**TESTONTWERP & -REALISATIE**

|  |  |
| --- | --- |
| **Naam** | Jay Bhageloe |
| **Studentnummer** | 22072039 |
| **Klas** | Se.H5 |

**OPDRACHT**

Geef hieronder een korte beschrijving van je opdracht (een samenvatting van je productvisie) zodat je docent OPT3 ook begrijpt met welke opdracht jij bezig bent.

Mijn opdracht helpt de productgever met het plannen dat een andere manier gebruikt dan wat de school aanbeveelt aan de studenten. Een applicatie waar je invoert hoeveel lessen, projecten, deadlines of toetsen je hebt. Integendeel tot een normale kalender kan je hier toevoegen hoeveel keer je nodig hebt voor een toets voordat je het lesmateriaal onder de knie hebt. Voor lessen krijg je herinneringen dat je je werkcollege of lesmateriaal voorbereiding moet doen ook al is het maar voor 15 minuten, het is handiger dat je een herinnering krijgt en een paar minuten besteed dat helemaal niets doet.

Voor projecten en deadlines krijg je ook een countdown hoeveel dagen je nog overhebt want een visuele representatie van wanneer iets af moet is veel motiverend voor de klant dan een normale kalender bericht. Je kan ook tijden aangeven wanneer je meestal dingen voor school aan het doen bent bijvoorbeeld laat in de avond of juist tussen 5 en 8 voor werk. De app kan in die tijd aangeven wat er gedaan kan worden in die tijd en of er iets urgents afgemaakt moet worden

**GITHUB**

Maak je GitHub-repo **private**, geef je eigen docent toegang tot je repo (zie paragraaf 2 in [de uitleg bij deze opdracht op Brightspace](https://brightspace.hhs.nl/d2l/le/lessons/56739/topics/736360)) en lever je repo *clean* en Maven-*aware* op (zie paragraaf 3 in [de uitleg bij opdracht 2](https://dehaagsehogeschool.sharepoint.com/:b:/r/sites/AgileOOProgrammingSemesterSE-S2_groups-Docenten/Gedeelde%20documenten/Docenten/H-SE-S2OPT3/Lesmateriaal%20(publiek)/OPT3%2014.1%20-%20Realisatie%20zonder%20code%20smells%20-%20Toelichting.pdf?csf=1&web=1&e=MHo6uM)). Stuur niet steeds een nieuwe link/uitnodiging, maar bouw in feature-branches verder aan de applicatie die je eerder hebt opgeleverd:

https://github.com/SchoolAccount22072/Project2OPT3/tree/Test-Branch

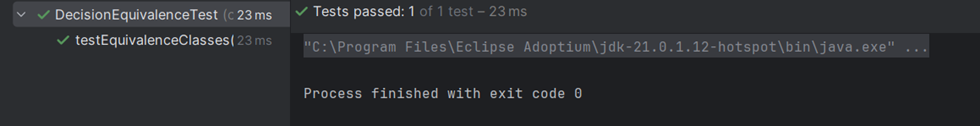
Bovenstaande repo is m.b.v. .gitignore ontdaan van IntelliJ-files en er is een Maven POM-file toegevoegd waarmee je applicatie in één keer gerund kan worden.

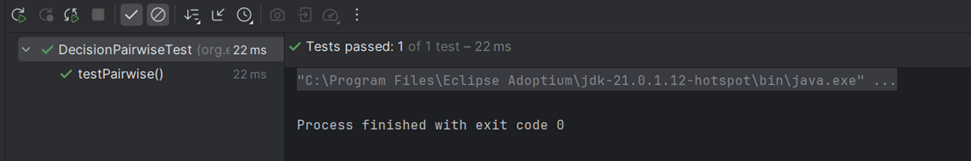
**GESLAAGDE TESTS**

Toon met een relevante screenshot aan dat alle onderstaande tests inderdaad zijn geslaagd:

A screenshot of a computer program

Description automatically generated





**SAMENGESTELDE DECISION**

In mijn software neem ik een besluit op basis van de volgende voorwaarden (maximaal 3 te testen):

Voorwaarde A: Een Gebruiker Is ingelogd

Voorwaarde B: De gebruiker is de docent

Voorwaarde C: De Gebruiker werkt tijdens schooluren

Het besluit is gebaseerd op de volgende combinatie van deze voorwaarden (bijv. D = A && (B || C)):

D = A && (B || C)

**CONDITION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Condition Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| True | True | True | True |
| True | True | False | True |
| True | False | True | True |
| True | False | False | False |
| False | True | True | False |
| False | True | False | False |
| False | False | True | False |
| False | True | False | True |
| <waarde> | <waarde> | <waarde> | <waarde> |

**DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| True | True | True | True |
| True | False | False | False |
| False | True | False | False |
| False | False | False | False |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |

**CONDITION/DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Condition/Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| True | True | True | True |
| True | True | False | True |
| True | False | True | True |
| True | False | False | False |
| False | True | True | False |
| False | True | False | False |
| False | False | True | False |
| False | False | False | False |
| <waarde> | <waarde> | <waarde> | <waarde> |

**MODIFIED CONDITION/DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Modified Condition/Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie). Geef met <l>/<f> aan met welke logische en fysieke test cases je werkt:

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| True | True | False | True |
| True | False | True | True |
| True | False | False | False |
| False | True | False | False |
| False | False | True | False |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |

**MULTIPLE CONDITION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Multiple Condition Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = <Decision**> |
| True | True | True | True |
| True | True | False | True |
| True | False | True | True |
| True | False | False | False |
| False | True | True | False |
| False | True | False | False |
| False | False | True | False |
| False | False | False | False |
| <waarde> | <waarde> | <waarde> | <waarde> |

**MODIFIED CONDITION/DECISION COVERAGE (JUNIT)**

Ik heb de testcases voor Modified Condition/Decision Coverage omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class DecisionTest {

private boolean decision(boolean A, boolean B, boolean C) {

return A && (B || C);

}

@Test

void testMC\_DC() {

assertTrue(decision(true, true, false));

assertTrue(decision(true, false, true));

assertFalse(decision(true, false, false));

assertFalse(decision(false, true, false));

assertFalse(decision(false, false, true));

}

}

Hiermee test ik de volgende methode (van het type boolean, waarin de waarde van D = <Decision> wordt bepaald):

public class Decision {

public boolean decide(boolean A, boolean B, boolean C) {

return A && (B || C);

}

}

**EQUIVALENTIEKLASSEN EN RANDWAARDEN**

Voor het nemen van een besluit in de software heb ik een domein dat bestaat uit de volgende equivalentieklassen (maximaal 3 te testen):

|  |  |  |
| --- | --- | --- |
|  | Logische weergave | Besluit |
| Equivalentieklasse 1 | De gebruiker is ingelogd, is docent en werkt tijdens schooluren. | True |
| Equivalentieklasse 2 | De gebruiker is ingelogd, is geen docent en werkt tijdens schooluren. | True |
| Equivalentieklasse 3 | De gebruiker is niet ingelogd | False |

Geef nu alle relevante fysieke testgevallen (incl. de meest intensieve test van randwaarden) en geef voor elk testgeval aan voor welke equivalentieklasse dit een test is (als je de laatste regels niet meer nodig hebt, kun je deze velden vullen met een spatie):

|  |  |  |
| --- | --- | --- |
| Equivalentieklasse | Fysiek testgeval | Besluit |
| 1 | A=true, B=true, C=true | Allowed |
| 2 | A=true, B=false, C=true | Allowed |
| 3 | A=false, B=false, C=false | Denied |
| 1 | A=true, B=true, C=false | Allowed |
| 2 | A=true, B=false, C=false | Denied |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |

**EQUIVALENTIEKLASEN EN RANDWAARDEN (JUNIT)**

Ik heb de testcases voor Equivalentieklassen en randwaarden omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class DecisionEquivalenceTest {

private boolean decision(boolean A, boolean B, boolean C) {

return A && (B || C);

}

@Test

void testEquivalenceClasses() {

assertTrue(decision(true, true, true));

assertTrue(decision(true, false, true));

assertFalse(decision(false, false, false));

assertTrue(decision(true, true, false));

assertFalse(decision(true, false, false));

}

}

Hiermee test ik de volgende methode (waarin de besluiten op basis van de equivalentieklassen worden bepaald):

public class Decision {

public boolean decide(boolean A, boolean B, boolean C) {

return A && (B || C);

}

}

**PAIRWISE TESTING**

Voor Pairwise Testing maak ik gebruik van de volgende parameters (op basis waarvan een besluit wordt genomen; vereenvoudig tot een set van 4 parameters die ieder 3, 2, 2 en 2 mogelijkheden hebben (anders wordt handmatig testen met Pairwise Testing veel te ingewikkeld; vul een spatie in, als je een optie niet wilt gebruiken):

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Optie 1** | **Optie 2** | **Optie 3** |
| Inloggen | Wel ingelogd | Niet ingelogd | <Optie 3> |
| Docent | Docent Ja of Nee | Docent Nee | <Optie 3> |
| Schooluren | Tijdens schooluren | Niet Tijdens schooluren | <Optie 3> |
| Deadlines | Deadline aanmaken | Deadline opvragen | <Optie 3> |

Na de tussenstappen in het proces heb ik de volgende testtabel met fysieke testgevallen opgesteld (vul spaties in, als er teveel testgevallen in de tabel staan voor jouw casus):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **<Parameter** 1> | **<Parameter** 2> | **<Parameter** 3> | **<Parameter** 4> | **Decision** |
| Wel ingelogd | Ja | Tijdens schooluren | opvragen | Allowed |
| Wel ingelogd | Ja | Niet Tijdens schooluren | Aanmaken | Allowed |
| Wel ingelogd | Nee | Tijdens schooluren | Aanmaken | Allowed |
| Wel ingelogd | Nee | Niet Tijdens schooluren | Opvragen | Denied |
| Niet ingelogd | Ja | Tijdens schooluren | Aanmaken | Denied |
| Niet ingelogd | Nee | Niet Tijdens schooluren | Opvragen | Denied |

**PAIRWISE TESTING (JUNIT)**

Ik heb de testcases voor Pairwise Testing omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class DecisionPairwiseTest {

private boolean decision(boolean A, boolean B, boolean C) {

return A && (B || C);

}

@Test

void testPairwise() {

assertTrue(decision(true, true, true));

assertTrue(decision(true, true, false));

assertTrue(decision(true, false, true));

assertFalse(decision(true, false, false));

assertFalse(decision(false, true, true));

assertFalse(decision(false, false, false));

}

}

Hiermee test ik de volgende methode (waarin het besluit op basis van de meegegeven parameters wordt bepaald):

public class Decision {

public boolean decide(boolean A, boolean B, boolean C) {

return A && (B || C);

}

}