calgon | Nijenoord 1

calgon

Joost Wagensveld-van Veen 1664713 Zehna van den Berg 1662506 Jessy Visch 1661709 Koen de Groot 1638079

Thema Opdracht 6 Technische Informatica

2016

# Samenvatting.

In dit document zal nader in gegaan worden op alle onderdelen van het proces die wij als groep hebben doorlopen om tot het eindproduct te komen. Er zal uitleg gegeven worden over de verschillende modellen die gebruikt zijn in het plannen en organiseren van het project. Er zal globaal ingegaan worden op de requirements, de *requirements architectuur*, de *solution architectuur* en de werking van de code. Deze documenten zullen als bijlagen bijgevoegd worden, ze zullen echter niet in detail behandeld worden in dit document.

Aan het einde van het verslag zal er nog kort stukje met per teamlid eigen bevindingen, wat ging goed want niet, te lezen zijn. Welke onderdelen waren uitdagender dan verwacht. Ook zal er kort wat zegen worden over hoe de samenwerking gegaan is.

De conclusie en de aanbeveling zullen het verslag afsluiten. In het hoofdstuk *Aanbevelingen* zal voornamelijk gekeken worden naar wat wij als verbeterpunten zien. We zullen ook kort uitweiden wat wij er graag bij zouden willen maken als we hier de tijd voor zouden hebben. In de conclusie zal kort samengevat worden wat onze persoonlijke bevindingen zijn. Natuurlijk word hier ook nog even terug gekeken naar het gehele proces en hoe wij denken hoe dat het in het vervolg beter kan gaan.

De conclusie en de aanbeveling zullen het verslag afsluiten. In het hoofdstuk *Aanbevelingen* zal er vertelt worden wat volgens ons de punten zijn die een volgend team nog toe zou kunnen uitvoeren. Dit zijn onderdelen waar wij zelf niet aan toe zijn gekomen. In de conclusie zal terug gekeken worden op het hele proces. Wat goed en wat fout ging en hoe het in het vervolg beter zou kunnen.

# Inhoudsopgave

[Samenvatting. 2](#_Toc440972508)

[Inhoudsopgave 3](#_Toc440972509)

[1. Inleiding 5](#_Toc440972510)

[2. De wasmachine 6](#_Toc440972511)

[2.1 Algemeen 6](#_Toc440972512)

[2.1.1 De opdracht 6](#_Toc440972513)

[2.1.2 Bediening 6](#_Toc440972514)

[2.1.3 Communicatie van de web browser en wasmachine 6](#_Toc440972515)

[2.1.4 Uitvoering nodige stappen wasmachine 7](#_Toc440972516)

[2.1.5 Log systeem 7](#_Toc440972517)

[2.2 Screenshots van de webinterface 7](#_Toc440972518)

[3. Ontwikkelomgeving 8](#_Toc440972519)

[3.1 Ontwerpen 8](#_Toc440972520)

[3.2 Programmeren 8](#_Toc440972521)

[3.3 Testen 8](#_Toc440972522)

[4. Solution architectuur 9](#_Toc440972523)

[4.1 Klassendiagram 9](#_Toc440972524)

[4.2 Concurrency diagram 9](#_Toc440972525)

[4.3 State transition diagram 9](#_Toc440972526)

[4.4 Taalstructurering 10](#_Toc440972527)

[4.5 Communicatie protocol webserver websocket 10](#_Toc440972528)

[5. Requirement Architecture 11](#_Toc440972529)

[5.1 Usecase diagram 11](#_Toc440972530)

[5.1.1 Wasprogramma uitvoeren: 11](#_Toc440972531)

[5.1.2 Wasprogramma stoppen: 11](#_Toc440972532)

[5.1.3 Update aanbieden: 11](#_Toc440972533)

[5.1.4 Activiteiten logs weergeven: 11](#_Toc440972534)

[5.1.5 Systeem logs weergeven: 11](#_Toc440972535)

[5.2 Activity diagram 11](#_Toc440972536)

[5.3 Constraints diagram 12](#_Toc440972537)

[6. Testen 13](#_Toc440972538)

[7. Conclusie 14](#_Toc440972539)

[8. Evaluatie 14](#_Toc440972540)

[9. Suggesties en aanbevelingen 15](#_Toc440972541)

[10. Bijlagen 16](#_Toc440972542)

# Inleiding

We zullen het hier gaan hebben over alles dat van belang was voor het ontwikkelen van de over internet bestuurbare wasmachine. We zullen het hebben over de test die we hebben gedaan om tot bepaalde beslissingen te komen. Ook zullen we uitgebreid praten hoe de uiteindelijke bediening en werking van de wasmachine is geworden en de keuzes die we hier in gemaakt hebben. We hopen hier door een duidelijk beeld te geven over het proces dat wij als groep hebben doorlopen. Als afsluiting zullen we onze bevindingen als groep en wat wij denken dat de wasmachine nog beter kan maken.

***\*\*\*De student maakt een voorblad paginanummering, hoofdstuk- en paragraafnummering, index, referenties volgens APA, titels en kopjes.\*\*\****

***\*\*\* De student gebruikt verwijzingen en bronvermeldingen volgens APA.\*\*\****

\*\*\*De student maakt een managementsamenvatting van het gehele document\*\*\*

*\*\*\*De student schrijft een inleiding met opening, doel en vooruitblik\*\*\**

***\*\*\*De student beschrijft op heldere wijze de requirements, de requirements architecture, de solution architecture, de werking van de code en de uitgevoerde tests. \*\*\****

***\*\*\*De student schrijft een afsluitend hoofdstuk met conclusie(s) en aanbevelingen.\*\*\****

***\*\*\*De student maakt gebruik van bijlagen\*\*\****

# De wasmachine

## Algemeen

in het volgende hoofdstuk zal er gekeken worden naar de eigenschapen en onderdelen van de wasmachine.

### De opdracht

De opdracht die wij van Swirl Industries hebben gekregen. Was het ontwikkelen van software en een wasmachine.

Deze wasmachine moest op een voor de gebruiker eenvoudige manier bestuurbaar zijn over het internet. Hier voor moest er gebruik gemaakt worden van een web browser. Vanaf deze browser moest het vervolgens mogelijk zijn om een wasprogramma te selecteren. Deze wasprogramma’s werden mee geleverd op de wasmachine. Wel moest het mogelijk zijn dat de gebruiker de tempratuur aan kon passen naar wat wenselijk was voor de was die zij wilde draaien. De gebruiker zou ook aan moeten kunnen geven dat de was om een bepaalde tijd moest starten.

De laatste optionele wens die Swirl Industries had voor deze wasmachine was dat er logs bijgehouden zou worden. Van deze logs zouden er 2 versies bestaan. Hier later meer over. Verder moet de wasmachine natuurlijk alles kunnen dat een gewone machine kan.

### Bediening

De bediening van de wasmachine wordt gedaan via het internet. Hier voor is er met HTML, CSS en javascript een webpagina gemaakt. Deze pagina voorzien van een gebruiksvriendelijke vormgeving en makkelijk te gebruiken bediening, zorgt er voor dat de gebruiker op eenvoud wijze het gewenste wasprogramma kan starten. Wij hebben qua vormgeving gekozen voor groten knoppen en niet overdreven veel informatie in een scherm. Dit vooral met het oog op tablet en smartphones.

### Communicatie van de web browser en wasmachine

Communicatie gaat zoals het woord al zegt 2 kanten op. De wasmachine kan dus niet alleen berichten ontvangen van af het web maar hij kan ze ook versturen. Het is hierdoor mogelijk om de gebruiker te voorzien van meer informatie dan bij een normale wasmachine. Op deze manier kan de gebruiker in het venster waar de was gestart wordt informatie zien. Informatie die bijvoorbeeld te zien is:

* Deur status (is de deur open of dicht);
* Deur vergrendeling (zit de deur op slot? ja/nee);
* Waterniveau (voor hoeveel procent is de trommel gevuld met water);
* Watertemperatuur (wat is de tempratuur van het water);
* Resterende tijd (wat de geschatte resterende tijd van het wasprogramma is).

Dankzij deze informatie kan de gebruiker een goed overzicht behouden wat de wasmachine doet.

### Log systeem

Het log systeem bestaat uit 2 verschilden logs die bijgehouden worden.

Een systeem dat alleen maar de hoofd activiteiten van de wasmachine bijhoudt. Deze is bedoeld voor de gebruiker. De gebruiker kan bijvoorbeeld in dit systeem opzoeken wanneer welke was gedraaid was en hoe lang dat geduurd heeft.

Een ander systeem dat alle processen en (fout-)meldingen bijhoudt. Dit gedeelte is alleen bedoeld voor onderhoudsmonteurs die aan de wasmachine moeten werken. In deze log files worden alle stappen die de wasmachine doorloopt bijgehouden. Stap voor stap wordt er opgeslagen wat hoe laat gestart is en hoe lang dit geduurd heeft. Deze log files zijn niet bedoeld voor de gebruiker maar voor onderhoud/reparatie van het apparaat. Zo moet het bijvoorbeeld mogelijk zijn dat een reparateur kan zien dat het verwarmingselement niet goed werkt. Dit zou een monteur kunnen zien doordat uit de gegevens blijkt het verwarmen van het water steeds langer duurt.

## Screenshots van de webinterface

# Ontwikkelomgeving

Een ontwikkelomgeving is alles van de computersoftware en de hulpmiddelen die de ontwikkelaar ondersteunen bij het ontwikkelen van de software.

De aspecten van een ontwikkelomgeving hangen af van de taken die moeten worden uitgevoerd. Er kan een onderscheidt gemaakt worden tussen het ontwerpen, programmeren en het testen van de software.

## Ontwerpen

Bij het ontwerpen zal de omgeving vooral bestaan uit modelleringgereedschappen en programma’s waarmee je modellen kan maken en aanpassen.

De modelleringgereedschappen bieden ondersteuning in het maken van bijvoorbeeld een UML, klassendiagram en dergelijke. Deze worden ontworpen met behulp van het modellering programma Software Ideas Modeler.

## Programmeren

Notepad++, PSPad

## Testen

Bij het testen wordt de ontwikkelde software getest. Voldoet de test aan de eisen dan wordt deze versie opgeslagen. Is dit niet het geval dan wordt de code aangepast en weer getest. Dit wordt herhaald totdat het geteste item naar behoren werkt.

Ook wordt de code door meerdere teamleden bekeken en getest. Dit om er voor te zorgen dat de code niet alleen goed werkt maar ook volgens de afspraken is opgemaakt.

# Solution architectuur

Solution architecturen worden gemaakt om een goed en overzichtelijk beeld te geven over wat de beoogde werking is van de software. Hoe is deze ingedeeld en hoe communiceert die met de anderen onderdelen. Ook word hier gekeken naar wat de prioriteit is van welke onderdelen. Om al deze vragen te beantwoorden wordt er gebruik gemaakt van een aantal diagrammen. Het zo genoemde klassendiagram, het *concurrency diagram* en het *state transition diagram*. Verder word er ook een taakstructurering en anderen nodige protocollen gemaakt.

## Klassendiagram

het *klassendiagram* wordt gebruik om een overzicht te geven van alle onderdelen van de software. De “klassen” zijn in dit geval de verschillende objecten die nodig zijn om de software te laten werken.

In dit diagram wordt is dus een overzicht te zien van hoe de verschillende objecten opgemaakt zijn, welke functionaliteit deze hebben en hoe ze met de anderen objecten kunnen praten. Zo is hier ook in terug te zien dat niet elke klassen (objecten) met elkaar verbonden zijn.

Dit is voor de programmeurs handig omdat ze dan snel kunnen terug zien hoe de verschillende onderdelen zijn opgebouwd. Zeker als er meerdere programmeurs tegelijk aan onderdelen werken zorgt dit er voor dat ze van tevoren al weten wat er ongeveer verwacht kan worden betreffende anderen klassen.

## Concurrency diagram

het concurrency diagram dient er voor om een beeld te geven over hoe bepaalde onderdelen berichten over en weer sturen naar mekaar. Dit zorgt er voor dat de programmeurs een duidelijk beeld kunnen vormen van hoe bepaalde onderdelen van de code de informatie waar zij mee moeten werken kunnen krijgen en zo nodig hoe zij deze terug sturen naar de genen die er om gevraagd heeft

## State transition diagram

het state transition diagram is een diagram dat in detail afbeeld in wat voor stadiums het programma terecht kan komen. met behulp van dit diagram kan er dus goed uitgebeeld worden wat de weg is die het programma kan nemen om zijn nodige stappen door te lopen. Het geeft ook gelijk weer welke keuzen en/of afwegingen het programma maakt terwijl het van stadium naar stadium gaat.

## Taalstructurering

De toolstructurering is een overzicht van alle taken die de verschillende objecten hebben. Hier word gelijk gekeken naar de prioriteit van deze taken. De prioriteit wordt beïnvloed door verschillende onderdelen. Het maakt namelijk een hoop uit hoe vaak een bepaalde taak uitgevoerd moet worden en wat de deadline is waar binnen deze taak klaar moet zijn met uitvoeren.

## Communicatie protocol webserver websocket

Het communicatie protocol tussen deze onderdelen zorgt voor een structuur waar de programmeurs snel aan kunnen zien hoe of wat ze zouden moeten versturen/ontvangen wanneer ze iets van of naar de webserver sturen. Door dit protocol op te stelen zorg je er voor dat op alle plekken in de code dit op soort gelijke manier gedaan kan/moet worden.

# Requirement Architecture

De naam zegt het al een beetje. De *Requirements Architecture* wordt gemaakt om voordat er geprogrammeerd word vast te kunnen stellen wat er ongeveer nodig gaat zijn om een goed werkten systeem te maken.

Hier bij word gebruik gemaakt van meerdere diagrammen namelijk: *Usecase Diagram*, *Activity Diagram* en *Constraints Diagram*.

## Usecase diagram

De *Usecase* is bedoeld om een beeld te kunnen vromen over hoe de ideale gebruiker gebruik maakt van het product. We hebben het hier alleen over de ideale gebruiker omdat wanneer je een product maakt je nooit alles kan voor spellen in wat een gebruiker zou kunnen proberen. Natuurlijk moet er bij er schrijven van de code voor gezorgd worden dat onvoorzien gedrag van een gebruiker niet het hele systeem om gooit. Maar dit is niet het doel dan bereikt moet worden met een *Usecase Diagram*.

*Usecases* worden gebruikt om snel duidelijk te maken wat de gebruiker richting het systeem kan doen en wat het systeem richting de gebruiker moet doen voor de wasmachine hebben we het dan over: Wasprogramma uitvoeren, Wasprogramma Stoppen, Update aanbieden, Activiteiten logs weergeven, System logs weergeven.

### Wasprogramma uitvoeren:

het kunnen selecteren van het gewenste programma die vervolgens door het systeem uit gevoed moet kunnen worden.

### Wasprogramma stoppen:

Het stoppen van het huidige wasprogramma dat uitgevoerd word.

### Update aanbieden:

Het aanbieden van een update voor het systeem en of de wasprogramma’s die gebruiker tot zijn beschikking heeft.

### Activiteiten logs weergeven:

Toont een log met alle activiteiten die de wasmachine door heeft uitgevoerd. Zoals eerder genoemd is dit log alleen bedoeld om de gebruiker te kunnen vertellen hoe veel wassen er gedraaid zijn en om hoe laat welke gestart is.

### Systeem logs weergeven:

Toont in groter detail wat het systeem gedaan heeft en hoe lang wat heeft gedaan. Denk bijvoorbeeld aan het op warmen van het water Of het vullen van de trommel met water.

## Activity diagram

Het *Activity Diagram* is bedoeld om een over zicht te geven van wat het systeem doet van begin tot einde. Deze “tekening” is nog niet een uitwerking van hoe het systeem (de wasmachine) echt gaat werken. Het is meer een globaal over zicht van welke stappen het systeem moet door lopen om tot het gewenste eind resultaat te komen. Er wordt hier dus gekeken naar het aanzetten van de machine en wat hij moet door lopen om met succes een was te draaien. Hoe lang welke stap moet duren is daar aan tegen iets dat op dit moment minder belangrijk is.

## Constraints diagram

Dit diagram wordt niet zo zeer gebruikt om vast te stellen wat het systeem moet doen. Dat wil niet zegen dat het hier niet ook voor gebruikt kan worden maar wat dit diagram voornamelijk zo belangrijk maakt is dat het een beeld geeft aan welke eisen het systeem moet vol doen (hardware en software). Bijvoorbeeld het meten en weergeven van de temperatuur van het water in de trommel mag niet al te veel afwijken. Het is voor het draaien van een was natuurlijk ook niet nodig dat ik op een honderdste graad precies kan meten. Maar het is ook zeker niet de bedoeling dat de sensor 20 graden meet terwijl het eigenlijk 120 zou moeten zijn. Dit is natuurlijk niet het meest realistisch voorbeeld en zal in deze extremen niet snel voorkomen. Het zijn wel dingen waar voordat je begint over na gedacht moet worden. Zeker als het gaat over dingen die je alleen maar kunt weten wanneer je dit specifiek test. Zo is natuurlijk wel een probleem wanneer je geen rekening houd met het feit dat je microprocessor te klein is voor de software die je geschreven hebt.

# Werking van de code

De software is geschreven in C++ aan de hand van het klassendiagram. De software is op te delen van verschillende onderdelen. De sensors, het wasprogramma uitvoeren en de websever.

De sensors :

Door middel van de SensorHandler word er gezorgt dat de sensoren beschikken over de meest recente data. Dit doet hij simpel weg door eens in de halve seconde alle sensoren de opdracht te geven nieuwe data op te vragen en deze te bewaren. Aan de sensoren kan vervolgens gevraagd worden wat de data is die zij hebben. Deze informatie kan dan verder in de software gebruikt worden om te bepalen wat de volgende stap moet zijn.

Het wasporgramma:

De webserver

# Testen

Er zijn tijdens het project meerdere tests/onderzoeken gedaan om tot sommige beslissingen te komen of om kennis op te doen over de werking van onderdelen.

Zo is er een onderzoek gedaan naar:

welke webserver software pakket het meesten pasten bij onze wens,

hoe de Linux queue functioneert,

hoe een wasprogramma er uit ziet,

hoe de raspberry pi zij ip-adress bekent kan maken,

hoe RTOS functioneert,

hoe de wasmachine emulator werkt.

In dit hoofdstuk zullen al deze onderzoeken aanbod komen. Er zal per onderzoek een korte samenvatting worden wat de einde conclusie is en waarom dit is besloten.

## welk webserver software pakker gaan we gebruiken?

In dit onderzoek hebben we gekeken naar 2 verschillende software pakketen voor webservers Apache en Nginx. Beiden software pakketen heeft zo zijn voordelen en nadelen. Zo is Nginx over het algemeen sneller. Er word namelijk niet voor iedere verbinding een nieuwe process gestart. Dit zorgt er ook voor dat hij minder geheugen gebruikt om de pi. Daar aan tegen is de mogelijkheden en documentatie van Nginx minder uitgebreid.

Hoewel Nginx sneller is en minder geheugen verbruikt is er als team toch gekozen voor apache. Deze keuze is genomen omdat de snelheid van Nginx voor onze doeleinden niet van belang was. Je zou namelijk echt pas wat van deze snelheid.,;.,;.,;,.;,.,.,.,.,..,.,.p;;;;.,.,;.,;.,;.,;.,;.,’’’’’’’’’’;.,.’ merken wanneer er meerdere verbindingen tegelijk open staan. Dit is bij de wasmachine niet het geval er zal namelijk altijd alleen maar een verbinding zijn van de server naar het web zijn en van de server naar de wasmachine. Ook de uitgebreidere documentatie en de al bestaande kennis binnen het team maakte apache de beste keuze voor ons.

## Hoe functioneert de Linux queue?

Tijdens het uitvoeren van dit onderzoek kon al snel worden geconcludeerd dat de Linux queue veel ingewikkelder was dan voor ons nodig. Hierom veranderde de richting van het onderzoek naar wat een betere optie was om ons doel te bereiken. We zijn toen uitgekomen bij de STD::queue. Deze queue is onderdeel van de STD libary. Omdat we deze libary toch al gebruikte was het natuurlijk een logische keuze. De queue werkt op het First in First out principe. De aderen bewerkingen vallen in de zelfde vorm en logica als anderen onderdelen van de libary. Dit zorgde er voor dat het makkelijk te begrijpen was en snel te implementeren.

## Hoe ziet een wasprogramma er uit?

Het doe van dit onderzoek was om een duidelijk beeld te krijgen van wat er kwam kijken tijdens het uitvoeren van een wasprogramma. Welke stappen loopt hij door en hoe lang duren deze stappen ongeveer. Dit moest er ook voor zorgen dat de verschillen tussen de wasprogramma’s duidelijk werden.

## Hoe kan de rasoberry pi zijn ip-adress bekent maken?

Er zijn verschillende manieren om dit te kunnen doen. De simpelste manier zou zijn om een serielele kabel aan te sluiten en via die verbinding het aan de rasberry pi te vragen. Maar dat moet je elke keer weer met die kabel en aansluiting rommelen. Na wat onderzoek op internet vonden we een Python script die als deze in het opstarten van de pi word uit gevoerd een e-mail stuurt met daar in zijn ip addres. Dit was voor ons de beste oplossing geen gerommel met kabels en anderen verbindingen. De rasberry pi vertelde het ons gewoon zelf terwijl wij ons op anderen onderdelen richten.

## Hoe functioneert RTOS?

## Hoe functioneert de wasmachine emuator?

# Conclusie

We hebben het in dit document gehad over alle onderdelen van het project. Van de opdracht tot de werking van onderdelen code. We hebben uitleg gegeven over de keuzes die we hebben gemaakt. Keuzes over wat volgens ons belangrijk is voor de wasmachine. Deze keuzes hebben ons geholpen bij het vormen van een beeld over hoe de software er uit moest komen te zien.

Tijdens dit project hebben we een hoop beslissingen genomen die hebben bepaalt hoe de software voor ons er uit ging zien. Dat begon allemaal met de keuzes van welke onderdelen *must-haves* en welke *should-haves* zouden zijn. Aan de hand hiervan is de *Requirement architectuur* gemaakt. Met deze *Requirement architecture* kon een duidelijk beeld gevormd worden voor het maken van de *Solution architectuur*. Nadat deze documenten klaar waren was er een goed beeld van hoe alle code samen zou komen en hoe de communicatie gaat verlopen. Hierna is de code van de wasmachine en webserver gemaakt en zijn we tot ons product gekomen.

# Evaluatie

# Suggesties en aanbevelingen

# Bijlagen