

SWEN 2022

De Zonnescherm regelaar



Student: Jasper Verheijen

Studentnummer: 0482018

Datum: 20 december 2022

Docent: Marco Dumont

Inleiding

Voor deze opdracht heb ik gekozen om het alternatieve onderwerp uit te werken:

De zonnescerm regelaar.

Dit bestand geeft uitleg over het ontwerp van de zonnescerm regelaar software.

De hoofdstukken zijn als volgt ingedeeld:

Inhoud

Inleiding.....	2
Eisen en aannames.....	3
Software ontwerp	4
State Machine	4
State Chart.....	5
Classen	6
Functies	7
Testen.....	9
Feedback studenten.	10
Conclusie	10
Aanbevelingen.....	10

Eisen en aannames

In document “Alternatieve Opdracht Blok 2” worden de eisen beschreven voor deze opdracht.

De functionele eisen zijn als volgt:

- Als zonneschermbewerking wil ik dicht als het te hard waait.
- Als zonneschermbewerking wil ik open als de zon schijnt.
- Als zonneschermbewerking wil ik dicht als het donker is.
- Als zonneschermbewerking wil ik dicht als het regent.
- Als zonneschermbewerking wil ik dicht als de binnentemperatuur lager is dan de bewoner heeft ingesteld.
- Als zonneschermbewerking wil ik tijdens het dichtgaan niet opengaan.
- Als zonneschermbewerking wil ik tijdens het opengaan niet dichtgaan.

En de technische eisen zijn als volgt:

- De software wordt geschreven en draait op QT creator versie 5.9.5
- State chart, use-case diagram en class diagram worden gemaakt met Plant UML.
- De zonneschermbewerking bevat:
 - o een binnensensor die de temperatuur meet,
 - o een UV-sensor (voor de zon)
 - o een buitensensor die lichtintensiteit, regen en windsnelheid meet.

Aannames:

Voor deze opdracht worden de volgende aannames gedaan:

- De zonneschermbewerking bevat twee contacten om de software van een signaal te voorzien wanneer het zonneschermbewerking geheel open of geheel gesloten is. Hiermee kunnen de functionele eisen op een robuuste manier worden behaald.
- Het zonneschermbewerking is dicht op het moment van starten software.
- Er wordt uitgegaan dat de bewoner de gewenste binnentemperatuur instelt op de binnensensor.
- De functionele eis “open als de zon schijnt” wordt bepaald aan de hand van zowel de uv sensor als de lichtintensiteit sensor.

Software ontwerp

In dit hoofdstuk worden de verschillende onderdelen van de software beschreven en de relatie tot elkaar.

State Machine

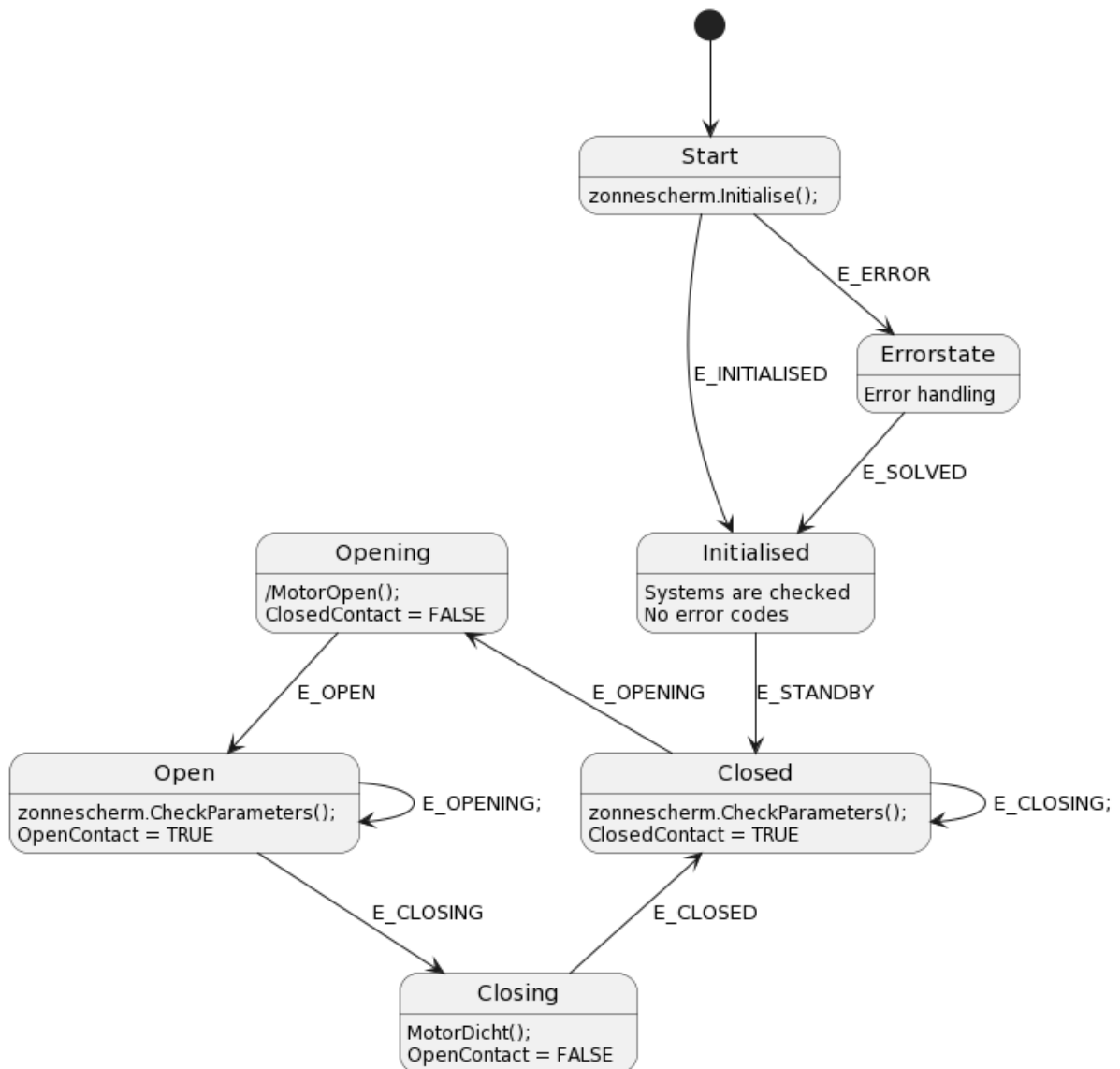
De software loopt op basis van een statemachine die het mogelijk maakt om de geschreven software in secties, zogenaamde “states” te doorlopen. De statemachine zal zorgen dat een state wordt aangehouden tot het moment dat beschreven parameters worden behaald en aan de hand van deze parameters zal een volgende state worden gestart.

De software is onderverdeeld in de volgende states:

- **Start:** dit is het startpunt van de state machine wanneer het zonnescherf wordt voorzien van stroom. Vanuit deze state zal de machine direct zijn hardwarecomponenten controleren. Wanneer alle hardwarecomponenten zijn gecontroleerd, zal het zonnescherf overgaan naar de status “Closed”. Het kan ook zijn dat er tijdens het controleren er een foutmelding op treedt, dan zal de software naar de “Errorstate” gaan totdat deze error is verholpen.
- **ErrorState:** Deze state kan alleen worden verlaten wanneer het systeem wordt gereset of de stroom van het zonnescherf wordt onderbroken.
- **Closed:** Het zonnescherf is in een dichte positie. In de “closed” state zal de state machine de beschreven parameters blijven controleren totdat er moet worden overgegaan op de “opening” state.
- **Opening:** De “opening” state wordt aangesproken wanneer het zonnescherf wordt geopend. In de state wordt de functie aangesproken die het openen van het zonnescherf simuleert. Deze state zorgt ervoor dat het zonnescherf niet dicht kan worden gemaakt totdat deze volledig open is. Wanneer de waarde “SchakelcontactOpen” true wordt, zal de state machine overgaan op de state “open”.
- **Open:** Het zonnescherf is in de “open”state. In de open positie zal de state machine de beschreven parameters blijven controleren totdat er moet worden overgegaan op de “closing” state.
- **Closing:** De “closing” state wordt aangesproken wanneer het zonnescherf wordt gesloten. In de state wordt de functie aangesproken die het sluiten van het zonnescherf simuleert. Deze state zorgt ervoor dat het zonnescherf niet open kan worden gemaakt totdat deze volledig dicht is. Wanneer de waarde “SchakelcontactDicht” true wordt, zal de state machine overgaan op de state “Closed”.

State Chart

State Chart Zonnescherm -V1-



De state chart geeft een overzichtelijk beeld van de states waarin de state machine zich kan bevinden en welke functies in de states en bij overgangen worden aangesproken.

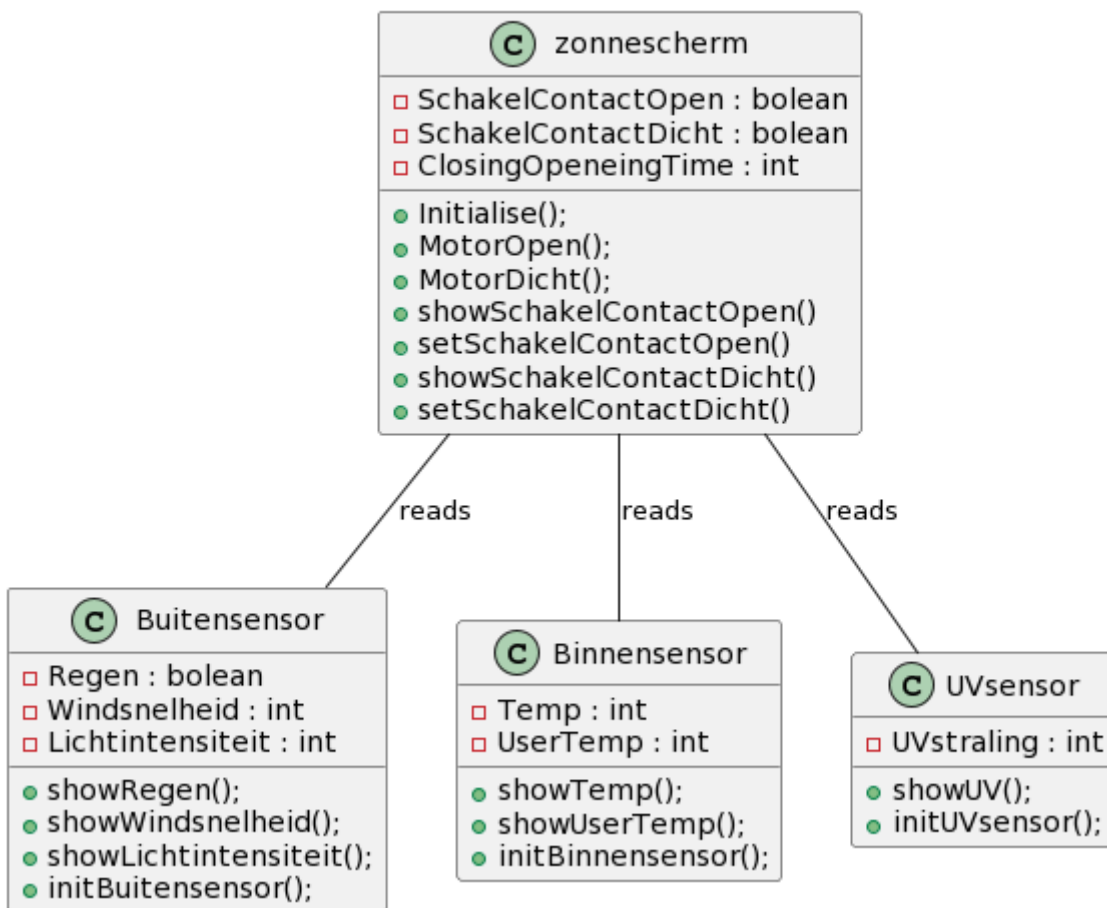
Hierbij zorgt de functie `zonnescherm.CheckParameters()` voor de event `E_OPENING` of `E_CLOSING` en `MotorOpen()` zorgt voor het event `E_OPEN` en `MotorDicht()` voor het event `E_CLOSED`. `Zonnescherm.Initilialise()` bepaald of de `E_INITIALISED` of `E_ERROR` wordt getriggerd.

Classen

In overeenstemming met de beoordelingscriteria is er gekozen om de gebruikte hardware koppeling onder te brengen onder classes, deze classes bevatten waarden en functies die kunnen worden aangesproken door de software. Er is bewust gekozen om deze voor deze objecten allemaal een unieke classe te gebruiken aangezien er geen overlap zit in de functies en waarde die zij gebruiken. Het werken met classes zorgt ervoor dat de software een modulair karakter krijgt en er geen aanpassingen in de “main” software hoeft te worden gemaakt wanneer de opdrachtgever besluit een andere sensor te implementeren, zo lang onderstaande beschreven functies & waarden maar gelijk blijven.

De software bevat de volgende classes:

Class Diagram Zonnescherm -V1-



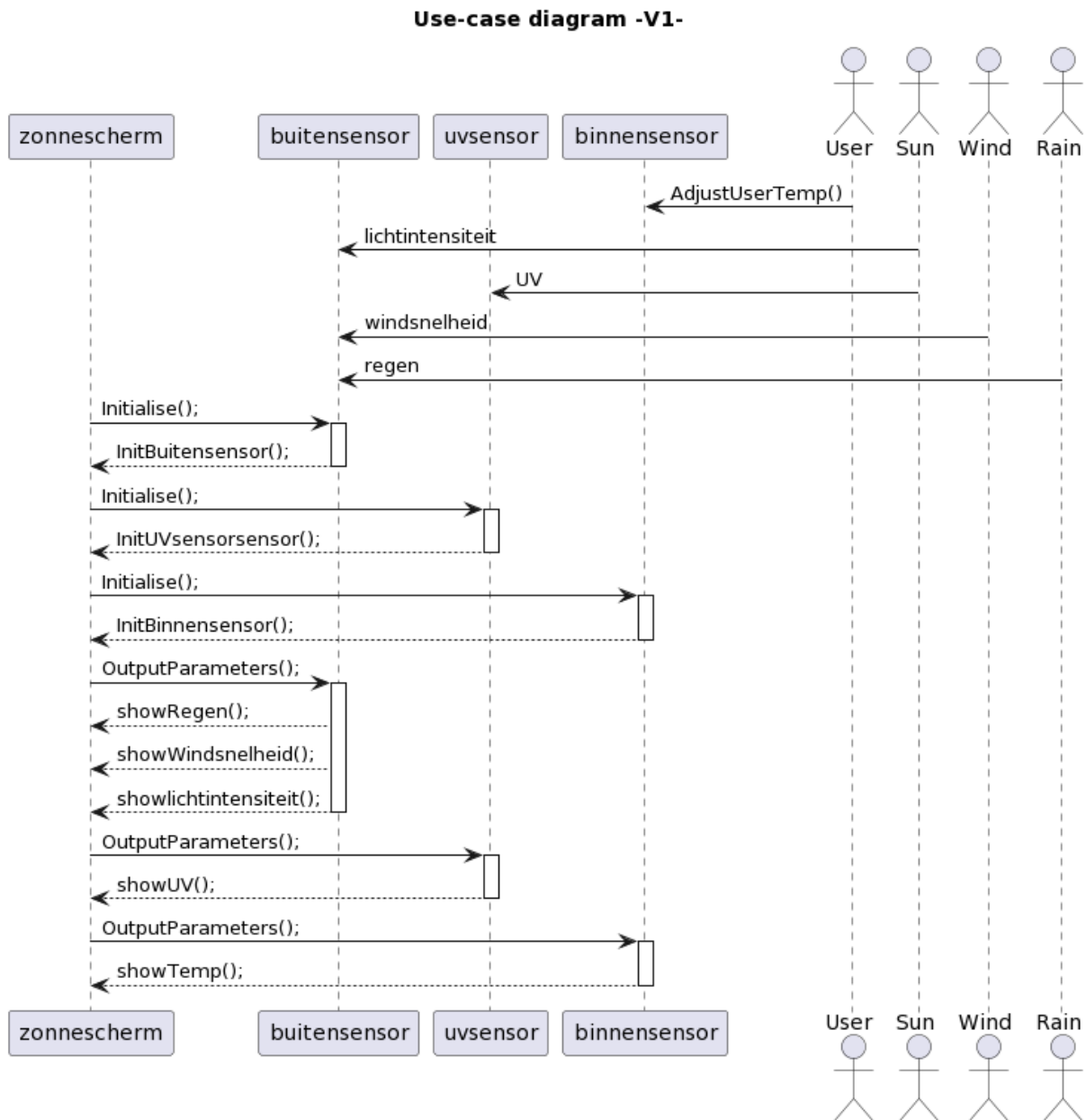
- Binnensensor
 - o (integer) Temp 0 t/m 30 Celsius(C°)
 - o (integer) UserTemp 10 t/m 30 Celsius(C°)
- UV sensor
 - o (integer) UVstraling 0 t/m 15 UV-index
- Buitensensor
 - o (integer) Lichtintensiteit 0 > 130 Lux (theoretisch getal)
 - o (boolean) Regen: true/false
 - o (integer) Windsnelheid 0-30 km/h
- Zonnescherm
 - o (integer) ClosingOpeningTime 15 t/m 30 tijd(s)

Functies

Om de leesbaarheid te vergroten is er gekozen om de functies uit bovenstaande klassen in een apart hoofdstuk te beschrijven.

- CheckParameters: Functie om de Parameters te controleren en te bepalen of het zonnescerm open of dicht moet zijn.
Om te zorgen dat de zonnescerm voorspelbaar functioneert, is er gekozen om de functionele eisen op volgorde van logische prioriteit te programmeren. Het belangrijkste is dat het zonnescerm beschermd wordt tegen wind en regen.
- OutputParameters(): shows current values of logged parameters.
- ShowTemp(): Functie om de waarde "Temp" in te lezen en door te geven aan de state machine.
- CompareTemp(): Functie om de waarde "Temp" te vergelijken met de ingestelde waarde "UserTemp".
- AdjustTemp(): Functie om de Private waarde "UserTemp" aan te passen.
"gebruiker kan de gewenste binnentemperatuur instellen".
- ShowUV(): Functie om de waarde "UVstraling" in te lezen en door te geven aan de state machine.
- MotorOpen(): Het elektrisch zonnescerm worden geopend. Schakelcontacten worden beide op open gezet. Tegelijk wordt er een timer gestart, wanneer deze timer gelijk is aan "ClosingOpeningTime" wordt de waarde "schakelcontactOpen" op True gezet.
- MotorDicht(): Het elektrisch zonnescerm worden gesloten. Schakelcontacten worden beide op open gezet. Tegelijk wordt er een timer gestart, wanneer deze timer gelijk is aan "ClosingOpeningTime" wordt de waarde "schakelcontactDicht" op True gezet.

Om de werking en relaties tussen de variabelen, functies en variabelen beter inzichtelijk te maken is hieronder een use-case diagram weergegeven.



Testen

De software wordt getest aan de hand van automatische testen. In dit hoofdstuk wordt er uitgelegd gegeven over de gemaakte testen en de geschreven software hiervoor.

De testsoftware bestaat uit een main function die wordt gedraaid, deze spreekt op volgorde van schrijven de verschillende functies aan.

De functies zijn apart buiten de main gemaakt, dit is een keuze gebaseerd op repetitie van "ResetValuesZonnescherm()" en de mogelijkheid om in de toekomst de huidige geschreven testen om te schrijven naar een enkele variabele test waarin testparameters kunnen worden ingevoerd.

```
int main(void) {  
    ResetValuesZonnescherm();  
  
    test1(); //checkparameters should now open zonnescherm  
    test2(); //checkparameters should now close zonnescherm  
    ResetValuesZonnescherm();  
  
    test3(); //testing initiliazation of hardware components.  
  
    cout << "\n***** End of testing. *****\n";  
  
    return 0;  
}
```

De testen worden beschreven in onderstaande tabel met hun pass en fail criteria.

Test	beschrijving	Condities
1	Zonnescherm start gesloten. Uvstraling gezet op 4. Lichtintensiteit gezet op 90. Waardes zouden nu het zonnescherm moeten openen.	Pass: Open Fail: Close
2	Zonnescherm start open. Binnentemp gezet op 16. Doordat de binnentemp lager is dan ingestelde gebruikers temperatuur, zou het zonnescherm moeten sluiten.	Pass: Close Fail: Open
3	Initialise functie wordt gestart en passed. Uvsensor wordt gezet op out of range waarde -10. Initialise functie zou nu moeten failen.	Pass: failed initialization Fail: all initalized

Feedback studenten.

Als onderdeel van de eindopdracht moet de geschreven code en documentatie worden beoordeeld door drie medestudenten. De beoordelingsformulieren zijn bijgevoegd als separate documenten onder dit verslag.

Korte feedback op de beoordelingen die de studenten hebben gegeven die ook naar diegenen zijn gecommuniceerd:

- Jolien: Duidelijke scores per onderdeel, helaas geen gebruik gemaakt van geschreven feedback waardoor sommige scores onduidelijk en niet onderbouwd zijn.
- Michelle: Geen duidelijke scores per onderdelen, alleen een eindscore gegeven, wel gebruik gemaakt van de optie om geschreven feedback te gebruiken.
- Johannes: Uitgebreide feedback in een zelf opgesteld document, hoofdstuk indeling komt overeen met beoordeling model, maar verdere score niet overzichtelijk en er lijkt geen gebruik te zijn gemaakt van de bijgeleverde softwaredocumentatie met o.a. Aanwezige UML charts.

Conclusie

Tijdens het maken van de software, ontstaat er steeds meer het inzicht dat de code niet moet worden gemaakt op basis van een simulatie, waarbij handmatig door user input de parameters kunnen worden veranderd zoals voorgaande modules, maar als een geheel werkend programma dat getest kan worden door een extern programma om de veranderingen in parameters te simuleren en zo het gewenste gedrag te testen. De overzichtelijke introductie tot het gebruik van classes en het introduceren van externe test software maakte dit een erg geschikte opdracht om de gestelde leerdoelen voor dit vak te bewijzen.

Aanbevelingen

Een van de aanbevelingen is om de software geheel in het Engels of Nederlands te programmeren. Dit is in eerste instantie niet gedaan doordat de documentatie en opdracht in het Nederlands zijn geschreven, maar het softwarepakket zelfs engels is.

Vanuit persoonlijke voorkeur en lessen oude docent is ervoor gekozen om getter/setters te gebruiken om private variabelen te kunnen gebruiken buiten de classes. Er kan worden gekozen om alle variabelen om te zetten naar public en de getter/setters te verwijderen.

Om de software te runnen moet de `include_directories` in de `makefile` worden aangepast naar de directory van de huidige gebruiker, dit zou mogelijk flexibel kunnen gemaakt worden voor de eindgebruiker.

	<4	Zwaar onvoldoende	Onvoldoende	Voldoende	Goed	Zeer goed		Weging D1	Weging D2	Cijfer	Punten onderdeel	Opmerkingen Goed	Verbeterpunten
Student kan C++ Klassen gebruiken in programma's	Geen structuur aanwezig of zeer onduidelijk (geen inheritance, incapsulatie of interfacing)	Er zijn shortcut's gemaakt (onvoldoende incapsulatie en of zeer hoge coupling)	Incapsulatie is 100% en er is sprake van overerving van klassen.	De objecten zijn een logisch gevolg van elkaar.	Er is gekozen voor een structuur (event driven, MCV) die is onderbouwt in het commentaar.	Structuur is een logische afleiding van het domein en volgt de engineering principes van pressmann	<ul style="list-style-type: none"> - Separation of concerns - cpp en headerfiles - commentaar bij elke functie - geen dataclasses - single responsibility and which provide a unique behavior. - methoden hebben een implementatie. Pressman: A design should exhibit an architecture that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics, and (3) can be implemented in an evolutionary fashion, thereby facilitating implementation and testing. 2. A design should be modular; that is, the software should be logically partitioned into elements or subsystems. 3. A design should contain distinct representations of data, architecture, interfaces, and components. 4. A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns. 5. A design should lead to components that exhibit independent functional characteristics. 6. A design should lead to interfaces that reduce the complexity of connections between components and	40%	30%				
Student gebruikt de juiste syntax en gebruik van IDE	Syntax is van een C programma geen Qtcreator gebruik.	Qtcreator gebruikt en C structuur.	Qt creator wordt gebruikt en C++ structuren	cmake, qmake c++ project te bouwen zonder qtcreator en volgt voor 50% de richtlijnen van clean code.	cmake, qmake c++ project te bouwen zonder qtcreator en volgt voor 100% de richtlijnen van clean code.	Project runt en compileert buiten de omgeving.		5%	5%				
Student maakt gebruik van Linux (Ubuntu 20.04 of 21.04)	Werkt onder een ander besturingssysteem.	Werkt een enkele keer in een ander besturingssysteem.	Werkt in een ander besturingssysteem.	Werkt in de virtual machine onder windows.	Werkt op Linux, zonder virtual machine.	Werkt op een Raspberry Pi.		5%	5%				
Student vangt de echte wereld in een klasse model (modellieren).	Geen, alles in een of een enkele klasse (God klasse)	40% van het gedrag is in de code opgenomen.	60% van het gedrag is opgenomen in de code.	80% van het gedrag is in de code opgenomen.	Gedrag van de opdracht is in zijn geheel in de code opgenomen.	Coupling en balans zijn in orde		30%	20%				
Student zorgt je dat het werkt (testen)	Geen testen gemaakt, code compileert wel.	Er is handmatig getest	Er wordt automatisch getest.	80% is automatisch getest, door middel van een framework	100% is automatisch getest, door middel van een framework	Student test zowel branches en coverage		20%	10%				
Reviews	Geen reviews	Een enkele review	Enkele reviews maar geen aanpassingen in de code	De meeste reviews aanwezig	De meeste reviews aanwezig met aanpassingen van de code	De meeste reviews aanwezig met aanpassingen van de code			10%				
UML	Geen UML	Een diagram	Alle class-diagram	Class, state en use-case diagram	Een volledig Class, state en use-case diagram	Een volledig Class, state en use-case diagram			20%				
								100%	100%				

Lanza, M., & Marinescu, R. (2007). Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media.

	<4	Zwaar onvoldoende	Onvoldoende	Voldoende	Goed	Zeer goed		Weging D1	Weging D2	Cijfer	Punten onderdeel	Opmerkingen Goed	Verbeterpunten
						Gebruik gemaakt van een statechart							
Student kan C++ Klassen gebruiken in programma's	Geen structuur aanwezig of zeer onduidelijk (geen inheritance, incapsulatie of interfacing)	Er zijn shortouts gemaakt (onvoldoende incapsulatie en of zeer hoge coupling)	Incapsulatie is 100% en er is sprake van overerving van klassen.	De objecten zijn een logisch gevolg van elkaar.	Er is gekozen voor een structuur (event driven, MCV) die is onderbouwt in het commentaar.	Structuur is een logische afleiding van het domein en volgt de engineering principes van pressmann	<ul style="list-style-type: none"> - Separation of concerns - cpp en headerfiles - commentaar bij elke functie - geen dataclasses - single responsibility and which provide a unique behavior. - methoden hebben een implementatie. Pressman: A design should exhibit an architecture that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics, and (3) can be implemented in an evolutionary fashion, thereby facilitating implementation and testing. 2. A design should be modular; that is, the software should be logically partitioned into elements or subsystems. 3. A design should contain distinct representations of data, architecture, interfaces, and components. 4. A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns. 5. A design should lead to components that exhibit independent functional characteristics. 6. A design should lead to interfaces that reduce the complexity of connections between components and with the external environment. 7. A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis. 8. A design should be represented using a notation that effectively communicates its meaning.	40%	30%		0	<p>Goede uitleg wat betreft keuzes.</p> <p>Systeem is modulair. (logische verdeling van de sensoren)</p> <p>C++ implementatie is SOLID, o.a. omdat elke class z'n eigen taak heeft. Het systeem kan "eenvoudig worden uitgebreid"</p> <p>Denk aan Engels en Nederlands door elkaar gebruiken. Is met name storend in de code.</p> <p>Zijn de weerstorstandigheden actors? Is user in deze case een actor?</p>	
Student gebruikt de juiste syntax en gebruik van IDE	Syntax is van een C programma geen Qtcreator gebruik.	Qtcreator gebruikt en C structuur.	Qt creator wordt gebuikt en C++ structuren	cmake, qmake c++ project te builden zonder qtcreator en volgt voor 50% de richtlijnen van clean code.	cmake, qmake c++ project te builden zonder qtcreator en volgt voor 100% de richtlijnen van clean code.	Project runt en compileert buiten de omgeving.		5%	5%		0		
Student maakt gebruik van Linux (Ubuntu 20.04 of 21.04)	Werkt onder een ander besturingssysteem.	Werkt een enkele keer in een ander besturingssysteem.	Werkt in een ander besturingssysteem.	Werkt in de virtual machine onder windows.	Werkt op Linux, zonder virtual machine.	Werkt op een Raspberry PI.		5%	5%		0		
Student vangt de echte wereld in een klasse model (modellieren).	Geen, alles in een of een enkele klasse (God klasse)	40% van het gedrag is in de code opgenomen.	60% van het gedrag is opgenomen in de code.	80% van het gedrag is in de code opgenomen.	Gedrag van de opdracht is in zijn geheel in de code opgenomen.	Coupling en balans zijn in orde		30%	20%		0		
Student zorgt je dat het werkt (testen)	Geen testen gemaakt, code compileert wel.	Er is handmatig getest	Er wordt automatisch getest.	80% is automatisch getest, door middel van een framework	100% is automatisch getest, door middel van een framework	Student test zowel branches en coverage		20%	10%		0		
Reviews	Geen reviews	Een enkele review	Enkele reviews maar geen aanpassingen in de code	De meeste reviews aanwezig	De meeste reviews aanwezig met aanpassingen van de code	De meeste reviews aanwezig met aanpassingen van de code			10%		0		
UML	Geen UML	Een diagram	Alle class-diagram	Class, state en use-case diagram	Een volledig Class, state en use-case diagram	Een volledig Class, state en use-case diagram			20%		0		
								100%	100%				
Lanza, M., & Marinescu, R. (2007). Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media.											0		

8,5

Feedback voor https://github.com/CoralCare/SWEN_2022/tree/10DEC22

Door J.G. Jeronimus

Jg.jeronimus@student.han.nl

1. Student kan C++ Klassen gebruiken in programma' s:

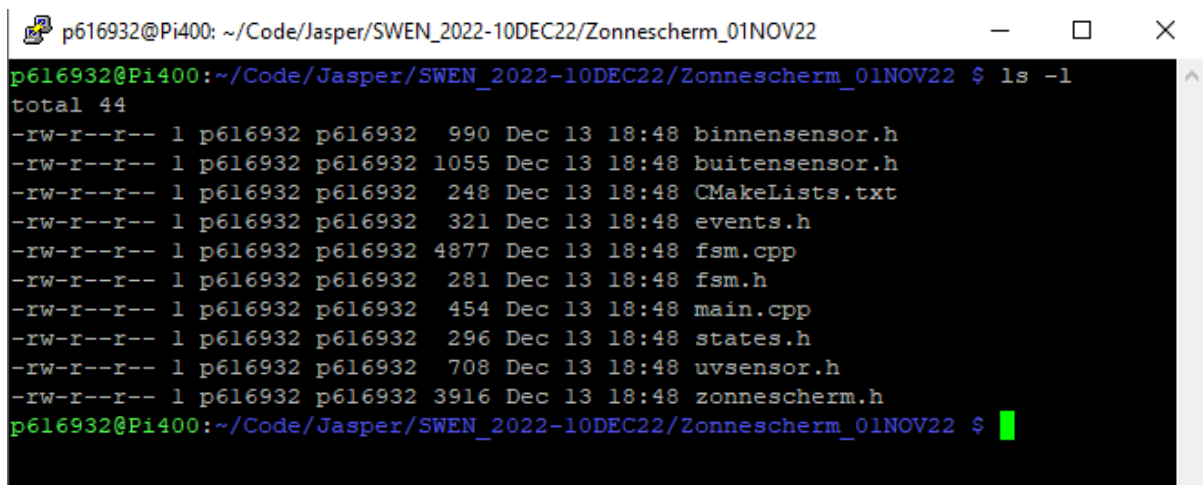
Er is gekozen voor een structuur (event driven, MCV) die is onderbouwt in het commentaar.

- Separation of concerns
- The product shows the correct use of .cpp and header-files.
- commentaar bij elke functie
- geen dataclasses
- single responsibility and which provide a unique behavior.
- methoden hebben een implementatie.

2. Student gebruikt de juiste syntax en gebruik van IDE:

<4 Syntax is van een C programma, geen Qtcreator gebruik(t?).

There are no traces of Qtcreator present. Could be made in Wordpad for what I am concerned.



```
p616932@Pi400: ~/Code/Jasper/SWEN_2022-10DEC22/Zonnescherm_01NOV22
p616932@Pi400:~/Code/Jasper/SWEN_2022-10DEC22/Zonnescherm_01NOV22 $ ls -l
total 44
-rw-r--r-- 1 p616932 p616932  990 Dec 13 18:48 binnensensor.h
-rw-r--r-- 1 p616932 p616932 1055 Dec 13 18:48 buitensensor.h
-rw-r--r-- 1 p616932 p616932  248 Dec 13 18:48 CMakeLists.txt
-rw-r--r-- 1 p616932 p616932  321 Dec 13 18:48 events.h
-rw-r--r-- 1 p616932 p616932 4877 Dec 13 18:48 fsm.cpp
-rw-r--r-- 1 p616932 p616932  281 Dec 13 18:48 fsm.h
-rw-r--r-- 1 p616932 p616932  454 Dec 13 18:48 main.cpp
-rw-r--r-- 1 p616932 p616932  296 Dec 13 18:48 states.h
-rw-r--r-- 1 p616932 p616932  708 Dec 13 18:48 uvsensor.h
-rw-r--r-- 1 p616932 p616932 3916 Dec 13 18:48 zonnescherm.h
p616932@Pi400:~/Code/Jasper/SWEN_2022-10DEC22/Zonnescherm_01NOV22 $
```

There is no "makefile" present in the main application.

In the TEST module is however an "makefile" present. This does not result in a successful compilation of the program on the CLI on a Raspberry Pi 400.

```
p616932@Pi400: ~/Code/Jasper/SWEN_2022-10DEC22/TEST_Zonnescherm_09NOV22
p616932@Pi400:~/Code/Jasper/SWEN_2022-10DEC22/TEST_Zonnescherm_09NOV22 $ ls -l
total 24
-rw-r--r-- 1 p616932 p616932 460 Dec 13 18:48 CMakeLists.txt
-rw-r--r-- 1 p616932 p616932 582 Dec 13 18:48 main.cpp
-rw-r--r-- 1 p616932 p616932 6460 Dec 13 18:48 Makefile
-rw-r--r-- 1 p616932 p616932 2168 Dec 13 18:48 test.cpp
-rw-r--r-- 1 p616932 p616932 398 Dec 13 18:48 test.h
p616932@Pi400:~/Code/Jasper/SWEN_2022-10DEC22/TEST_Zonnescherm_09NOV22 $ make -B
CMake Error: The source directory "/home/student/Desktop/2022SWEN/SWEN_2022/Zonnescherm Software Ubuntu/TEST_Zonnescherm_09NOV22/TEST_Zonnescherm_09NOV22" does not exist.
Specify --help for usage, or press the help button on the CMake GUI.
make: *** [Makefile:218: cmake_check_build_system] Error 1
p616932@Pi400:~/Code/Jasper/SWEN_2022-10DEC22/TEST_Zonnescherm_09NOV22 $
```

This is easily fixable by using a relative directory in the "Makefile".

Tips:

- Start every file with a personal text block where you tell who you are, to what purpose this software is, how to contact you, what dev-tools are used and what version of the file this is!

- Choose one language. This mix between Dutch and English is peculiar.

3. Student maakt gebruik van Linux (Ubuntu 20.04 of 21.04):

I was not able to find any evidence of this in the program. See, #2.

I found minor traces of Linux-usage in the makefile for the test.

Coding is clean, only minor issues in Qt Creator 8.0.1, mostly depending on the sensor-data not changing. Does go away when compiling. No errors in Raspbian.

4. Student vangt de echte wereld in een klasse model (modelleren):

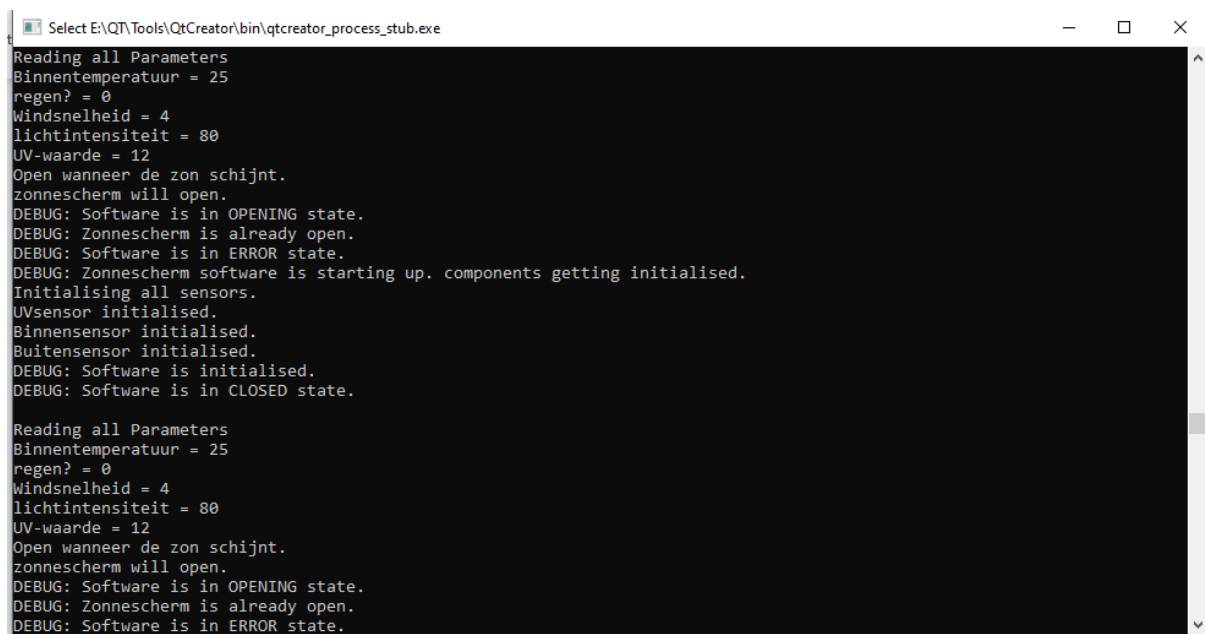
Zeer goed: Coupling en balans zijn in orde.

No comments.

5. Student zorgt je dat het werkt (testen):

After making some minor tweaks to the program and the test I was able to compile and test them both on a Raspbian and Windows 10 environment.

The program works. Until some values are changed:



```
Select E:\QT\Tools\QtCreator\bin\qtcreator_process_stub.exe
Reading all Parameters
Binnentemperatuur = 25
regen? = 0
Windsnelheid = 4
lichtintensiteit = 80
UV-waarde = 12
Open wanneer de zon schijnt.
zonnescrm will open.
DEBUG: Software is in OPENING state.
DEBUG: Zonnescrm is already open.
DEBUG: Software is in ERROR state.
DEBUG: Zonnescrm software is starting up. components getting initialised.
Initialising all sensors.
UVsensor initialised.
Binnensensor initialised.
Buitensensor initialised.
DEBUG: Software is initialised.
DEBUG: Software is in CLOSED state.

Reading all Parameters
Binnentemperatuur = 25
regen? = 0
Windsnelheid = 4
lichtintensiteit = 80
UV-waarde = 12
Open wanneer de zon schijnt.
zonnescrm will open.
DEBUG: Software is in OPENING state.
DEBUG: Zonnescrm is already open.
DEBUG: Software is in ERROR state.
```

This unsuspected behavior does not halt the State Machine in an error state.

Testing: Zeer goed: Testing is performed 100% automatically and the program tests branches and coverage.

6. Reviews:

Not applicable.

7. UML:

Not applicable in the provided directory, a little snooping revealed that a technical document exists. 😊