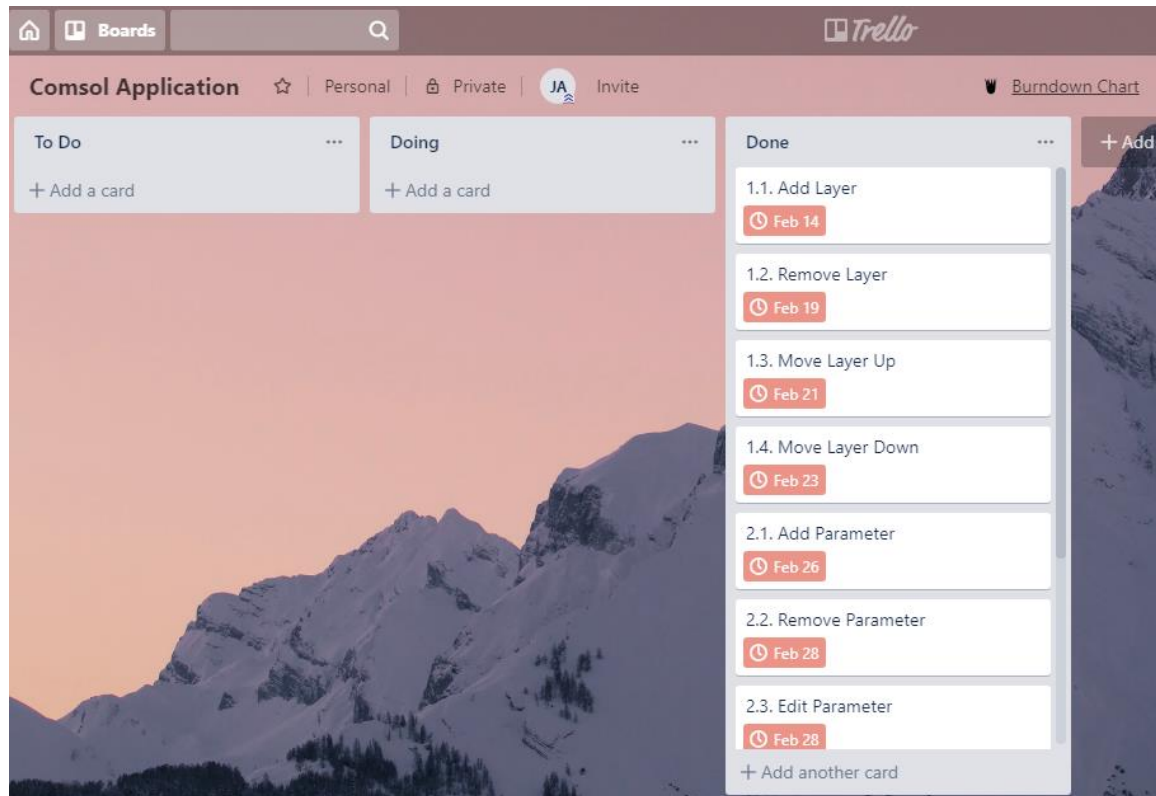


Figure 5 - Trello

Figure 5 shows an example of a Scrum Board created using Trello, with the three task lists frequently used throughout the project.

Using this methodology helped make sure that every requirement was met, while avoiding the situation where development of the initial requirements is finished early and the remaining time is not properly used. The reason why an agile methodology had to be chosen was because of the possibility of new requirements being added in each sprint due to requests from the client.

3.2 Work Plan

Some constraints were considered when defining what the work plan should look like. The following dates had been defined before the start of the project:

- 10th September** – Beginning of Internship
- 21st January** – Delivery of Intermediate Report
- 29th January** – Intermediate Oral Presentation and Defence
- 14th June** – End of Internship
- 1st July** – Delivery of Final Report
- 8th to 12th July** – Final Oral Presentation and Defence

Figure 10 shows the initial plan of how the different stages of creating the project would be spread through time:

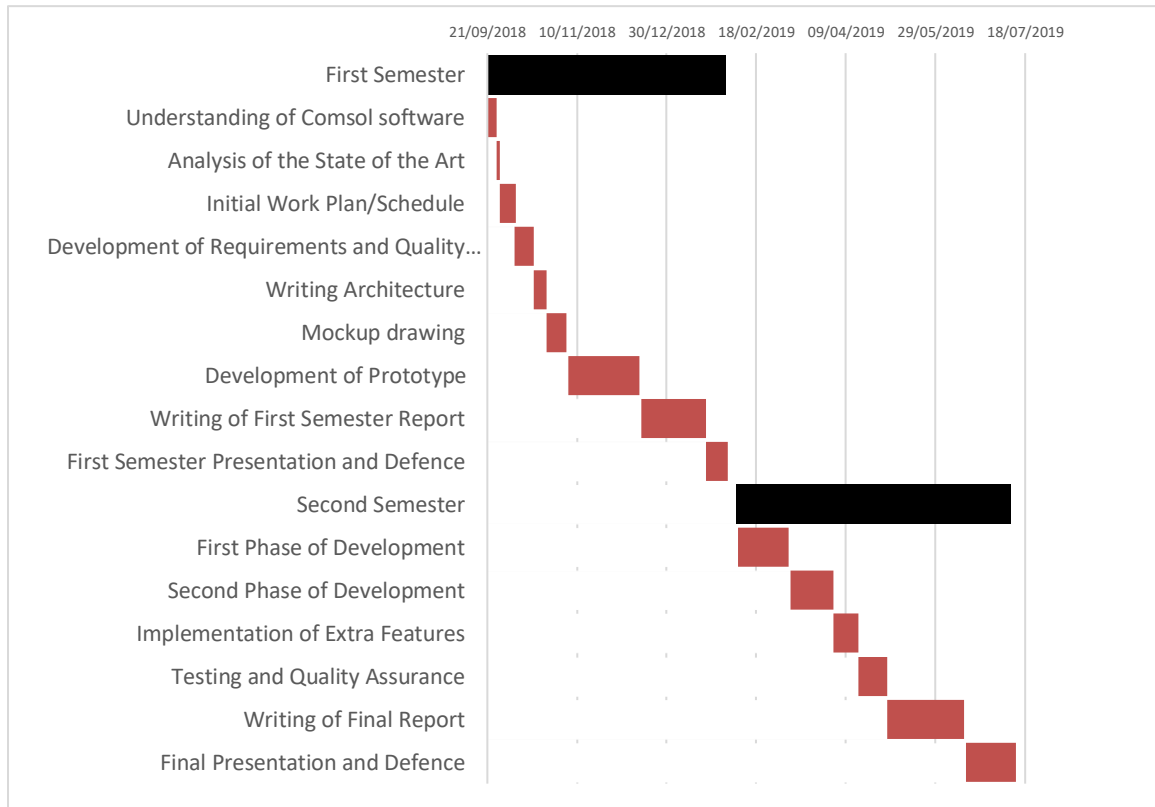


Figure 6 - Initial Work Plan

While the plan for the first semester was mostly followed according to Figure 6, some adaptations had to be made since the plan to create a prototype was scrapped due to request from the client. As such, time that was supposed to be spent on the prototype was spread along the other tasks, mostly the development of the requirements document and the writing of the intermediate report.

Figure 7 shows what the work plan looked like at the end of the first semester. Table 2 has been added for clarity. The first semester part was changed according to how time was spent, and some adjustments were made to the second semester section, as too much time was initially attributed to the writing of the final report. In addition, both phases of development were merged into a single one, as it was considered that having two separate phases did not provide any important information:

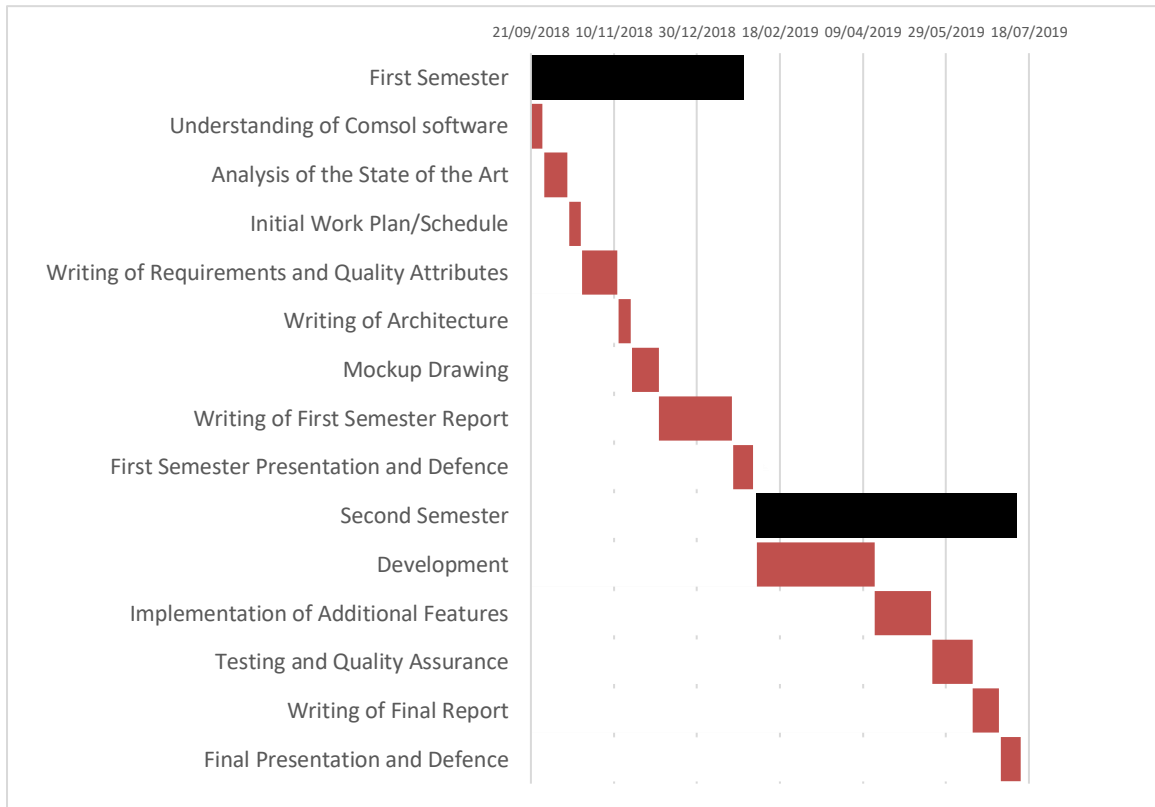


Figure 7 - End of First Semester Work Plan

Activity	Start	End	Duration
First Semester	21/09/2018	01/02/2019	128
Understanding of Comsol software	21/09/2018	28/09/2018	7
Analysis of the State of the Art	29/09/2018	13/10/2018	14
Initial Work Plan/Schedule	14/10/2018	21/10/2018	7
Writing of Requirements and Quality Attributes	22/10/2018	05/11/2018	21
Writing of Architecture	13/11/2018	20/11/2018	7
Mockup Drawing	21/11/2018	06/12/2018	16
Writing of First Semester Report	07/12/2018	20/01/2019	44
First Semester Presentation and Defence	21/01/2019	01/02/2019	12
Second Semester	04/02/2019	12/07/2019	157
Development	04/02/2019	16/04/2019	71
Implementation of Additional Features	16/04/2019	20/05/2019	34
Testing and Quality Assurance	21/05/2019	13/06/2019	24
Writing of Final Report	14/06/2019	30/06/2019	16
Final Presentation and Defence	01/07/2019	12/07/2019	12

Table 2 – End of First Semester Work Plan

The following list shows the purpose of each task:

1. Understanding of Comsol Software – Reading all the available Comsol Multiphysics documentation and experimenting by making very simple test programs.

2. Analysis of the State of the Art – Researching past projects within the area, discovering popular and high-quality simulation software used in simulating solar cells.
3. Initial Work Plan/Schedule – Estimating how much time each task should take and presenting it graphically.
4. Writing of Requirements and Quality Attributes – Writing the first version of the Requirements document, with frequent feedback from the client.
5. Writing of Architecture – Defining elements of the software architecture, including Client-Server interactions
6. Mockup Drawing – Initial designs for the GUI, following good UI design heuristics and principles.
7. Writing of First Semester Report
8. First Semester Presentation and Defence
9. Development – Development process, following Scrum methodology
10. Implementation of Additional Features – Technically part of the development phase, extra time to implement new features that the client might consider necessary.
11. Testing and Quality Assurance – Quality Assurance tests. Some extra time was allocated to allow for changes to be made according to the results of these tests.
12. Writing of Final Report
13. Final Presentation and Defence

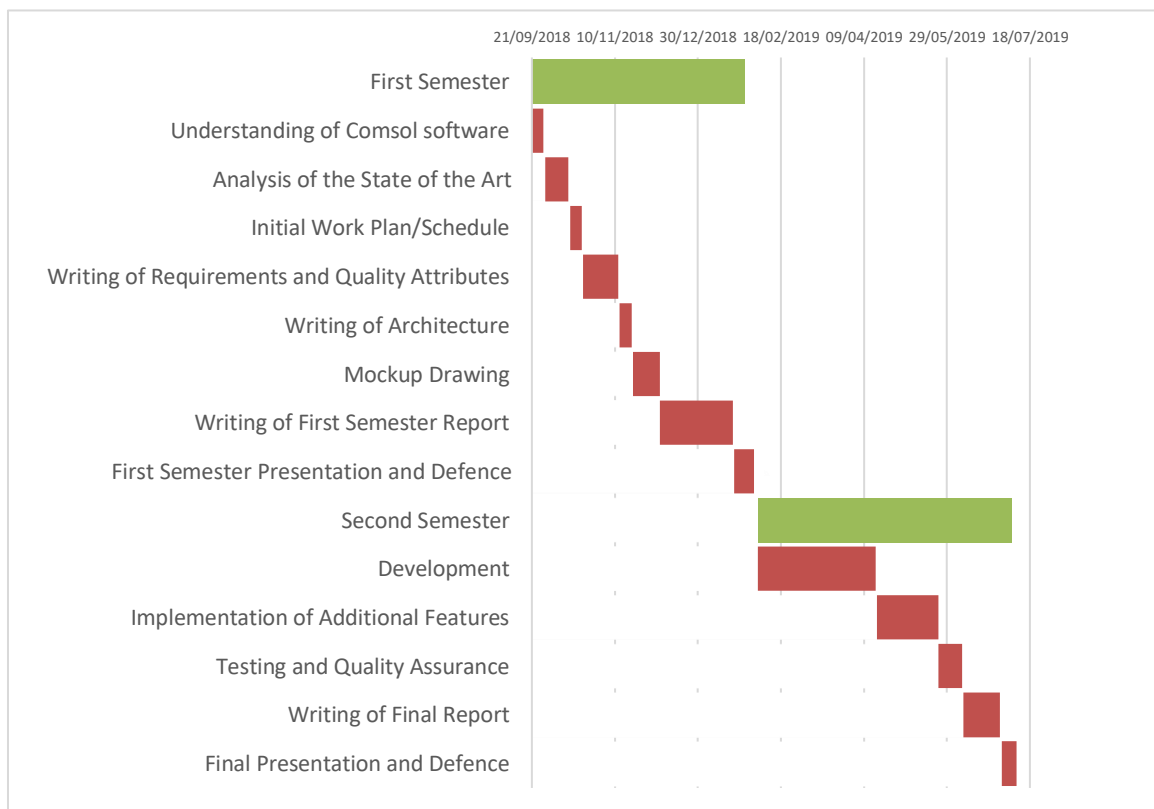


Figure 8 - End of Second Semester Schedule

Activity	Start	End	Duration
First Semester	21/09/2018	01/02/2019	128
Understanding of Comsol software	21/09/2018	28/09/2018	7
Analysis of the State of the Art	29/09/2018	13/10/2018	14
Initial Work Plan/Schedule	14/10/2018	21/10/2018	7
Writing of Requirements and Quality Attributes	22/10/2018	05/11/2018	21
Writing of Architecture	13/11/2018	20/11/2018	7
Mockup Drawing	21/11/2018	06/12/2018	16
Writing of First Semester Report	07/12/2018	20/01/2019	44
First Semester Presentation and Defence	21/01/2019	01/02/2019	12
Second Semester	04/02/2019	09/07/2019	153
Development	04/02/2019	16/04/2019	71
Implementation of Additional Features	17/04/2019	23/05/2019	37
Testing and Quality Assurance	24/05/2019	07/06/2019	14
Writing of Final Report	08/06/2019	30/06/2019	22
Final Presentation and Defence	01/07/2019	09/07/2019	9

Table 3 – End of Second Semester Work Plan

Figure 8 and Table 3 show how the schedule of the project ended up being organized. Although the final schedule doesn't differ much from the original plan, there were a few changes to note. The Testing and QA phase was slightly shortened, to allow time for the writing of this Final Report. The end of the project was also changed to reflect the official date for the Oral Presentation. Otherwise, the remaining changes were very minor, generally consisting of a two or three day's difference at most. The labels for the table can be consulted below Table 2, as both tables share the same activity names.

3.3 Development Schedule

Figure 9, in the next page, shows the initial plan for the development phase of the project. Each requirement is taken as a different task, and an estimated timeslot was attributed based on its complexity. To see details about what each task entails, consult **Attachment 3 – Functional Requirements**. This schedule was used to plan each sprint. It's important to note that the Report task refers to the report generation feature in the application, not to the internship report.

This schedule focuses only on the development phase. As such, information about testing is not included in the figure, as the testing phase happened afterwards. However, it's also important to note that the development of each task includes a short period of testing at the end. More information can be found in the “7. Testing and Quality Assurance” section of this report.

On the following page, Figure 10 contains the final development schedule. By comparing it to the previous figure, we can observe the changes that were made to the schedule during the development process.

The order the tasks were developed in was changed in some places. For example, the Compute EIS and Parameter Fitting tasks were delayed and other tasks were done earlier in their place, while waiting for certain feedback from the client.

It's possible to note that some estimations ended up being inaccurate. For example, layer parameter tasks took less time than expected, while computation tasks took longer. In the end, the two compensated for each other and the overall time the development process took was the same as initially estimated.

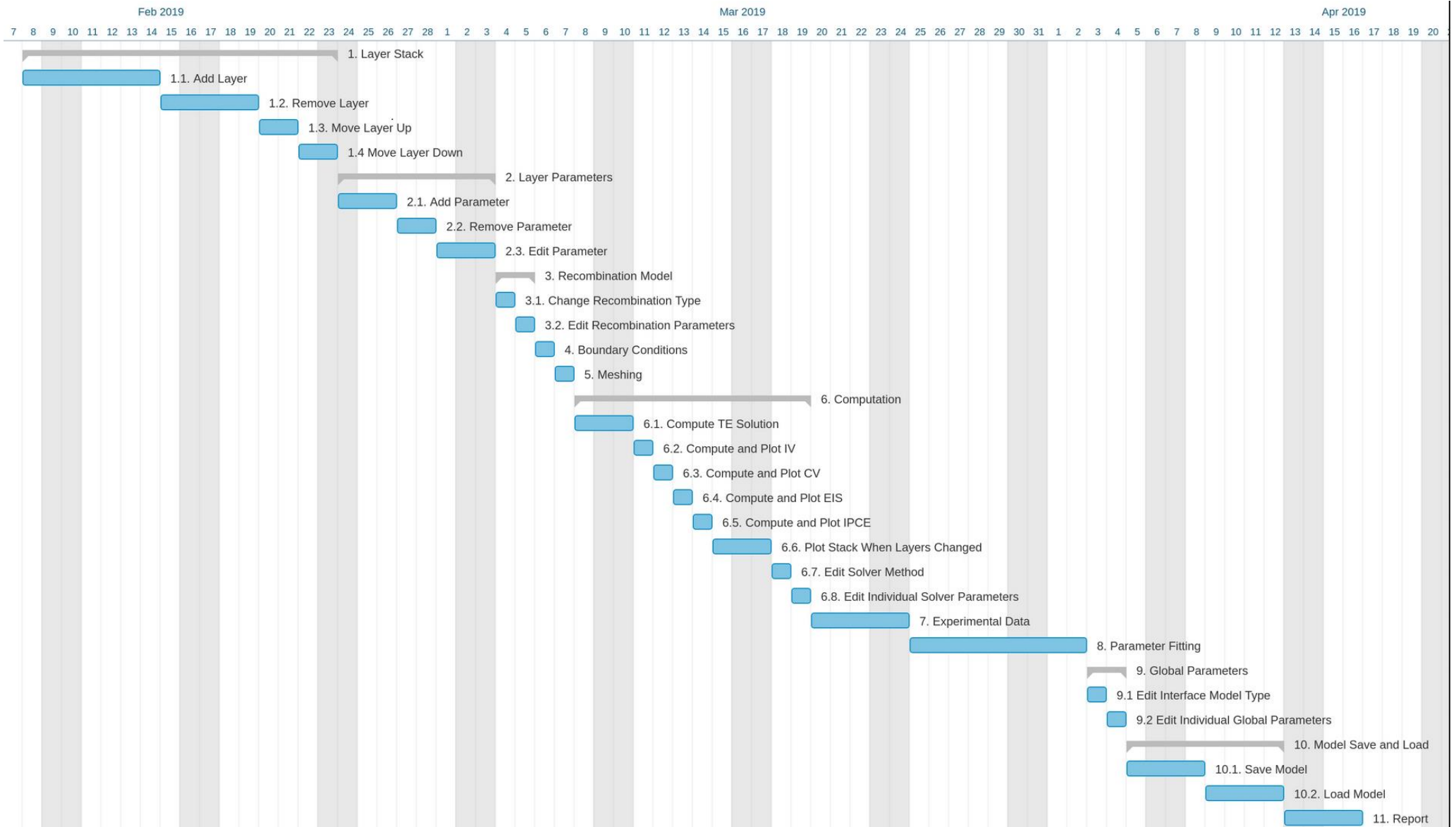


Figure 9 – Initial Development Plan

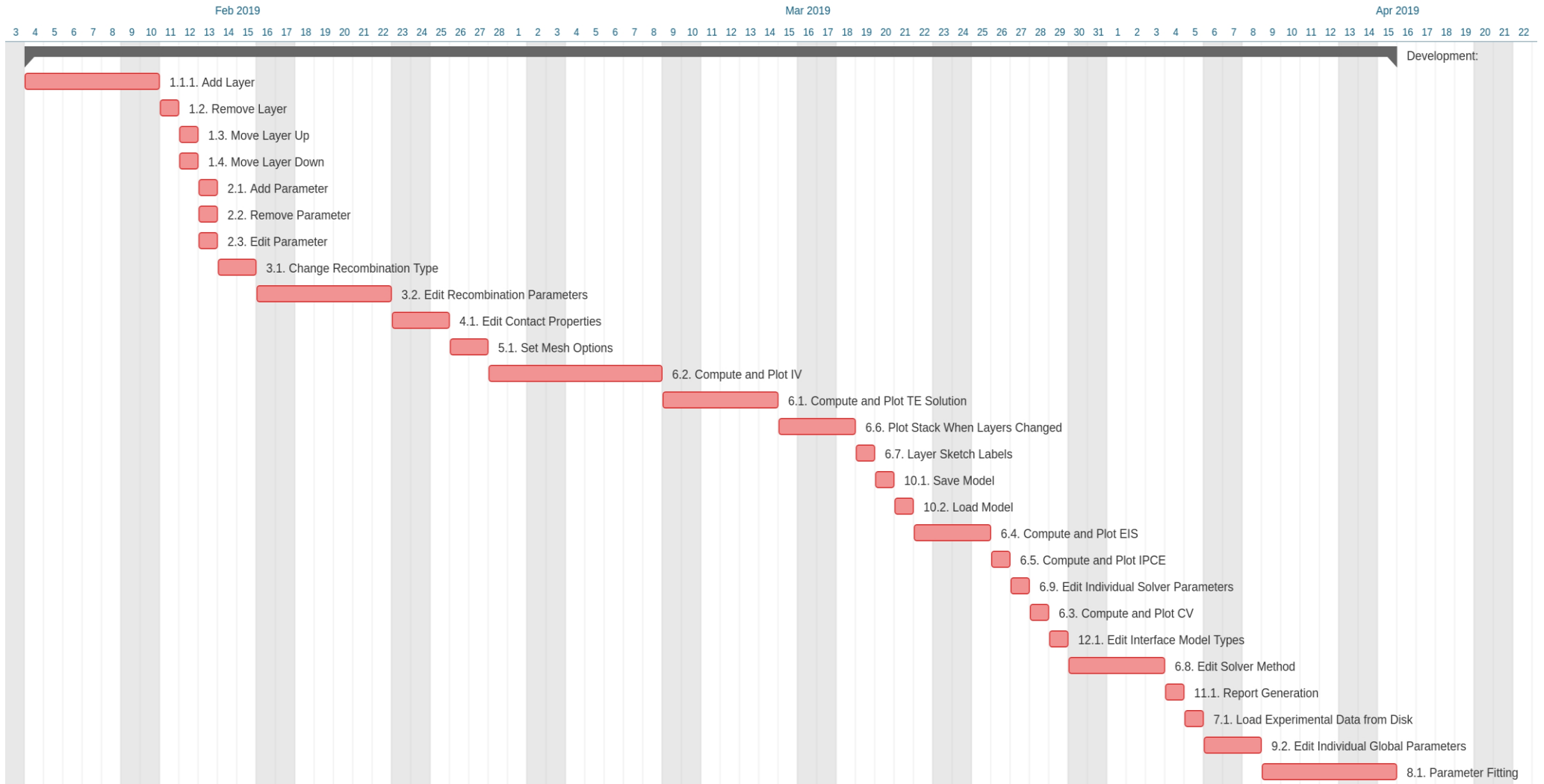


Figure 10 – Final Development Plan

During the development phase, the time it took to complete each task from each sprint was recorded. At the end of each sprint, a burndown chart was generated to help evaluate whether or not work was being done in the right amount of time.

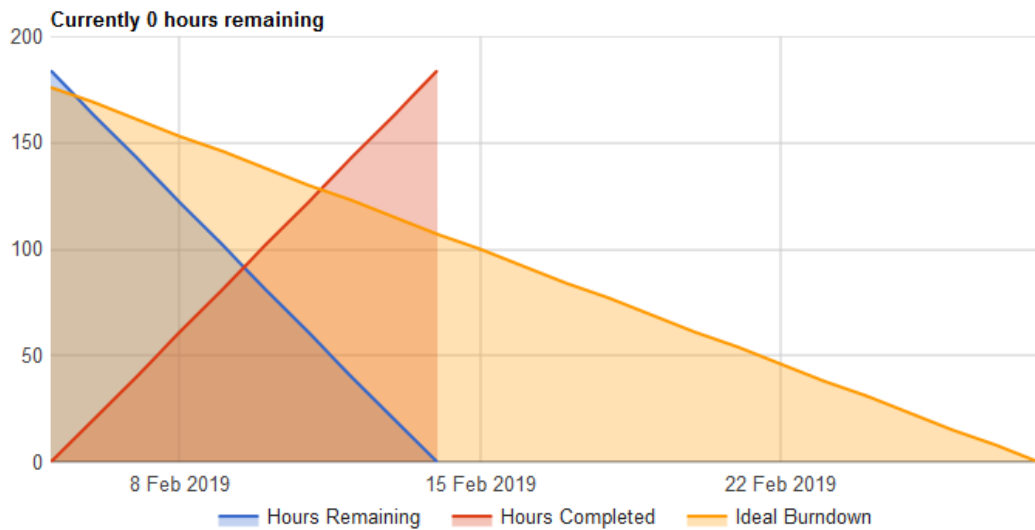


Figure 11 – Sprint 0 Burndown

Figure 11 shows the burndown chart from the first sprint. This sprint lasted one month and, as can be observed, work was completed significantly earlier than expected. With these results, the decision was made to change the duration of future sprints into two weeks. The burndown charts corresponding to the following months can be seen in the next figures.

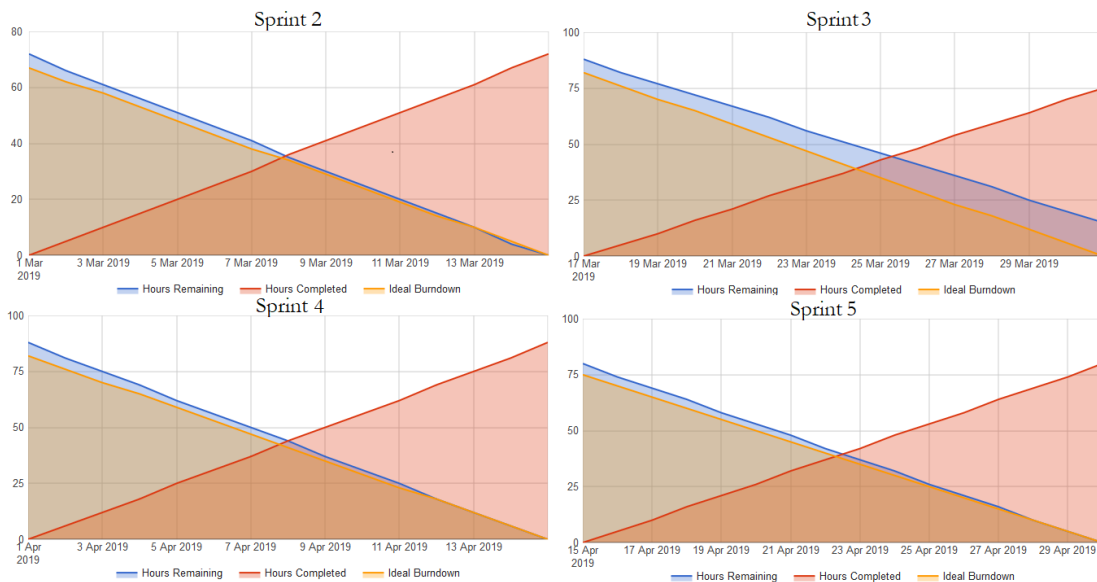


Figure 12 – Remaining Sprints Burndown

As can be observed in Figure 12, the following sprints went largely according to expected, with the amount of work completed being very close to the expected amount. As such, the same methodology with roughly the same workload per sprint was maintained throughout development.

Also worth noting is the fact that the dates for the last sprint shown fell outside of the timeframe for the development phase. This is because that timeframe corresponds to the first few weeks of the project phase that were reserved for the addition of extra features.

The following list will detail some information about each sprint:

Sprint 1 (Feb 4 to Feb 28) – As the first sprint, this was where the foundation of the application was developed. Areas that a lot of different requirements had as dependencies (especially areas related to layers and parameters) were worked on.

Sprint 2 (March 1 to March 14) – Implementation and testing of IV and TE solver computation and plotting.

Sprint 3 (March 15 to March 31) – Multiple requested changes, remaining features related to the layer stack, and completion of the solver computation and plotting requirements.

Sprint 4 (April 1 to April 14) – Completion and testing of all the initial requirements.

Sprint 5 (April 14 to April 30) – Focus on additional requirements added through feedback from client.

Regarding the additional requirement phase, while the adapted SCRUM methodology that was used up to that point was helpful for Sprint 5, it soon became clear through conversations with the client that the remaining time for this phase would need to follow a different process. This is due to the need that arose to have short iteration cycles where small changes to the application and feedback from the client would be intercalated. As such, the move to an iterative methodology consisting of small iterations of just a few days each was made. In order to organize this, a Kanban board where tasks would be added when feedback from the client came in and removed once they were completed was created.

This move to a more iterative process proved to be helpful for this phase, as it simplified the process of rapidly iterating new versions with client feedback while keeping the remaining tasks well organized.

3.4 Risk Assessment

A list of potential risks, with their importance and ways to prevent them, was written in Table 4. “Risk” describes the nature of the risk itself, concern describes its potential impact, “Ranking” is based on both likelihood and impact (1 being the lowest risk and 3 being the highest), and “Mitigation Strategy” describes the steps taken to combat said risk:

Risk	Concerns	Likelihood	Impact	Ranking	Mitigation Strategy
Unfamiliarity with development tool	Could result in an unexpected increase in development time	3	2	2	Two weeks of preparation were given just for getting used to Comsol
Overly generous time estimates	Could force some requirements to be cut out completely	1	3	1	Estimates should be re-evaluated at the end of every sprint
Addition of too many new requirements	Could require more time to be developed than is available	1	2	1	When a new requirement is added, other requirement's time estimates are taken into account
Lack of theoretical physics knowledge	Could require extra time studying materials	3	1	1	Client will provide help in these situations, as required
Problems related to Comsol version	Could make some features impossible to implement	2	3	2	The University will attempt to purchase licence to the latest version
Comsol Application Builder limitations	Could make some features impossible to implement	2	3	2	Comsol Multiphysics manuals were consulted when designing requirements, making sure the features we need are available

Table 4 – Risk Assessment

At the end of every sprint, the development of every risk was analysed, in order to find out if it needed more careful inspection or if it was no longer a risk and could be safely removed without worry.

In the end, thanks to the mitigation strategies chosen, most of these risks ended up being mitigated successfully. However, there were two exceptions that did cause an impact on the development process.

The first problem was the addition of too many new requirements. Due to demands from the client, some requirements required more work time than expected to be fulfilled. This did not, however, significantly impact the development schedule.

The second problem was related to the Comsol version. Since the university did not have access to a Comsol Server licence, the features related to the planned client-server architecture had to be cancelled.

4 Requirements and Architecture

This chapter contains information about both the functional and non-functional requirements for this project, as well as a detailed explanation of the project’s architecture.

The list of requirements was developed based on the results from interviews with the client, a short Specifications document, and paper sketches. An initial version was written to try and fulfil every demand. From then on, four revisions were done with the client until reaching the list shown in this document.

Each requirement is structured by ID, Dependencies and Description. Note that Dependencies only consider other requirements, and not aspects of the original Comsol model.

Functional and Non-Functional Requirements were divided into two different sections. Within the Functional Requirements, multiple different sub-sections were created.

Since the development of this project followed an adapted version of the SCRUM methodology, the end of each sprint served as time to review the requirements document, and to make sure that the previously written requirements were respected.

All requirements listed were considered “Must”. As such, the threshold of success for this project was the completion of every requirement listed in this section.

In order to understand this list of requirements, it’s important to understand how the Comsol Application Builder works. The Application Builder is a feature within Comsol Multiphysics that allows the developer to create an app that users can later interact with, with their action affecting a saved model. In the particular case of this project, the model simulates the charge transport inside a PEC or PV Solar cell. This solar cell simulates all the specific aspects of real solar cells, including their different layers, as well as each layer’s properties.

Since the user must have control over the details of the simulation, it’s important to parameterize as many aspects of it as possible. Many of the listed requirements will show exactly that, parameters that the user can change to alter the simulation, and observe their impact in the results.

For more information, an explanation about how Comsol works, and what steps need to be taken to add a new functionality in Comsol Application Builder, has been included in the Attachments section of this report (**Attachment 1**). This explanation serves to try and offer the reader a more intuitive idea about how development in the Comsol Application Builder works, which should make the requirements of this project easier to understand. The previous “2.2. Framework” section of this report also contains important information regarding this subject.

4.1 Functional Requirements

This section will show a shortened list of the functional requirements for this project in Table 5. A more in-depth explanation of each of the requirements can be consulted in the Attachment section of this report (**Attachment 3**).

Requirement	ID	Dependencies
1.1. Layer Stack		
1.1.1. Add Layer	FUN01	None
1.1.2. Remove Layer	FUN02	FUN01
1.1.3. Move Layer Up	FUN03	FUN01
1.1.4. Move Layer Down	FUN04	FUN01
1.2 Layer Parameters		
1.2.1. Add Parameter	FUN05	FUN01
1.2.2. Remove Parameter	FUN06	FUN05
1.2.3. Edit Parameter	FUN07	FUN05
1.2.4. Rename Layer	FUN08	FUN01
1.3. Recombination Model		
1.3.1. Change Recombination Type	FUN09	FUN01
1.3.2. Edit Recombination Parameters	FUN10	FUN09
1.4. Boundary Conditions		
1.4.1. Edit Boundary Conditions	FUN11	None
1.5. Meshing		
1.5.1. Set Mesh Options	FUN12	None
1.6. Computation		
1.6.1. Compute and Plot TE Solution	FUN13	None
1.6.2. Compute and Plot IV	FUN14	None
1.6.3. Compute and Plot CV	FUN15	None
1.6.4. Compute and Plot EIS	FUN16	None
1.6.5. Compute and Plot IPCE	FUN17	None
1.6.6. Plot Stack When Layers Changed	FUN18	FUN02-FUN10
1.6.7. Layer Sketch Labels	FUN19	FUN18
1.6.8. Edit Solver Method	FUN20	FUN13-FUN17
1.6.9. Edit Individual Solver Parameters	FUN21	FUN13-FUN17
1.6.10. Edit Illumination Value	FUN22	FUN13-FUN17
1.6.11. Plot Solution	FUN23	FUN13-FUN17
1.6.12. Change current plot	FUN24	FUN23
1.6.13. Change displayed parameter sweep values	FUN25	FUN24
1.7. Experimental Data		
1.7.1. Load Experimental Data from Disk	FUN26	FUN13-FUN17
1.8. Parameter Fitting		
1.8.1. Perform Automatic Parameter Fitting	FUN27	FUN05, FUN20, FUN21
1.9. Global Parameters		
1.9.1. Import Spectrum Photon File	FUN28	None
1.9.2. Edit Individual Global Parameters	FUN29	None
1.10. Model Save and Load		
1.10.1. Save Model	FUN30	None
1.10.2. Load Model	FUN31	FUN30
1.11. Report		
1.11.1. Report Generation	FUN32	FUN13-FUN17
1.12. Heterointerfaces		
1.12.1. Edit Interface Model Types	FUN33	FUN29

Table 5 - Functional Requirements

4.2 Non-Functional Requirements

4.2.1. Usability

ID: NONFUN01

Description: A user with knowledge in physics but with no experience with Comsol or programming should be able to perform the tasks allowed by the application without needing to be guided. In the sections “5.3. Design Walkthrough” and “7.6. Usability Testing” of this report, details about how this was measured are detailed.

4.2.2. Extensibility

ID: NONFUN02

Description: The application should allow for an easy implementation of new functions, without breaking previously implemented ones. For example, if a change to the experimental data import format is requested, it should be possible to change the system accordingly in less than 20 work hours.

4.2.3. Maintainability

ID: NONFUN03

Description: All defects should take, on average, less than 3 days for a developer familiar with the application’s code to correct.

4.2.4. Accuracy

ID: NONFUN04

Description: Application simulation results should never differ more than 1% from regular Comsol model results. During quality assurance, model data will be used to ensure this requirement is met.

4.3 Architecture

SolCelSim can be used as a desktop application by users with access to a Comsol Multiphysics licence. The app indirectly connects the user to a Comsol model. This section details that connection, with specific examples that illustrate how the architecture of the project works.

Initially, the architecture was planned to include a server that could be accessed remotely from a web browser or from the Comsol Client App. However, since the University where this project was developed in wasn't able to get access to the Comsol Server, this feature was cancelled and SolCelSim was changed to be a standalone application. Details about these Client-Server architecture plans can be found in **Attachment 4 – Server Architecture**.

Following those changes, Figure 13 represents the general architecture of the project:

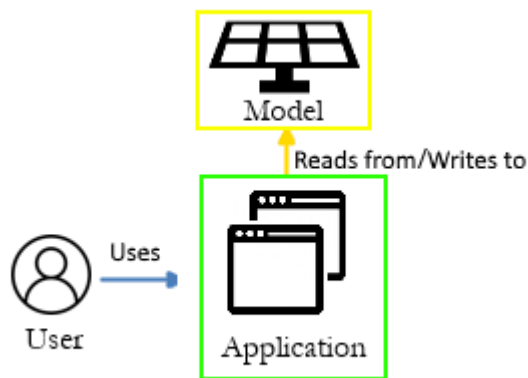


Figure 13 – General Architecture

Figure Element Labels:

- *Blue Arrow:* Interactions between user and application. Includes things such as parameter inputs, addition of layers, etc.
- *Yellow Arrow:* Connection between model and application. Includes both writing information to the model and retrieving information from it.
- User – Person interacting with the application.
- Application – The software simulator described in this report.
- Model – The underlying model of the solar cell, with which the application interacts.

The developer was responsible for developing the part highlighted in green. The part highlighted in yellow was previously created by the research team, but had some changes made over time both by the team and by the software developer.

This general architecture was chosen to maximize accuracy, according to the non-functional requirement NONFUN04. Since data is retrieved from the model, results obtained by the user while using the application should be as accurate as if they were performing Comsol Multiphysics simulation directly through the Model Builder. To verify this accuracy, accuracy testing was performed, and is detailed in section “7.3. Accuracy Testing/Boundary Value Testing”.

The application seen in Figure 13 was created using the Comsol Application Builder. More information about it can be found in section “2.2.2. Comsol Application Builder” of this report.

The user only interacts with the User Interface of the Application. From there, the app automatically reads information from the model, displays it, and changes it according to user input.

Regarding the coding of the application, multiple methods were implemented. This way of organizing code, with separate methods that interact with the model directly, should ensure good extensibility (NONFUN02). New methods can easily be added, both by the developer of this project and by future developers, without interfering with the rest of the application. Frequent use of comments will also help allow future developers understand the code and implement new features quickly.

Since data is stored locally, there is also no need for the creation of a central database. Model files will be stored in .mph format; experimental data files will be stored in .csv format and report files will be stored in .pdf format.

Model files can be saved and loaded through the application. When the user finishes working on a model, they can save it locally, and later resume work. The entire model is saved this way, including both the state of the application (for example, parameter sweep values to be used later) and the information stored in the model itself.

The overall flow of execution when the user presses something involves the correct method being activated, the parameters being checked, and information being sent and saved to the model. When things are changed, information is sent from the model to the application and shown on the user’s screen.

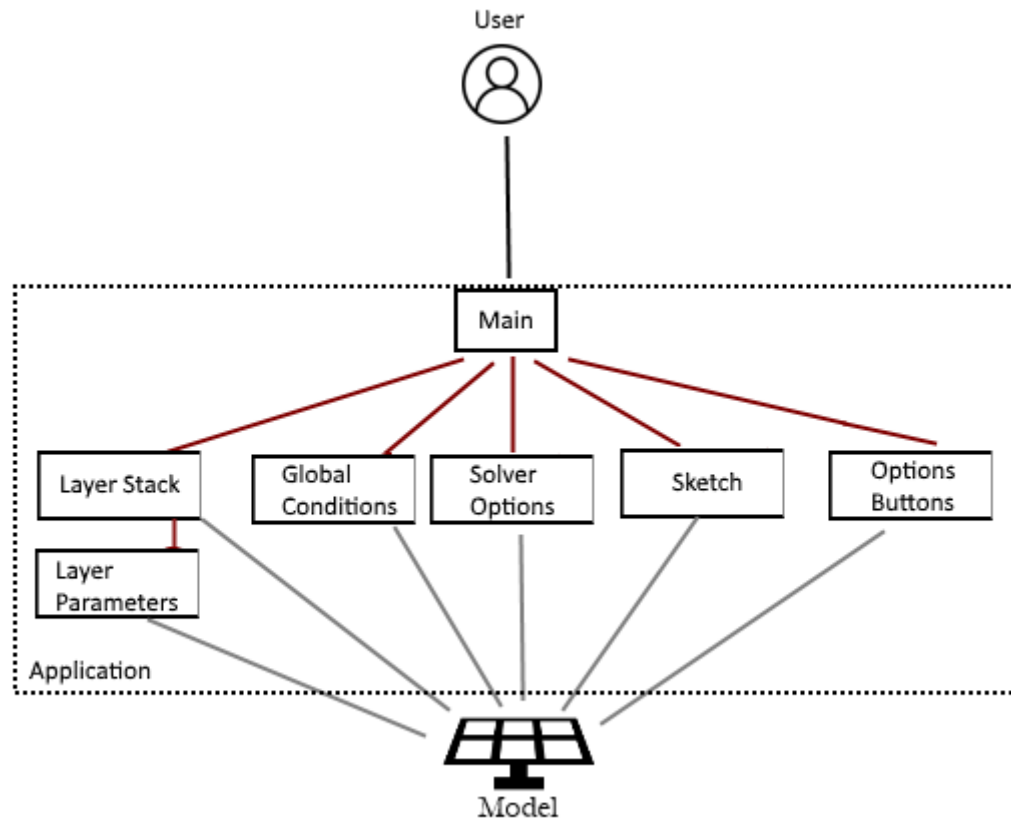


Figure 14 – Diagram with Software Components

Figure Element Labels:

- *Black Line:* Interaction between user and application.
- *Brown Lines:* Connection between “Main” form and other subforms.
- *Grey Lines:* Connections between model and application.
- User – Person interacting with the application.
- Main – Main form of the application, giving access to the features enabled by the different methods.
- Layer Stack – Controls the multiple simulated layers and the options for editing them.
- Layer Parameters – Allows altering of each layer’s parameters.
- Global Conditions – Allows the user to change the global parameters that affect the entire model.
- Solver Options – Allows editing of the parameters used when computing solutions.
- Sketch – Displays the sketch of the simulated multi-layer solar cell.
- Options Buttons – Additional options related to the Layer Stack, including creation of new layers.
- Application – Includes a selection of methods present in the application.
- Model – The Comsol Multiphysics model, which receives information from the previously mentioned methods.

In Figure 14, we take a deeper look at the application, and specifically the groups of methods created with the application builder. The brown lines show interactions between the groups, while the grey lines show retrieval and delivery of information between the

application and the model. From the “Main” menu, different modules are accessed, which then interact directly with the model and alter or retrieve relevant information.

The main requests from the client were that users could dynamically change the number of layers in the simulated model, along with each layer’s parameters and their values.

A number of solutions (TE, IV, IPCE, EIS and CV) of the model is computed and for every one of them users can import experimental data and perform automatic Parameter Fitting. They can also select the number of parameters of simulated data. The simulation takes the layers added by the user into account. Details about the implementation of these features is further detailed in the Requirement section of this report.

In Figure 15, we will look at the Application Flow Diagram that describes user’s interactions with the entire application. By looking at this figure, it should be possible to understand some of the choices made to ensure good usability (NONFUN01). For example, in section “5.2 Principles” of this report the idea of managing a good balance between Discoverability and Constraints was mentioned. By looking at the diagram, we can notice that even completely unrelated features are never more than 6 clicks away from one another, ensuring that it’s easy for users to get to different sections of the application without getting lost. At the same time, the user doesn’t see every single feature at the same time, ensuring that they don’t get overwhelmed and confused. Usability tests were performed to make sure these measures were effective, as can be seen in section “7.6. Usability Testing” of this report.

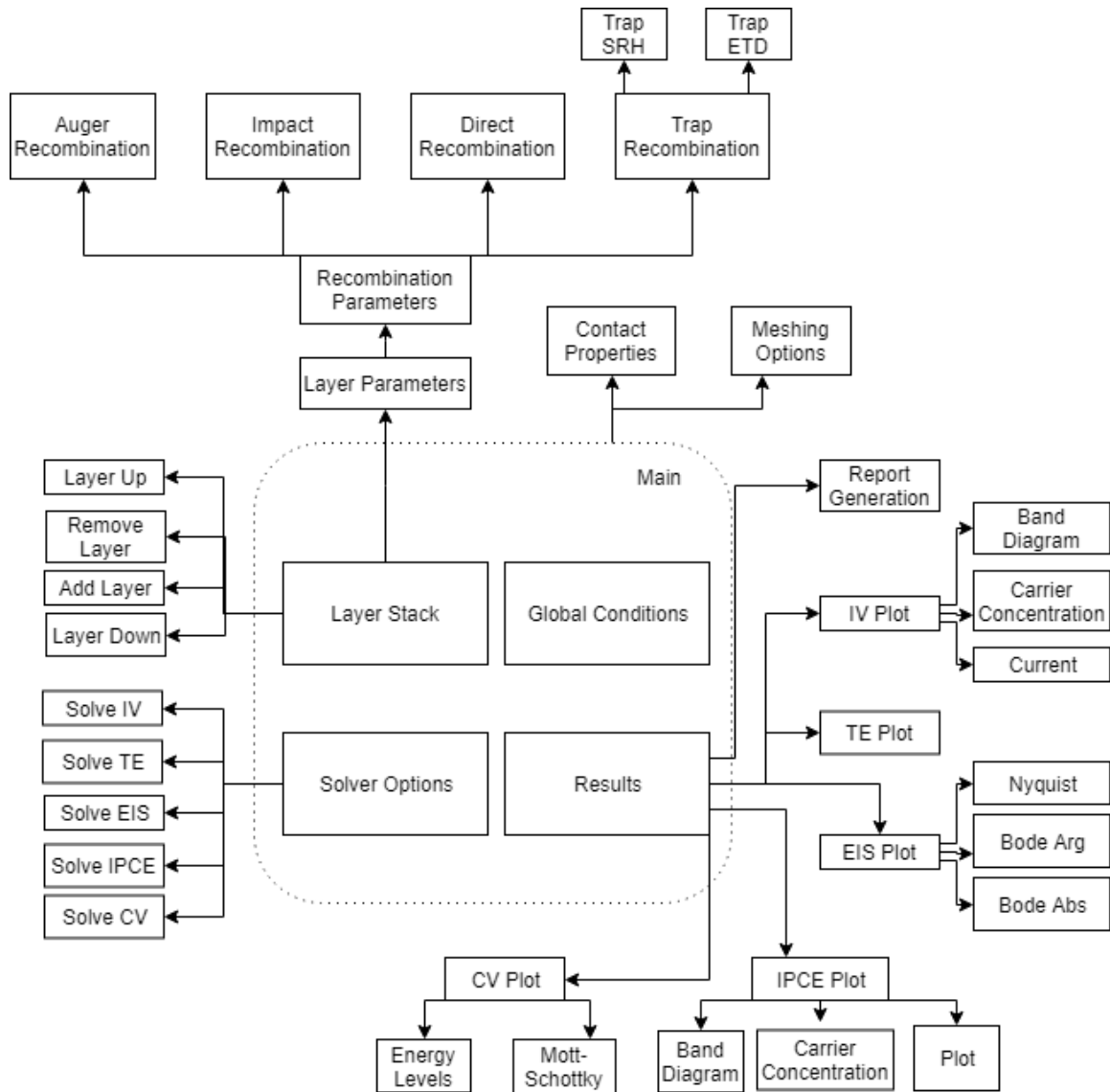


Figure 15 – Application Flow Diagram

Figure Element Labels and Associated Functional Requirements. Functional Requirement tags are included under squared brackets:

- Arrows: Each arrow in the figure represents the ability of the user to move from one part of the application to the other. The user is always able to return to the previous section.
- Recombination Parameters – [FUN09] Includes the different recombinations types.
- Auger, Impact, Direct, Trap Recombinations – [FUN10] Each section allows for editing of the parameters for that specific recombination type.
- Trap SRH, Trap ETD – [FUN10] Two available types of Trap Recombination.
- Layer Parameters – [FUN05 – FUN08] List of individual layer parameters.
- Main – [FUN31 – FUN32] Sections that can be immediately accessed upon opening the application.
- Layer Stack – [FUN01 – FUN04, FUN18 – FUN19] Includes the currently simulated solar cell layers.
- Layer Up, Remove Layer, Add Layer, Layer Down – [FUN01 – FUN04] Different available solar cell layer options.
- Solver Options – [FUN20 – FUN22, FUN27] Available settings that affect solution computation.

- Solve IV, Solve TE, Solve EIS, Solve IPCE, Solve CV – [FUN13 – FUN17] Settings specific to each solver type. The user can perform computation here.
- Contact Properties – [FUN11] Left and Right Metal Contact properties of the model. Settings of each individual Contact can be changed.
- Meshing Options – [FUN12] Settings that affect the meshing of the model.
- Global Conditions – [FUN26, FUN28 - FUN29, FUN33] General parameters that affect the entire model.
- Results – [FUN23 – FUN25] Displays results from previously computed solutions from the “Solver Options” section.
- Report Generation – [FUN32] Creates a PDF file with information about the model and the computed results.
- IV Plot, TE Plot, EIS Plot, IPCE Plot, CV Plot – [FUN23 – FUN25] Displays the plot generated by each of the solvers.
- Band Diagram, Carrier Concentration, Current, Nyquist, Bode Arg, Bode Abs, Plot, Energy Levels, Mott-Schottky – [FUN24] All of the different available result plots for each individual solver. Allow access to each plot without re-computing the solution.

Figure 15 expands the application portion of Figure 14 into the entire application. This diagram shows how the user can access all of its individual screens.

All parts of the application seen in the figure were developed by the intern, in Java code, using the Application Builder. Initially, there was no application, only the underlying Comsol model. The application was developed individually during the second semester of this project.

The “Main” section includes the different tabs that can be activated in the main menu. From there, some sections, such as the Contact Properties, can be accessed from any tab, while others are specific to certain tabs. When the user moves to a different screen, they can return to the previous screen by pressing the “Back” button at all times. To ensure good extensibility and maintainability (NONFUN02 and NONFUN03), the application’s code is designed so that the different “Main” sections aren’t dependent on one another. This means that developers can make changes to one without breaking the others.

The user is also able to alternate between different plots (from CV Plot to IPCE Plot and vice-versa, for example).

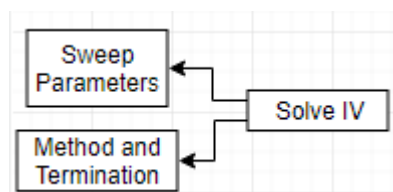


Figure 16 – Application Flow Diagram (Solver)

Figure 16 details the screens accessible from each Solver. This information was omitted from Figure 15 for the sake of simplicity, as each individual solver (IV, TE, EIS, IPCE and CV) has access to different versions of these screens.

SolCelSim was developed using the Comsol Multiphysics Application Builder, using the JAVA programming language. Users are able to access it on Windows, Linux and Mac, assuming they have access to a valid Comsol Multiphysics licence. The application was created by a single developer, as an internship project.

5. Interaction Design

In this chapter of the report, we will look at the decisions made when designing the User Interface of SolCelSim. Principles used, how design walkthroughs were performed, and an example of a Use Case Diagram are shown. Finally, the initial mock-ups are shown, with detailed comments about each of them.

5.1 Interface Design

This section will go into detail about the principles considered and the methods used to ensure the best possible design of the GUI and the best possible user experience.

Good interface design was especially important in this project, as our objective was to create a piece of software that can be understood in a relatively short amount of time with as little frustrations as possible, serving as a good alternative to the difficulty of learning to work with Comsol Model Builder.

Some methods used to evaluate the project from an interface design perspective will be listed in the following subsections.

5.2 Principles

This section will detail some common UI design principles¹⁴ that were used in the design of the User Interface:

1. Discoverability: The user should immediately have a general idea of how to access any feature he might want to use. This was done by showing all the generic options on the main menu, and through the use of icons, when their meaning is obvious, and labels otherwise.
2. Feedback: The user should never be confused about the state of the application – if something is loading, the application should make that very clear, through loading bar or messages. If there is an error, an error box explaining exactly what happened should appear. Finally, every single action should have an immediate, visible effect.
3. Constraints: Not every option should appear at all times. In each different menu, the user should only have access to the options that are relevant for that menu.
4. Gather Similar Options: Similar tasks should be close to each other on the UI, so the user doesn't have to move the cursor through the whole screen to perform a predictable, linear sequence of tasks.

5.3 Design Walkthrough

Once new features are implemented into the project, it was necessary to test whether users could quickly access them through the User Interface without being guided by someone else. As such, a sequence of tasks was designed, and users were asked to perform them without any extra help. The steps they took, along with their thought process and the time it takes to

¹⁴ <https://blog.prototypr.io/boost-your-ux-with-these-successful-interaction-design-principles-e2f0c2b49050>

perform each step, was recorded and analysed. One of the main objectives was to measure learning time.

The expectations in these walkthroughs were that a new user should be able to perform a series of 7 tasks, in less than an hour, without guidance.

At the end of any walkthrough, questionnaires were used to try to measure user satisfaction. They used Likert Scales with cross-checking questions and generally contained simple questions about the feature being evaluated. The user was expected to feel comfortable using any feature within the application after two hours. The previously mentioned questionnaires were used to measure whether or not that objective has been met.

Information about the results from these tests can be seen in section “7.6. Usability Testing” of this report.

5.4 Use Case Diagram

Figure 17 shows a brief overview of how each user’s interactions with the system works. It was made to make sure that the initial idea for user interaction was followed during the design phase.



Figure 17 – System Use Case Diagram

5.5 Mockups

A number of mockups were developed, based on feedback from the client and with the objective of following every requirement. These mockups went through 3 revisions with the client before reaching the final state, shown in the next figures.

The decision to include these mockups in this report was due to the high importance of interface design and usability in this project.

The requirements that each mockup looks to fulfil will be mentioned by their labels in each mockup's description.

When designing these mockups, the objective was to obey the good UI design principles listed in section 5.2. Some examples of how each principle were considered will be mentioned in this section.

The main menu serves as the centre from which all features can be accessed. From here, the user has access to every option related to computation and the use of experimental data. They can save (FUN30) and load (FUN31) the model. They also have access to four tabs (Stack, Solve, Global Options, Results).

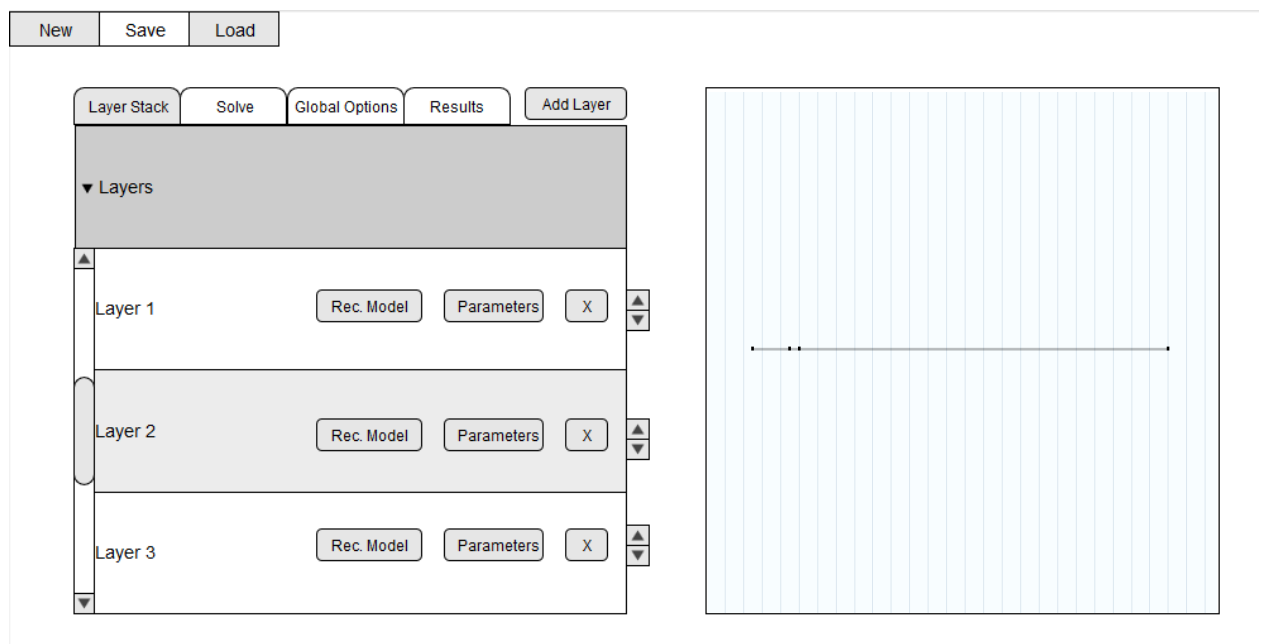


Figure 18 – Main Menu

The principle 4. Gather Similar Options was considered while designing every element on the UI, which can be easily seen from the mockups. Parameter values that can be edited, confirmation buttons, plot displays and report-related buttons all have their own specific part of the UI reserved, which stay consistent throughout.

The principle 3. Feedback can't be directly seen in the mockups. However, as requested by the client, a separate window is seen whenever there is any kind of error, or while plots and reports are being generated.

When designing the main menu, it was important to find a good balance between the principles 1. Discoverability and 3. Constraints. In order to achieve this, the idea was to make the four tabs above the layer stack serve as a clear indication of which options the user can access, without actually displaying every single feature on the home page. As such, they have clear and unambiguous names, and clicking them leads to different screens with more specific options. The use of labels for every single option should also increase discoverability.

In order to preserve both principles, the list of tabs is always visible when using the program. What changes is the area below the tabs. This way, discoverability is maintained because the user can always see how to access different options in the tab list, but there is still some element of constraint present by limiting what the user can see while each tab is selected.

The first tab is the layer stack, seen in Figure 17, which allows the user to dynamically adjust the number of layers (FUN01, FUN02), as well as changing their position (FUN03, FUN04). It also gives the user the possibility of changing individual parameters within each layer (FUN05, FUN06, FUN07) and altering its recombination type (FUN09) and recombination parameters (FUN10). When a layer is changed, the plot shown in the main menu updates automatically (FUN18).

▼ Parameters	▼ Expression	▼ Unit	▼ Remove
Donor/Acceptor Concentration [Doping]	N1	1/m ³	<input type="button" value="X"/>
Effective Density of States in CB [NC]	2.43e19[1/cm ³]	1/m ³	<input type="button" value="X"/>
Effective density of states in VB	1.34e19[1/cm ³]	1/m ³	<input type="button" value="X"/>
Bandgap energy	2.1[eV]	J	<input type="button" value="X"/>
Electron affinity Brandt	3.2[eV]	J	<input type="button" value="X"/>
Rel. permittivity	7.6		<input type="button" value="X"/>

Figure 19 – Layer Parameters

The available recombination types are Trap-Assisted, Auger, Impact Ionization and Direct Recombination Models.

▼

Figure 20 – Recombination Type

Once the user selects the recombination type in the menu in the previous picture, the list of parameters that appears when they click the “Parameters” button changes.

▼ Parameters	▼
Equation	light IV ▼
Trapping Model	Shockley-Rea ▼
Electron Lifetime	From material ▼
Hole Lifetime	From material ▼
Energy Difference	0[V] ▼

Figure 21 – Recombination Parameters

The second tab allows the user to solve for specific numerical solutions of the model. Figure 21 shows the options available for customizing global parameters (FUN29).

New
Save
Load

Layer Stack

Solve

Global Options

Results

▼ Parameters	▼	
Interface Model	Thermionic ▼	
Va	Value	IV
kT	Value	CV
cp	Value	EIS
Area	Value	IPCE
		TE

Figure 22 – Solve

When the user clicks one of the solution buttons, a window requesting a number of inputs will appear. This data can also be used for Parameter Fitting (FUN27) if the user desires.

▼ Parameters	▼
Voltage Sweep Min Value	<input type="text" value="Value"/>
Voltage Sweep Max Value	<input type="text" value="Value"/>
Voltage Sweep Step Size	<input type="text" value="Value"/>
Illumination	<input type="text" value="Value"/>
Parameter Sweep	<input type="button" value="Edit"/>
	<input type="button" value="Solve IV"/>

Figure 23 – IV Inputs

If they press Solve IV (FUN14), a window with solution progress appears, and when the solution is finished the Results tab is activated. Selecting CV (FUN15), EIS (FUN16) or IPCE (FUN17) shows similar windows, with their respective curves as an added option. In the case of EIS, the user can also see the Nyquist, Boder Arg and Boder Abs curves.



Figure 24 – Results

The user can alter the Solver Method and Termination settings (FUN20). These changes are reflected in any future operation the user does, as well as any currently displayed plots.

▼ Parameters	▼
Method	Nelder-Mead ▼
Optimality tolerance	Value
Maximum number of objective evaluations	Value
Max. Evaluations per Parametric sweep	Value

Figure 25 – Method and Termination

6. Final Interface

This chapter will serve to show what SolCelSim looks like, the changes that were made from the initial mockups and the reasons for those changes. Some examples that were considered important to illustrate the general interface will be shown.

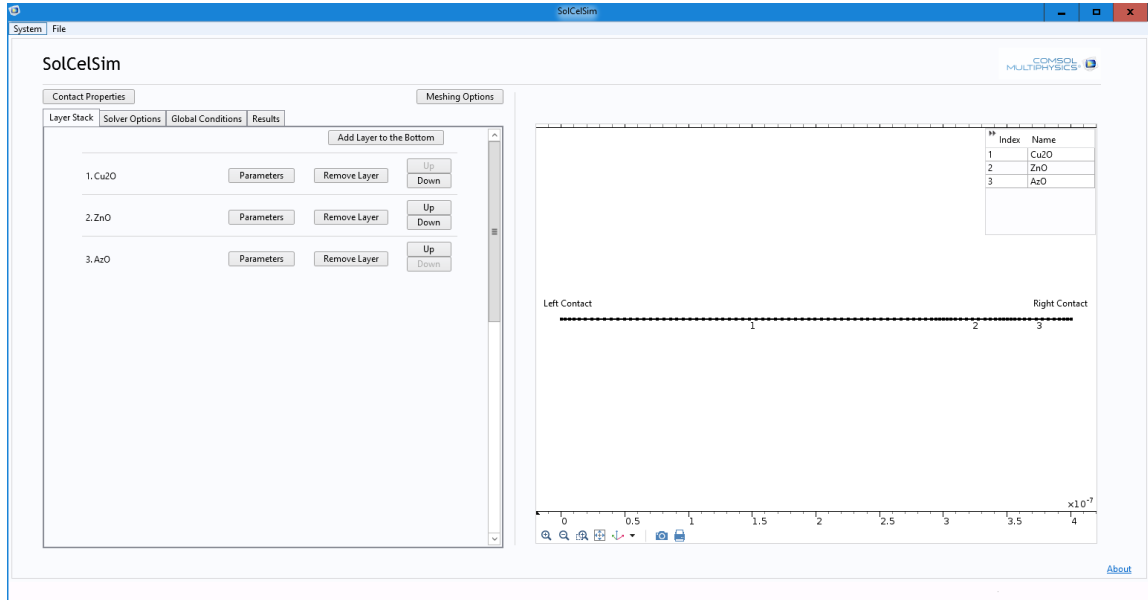


Figure 26 – Main Menu

Figure 26 shows what the main menu of the application looks like. While the general design was kept from the initial mockups, some additions were also made. For example, a list with the current layers and their indices is displayed on the top right corner of the sketch area. Inside that area, the corresponding index for each layer is shown, along with the positions of the right and left contacts. Otherwise, some slight aesthetic improvements were made, such as lines separating the different sections, and logos on top of the menu screen.

This screen serves as the basis of the user's interaction with the application. Clicking the tabs above the layer section changes what appears inside that section, along with the graph on the right panel, but the general structure of the screen stays the same throughout. The next figure will serve to exemplify this behaviour.

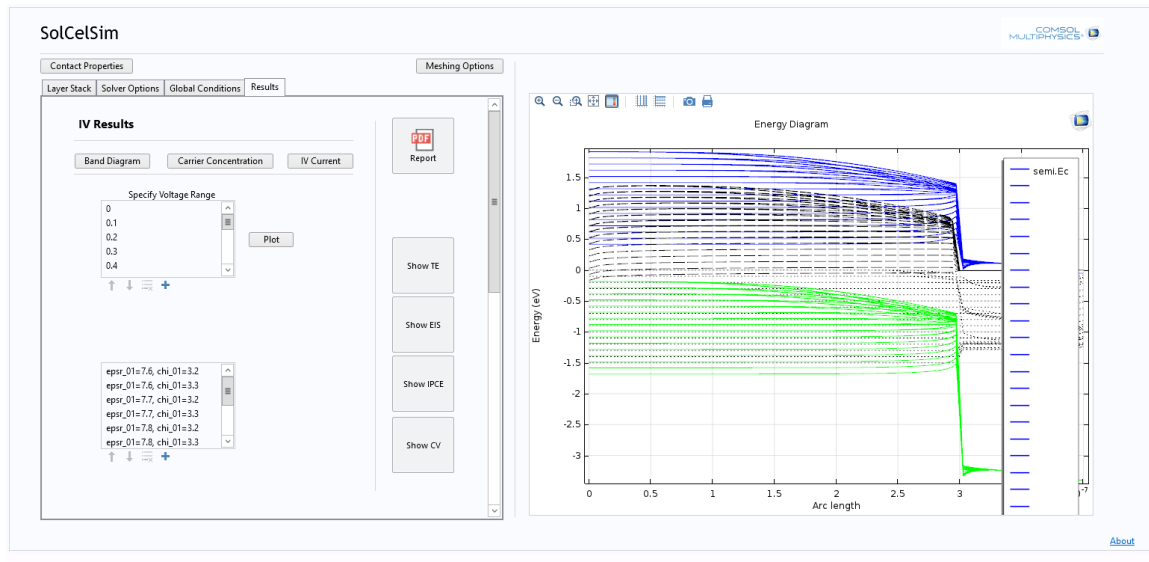


Figure 27 – Results Screen

Figure 27 shows what the application displays when the user presses the “Results” tab. As is possible to see, the general layout of the menu remains the same, but what is displayed on the left and right panel changes appropriately.

This is the area of the interface that has changed the most since the initial mockups were designed. While the initial idea was to have only the graph, plot selection, and “export data” buttons, significant changes were requested by the client in the meantime.

Firstly, the currently displayed plot is written in text above the options, to inform the user of what graph they are currently seeing. If they want to move to a different study, they can press, for example, “Show TE”, and the TE graph will appear, along with the available options regarding TE plots. These buttons do not re-compute the solution, so the user can quickly change between plots without a long waiting time.

Similarly to the prototype, the ability to change between plot types within the same study (in the case of IV, this means Band Diagram, Carrier Concentration and IV Current) is available. When the user clicks on of these buttons, the graph displayed changes, along with the sweep range selections on the left. For some of these plot types, the user is able to import experimental data from a .csv file, in order to compare the real results with the simulated ones.

The previously mentioned sweep range selections allow users to specify which sweep values should be considered for plotting. The user can specify, for example, that they want to show Voltage Sweep value for 0 and 0.1, and the plot will be shown only for those values. While the Voltage Range selection was added early in development, the second table, which allows the user to specify which Parameter Sweep values to plot for, was the last requirement to be added to the project. The two selections work at the same time, meaning that users can select, for example, two Voltage sweep values and three Parameter Sweep values, and the plot will change accordingly. When a solution is computed without Parameter Sweep, the second list is empty and a message will appear informing the user that there are no parameters to show.

Left Contact Properties	Ohmic	
Barrier height	Ideal	10.67[V]
Metal work function	114.5[V]	
Thermionic currents	Surface recombination velocities	
Surface recombination velocity, electrons	121605[m/s]	
Surface recombination velocity, holes	119006[m/s]	
Effective Richardson constant for electrons	1110[A/(K*cm)^2]	
Effective Richardson constant for holes	190[A/(K*cm)^2]	
Right Contact Properties	Ohmic	
Barrier height	Ideal	0.67[V]
Metal work function	4.5[V]	
Thermionic currents	Surface recombination velocities	
Surface recombination velocity, electrons	21605[m/s]	
Surface recombination velocity, holes	19006[m/s]	
Effective Richardson constant for electrons	110[A/(K*cm)^2]	
Effective Richardson constant for holes	90[A/(K*cm)^2]	
		Back

Figure 28 – Contact Properties

Figure 28 serves as an example of the design of the parameter input sections of the application. Here we can see the Contact properties of the simulated solar cell. By default, most of the options are disabled, as the “Ohmic” contact property does not allow for specification of the parameters listed. If the user changes “Ohmic” to “Schottky”, the related options will become available and the user can now edit them at will. Some of the options like “Thermionic currents” also enable and disable certain fields depending on what option is currently selected.

Nonlinear Method	Automatic (Newton)
Initial Damping Factor	0.1
Minimum Damping Factor	1.0E-4
Restriction for Step-Size Update	10
Use Recovery Damping Factor	Automatic
Recovery Damping Factor	0.001
Termination Technique	Tolerance
Maximum Number of Iterations	400
Number of Iterations	1
Tolerance Factor	1
Termination Criterion	Solution
Residual Factor	1000

Figure 29 – Method and Termination

Figure 29 shows the Method and Termination selection page. While similar in functionality to the Contact Properties page, this one has the peculiarity of completely changing which options are displayed. This means that changing “Non-Linear Method” not only disables some options, but completely changes the page to display a different list of options instead.

Figure 30 – IV Solver Options

Like what was listed in requirements FUN20 through FUN22, the user should be able to specify multiple aspects of the solving configuration. This page serves as a “hub” for those options, from where the user can access other more specific features such as the Method and Termination options shown in Figure 29 and the Parametric Sweep options. Pressing “Solve IV” will take the user back to the results section of the main menu, where the results of the computation will be displayed.

Name	Minimum	Maximum	Step Size	Unit
epsr_01	7.6	8	0.1	1
chi_01	3.2	3.3	0.1	eV

Figure 31 – Parameter Sweep Options

If the user chooses to perform parametric sweep, he must enable the option on this screen and select the desired parameters through the menu. The first box displays the list of layers, while the second displays the list of parameters for that layer. For example, if the user selects layer “Cu2O”, a list of every parameter from that layer will show up in the second box. There, the user can select one of those parameters and press “Add” to add it to the list of sweeping parameters. Once the parameter is added, the user can specify the minimum, maximum and step size of the sweep, along with the appropriate unit. This information is initially stored by the app. When the user computes the solution, the application automatically converts this information into a format that the model can recognize and creates the necessary temporary variables. Afterwards, this data is cleaned up from the model but kept in the application.

Recombination Type: Trap Assisted Change Rec. Parameters

Layer Name: Save

Thickness: m

Absorption Coefficient: Browse Save

Name	Value	Description
NDoping	0	"Donor concentration"
NC	2.43e19[1/cm^3]	"Effective density of states in CB"
NV	1.34e19[1/cm^3]	"Effective density of states in VB"
Eg	2.1[eV]	"Bandgap energy"
chi	3.2[eV]	"Electron affinity Brandt"
epsr	7.6	"Rel. permittivity"
mue	200[cm^2/V/s]	"Electron mobility, computed from diffusion length an..."
muh	100[cm^2/V/s]	"Hole mobility, lower estimate in paracchino p.64"
tn	1/Ntd/vthn/sigman	""
tp	1/Ntd/vthn/sigmap	""
Ntd	1e13[1/cm^3]	""
sigman	5e-13[cm^2]	""
sigmap	1e-15[cm^2]	""
vthn	sqrt(3*kT/mn)	"memming p.19"
Ge	IllumIntensity*integrate(alp...	""
mn	NC^(2/3)/2*h_const^2/2/pi...	"eff. electron mass"
mp	NV^(2/3)/2*h_const^2/2/pi...	""
vthp	sqrt(3*kT/mp)	""
PDoping	1e16[1/cm^3]	"Acceptor concentration"
Gipce	alpha1(lambda)*incidentPh...	

↑ ↓ + ≡ 📁 💾 🗑️ 🔄

Save Changes Back

Figure 32 – Layer Parameters

In Figure 32, we can see the section where the user can edit the parameters of one layer. Depending on the layer selected, the parameters displayed will change. This screen looks similar to the one from the initial prototype, but with a few new options. Importing of the absorption coefficient file can be done on the same page, instead of there being a button to take the user to a different page as initially planned. The user can select a different absorption coefficient file for each layer.

Layer names can also be renamed in this section, and the application checks to make sure the new name is valid and not repeated.

The recombination type selection was moved to this section, to allow the individual selection of different recombination types for each layer.

Regarding the parameter list, the user has multiple options besides just individually editing each parameter. With the options below the list, they are able to add or remove parameters, clean the list, save the currently saved list or load a previously saved one into the application.

7. Testing and Quality Assurance

Careful testing is extremely important in a simulation project like this one. It is imperative that results obtained from the simulation correspond to reality as closely as possible. In addition, the user-friendly nature of the software can't be fully realized if the user encounters frequent bugs and faults. Since this project is individual, there could not be a separate team for testing. As such, testing was intercalated with development, meaning that there was a software review process after the development of every task.

Defects had to be detected as early as possible, in order to ensure a smooth development process where the detection of defects didn't interrupt the development of new tasks.

In order to allow us to intercalate development and testing in this manner, a number of testing techniques, listed below, were used:

- **Boundary Value** – Black box testing (tests designed previously based on requirements, not code) technique. Consists of specifically testing the values near the boundaries of the allowed ranges for each parameter. This technique was useful since parameterization was very important in this project.

- **Regression testing** – Previous tests were run after the implementation of each new feature, ensuring that the new code is not interfering with past work.

- **Maintainability Testing** – Time required to diagnose and fix faults was recorded.

In addition, other non-functional tests such as usability, acceptance, exploratory, negative and accuracy testing were also important in order to obey the non-functional requirements of this project. More information about usability tests is available in subsection “5.4 Design Walkthrough” of this report.

The next section will go into detail regarding how Quality Assessment was performed, divided into different subsections for the types of tests performed.

7.1 Alpha Testing

Alpha testing was performed to try and identify issues and defects within the application. In order to achieve that, the application was made available to a small number of individuals within the research group. Through regular use, some problems were identified, sent to the developer of the application and subsequently fixed. While the main objective of alpha testing was to find and fix bugs within the code, small design changes were also made when appropriate.

7.2 Acceptance Testing

Acceptance testing was performed by the client, in order to make sure that the application fulfilled all of the requirements. The software was considered accepted once all the desired features had been implemented and worked as expected.

7.3 Accuracy Testing/Boundary Value Testing

These two types were performed simultaneously and, thus, are included in the same subsection.

For this specific application, it is not only important that the application behaves correctly and without crashes for valid inputs, but the results obtained should accurately reflect the simulations ran in the model.

As such, a list of valid inputs was created, based on the boundary level of the inputs the user can set. For example, in values that must be positive, inputs near zero were used for testing.





Afterwards, the results obtained by the application were compared to the results obtained by simulating the underlining model by itself; and it was considered that tests were passed when the results obtained from both were within 1% or less from each other.

All the inputs and outputs obtained, along with comments about the accuracy of the results, were written and stored in a .xlsx format file.

Section	Parameter	Input 1	Input 2	Input 3	Result 1	Result 2	Result 3
New Layer	Name	Test	Layer	123			
	Thickness	5.00E-08	5.20E-08	3.00E-07			
	Donor Doping	1.00E+20	1.20E+20	0			

Table 6 – Accuracy Testing Example

Table 6 has a very small example of some of the data obtained from the tests and present in the previously mentioned Excel file. These tests were repeated for every parameter within the application, with multiple inputs. The results are represented by a different colour depending on the expected behaviour and the real behaviour. This color system is explained by the label below:

	Positive Result, Equal Result to Model
	Expected Negative Result, Equal To Model
	Positive Result, Different from Model
	Negative Result, Different from Model

7.4 Exploratory Testing

After the development of each new feature, a period of exploratory testing was performed.

This involved exploring the application and experimenting with different features in the search for potential defects. Generally, the focus during these tests would be in the most recently implemented feature, although older features were also tested at the same time in order to make sure that new updates did not break previous code.

7.5 Negative Testing

This technique was used to verify the behaviour of the application when the user inputs incorrect data. In the specific case of this application, this meant writing inputs that, while having the correct data type and being within acceptable boundaries, cannot generate a valid physics model and thus can't be computed by Comsol.

The expected behaviour in these circumstances is that the application accepts the inputs but gives the user an error message when he tries to perform one of the “Study Solver” options. When this happens, the “Results” section of the application is changed to represent the fact that the current parameters should be changed. If the user tries to compute the “Study Solver” again with correct parameters, the application will display the results correctly.





A second phase of negative testing was also performed. These second tests focused on invalid, rather than incorrect inputs. This means ridiculous values, such as String inputs in Integer input boxes, long sets of non-alphanumerical characters, and extremely high numbers that are far outside of what should be expected for each parameter.

The inputs and outputs obtained in both phases were saved in the same file .xlsx mentioned in subsection 7.3., each in a separate tab. Comments and appropriate labels were added, comparing the results obtained with the expected behaviour. In cases where the application behaved differently than expected, the defect was identified and fixed, and the tests were repeated.

Parameter	Input 1	Input 2	Input 3	Result 1	Result 2	Result 3
Name	<empty string>	a <100 times>	a123<123 repeated 20 times>	Invalid Parameter Inserted	Invalid Parameter Inserted	Invalid Parameter Inserted
Thickness	\n\r@"!=#\$) `*a\r\n	Letters	<empty string>	Invalid Parameter Inserted	Invalid Parameter Inserted	Invalid Parameter Inserted
Donor Doping	\n\r@"!=#\$) `*a\r\n	Letters	<empty string>	Error while computing	Error while computing	Invalid Parameter Inserted

Table 7 – Negative Testing Example

Table 7 contains a small example of some of the data obtained from Negative Testing. This table is generally analogous to Table 6, with the difference that the inputs chosen were intended to be invalid and force the application to return an error. As the following label shows, the results obtained in this example show the application behaving as expected, and recovering correctly from the forced errors:

	Positive Result
	Expected Negative Result, Recovered Correctly
	Expected Negative Result, Did Not Recover
	Unexpected Negative Result

7.6 Usability Testing

Early in development, before implementing most of the methods related to the backend of the application, usability tests were designed and performed.

In order to do this, a small prototype was developed, and users were asked to perform a set of tasks.

Only two users participated in these tests. While this number is quite low, and thus results can't be fully generalized, they helped give us an idea of the usability of the application. User 1 had some general idea of how the application functioned before the tests began, while User 2 was completely unfamiliar with it. User 1 was also more familiar with aspects of solar cell simulation in general than User 2.

The users were monitored during this time. The time they took to perform these tasks, along with the steps they took, were analysed.

The following list shows the list of tasks required, in order:

1. Add a new layer. Move it up and then down. Remove it.
2. Add a new layer parameter to any layer and save your changes. Remove it and save your changes.
3. Set Right Contact Properties to Schottky. Set Metal Work function to 1. Set Right Contact Properties back to Ohmic.
4. Set any Layer Recombination Properties to Auger. Change Electrons Recombination Factor to any value. Set it back to Trap Recombination.
5. Solve for IV.
6. Set Meshing Options Sequence Type to Physics Controlled.
7. Close the application.

After the users performed the tasks listed, they were asked for their thoughts regarding the design of the user interface, and asked to answer 10 questions from the System Usability Scale (SUS), a usability measuring tool created by John Brooke¹⁵:

1. *I think that I would like to use this system frequently.*
2. *I found the system unnecessarily complex.*
3. *I thought the system was easy to use.*
4. *I think that I would need the support of a technical person to be able to use this system.*
5. *I found the various functions in this system were well integrated.*
6. *I thought there was too much inconsistency in this system.*
7. *I would imagine that most people would learn to use this system very quickly.*
8. *I found the system very cumbersome to use.*
9. *I felt very confident using the system.*
10. *I needed to learn a lot of things before I could get going with this system.*

The questions were made available in an online questionnaire, where they were presented on a Likert scale with five possible values for each. Something to note is that the questions in the SUS were designed with some having a more positive connotation, while others have a more negative one. This gives us more confidence in the user's responses, as they can't simply select 1 or 5 for every answer, forcing them to think about each individual question.

¹⁵ <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>

Task	User 1	User 2	Average
1	01:03	03:06	02:04
2	01:31	02:51	02:11
3	00:36	01:42	01:09
4	00:47	01:23	01:05
5	00:13	00:48	00:30
6	00:30	00:38	00:34
Total	04:40	10:28	07:34

Table 8 – Task Time to Complete per User

In Table 8, we can see the time each user took to complete each task. The expected average for each task was around 5 minutes, so the results were significantly better than expected. With both users, it's possible to note that there was a short period of getting used to the application, which explains why times for the first tasks were higher than times for later tasks. As expected, the more experienced user (User 1) performed the tasks in a lower amount of time.

Question	User 1	User 2	Average
1	3	3	3
2	3	4	3.5
3	3	4	3.5
4	4	2	3
5	2	3	2.5
6	3	4	3.5
7	4	3	3.5
8	4	4	4
9	3	3	3
10	4	3	3.5
Total Rank	82.5	82.5	82.5

Table 9 – Questionnaire Results Rankings

In Table 9, we can observe the rankings for the answers given by the users in the Questionnaire. It's important to note that the numbers in the table don't correspond directly to the ratings given by the users. Since in the System Usability Scale questions odd numbered questions are positive and even numbered questions are negative, some adaptations have to be made before analysing results. In order to obtain the SUS Usability Score, three steps are taken. First, the score of odd numbered questions is subtracted by one. Then, the score of even numbered questions is subtracted from 5. Finally, the sum of every score is multiplied by 2.5. With this, we obtain the SUS score, on a scale from 1 to 100. In this case, the score for both users was 82.5, which can be considered very positive.

The results obtained from these usability tests were very positive. The time users took to perform each task was significantly lower than expected. As such, only some very minor changes were made to the user interface based on feedback received from the user's comments, and the final UI design was accepted, allowing proper development of the application to begin.

7.7 Others

There are a few other aspects of the testing and quality assurance part of this project that are worth mentioning.

Regression tests were used to make sure further development did not break previously implemented parts of the project.

Since one of the functional requirements of this project was Maintainability, Dynamic maintainability testing was performed by measuring the time needed to perform corrective maintenance. These tests revealed that the application fulfils the requirement. Similarly, extensibility was tested by measuring time required to implement new features, also showing positive results.

8. Challenges

This chapter will go into detail about what challenges emerged during the development of this project, and how it was possible to deal with them and achieve positive results.

Layer Stack: The management of the layer stack is not a trivial problem in this project. The physical concept of a solar cell layer is represented by multiple different parts of the model.

Some of the defining characteristics of each layer are the “Variable” and “Interval” parts of the model. The former stores the parameters of the layer, taking into account the data stored in the Absorption Coefficient file. The latter stores the thickness and position of each layer, and represents it graphically. For example, the interval for “Layer 3” would start at d_1+d_2 and end at $d_1+d_2+d_3$, where d_X represents the thickness of Layer X. Consequently, the interval for “Layer 4” would start at $d_1+d_2+d_3$ and end at $d_1+d_2+d_3+d_4$.

Each interval defines a “domain”. The second big challenge related to the layer stack is creating the correct semiconductor properties and associating them with the correct domains. For example, if we have two layers in the model and the user wants to create a third one, we need to create the semiconductor property “Trap Assisted Recombination 3”, and associate it with Layer 3. This process needs to be repeated not just for recombination, but for all the semiconductor properties, including the “Doping Model” and “User Defined Generation”, among others.

With the creation of a new layer, the application needs to detect what the current highest layer number is, create a new absorption coefficient with a new associated table, a new interval, a new variable, and all the necessary new semiconductor properties (which must be associated to the new domain). The domains of certain properties, such as the “Right Metal Contact”, need to be adjusted to include only the new highest layer.

The problem then becomes more complicated when it involves removing and moving layers. For example, if we have 4 layers, and the user tries to remove Layer 2, we can’t simply remove that layer and leave the rest of the model as is. If we did that, then the model would jump from Layer 1 to Layer 3 and the simulated solar cell would have a “hole” where the second layer used to be.

Thus some adjustments need to be made automatically. Generally for removal, parts associated to layers of higher number need to be “moved” accordingly. Using the duplication functions of the Comsol API, we iteratively remove layers and copy the contents of the subsequent layers into the previous positions. Domains are then adjusted for every part of the model, including every semiconductor property of every layer. When moving layers up and down, the application also needs to create temporary objects to store the properties of the layer being moved before making the rest of the adjustments.

Parameter Sweep: A “Parameter Sweep” object, along with its solution and job configurations, is kept in the model at all times, with one for each different solver. The user can disable and enable it through the application’s user interface.

It was requested that users are able to perform parameter sweep on individual parameters from each layer. However, Comsol only allows users to sweep *global* parameters. As such, some adjustments need to be made whenever the user runs a solver with Parameter Sweep enabled.

The best solution that was found was to keep Parameter Sweep values inside the application. When the user tries to compute a solution with Parameter Sweep enabled, temporary global parameters are created based on the sweep values (for example, if “epsr” from Layer 1 is a sweep value, a temporary global parameter “epsr01” is added). Then, the layer parameter associated with that value is properly adjusted (in the previous example, layer parameter “epsr” would get “epsr01” as its value). With all of this done, the Parameter Sweep object can now be filled with the information entered by the user. The name of the parameter is added to the sweep, followed by an automatically generated “range(X,Y,Z)” string, where X, Y and Z are sweep values input by the user, and finally the parameter unit.

When these preparations are complete, the solution is computed, and table datasets are adjusted to display the correct results. Some Boolean variables in the application are also updated so that SolCelSim “remembers” to show the appropriate sweep results in the future, even if the user interrupts their work. The second table in the left side of the Results tab of the application then allows users to select which sweep values to plot for.

At the end of computation, or if the computation is interrupted by the user or an internal error, the temporary global parameters that were stored earlier are deleted from the model, but kept in the application.

Error Handling: Initially, when a computation for any solver was interrupted, certain tables in the user interface would point to data from an unfinished computation and, as such, would retrieve null values and break the normal functioning of the application. As an answer to this problem, error detection was implemented in areas that involved solver computation. When an error is detected, certain parts of the application are automatically adjusted. The used dataset for the failed solution is changed to a “Safety Solution” dataset, which the user cannot interact with, and serves simply to give the tables and graphs in the Results section a valid data source. The user doesn’t actually see the tables associated to this dataset, as an error message is displayed instead of the Results page when the last computation of a certain Solver has failed.

This allows SolCelSim to recover effectively from any failed or interrupted computation. Important to note is that this doesn’t affect successful computations. For example, if the IV Solver fails, an error message is shown on the IV Solver Results page, but the user can still access TE, EIS, IPCE or CV Solver Results pages normally. If the user then performs a correct computation of the IV Solver, the error message is removed, datasets are reverted to their regular values, and the user can continue using SolCelSim as usual.

Others: Some features such as the Results Sweep selection, Heterointerfaces and Absorption Coefficient file import required some deep understanding of the Comsol API documentation and some creative thinking. Since the Comsol Application Builder limits what programmers can directly interact with in some regards, especially when it comes to the User Interface, some problems that initially seemed trivial ended up requiring a significant investment of time and some unconventional thinking to be achieved in spite of those limitations.

Conclusion

Future Work

Something that was noted near the end of development was that the application had a bigger filesize than expected and took relatively long to start up. Performance wasn't initially considered one of the non-functional requirements of the project, and this problem was only noticed during the last few weeks of the project. As such, there wasn't enough time to optimize both the size of application and its start up time. While some small changes were made to try and combat the problem, any more significant changes would require a long period of development, followed by regression tests to make sure previous features hadn't been broken by the changes. Because of this, there is the possibility that future work can be done in the area of optimization.

One feature that was cut was the use of the Comsol Server, due to the lack of licence. If a Comsol Server licence becomes available to the university, then the application can be uploaded to the server and then accessed either from a Comsol Client application or from any Web Browser.

As needed, new features that researchers find important to simulating solar cells can be added. In order to make sure different developers are able to continue work on the application and add those features, extensibility was made a priority. SolCelSim's architecture being friendly to the addition of new modules, along with the detailed comments throughout the entire code, should make future developers' jobs easier in this regard.

Final Comments

The first semester of this project was focused mostly on planning and research. In order to ensure a smooth development process, it was important that a work schedule estimate was created, to serve as a general guideline. There was also extensive study done on state of the art, software design methodologies, human-computer interaction and quality assurance. Developing a good set of design principles, and a solid methodology prevented the necessity of wasting development time fixing mistakes that could have been avoided from the start.

The second semester encompassed the development phase, while leaving a few weeks for the addition of new features and quality assurance. There was also some time at the end dedicated to the writing of the final internship report. New requirements were added throughout the second semester, according to the client's wishes, ensuring that all of the development time was used efficiently, and avoiding the situation where estimates are more strict than necessary and all tasks are completed before expected.

While both the initial requirements and all the added requirements were fulfilled, there is still space for future work. As such, the application's code includes very detailed comments, so that future developers can continue working on the project and add new features as necessary. Overall, proper extensibility was made a priority to facilitate future work on SolCelSim.

From the intern's point of view, this project was extremely valuable as a way to learn about proper User Interface design, the use of SCRUM methodology and Quality Assurance testing.

The resulting application is available to all users with a Comsol licence since June 2019.

Bibliography

- [1] Jia, J. *et al.* Solar water splitting by photovoltaic-electrolysis with a solar-to-hydrogen efficiency over 30%. *Nat. Commun.* 7, 13237 doi: 10.1038/ncomms13237 (2016).
- [2] Cendula, P. *et al.* Calculation of Energy Band Diagram of a Photoelectrochemical Water Splitting Cell, Zurich University of Applied Sciences (2014) (<https://arxiv.org/pdf/1407.5774.pdf>)
- [3] Delgado, E. et al. From solar to hydrogen energy: Modelling, design, and construction of a system for hydrogen production using photovoltaic panels, Delft University of Technology (2018)
- [4] Dumortier, M. et al. Holistic design guidelines for solar hydrogen production by photo-electrochemical routes, *Energy Environ. Sci.*, 2015, 8, 3614
- [5] Ager, J. et al., Experimental demonstrations of spontaneous, solar-driven photoelectrochemical water splitting, *Energy Environ. Sci.*, 2015,8, 2811-2824
- [6] Bonkge, S. et al, Renewable fuels from concentrated solar power: towards practical artificial photosynthesis, *Energy Environ. Sci.*, 2015,8, 2791-2796
- [7] Nakamura, A., A 24.4% solar to hydrogen energy conversion efficiency by combining concentrator photovoltaic modules and electrochemical cells, *Applied Physics Express*, 8, 10
- [8] Jong, W.A., PEC versus PV - E A Future Potential Comparison, Utrecht University

Attachment 1 – Comsol Explanation

Comsol MultiPhysics

Comsol MultiPhysics is a cross-platform physics simulation software. While the “core” of the program is its Model Builder, version 5.0 introduced the new Application Builder, which allows the creation of desktop apps with Java. The Application Builder was the main focus of this project, although some interaction with the Model Builder was necessary to ensure the application works properly.

Comsol Model Builder

Figure 33 shows the Comsol Model Builder. This mode allows the user to easily make changes to the model using its interface. It allows the creation of complex Multiphysics models.

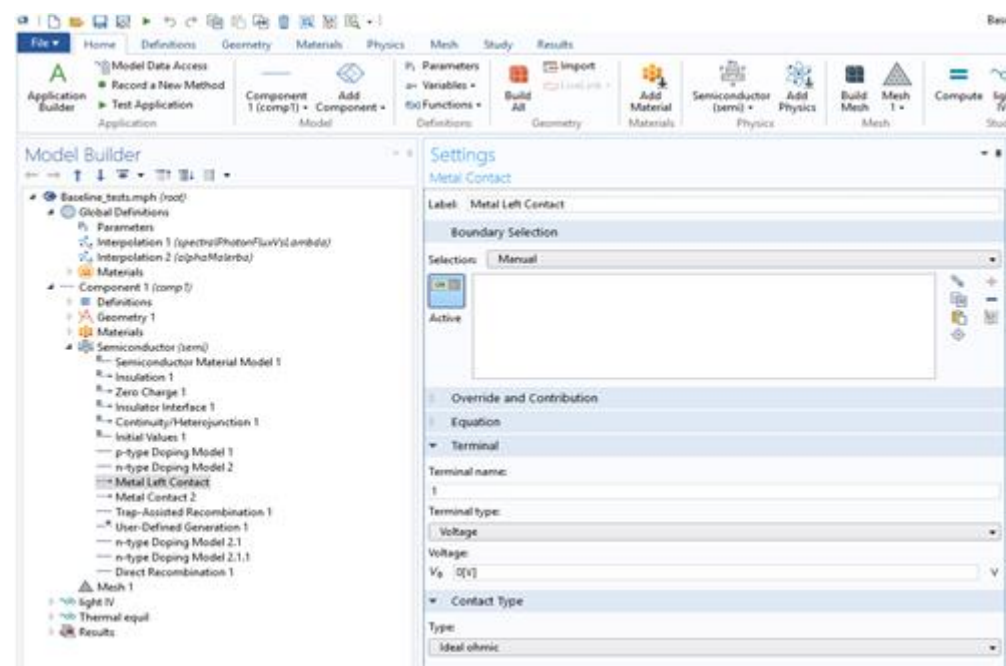


Figure 33 – Model Builder

Comsol Application Builder

Pressing the A button of the top left corner of the Model Builder takes the developer to the Application Builder. This takes us to the programming side of things, where we can write methods that activate when the user presses a specific button or enters a specific page.

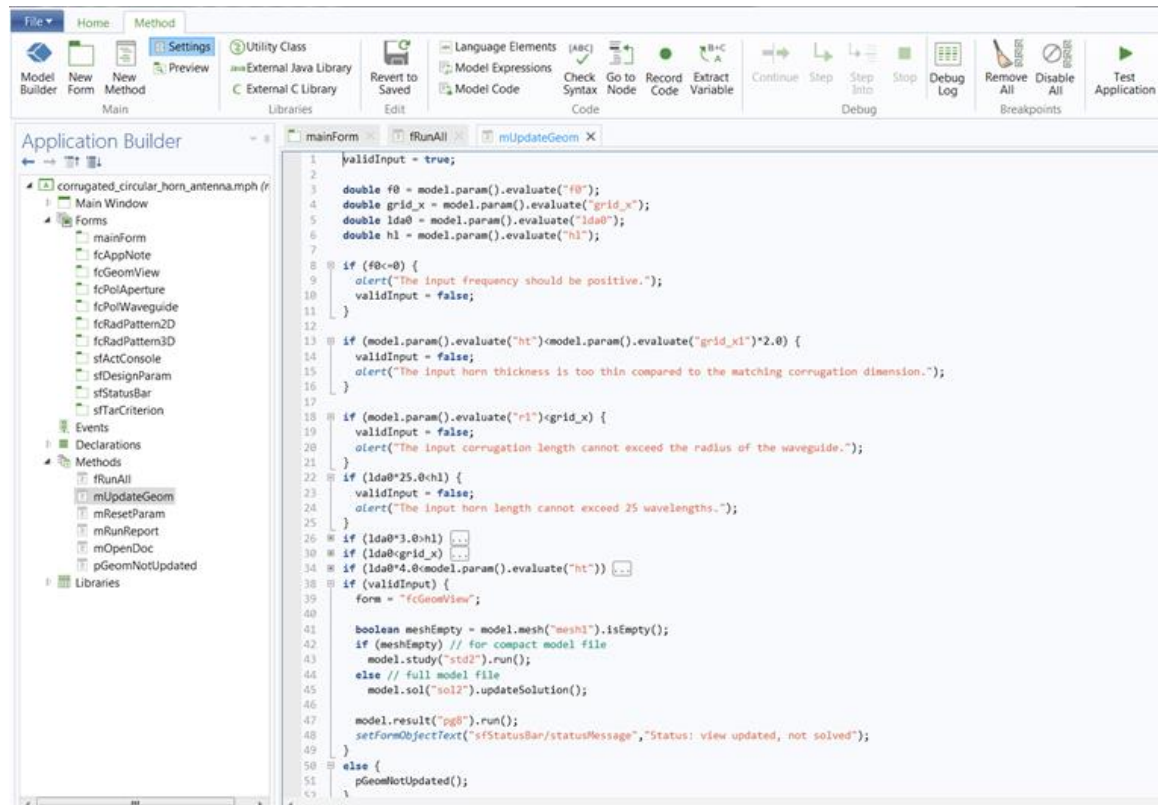


Figure 34 – Application Builder

Figure 34 shows a generic example of what a common screen will look like when interacting with the Application Builder. Pressing “Test Application”, on the top bar, allows the developer to immediately test the changes they’ve made.

In this piece of code, the keyword ‘model’ can be seen often. This is used for all interactions with the model which we saw in the Model Builder part of this document. For example, the first few lines get parameters from Global Definitions -> Parameter, which we could see on the left toolbar of Figure 33.

Features mentioned in the Requirements Document were implemented in this Application Builder.

It’s worth noting that the examples in this section serve purely as a demonstration of the functioning of the Application Builder, and are not representative of the final state of the application. Some code displayed here might thus be outdated.

Example – Add Layer

While learning how to work on the Application Builder, an initial version of the method for addition of layers was written. As a way of explaining how working with Comsol works, a step-by-step explanation of the implementation of this feature should be helpful.

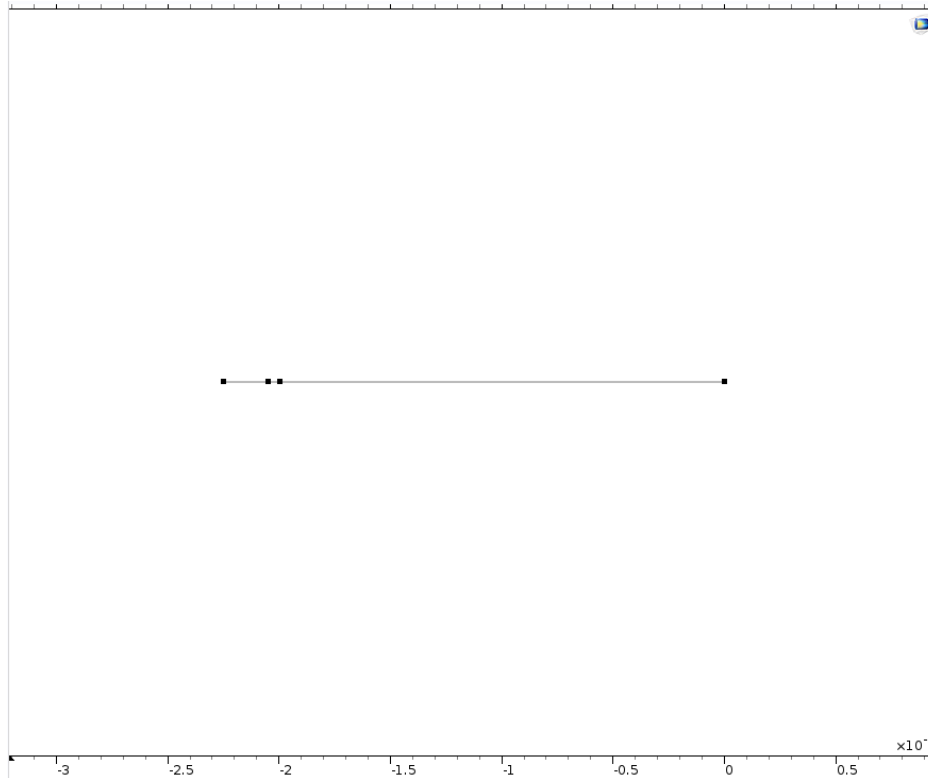


Figure 35 – Layers

First, this is a graphic that shows the layers present in the model. Each of the dots represents the limits of one layer. Each layer has a different number of parameters that affect its interaction with the other layers. Thus, any small change to each layer will completely change the graphic. As such, whenever a layer is added, a number of changes have to be made to the model, which will be shown next.

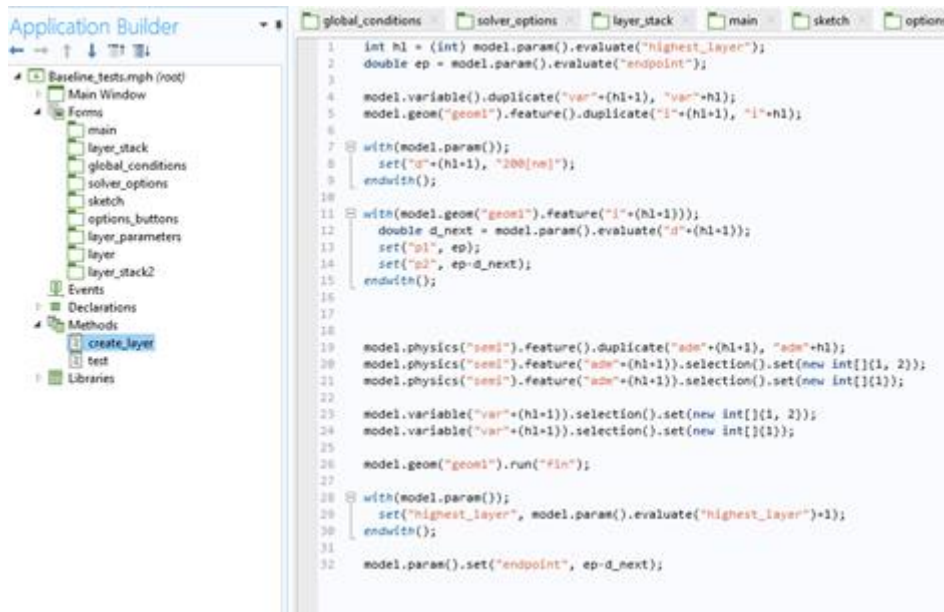


Figure 36 – Add Layer

Next, we will take a look at each part of this code, step by step, along with what changes are made automatically to the model.

```
int h1 = (int) model.param().evaluate("highest_layer");
double ep = model.param().evaluate("endpoint");
```

highest_la...	4	4	
endpoint	-d1-d2-d3-d4	-2.251E-6 m	

Figure 37 – Getting two parameters

These lines are basically Getters, obtaining the current highest numbered layer and its distance from the lowest numbered layer.

```
model.variable().duplicate("var"+(h1+1), "var"+h1);
model.geom("geom1").feature().duplicate("i"+(h1+1), "i"+h1);
```

Variables

- a= Cu2O
- a= ZnO
- a= Cu2O 1
- a= Cu2O 1.1

Figure 38 – Duplicating a default layer

These lines duplicate a default layer, creating the new layer that we will edit next. In this case, its number is an increment over the current highest numbered layer.

```
with(model.param());
  set("d"+(h1+1), "200[nm]");
endwith();
```

d4

200e-9

2E-7

AZO

Figure 39 – Adding Distance Value

Assuming the layer added is Layer 4, a new parameter, d4, is added with its distance from the previous layer.

```
with(model.geom("geom1").feature("i"+(h1+1)));
  double d_next = model.param().evaluate("d"+(h1+1));
  set("p1", ep);
  set("p2", ep-d_next);
endwith();
```

- Geometry 1
 - Interval 1 (*i1*)
 - Interval 2 (*i2*)
 - Interval 3 (*i3*)
 - Interval 4 (*i4*)
 - Form Union (*fin*)

Settings

Interval

Build Selected ▾ Build All Objects

Label: Interval 4

Interval

Number of intervals: One ▾

Left endpoint: -d1-d2-d3 m

Right endpoint: -d1-d2-d3-d4 m

Figure 40 – Setting Endpoints

Based on the value previously added, d4, the written code automatically sets the new left and right endpoints for the new layer. In this case, the left endpoint would be ep (the previous highest endpoint) and the right endpoint would be ep – d4. In practice, we will be able to see, in the graphic, that this new layers starts at the point where the previous layer ended, and ends at that point minus d4 (200 nm, as defined in Figure 38).

```

20 model.physics("semi").feature().duplicate("adm"+(h1+1), "adm"+h1);
21 model.physics("semi").feature("adm"+(h1+1)).selection().set(new int[]{1, 2});
22 model.physics("semi").feature("adm"+(h1+1)).selection().set(new int[]{1});
23

```

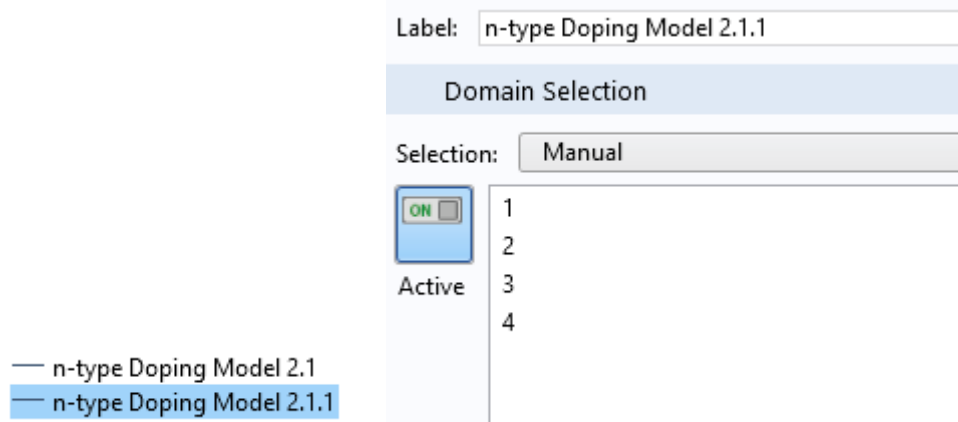


Figure 41 – Adding New Doping Model

Each layer has an associated doping model. This part of the code creates a new one, with the parameter values of the default doping model. In addition, since every layer has an associated Domain, it also adds the new layer's domain to the model.

```

24 model.variable("var"+(h1+1)).selection().set(new int[]{1, 2});
25 model.variable("var"+(h1+1)).selection().set(new int[]{1});
26

```

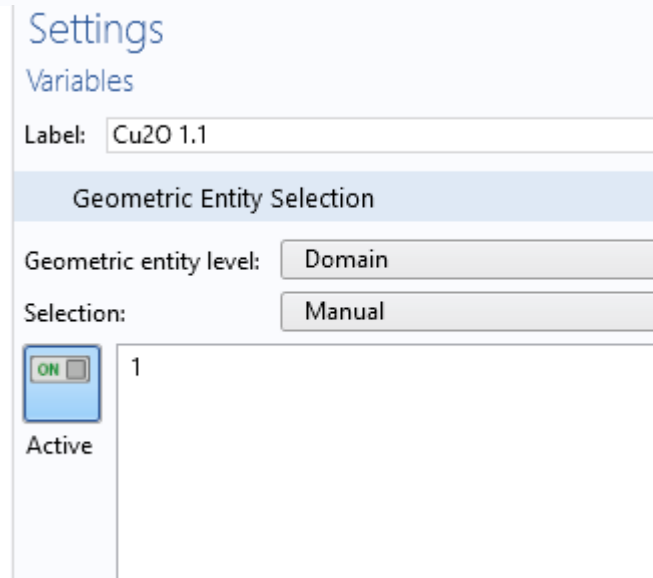


Figure 42 – Domain

The correct domain is then associated with the new layer.


```
model.geom("geom1").run("fin");  Build All
```

Figure 43 – Build

Builds the whole geometry so the graphic can be updated automatically.

```
with(model.param());
  set("highest_layer", model.param().evaluate("highest_layer")+1);
endwith();

model.param().set("endpoint", ep-d_next);
```

highest_la...	4	4	
endpoint	-d1-d2-d3-d4	-2.251E-6 m	

Figure 44 – Updating Global Parameters

This is used for internal purposes within the application. The number of the highest layer and its right endpoint are recorded, so they can be used when the user decides to create a new layer.

This section showed how to add a single feature using the Comsol Application Builder. It should hopefully give the reader a more intuitive idea of how Comsol works, and what can be done in the Application Builder.

Attachment 2 – Solar Panel Software

Purdue University has also developed a MatLab model¹⁶ for a-Si solar cells, as well as two other simulation tools, PV Analyser and PVPanelSim.

PV (Photovoltaic) Analyser¹⁷ provides “rapid data analysis and parameter extraction for solar cell measurements”. It extracts diode and shunt parameters from the dark IV characteristics of a solar cell. All the data and fit parameters can be later downloaded as text files.

PVPanelSim¹⁸ provides “two-dimensional SPICE (Simulation Program with Integrated Circuit Emphasis) simulation of thin-film solar panels”. The user can see the effects of partial shadowing caused by objects nearby. It can show various plots, such as the IV curve, PV cuve, 2D node voltage drop and 2D power generation

PVPlanner¹⁹ is an older application (from 2010) that allows users to estimate energy loss, performance and efficiency of the system they design. It simulates two-dimensional thin film solar panels. Losses due to terrain shading, snow, dirt, among others, are considered in the simulation.

PVPlanner boasts ease-of-use as one of its strong points, without disregarding accuracy of results. Since this is also one of the goals of the current project, PVPlanner’s design will be used for inspiration, with some of its ideas being implemented whenever possible.

¹⁶ <https://nanohub.org/publications/20/1>

¹⁷ <https://nanohub.org/resources/pvanalyzer>

¹⁸ <https://nanohub.org/resources/pvpanelsim>

¹⁹ <https://solargis.com/products/pvplanner/overview/>

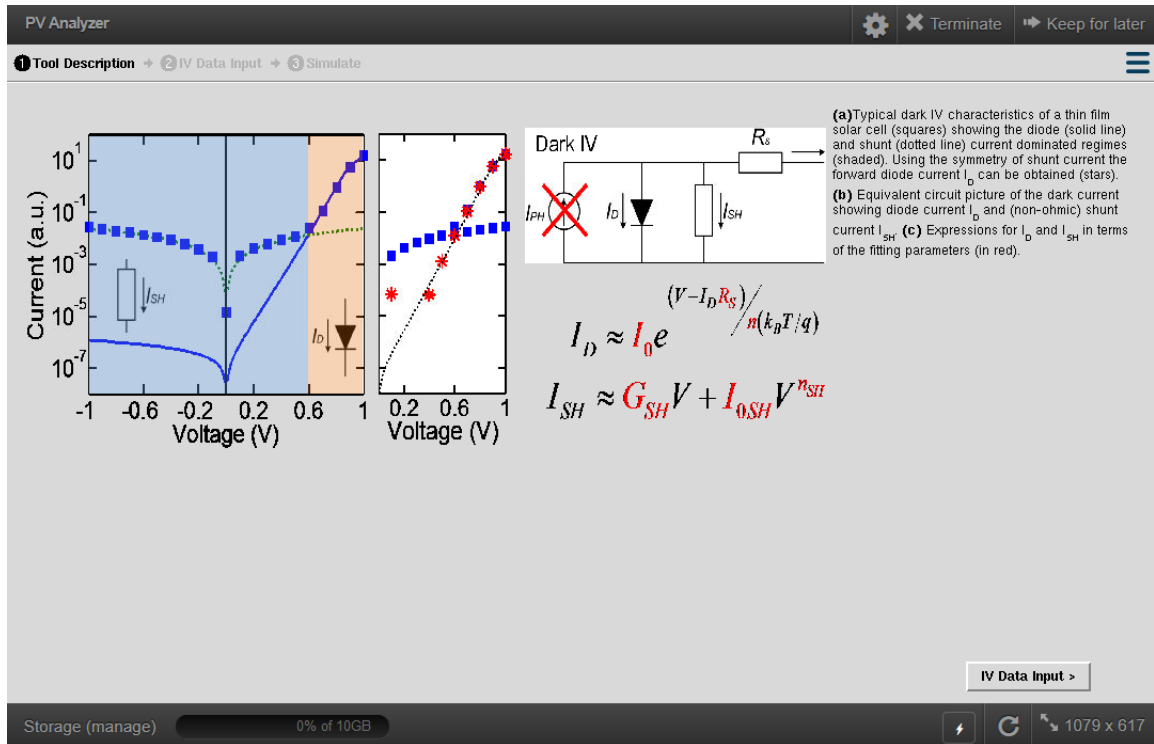


Figure 45 - PV Analyser

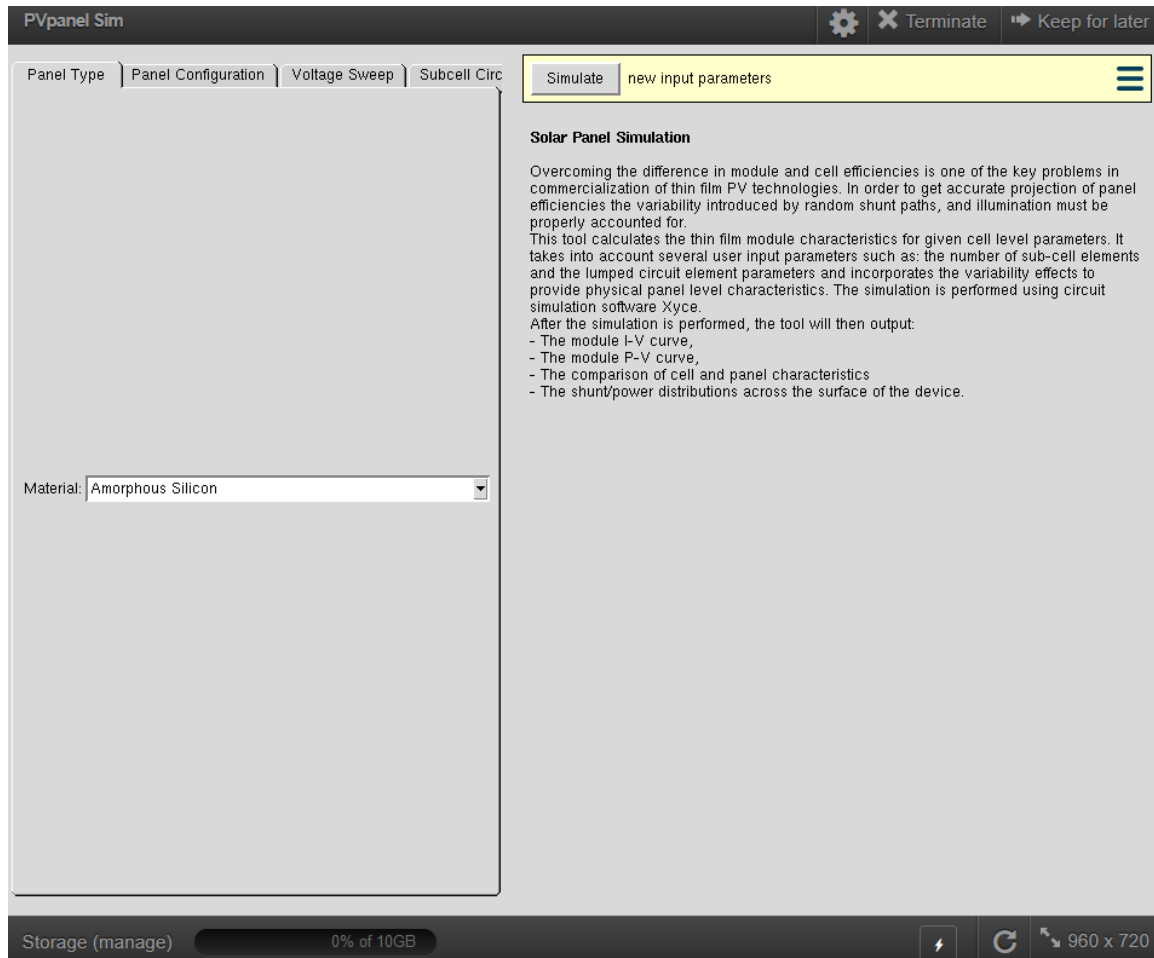


Figure 46 - PV PanelSim

The Solar Energy Simulator (SES)²⁰, despite having a somewhat different overall purpose, also has some ideas that can be applied to the current project. Particularly, the ability to add new systems and appliances and have the results change accordingly is similar to what we want to achieve. This application, however, is most useful for people who are interested in using solar energy to power their own house, for example. Data about electrical consumption and costs can be quickly obtained. It is also able to accurately model battery life, and generate a variety of reports and analysis documents.



Figure 47 – SES

System Advisor Model (SAM)²¹ can estimate performance and energy costs, based on where the user wants to install the system, with the ability to generate a large number of reports. It was developed by the National Renewable Energy Laboratory of the United States in 2005. It is generally not as simple to use as other examples in this section.

Its main purpose is to help facilitate decisions in the renewable energy industry, by predicting performance and energy costs based on the user inputs.

After a simulation, SAM can show metrics such as first year annual production, detailed annual cash flow and hourly performance data. It also allows the user to directly compare multiple different projects. The reports and tables generated by this application are extremely in depth. There is also a custom scripting language available for the user to write their own scripts within the user interface.

Different performance models and financial models with their own inputs and outputs are included within the program, and used for performing calculations.

²⁰ <http://www.hybridsunshine.com/solar-energy-simulator/>

²¹ <https://sam.nrel.gov/download>

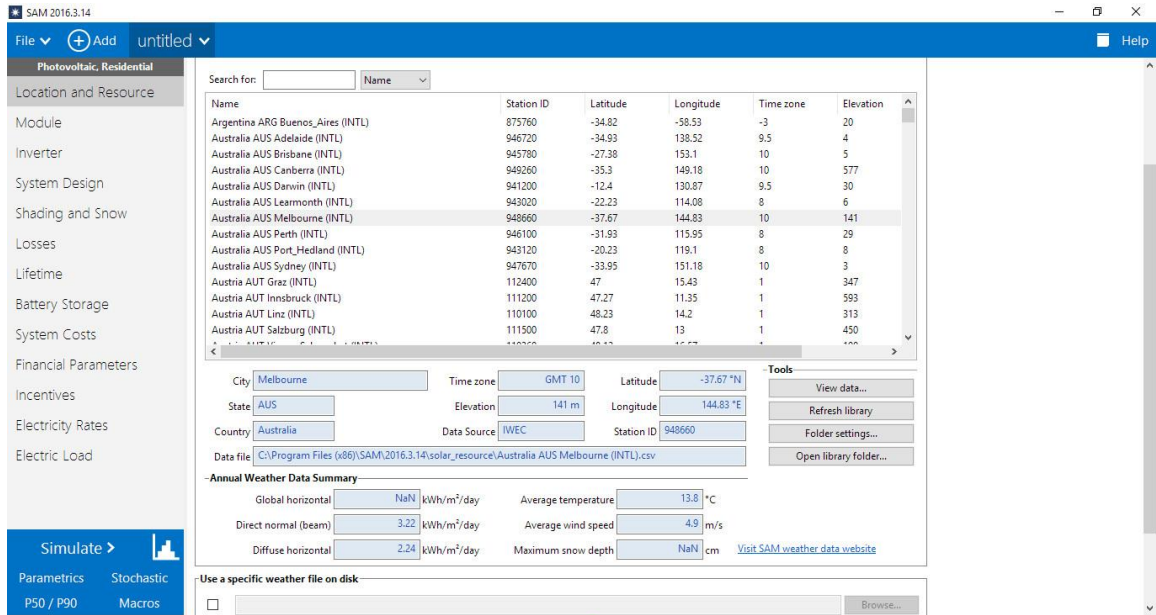


Figure 48 - System Advisor Model

Solar Pro²² offers a 3D visualization of the system, with changes being computed and displayed in real time. Some of its advantages are a user-friendly UI, data from weather stations around the world and accurate formulas. These qualities make it a very strong option as far as PV solar panel simulation goes.

The following chart summarizes the difference between the different software analysed:

Software Name	Price	Report Generation	Weather Data	3D Plots
PVAnalyser	Free	Yes	None	No
PVPanelSim	Free	No	None	No
PVPlanner	\$560 to \$3600	Yes	Monthly and Annual Satellite Data	No
SES	Free	Yes	Solar Irradiation data from 2000 to 2013	No
SAM	Free	Yes	Data from the National Solar Radiation Database of U.S.	Yes
SolarPro	\$1300 to \$2050	Yes	Hundreds of Weather Stations	Yes

Table 10 – Related Work Comparison

²² www.lapsys.co.jp/english/products/where_to_buy/index.html

Looking at Table 10, there are a few important ideas we can take. First off, the choice to make the application free seems to be the correct one, as the programs most similar to the one we're trying to build are also free, and raising the price would create an unnecessary barrier. From looking at the report generation feature in the listed software, it was decided that the possibility to generate a report describing results from simulations would be important and serve an advantage over GPVDM and PVPanelSim.

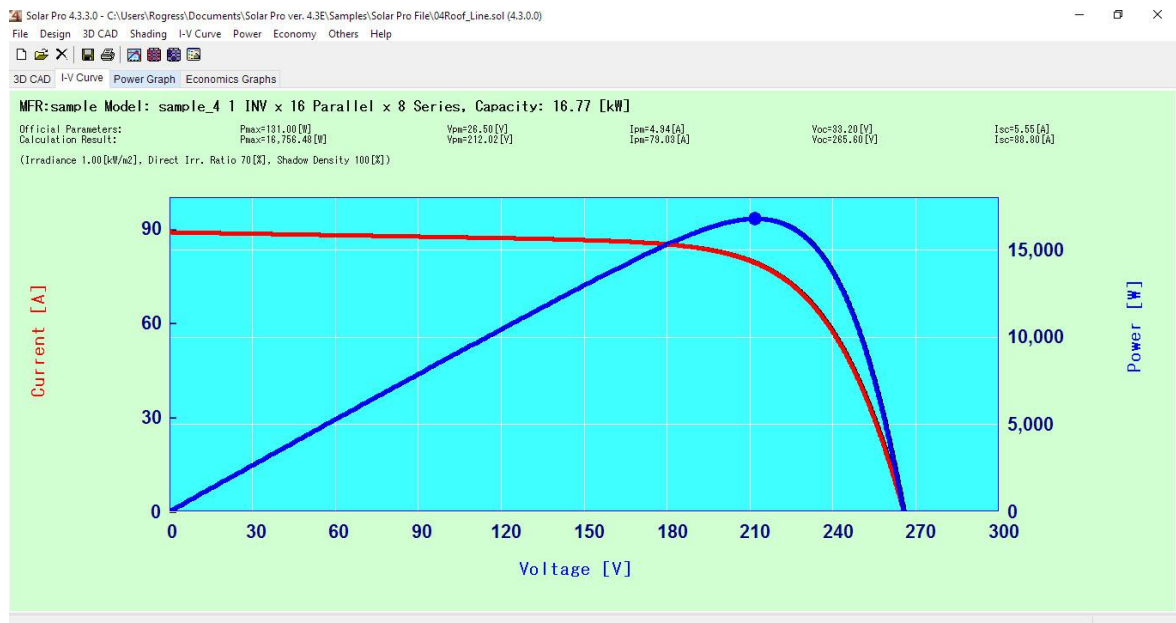


Figure 49 – Solar Pro IV Curve Graph

Attachment 3 – Functional Requirements

1.1. Layer Stack

1.1.1. Add Layer

ID: FUN01

Dependencies: None

Description: The user should be able to add a new layer at the push of a single button. The application automatically makes all the necessary adjustments to the model: adding a new variable, domain, distance parameter, interval and form union; adding associated domain to the necessary parts of the model. The user is requested to add certain parameters (Thickness, Donor Doping, Acceptor Doping, Eg, NV, NC, epsr, mue, muh, chi, ntd), while others are added automatically with default values (Gipce, vhtp, Ge, sigman, sigmap, tn, tp, vthp).

Note: Layers compose an individual solar cell. In this project, it's important to be able to deal with a variable number of layers. Figure 50 shows the layers of a typical solar cell. It's important to note that layers are a component of a solar cell, not a Comsol feature. Comsol will simply be used to simulate the multiple layers.

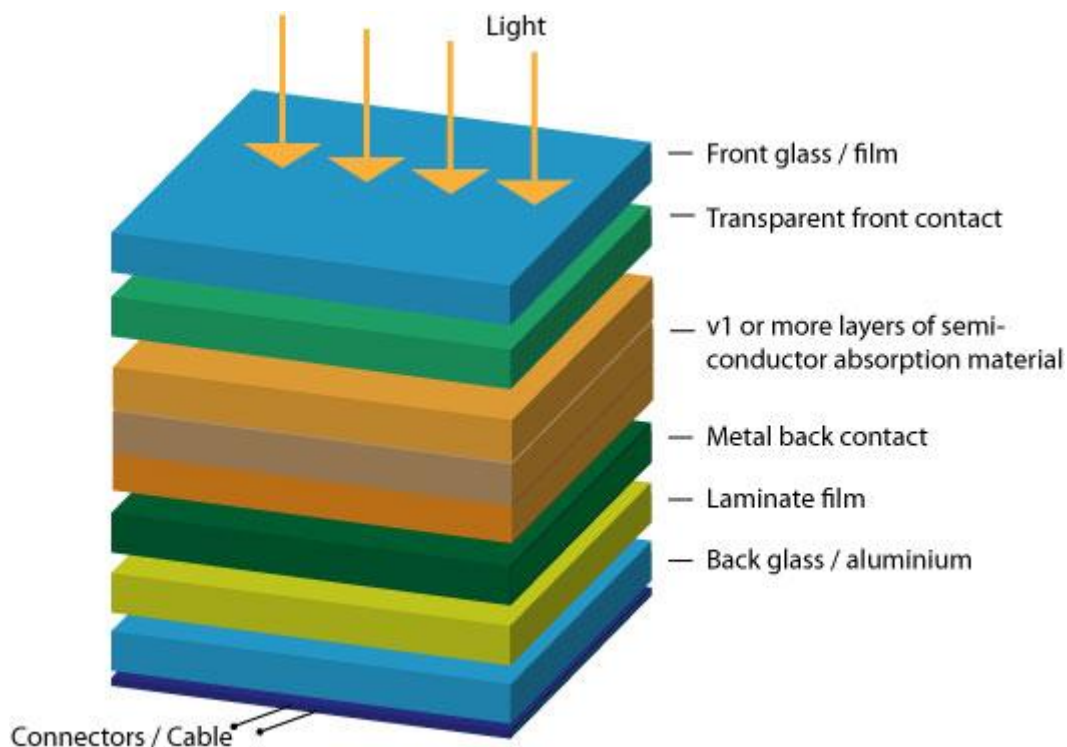


Figure 50 - Solar Cell Layers²³

²³ http://www.greenrhinoenergy.com/solar/technologies/pv_cells.php

1.1.2. Remove Layer

ID: FUN02

Dependencies: FUN01

Description: The user should be able to remove any specific layer at the push of a single button. The application automatically removes all the changes made to the model when the layer was added and adjusts the domain and intervals of higher-numbered layers. For example, if layer3 is removed, layer 4 will now be called layer 3, and its domain and interval will be adapted to reflect this change.

1.1.3. Move Layer Up

ID: FUN03

Dependencies: FUN01

Description: The user should be able to decrease the number of a single layer. For example, if the user selected this option on layer3, it would become layer2, and layer 2 would become layer 3. Adjustments to domain and interval will be done automatically as well.

1.1.4. Move Layer Down

ID: FUN04

Dependencies: FUN01

Description: The user should be able to increase the number of a single layer. For example, if the user selected this option on layer2, it would become layer3, and layer 3 would become layer 2. Adjustments to domain and interval will be done automatically as well.

1.2. Layer Parameters

1.2.1. Add Parameter

ID: FUN05

Dependencies: FUN01

Description: The user can add a default parameter with blank values, which can be edited at any time in FUN07.

1.2.2. Remove Parameter

ID: FUN06

Dependencies: FUN05

Description: The user can remove any parameter at the push of a single button.

1.2.3. Edit parameter

ID: FUN07

Dependencies: FUN05

Description: The user can alter the values of a parameter's name, expression and unit.

1.2.4. Rename Layer

ID: FUN08

Dependencies: FUN01

Description: In the Layer Parameter section, the user can alter the name of each layer.

1.3. Recombination Model

1.3.1. Change Recombination Type

ID: FUN09

Dependencies: FUN01

Description: The user can select between the Trap-Assisted (Recombination when an electron falls into a certain energy level), Auger (Non-radiative process involving three carriers), Impact Ionization (Process in which energetic charge carriers lose energy to create others) and Direct Recombination Models (Recombination through emission or absorption of a photon).

The following quote explains the concept of Recombination:

*“Any electron which exists in the conduction band is in a meta-stable state and will eventually stabilize to a lower energy position in the valence band. When this occurs, it must move into an empty valence band state. Therefore, when the electron stabilizes back down into the valence band, it also effectively removes a hole. This process is called recombination.”*²⁴

1.3.2. Edit Recombination Parameters

ID: FUN10

Dependencies: FUN09

Description: Depending on the Recombination Model selected, a different list of parameters will appear. The user cannot change their names, but they can select their values from a list.

²⁴ <https://www.pveducation.org/pvcdrom/pn-junctions/types-of-recombination>

1.4. Boundary Conditions

1.4.1. Edit Contact Properties

ID: FUN11

Dependencies: None

Description: The user can choose between Ohmic (Junction between conductors with linear IV curve) and Schottky (Electron barrier formed at metal-semiconductor junction) contact properties and edit their parameters, for both left and right contacts. Left and right contacts are both displayed in the layer sketch.

1.5. Meshing

1.5.1. Set Mesh Options

ID: FUN12

Dependencies: None

Description: The user can choose Mesh Options in the model.

1.6. Computation

1.6.1. Compute TE Solution

ID: FUN13

Dependencies: None

Description: The user can press a button to compute the TE (Thermal Equilibrium) Solution

1.6.2. Compute and Plot IV

ID: FUN14

Dependencies: None

Description: The user can have the application compute and plot Current-Voltage (IV) after specifying Voltage Sweep (Min/Max Value, Step Size), Illumination and (optionally) Parameter Sweep. This last option should allow a dynamic increase in number of sweep parameters and the selection of min/max values and step size for all of them. A window with the solution progress from the model should appear during computation. Meshing of individual domains needs to have automatic quality.

1.6.3. Compute and Plot CV

ID: FUN15

Dependencies: None

Description: The user can press a button to calculate and display a plot with the Capacitance-Voltage (CV). The user chooses its frequency value and voltage sweep (min/max value and step size).

1.6.4. Compute and Plot EIS

ID: FUN16

Dependencies: None

Description: The user can press a button to calculate and display a plot with the Electrochemical Impedance Spectroscopy (EIS).

1.6.5. Compute and Plot IPCE

ID: FUN17

Dependencies: None

Description: The user can press a button to calculate and display a plot with the Incident Photon to Current Efficiency (IPCE).

1.6.6. Plot Stack When Layers Changed

ID: FUN18

Dependencies: FUN02, FUN03, FUN04, FUN06, FUN07, FUN08, FUN09, FUN10

Description: When changes are made to the layers (addition, removal, position change, parameter change, recombination model change or global condition change), the plot should change accordingly.

1.6.7. Layer Sketch Labels

ID: FUN19

Dependencies: FUN18

Description: A window on the top right of the label sketch displays the name of each layer at all times. When changes are made to the labels, the values shown in this window reflect those changes.

1.6.8. Edit Solver Method

ID: FUN20

Dependencies: FUN13, FUN14, FUN15, FUN16, FUN17

Description: The user can adjust the method used by each solver.

1.6.9. Edit Individual Solver Parameters

ID: FUN21

Dependencies: FUN13, FUN14, FUN15, FUN16, FUN17

Description: The user can adjust individual parameters used by the solver.

1.6.10. Edit Illumination Value

ID: FUN22

Dependencies: FUN13, FUN14, FUN15, FUN16, FUN17

Description: The user can change the value for Illumination that affects every study solver.

1.6.11. Plot Solution

ID: FUN23

Dependencies: FUN13, FUN14, FUN15, FUN16, FUN17

Description: The user can plot a specific graph without re-computing the solution.

1.6.12. Change current plot

ID: FUN24

Dependencies: FUN23

Description: The user can press a button select a different Result plot within each study. The following list shows the available Result plot for each study:

IV: Band diagram, Carrier concentrations, IV current

TE: Band diagram, Carrier concentrations

EIS: Nyquist plot, Bode arg, Bode abs

IPCE: Band diagram, Carrier concentrations, IPCE plot

CV: Energy Levels, Mott-Schottky

1.6.13. Change displayed parameter sweep values

ID: FUN25

Dependencies: FUN24

Description: The user can select which sweep values should be taken into account for plotting after computing each solution. As an example, when plotting IV they can select which voltage sweep values to consider. If they run any additional parametric sweep, they can also select which values from that sweep should be considered for plotting.

1.7. Experimental Data

1.7.1. Load Experimental Data from Disk

ID: FUN26

Dependencies: FUN13, FUN14, FUN15, FUN16, FUN17

Description: The user should be able to load a .csv file with real experimental data. The format and units of the loaded data file should be displayed. Table 7 describes the possible formats. Absorption Coefficient values can be imported for every layer and are automatically integrated into the model.

Name	First Column	First Unit	Second Column	Second Unit
General				
Abs_coeff.csv	Wavelength	nm	Absorption Coeff	1/m
IV.csv	Voltage	V vs RHE	Current	mA/cm ²
CV.csv	Voltage	V vs RHE	Capacitance	F/cm ²
IPCE.csv	Wavelength	nm	IPCE	1
EIS only				
Nyquist.csv	Real Part of Complex Impedance	Ohm/cm ²	Imaginary Part of Impedance	Ohm/cm ²
BoderArg.csv	Frequency	Hz	Complex Impedance	Radian
BoderAbs.csv	Frequency	Hz	Absolute Value of Complex Impedance	Ohm/cm ²

Table 11 - Experimental Data

1.8. Parameter Fitting

1.8.1. Perform Automatic Parameter Fitting

ID: FUN27

Dependencies: FUN05, FUN20, FUN21

Description: Using the input data from the user, the application automatically performs parameter fitting. The user can access this feature in any of the “Compute” nodes. The user can choose however many parameters from the list of every parameter from every layer, and they will be considered for this calculation.

1.9. Global Parameters

1.9.1. Import Spectrum Photon File

ID: FUN28

Dependencies: None

Description: The user can import information from a file on their computer into the “spectralPhotonFluxVsLambda” table in the model.

1.9.2. Edit Individual Global Parameters

ID: FUN29

Dependencies: None

Description: The user should be able to adjust values for global parameters that affect the whole model.

1.10. Model Save and Load

1.10.1. Save Model

ID: FUN30

Dependencies: None

Description: The user can save the model in its current state in a file to resume work later.

1.10.2. Load Model

ID: FUN31

Dependencies: FUN30

Description: The user can load a previously saved model.

1.11. Report

1.11.1. Report Generation

ID: FUN32

Dependencies: FUN13, FUN14, FUN15, FUN16, FUN17

Description: A PDF document is created with relevant information about the simulation and saved in the user-specified folder.

1.12. Heterointerfaces

1.12.1. Edit Interface Model Types

ID: FUN33

Dependencies: FUN29

Description: The user should be able to choose Thermionic Emissions (Thermally induces flow of charge carriers) or Quasi-Fermi Levels (Population of electrons when displaced from equilibrium) Interface Model Type. The user should be able to do this for every interface. The number of interfaces should be equals to the number of layers minus one.

Attachment 4 – Server Architecture

This attachment shows the initial plans for the cancelled Client-Server architecture:

The application will be uploaded to a server using the Comsol Server software. This will allow any user to quickly have access to it from any web browser or from the Comsol Client for Windows desktop application. The user can then locally save and load model files from his own computer, but all physics computations are done on the server computer. They are also able to work with a default model file. Any change made to the model through the application, such as the addition of a new layer, is recorded and can be saved in an .mph file for future use.

Having the users access the application through their web browser or Comsol Client rather than downloading it will have two major advantages. The first one is the fact that changes made to the application can be easily and quickly uploaded to the server, ensuring that users are always accessing the most recent version. The other big advantage is that users do not need a powerful computer to run the application, as the most computationally demanding processes are all done by the server.

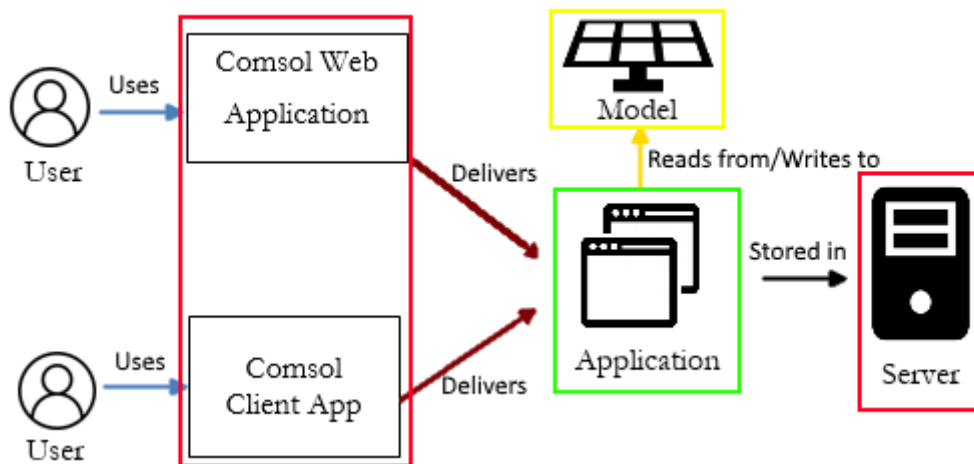


Figure 51 – Connections from User to Server

Rectangle Color Labels:

- Red – This part of the architecture will be done automatically by Comsol.
- Yellow – This part of the architecture has been previously written, but might require slight adjustments by the developer.
- Green – The developer will make this part of the architecture.

Figure Element Labels:

- User – Person interacting with the application.
- Comsol Web Application – Web Browser used to remotely access the application
- Comsol Client App – Desktop Application used to remotely access the application.
- Application – The software simulator described in this report.
- Model – The underlying model of the solar cell, with which the application will interact.
- Server – The server where the application will be stored and which will perform all computations.

In practical terms, we'll have the user interacting with the application through one of the two available clients. Their actions will activate different methods inside the application, which alter the model being used. Since the model is changing, what is shown on the application, such as the solar cell plot, for example, changes as well. The user will be able to observe this in real time.

The Comsol Server application largely automates the process of uploading the application created by the developer to the server. Since the server-client interactions do not need to be coded, work will focus on coding and testing the application itself. Servers will be hosted by the University of Žilina. It will be important to perform availability testing with these servers at the end of development, to ensure above 99% availability.