2023

ATP | Project- en Testplan

SLAAPKAMER COMFORT-REGELSYSTEEM

AHMET SERDAR ÇANAK

Datum:	14-4-2024
Cursuscode:	TCTI-VKATP-21
Naam:	Ahmet Serdar Çanak
Studierichting:	Technische Informatica
Specialisatie:	Embedded Systems Engineering
Studentennummer:	1760039

Inhoudsopgave

1. Inleiding

In het kader van de voortschrijdende ontwikkelingen in slimme thuisautomatisering, wordt in dit verslag een “Slaapkamer Comfort-Regelsysteem” geïntroduceerd en geanalyseerd. Dit prototypesysteem heeft als doel de leefomstandigheden in een slaapkamer te optimaliseren, waarbij gebruik wordt gemaakt van twee sensoren, namelijk een lichtsensor en een temperatuursensor, in combinatie met twee actuatoren: een generieke rolluikmotor en koelunit.

Deze rapportage belicht niet alleen de conceptuele en technische aspecten van het Regelsysteem, maar benadrukt tevens de cruciale rol van kwaliteitseisen, testen en validatie in het ontwikkelingsproces. Het gepresenteerde testplan is ontworpen om de effectiviteit, prestaties en betrouwbaarheid van het systeem te beoordelen. Het hoofddoel van deze testen is om te verzekeren dat het regelsysteem in staat is om het comfort van de gebruikers te handhaven.

In de aankomende secties zal het regelsysteem worden toegelicht met bijbehorende componenten, zullen de architectuurschetsen worden gepresenteerd en besproken, zal het testplan in detail worden behandeld. Tot slot heb ik verwijzing naar de datasheets van de desbetreffende componenten genoteerd.

2. Regelsysteem

Slaapkamer Comfort-Regelsysteem:

Het Slaapkamer Comfort-Regelsysteem is een automatiseringssysteem ontworpen om het comfort en welzijn in de slaapkamer te verbeteren. Het systeem maakt gebruik van twee belangrijke subsystemen voor temperatuurregeling en ochtendwekkerfunctionaliteit.

1. Temperatuurregeling met Ventilator en Temperatuursensor:

Dit systeem is gericht op het handhaven van een aangename slaapomgeving door temperatuurvariaties te beheren. Het bevat de volgende componenten:

- **Temperatuursensor**: Een temperatuursensor meet voortdurend de omgevingstemperatuur in de slaapkamer.
- **Koelunit**: Een koelunit is geïntegreerd in het systeem en wordt geactiveerd als de temperatuur in de slaapkamer boven een vooraf ingestelde drempel stijgt.
- **Automatische Aanpassing**: Als de omgevingstemperatuur weer binnen het gewenste bereik valt, zal de koelunit automatisch worden uitgeschakeld om onnodig energieverbruik te voorkomen.

2. Ochtendwekker met Zonlichtdetectie en Rolluikmotor:

Dit systeem fungeert als een intelligente ochtendwekker en is gebaseerd op de gecombineerde functionaliteit van een lichtsensor en een rolluikmotor. Het werkt als volgt:

- **Tijdinstelling**: De gebruiker kan een specifieke tijd instellen waarop ze wakker willen worden.
- **Lichtsensor**: Een lichtsensor detecteert de hoeveelheid omgevingslicht bij zonsopgang.
- **Alarmactivering**: Een uur voor de door de gebruiker ingestelde tijdstip, wordt het alarm geactiveerd. Als er voldoende daglicht aanwezig is, zorgt het systeem ervoor dat de rolluiken automatisch worden geopend met behulp van de rolluikmotor indien de rolluiken gesloten waren.
- **Ontwaken in Natuurlijk Licht**: De combinatie van het alarm en het openen van de rolluiken zorgt ervoor dat de gebruiker ontwaakt in een kamer die wordt verlicht door natuurlijk daglicht, wat het ontwaken aangenamer maakt.

2.1. Sensoren

- **Lichtsensor**: Als lichtsensor heb ik gekozen voor de BH1750, omdat het aan te sturen is via I^2C en omdat het voldoet aan de eis om lichtintensiteit te kunnen meten.
- **Temperatuursensor**: Als temperatuursensor heb ik gekozen voor de DHT11, omdat het aan te sturen is via een data-pin en het voldoet aan de eis om de omgevingstemperatuur te meten.

2.2. Actuatoren

- **Rolluikmotor**: Aangezien er geen rolluikmotoren zijn, heb ik gekozen voor een generiek apparaat als rolluikmotor wat aan en uitgezet kan worden via een data-pin.
- **Koelunit**: Aangezien er geen koelunits zijn, heb ik gekozen voor een generiek apparaat als koelunit wat aan en uitgezet kan worden via een data-pin.

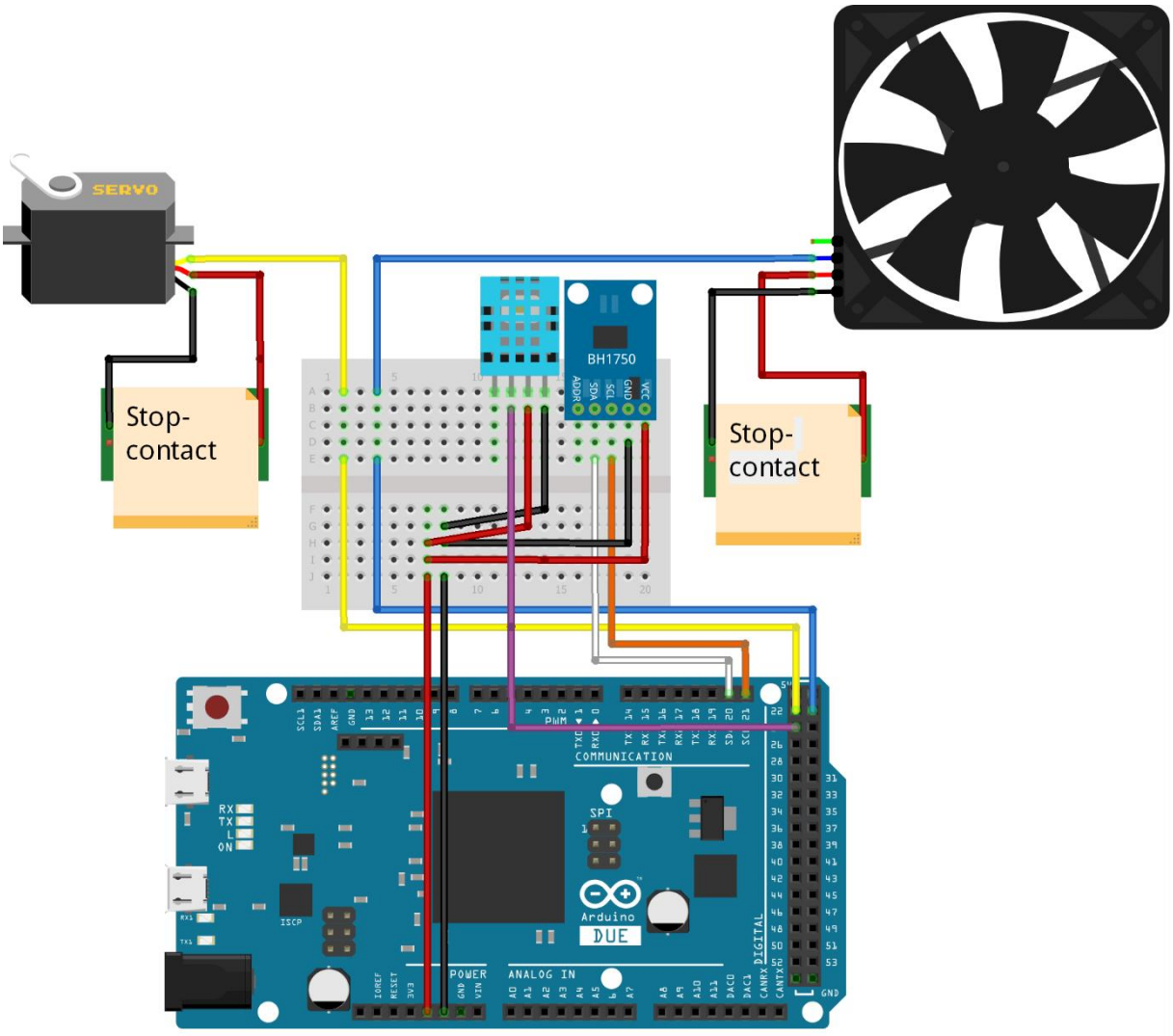
3. Architectuurschetsen

3.1. Hardware

In de Hardware-schets afgebeeld in figuur 1, is een Arduino Due microcontroller gebruikt in combinatie met een aantal sensoren en actuatoren om een geautomatiseerd systeem te realiseren. De Arduino Due fungeert als het brein van het systeem en stelt gebruikers in staat om verschillende aspecten van hun omgeving te meten en te controleren.

De schets omvat de aansluiting van een koelunit, een rolluikmotor, een BH1750-lichtsensor en een DHT11-temperatuur- en vochtigheidssensor. De koelunit, rolluikmotor en DHT11-sensoren zijn aangesloten via digitale-pinnen voor gegevensoverdracht en de BH1750-lichtsensor maakt gebruik van de I²C-communicatie-interface voor gegevensoverdracht. De koelunit en rolluiktmotor hebben aparte stroombronnen en worden dus alleen maar aangestuurd via de arduino.

De schets is bedoeld om een praktische demonstratie te bieden van de integratie van verschillende sensoren en actuatoren met een Arduino-microcontroller, waardoor gebruikers in staat worden gesteld om de omgevingsvariabelen te bewaken en te beïnvloeden.



Figuur 1: Hardware schets

3.2. Software

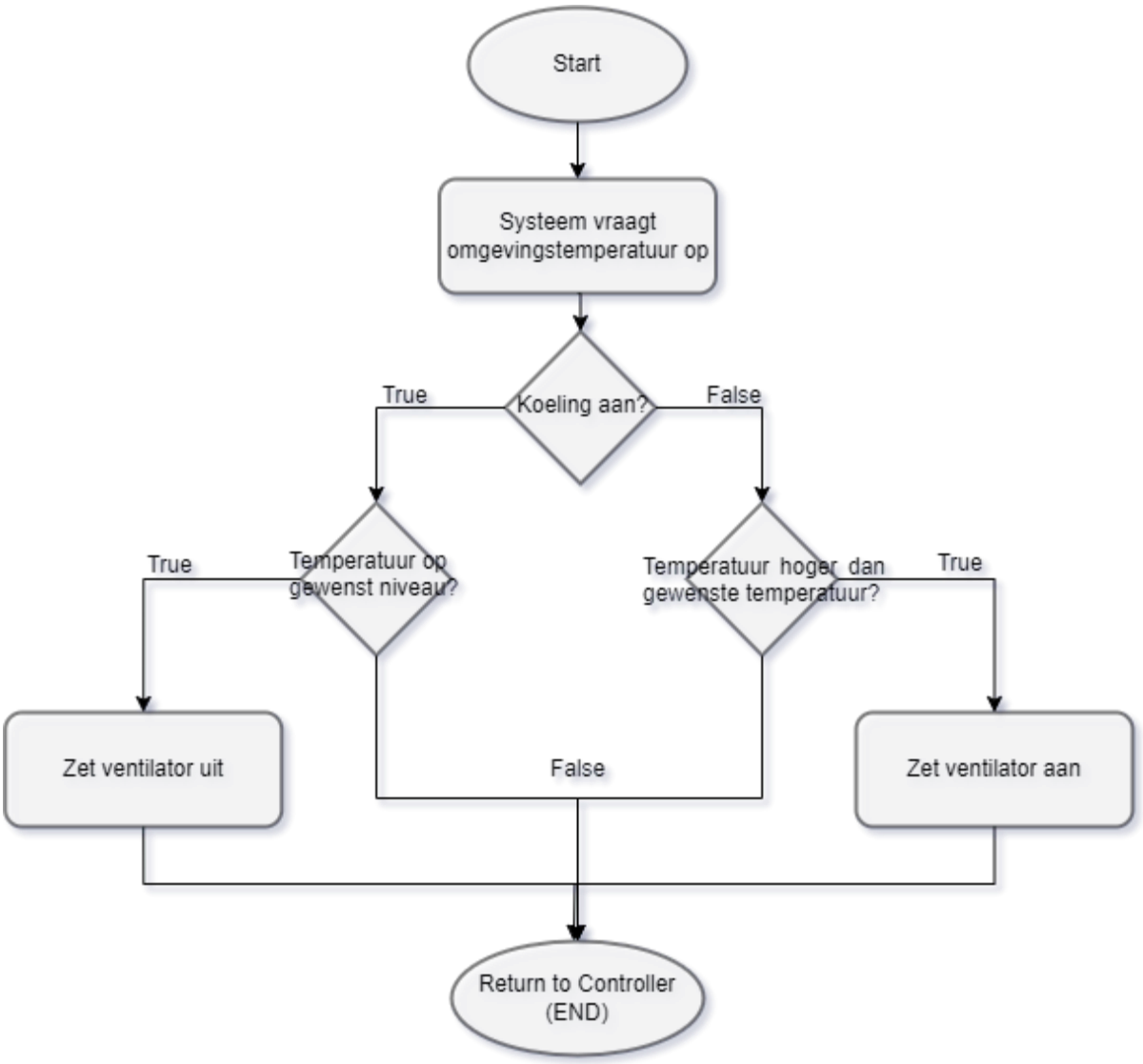
3.2.1. Flowcharts

De onderstaande flowcharts geven weer hoe de flow van de subsystemen eruit zullen zien.

3.2.1.1. Temperatuurregeling-subsysteem

Het Temperatuurregeling-subsysteem vraagt als allereerste de huidige omgevingstemperatuur op. Vervolgens wordt gecontroleerd of het koelsysteem reeds actief is. Als dit niet het geval is, wordt gekeken of de omgevingstemperatuur hoger is dan de gewenste temperatuur. Als dat het geval is, wordt het koelsysteem geactiveerd. Als dit niet het geval is, blijft de huidige toestand gehandhaafd.

Als het koelsysteem al actief is, wordt gecontroleerd of de omgevingstemperatuur de gewenste temperatuur heeft bereikt. Als dat niet het geval is, blijft de koelunit draaien om de gewenste temperatuur te handhaven. Zodra de gewenste temperatuur is bereikt, wordt de koelunit uitgeschakeld om energie te besparen.

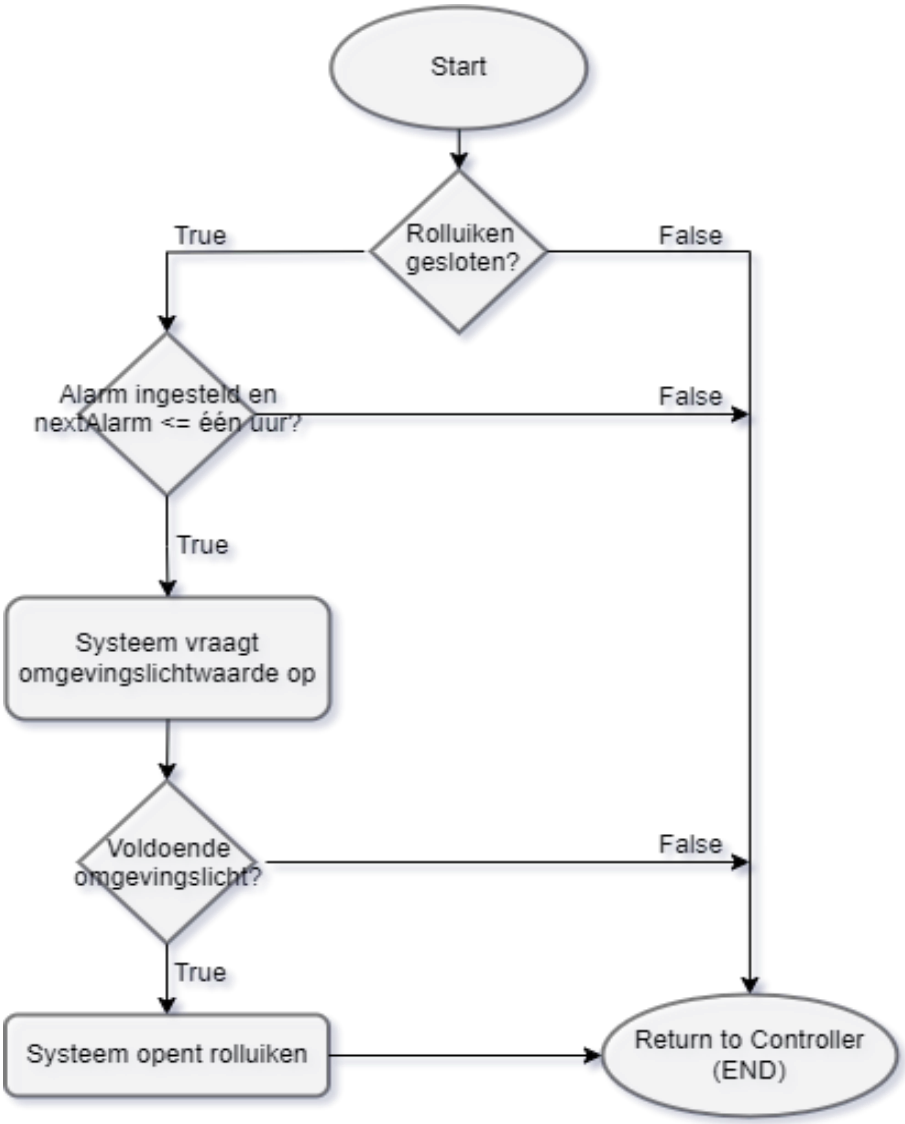


Figuur 2: Temperatuurregeling-subsysteem

3.2.1.2. Ochtendwekker-subsysteem

Het Ochtendwekker-subsysteem begint met een controle of het rolluik is gesloten. Nadat is geverifieerd dat het rolluik gesloten is, kijkt het systeem naar de ingestelde wekkertijd. Als er een alarm is ingesteld dat binnen het komende uur moet afgaan, wordt het proces voortgezet.

Op dit punt roept het systeem de lichtsensor om de actuele omgevingslichtwaarde te verkrijgen. Met deze informatie wordt gecontroleerd of de huidige lichtomstandigheden voldoende zijn om op een natuurlijk wijze wakker te worden. Indien er voldoende omgevingslicht is, initieert het systeem het openen van het rolluik. Door dit te doen, wordt de ruimte gevuld met natuurlijk zonlicht, waardoor het mogelijk wordt voor de gebruiker om aangenaam en geleidelijk wakker te worden.



Figuur 3: Ochtendwekker-subsysteem

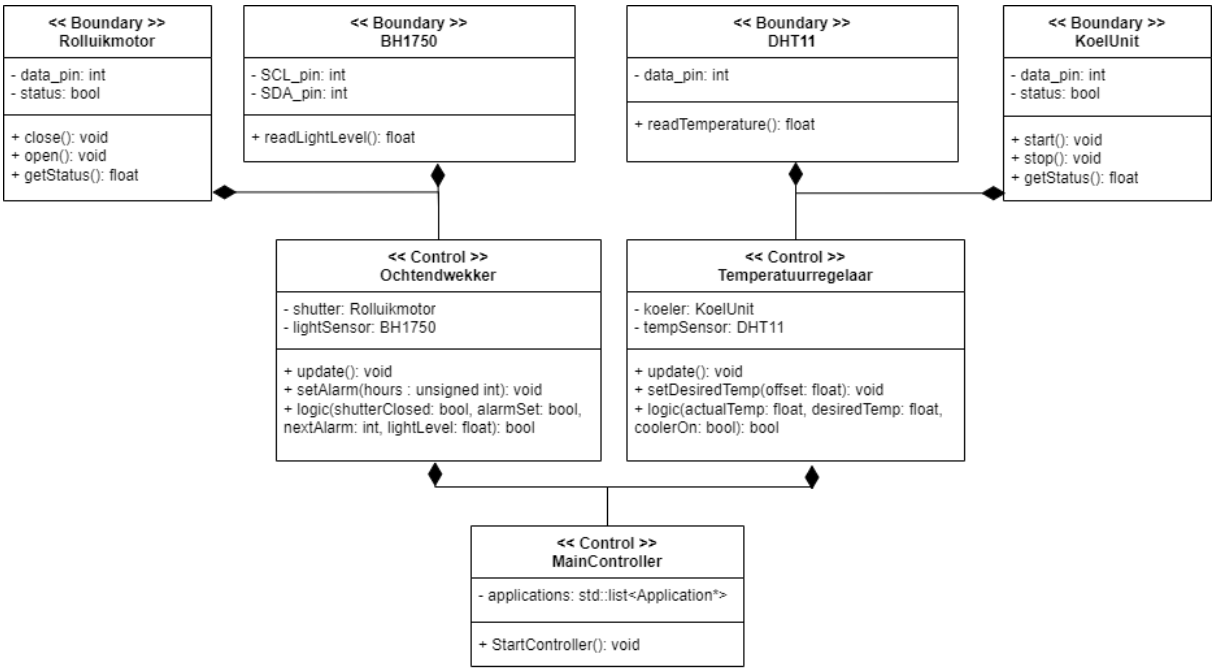
3.2.2. Klassen diagram

Zoals te zien in de klassen diagram afgebeeld in figuur 4, zal de “hardware” aansturing gedaan worden met interfaces. De code omtrent deze interfaces zullen geschreven worden in C++ en de kamer zal worden gesimuleerd in Python en dit allemaal zal worden gekoppeld met een binding zoals PyBind11.

De besturing van de sensoren en actuatoren worden in C++ geschreven. De realisatie van het controlesysteem zal in Python worden geschreven. Dit zal de student vervolgens met elkaar laten communiceren door een passende C++ Python koppeling te realiseren.

De boundary libraries zullen in C(++) geschreven worden. Het regelsysteem (controllers) zullen in python geschreven worden. Deze twee modules zal de student vervolgens met elkaar laten communiceren door een passende C++ Python koppeling (CDLL) te realiseren.

Daarnaast zullen ook de tests geschreven worden in Python door middel van een ondersteunende unit test library genaamd unittest.



Figuur 4: Klassen diagram C++ code met alle verschillende klassen, bijbehorende relaties en functies.

4. Kwaliteitscriteria

In de wereld van informatietechnologie is het leveren van kwalitatief hoogstaande software essentieel voor het succes van elk project. Normen zoals ISO 25010 [1] stellen criteria vast voor verschillende kwaliteitsaspecten, zoals functionaliteit, betrouwbaarheid en bruikbaarheid. Deze criteria zijn van cruciaal belang om de algehele kwaliteit van een IT-oplossing te waarborgen. Door deze normen te volgen, kunnen organisaties risico's verminderen, de tevredenheid van gebruikers verbeteren en de algehele efficiëntie van het project vergroten. Kortom, het integreren van kwaliteitscriteria is een sleutelpraktijk om het succes van IT-projecten te verzekeren.

Om de codekwaliteit te garanderen heb ik alle aspecten van ISO 25010 geprioriteerd, aangezien het regelsysteem relatief simpel systeem is, hebben sommige aspecten van ISO 25010 een ~~lage prioriteit~~ en zijn dus weggelaten, terwijl andere aspecten een hoge prioriteit hebben:

- Functionele Geschiktheid:
    - Functionele Volledigheid: De mate waarin de set functies alle gespecificeerde taken en gebruikersdoelen omvat.
    - Functionele Correctheid: Het vermogen van de software om de juiste resultaten te leveren.
    - Functionele geschiktheid: Mate waarin de functies het uitvoeren van gespecificeerde taken en doelstellingen vergemakkelijken.
- Betrouwbaarheid:
    - Volwassenheid: De frequentie van het falen van een product.
    - Beschikbaarheid: Mate waarin een systeem, product of onderdeel operationeel en toegankelijk is wanneer het nodig is voor gebruik.
- Onderhoudbaarheid:
    - Testbaarheid: Mate van effectiviteit en efficiëntie waarmee testcriteria kunnen worden vastgesteld voor een systeem, product of onderdeel, en tests kunnen worden uitgevoerd om te bepalen of aan die criteria is voldaan.

4.1 Belang van kwaliteitscriteria

Op basis van de prioriteiten heb ik de volgende relatieve belangstellingen toegewezen die ik nader zal toelichten.

Kwaliteitscriterium	Relatieve belangstelling (%)
1. Functionele Geschiktheid	40
2. Prestatie-efficiëntie	0
3. Compatibiliteit	0
4. Gebruiksvriendelijkheid	0
5. Betrouwbaarheid	30
6. Veiligheid	0
7. Onderhoudbaarheid	30
8. Overdraagbaarheid	0

- 1. Functionele Geschiktheid: Het systeem moet aan de vooraf gestelde eisen voldoen van de gebruiker anders kan je het project eigenlijk gelijk afschrijven.
- 2. Prestatie-efficiëntie: Voor dit systeem is de optimalisatie van code niet van belang aangezien het een relatief simpel systeem is.
- 3. Compatibiliteit: Dit product heeft geen predescessors en zal geen successors krijgen waar het rekening mee zal moeten houden.
- 4. Gebruiksvriendelijkheid: De gebruiker hoeft dit product niet al te makkelijk te kunnen gebruiken aangezien het product grotendeels geautomatiseerd is.
- 5. Betrouwbaarheid: Het product moet eigenlijk ten alle tijden beschikbaar zijn als het gedeployed is anders heb je niet echt veel aan het systeem.
- 6. Veiligheid: Het systeem beschikt zich niet over belangrijke data waardoor deze criteria niet echt van belang is.
- 7. Onderhoudbaarheid: Het is belangrijk in elk systeem om modulaire, herbruikbare, aanpasbare en testbare code te bevatten en daarom heeft het ook de hoogste prioriteit gekregen.
- 8. Overdraagbaarheid: Aangezien het product een one off product is hoeft de code niet overdraagbaar te zijn en heeft het dus ook geen prioriteiten gekregen.

5. Testen

Voor dit project moet één unit test, één integratietest en één systeemtest worden uitgevoerd. Alleen deze testen worden dan ook in dit testplan verder uitgewerkt en beschreven.

5.1. Unit test

Het doel van een unit test is om losse hardware of software componenten te testen. In mijn geval wil ik één van mijn subsystemen testen op 'functionele correctheid'.

Ik heb er voor gekozen om de logica van mijn temperatuurregeling subsysteem te testen, omdat een deel van het grotere systeem niet zal functioneren als deze sub-module foutieve resultaten oplevert.

De voorgestelde unit test voor het temperatuurregelingssubsysteem van het slaapkamer comfortsysteem, waarbij de temperatuur wordt gereguleerd door een koelunit en het slaapcomfort wordt beïnvloed, is van essentieel belang om te voldoen aan de functionele correctheidseisen van de ISO 25010-norm. Dit systeem speelt een cruciale rol bij het waarborgen van het comfort en welzijn van de gebruikers tijdens de slaap.

Testuitvoering:

1. De functie ontvangt drie waarden: de omgevingstemperatuur, de gewenste temperatuur en de status van de koelunit.
2. Vervolgens berekent de functie wat er moet gebeuren met de koelunit.
3. Na de berekening retourneert de functie de nieuwe gewenste staat voor de koelunit.

Dit proces wordt tweemaal uitgevoerd met nieuwe waarden voor elke flow in de flowchart, wat resulteert in acht unit tests voor de desbetreffende sub-module.

Slagingscriteria:

- De test is succesvol als de output van de functie overeenkomt met de flowchart in figuur 2.
    - Als de koelunit uit staat en de temperatuur hoger is dan gewenst, moet de ventilator aangaan (de functie retourneert 'start').
    - Als de koelunit aan staat en de temperatuur gelijk is aan de gewenste temperatuur, moet de ventilator uitgaan (de functie retourneert 'stop').
    - Als de koelunit uit staat en de temperatuur gelijk is aan de gewenste temperatuur, blijft het systeem in dezelfde staat (de functie retourneert 'none').
    - Als de koelunit aan staat en de temperatuur nog niet gelijk is aan de gewenste temperatuur, blijft het systeem in dezelfde staat (de functie retourneert 'none').

Potentiële risico's:

Een defect in de werking van dit subsysteem kan diverse nadelige gevolgen hebben, variërend van ongemak tijdens de slaap tot verstoring van de slaappatronen en zelfs gezondheidsproblemen. Dit komt voornamelijk doordat het koelsysteem mogelijk niet wordt geactiveerd onder de juiste omstandigheden, waardoor de gewenste temperatuurregeling niet wordt bereikt.

De unit test, die de functionaliteit van het temperatuurregelingssubsysteem valideert volgens de specificaties in de flowchart, is daarom van cruciaal belang. Het identificeert potentiële fouten in de logica van het systeem, zoals onjuiste reacties op temperatuurwensen. Het niet uitvoeren van deze test kan leiden tot een verminderde slaapkwaliteit en comfort voor de gebruikers, waardoor het systeem zijn beoogde doel niet kan bereiken en dus niet functioneel correct is.

5.2. Integratie test

Het doel van de integratietest is om de interactie tussen verschillende componenten van het systeem te valideren en te verifiëren. Deze testen richten zich op het testen van de samenwerking tussen softwarecomponenten en externe hardware, zoals sensoren en actuatoren. In mijn geval wil ik één van mijn subsystemen testen op 'functionele geschiktheid'.

Ik heb er voor gekozen om mijn unit test uit te breiden met echte waardes die komen uit de DHT11 in plaats van gedefinieerde temperatuur waardes. Om er zeker van te zijn dat de verschillende modules van de submodule correct met elkaar kunnen communiceren en samenwerken. Deze testen helpen om eventuele problemen of inconsistenties in de interactie tussen componenten op te sporen en te verhelpen, wat bijdraagt aan de algehele betrouwbaarheid en prestaties van het systeem.

Testuitvoering:

1. De functie ontvangt net als de unit test hiervoor drie waarden, maar dit keer is de omgevingstemperatuur daadwerkelijke sensor reading.
2. Vervolgens berekent de functie weer wat er moet gebeuren met de koelunit.
3. Na de berekening retourneert de functie de nieuwe gewenste staat voor de koelunit.

Dit proces wordt tweemaal uitgevoerd.

Slagingscriteria:

- De test is succesvol als de logic functie een valide output kan generen.
    - "start", "stop" of "none"

Potentiële risico's:

Zonder integratietests bestaat het risico dat problemen of inconsistenties in de interactie tussen de componenten niet worden opgemerkt en verholpen. Dit kan leiden tot verstoringen in de functionaliteit van het systeem, zoals het niet correct regelen van de temperatuur door de koelunit, wat kan resulteren in ongewenste omgevingsomstandigheden.

De integratie test, die de functionaliteit van onderdelen van de thermostaat submodule valideert volgens de specificaties in de flowchart, is daarom van cruciaal belang. Het identificeert potentiële fouten in samenwerking van modules van het systeem, zoals onjuiste reacties op temperatuurwensen. Het niet uitvoeren van deze test kan leiden tot een verminderde slaapkwaliteit en comfort voor de gebruikers, waardoor het systeem zijn beoogde doel niet kan bereiken en dus niet functioneel geschikt is.

5.3. Systeem test

Het doel van de systeemtest is om het gehele systeem als één geheel te testen en te verifiëren om ervoor te zorgen dat het voldoet aan de functionele en niet-functionele eisen zoals gespecificeerd in de systeemspecificaties.

Ik heb ervoor gekozen om mijn temperatuurregeling subsysteem in zijn volledigheid te testen op 'functionele correctheid'. De systeemtest is van cruciaal belang om de algehele prestaties, betrouwbaarheid en bruikbaarheid van het systeem te valideren en te verifiëren. Het testen van het systeem in zijn geheel helpt om eventuele gebreken of tekortkomingen op te sporen en ervoor te zorgen dat het systeem voldoet aan de verwachtingen van de eindgebruikers.

Testuitvoering:

1. De gewenste temperatuur wordt ingesteld.
2. De test simuleert het gedrag van het systeem gedurende een periode van 24 uur (24 seconden).
3. Tijdens elke update wordt gecontroleerd of de status van de koelunit overeenkomt met het verwachte gedrag op basis van de temperatuurmetingen en de gewenste temperatuur.

Slagingscriteria:

- De test is succesvol als gedurende de simulatieperiode de status van de koelunit overeenkomt met het verwachte gedrag op basis van de temperatuurmetingen en de gewenste temperatuur.
    - Als de koelunit is ingeschakeld (status True), moet de gemeten omgevingstemperatuur hoger zijn dan de gewenste temperatuur min een marge van 1.0 graad Celsius.
    - Als de koelunit is uitgeschakeld (status False), moet de gemeten omgevingstemperatuur lager zijn dan de gewenste temperatuur plus een marge van 1.0 graad Celsius.

Zonder systeemtests is er geen garantie dat het systeem als geheel correct functioneert onder echte omstandigheden. Dit kan leiden tot onverwachte fouten of storingen in de productieomgeving, wat de bruikbaarheid en betrouwbaarheid van het systeem negatief kan beïnvloeden.

Soms kunnen fouten optreden tijdens de integratie van afzonderlijke componenten die niet worden gedetecteerd tijdens de ontwikkeling of unit testing. Deze fouten kunnen sluimeren en later tot problemen leiden wanneer het systeem als geheel wordt uitgevoerd. Systeemtests helpen om deze verborgen fouten op te sporen voordat het systeem wordt vrijgegeven voor productie.

6. Decorators

Bij dit project zal er gebruikt gemaakt worden van vier decorators: measure time, validate input, memoize, log.

6.1. Measure time

De decorator ‘measure_time’ meet de uitvoeringstijd van een functie en geeft deze informatie weer door het te loggen.

Wanneer deze decorator wordt toegepast, kunnen we de prestaties van verschillende functies binnen het systeem evalueren en eventuele knelpunten in de uitvoeringstijd identificeren.

Zonder de ‘measure_time’ decorator zouden we geen inzicht hebben in de uitvoeringstijd van de verschillende functies binnen het systeem. Dit zou het moeilijker maken om prestatieproblemen op te sporen en te optimaliseren en/of verbeteren.

```
def measure_time(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        print(f"\nFunction '{func.__name__}' took {time.time() - start_time:.6f} seconds to execute.")
        return result
    return wrapper
```

Figuur 5: decorator measure_time.

6.2. Validate input

De decorator ‘validate_input’ valideert de invoerparameters van een functie op basis van de opgegeven datatypen.

Wanneer deze decorator wordt toegepast, zorgt het ervoor dat de functie alleen wordt uitgevoerd als de invoer voldoet aan de verwachte typen. Als de invoer niet overeenkomt met de verwachte typen, wordt een ‘TypeError’ gegenereerd, wat helpt om fouten in de invoer te voorkomen en de robuustheid van het systeem te verbeteren.

Zonder deze decorator zou de functie onjuiste invoer accepteren, wat kan leiden tot onverwacht gedrag of fouten.

```
def validate_input(*valid_args):
    def decorator(func):
        def wrapper(*args, **kwargs):
            for arg, valid_arg in zip(args, valid_args):
                if not isinstance(arg, valid_arg):
                    raise TypeError(f"'{type(arg).__name__}' is not of expected type '{valid_arg.__name__}'")
            return func(*args, **kwargs)
        return wrapper
    return decorator
```

Figuur 6: decorator validate_input.

6.3. Memoize

De decorator ‘memoize’ verbetert de prestaties van een functie door reeds berekende resultaten op te slaan en ze opnieuw te gebruiken wanneer dezelfde invoer wordt gegeven.

Door het toepassen van deze decorator wordt de berekening van resultaten verminderd, wat vooral nuttig is voor functies met dure berekeningen of herhaalde oproepen met dezelfde invoer.

Zonder deze decorator zou de functie herhaalde berekeningen uitvoeren, wat onnodige overhead met zich meebrengt en de algehele prestaties van het systeem kan verlagen.

```
def memoize(f):
    memo = {}
    def helper(x):
        if x not in memo:
            memo[x] = f(x)
        return memo[x]
    return helper
```

Figuur 7: decorator memoize.

6.4. Log

De decorator ‘log’ biedt logging-functionaliteit voor functieoproepen, waardoor een traceerbaar spoor van de uitvoering van het programma wordt gecreëerd.

Bij gebruik van deze decorator worden de argumenten en het retourresultaat van een functie opgenomen in de logboeken, wat nuttig is voor het debuggen en analyseren van het systeemgedrag.

Zonder deze decorator zouden dergelijke gegevens niet automatisch worden vastgelegd, wat het moeilijker zou maken om de oorzaak van fouten te achterhalen en het algemene systeemgedrag te begrijpen.

```
def log(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(f"Called '{func.__name__}' with args: {args}, kwargs: {kwargs},
returned: {result}")
        return result
    return wrapper
```

Figuur 8: decorator log.

7. Functional Programming

In dit project zullen twee principes uit functioneel programmeren worden toegepast: pure functies en hogere-orde functies.

7.1 Pure functies

Een belangrijk concept dat ik heb toegepast, is het gebruik van pure functies, zoals te zien bij de thermostaatLogic-functie in figuur 8. In deze functie accepteert thermostaatLogic de nodige parameters (actualTemp, desiredTemp, coolerOn) en retourneert een waarde op basis van die parameters. Door de afwezigheid van bijwerkingen op de interne toestand van een object, wordt thermostaatLogic beschouwd als een pure functie.

Door dergelijke functies te gebruiken, wordt de code voorspelbaar en gemakkelijk te begrijpen. Dit maakt het makkelijker om te testen en debuggen, en biedt voordelen zoals verbeterde parallelle uitvoering en verminderde complexiteit van het systeem.

```
def thermostaatLogic(actualTemp, desiredTemp, coolerOn):
    if coolerOn:
        if actualTemp <= desiredTemp - 1.0:
            return "stop"
    else:
        if actualTemp >= desiredTemp + 1.0:
            return "start"
    return "none"
```

Figuur 8: pure functie thermostaatLogic.

7.2. Hogere-orde functies

Een belangrijk concept dat ik heb toegepast, is het gebruik van hogere-orde functies in de vorm van decorators te zien in hoofdstuk zes decorators. Laten we eens kijken naar hoe elk van de decorators voldoet aan dit principe.

1. Measure time: Deze decorator neemt een functie als argument (func) en retourneert een nieuwe functie (wrapper), die het gedrag van de originele functie wijzigt door timinginformatie logs toe te voegen.
2. Validate input: Net als measure_time, accepteert deze decorator een functie en retourneert een nieuwe functie die extra validatie uitvoert voordat de originele functie wordt aangeroepen.
3. Log: Deze decorator gedraagt zich op dezelfde manier als measure_time en validate_input. Het accepteert een functie en retourneert een nieuwe functie die het gedrag van de originele functie wijzigt door loginformatie toe te voegen.
4. Memoize: Deze decorator is iets anders. In plaats van een nieuwe functie terug te geven, retourneert memoize een innerlijke functie (helper). Deze innerlijke functie wordt vervolgens geretourneerd als de wrapper.

Elk van deze decorators past dus binnen het concept van higher order functions omdat ze ofwel een functie accepteren en een nieuwe functie retourneren, ofwel een functie retourneren die later kan worden aangeroepen. Dit maakt ze krachtige hulpmiddelen voor het aanpassen en uitbreiden van de functionaliteit van andere functies in je programma.

Datasheets

[1] ISO, „ISO 25010,” [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. [Geopend 9 November 2023].

[2] R. Semiconductor, „BH1750FVI : Sensor ICs,” November 2011. [Online]. Available: https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf. [Geopend 25 September 2023].

[3] L. Aosong Electronics Co., „DHT11-Temperature-Sensor.pdf,” [Online]. Available: https://components101.com/sites/default/files/component_datasheet/DHT11-Temperature-Sensor.pdf. [Geopend 25 September 2023].