

Persoonlijk Ontwikkelings Rapport.

Cursus Application Programming.

Semester 2.

Student: Jan van Hest.

Applicatie: LingoPartner.

Introductie.

Als student Application Programming ben ik, Jan van Hest, begonnen aan het ontwikkelen van de applicatie LingoPartner dit semester. Mijn doel is om mijn vaardigheden als programmeur te verdiepen en te verbeteren. LingoPartner, gericht op het versterken van taalvaardigheden door interactieve oefeningen en games, wordt ontworpen en geïmplementeerd met C# en UML. Dit project sluit aan bij de leerdoelen van de cursus, zoals functioneel en technisch ontwerp, en speelt een cruciale rol in mijn professionele ontwikkeling.

Inhoudsopgave

Introductie.....	2
PDR evaluaties.....	5
<i>Development report: Functioneel Ontwerp.....</i>	<i>6</i>
Toelichting:.....	6
Eerste Evaluatie.....	6
Tweede evaluatie.....	7
Derde evaluatie.....	9
<i>Development report: Technisch ontwerp.....</i>	<i>10</i>
Toelichting.....	10
Eerste Evaluatie.....	10
Tweede evaluatie.....	11
Derde evaluatie.....	13
<i>Development report: Implementatie.....</i>	<i>14</i>
Toelichting:.....	14
Eerste Evaluatie.....	14
Tweede evaluatie.....	15
Derde evaluatie.....	15
<i>Development report: Algoritmiek.....</i>	<i>17</i>
Toelichting:.....	17
Eerste Evaluatie.....	17
Tweede evaluatie(mei).....	17
Derde evaluatie (juni).....	18
<i>Development report: Kwaliteit.....</i>	<i>19</i>
Toelichting:.....	19
Eerste Evaluatie.....	19
Tweede evaluatie.....	19
<i>Development report: Professional Skills.....</i>	<i>21</i>
Toelichting:.....	21
Eerste Evaluatie.....	21
Tweede evaluatie.....	22
Derde Evaluatie.....	23
Conclusie Report.....	24
<i>TerugBlik.....</i>	<i>24</i>
<i>Functioneel Ontwerp.....</i>	<i>24</i>
<i>Technisch Ontwerp.....</i>	<i>24</i>
<i>Implementatie.....</i>	<i>24</i>
<i>Algoritmiek.....</i>	<i>24</i>
<i>Kwaliteit.....</i>	<i>25</i>
<i>Professionele Vaardigheden.....</i>	<i>25</i>
<i>Samenvattende Conclusie.....</i>	<i>25</i>
<i>Concrete Voorbeelden van Vooruitgang:.....</i>	<i>25</i>
Herstructurering van de Administration-klasse:.....	25
Gebruik van Dependency Inversion:.....	25
Uitbreiding van Acceptatietesten:.....	25
Implementatie van de LearningStreakService-klasse:.....	26
Gedetailleerde Documentatie:.....	26

Bijlage.	27
<i>Beoordelingen.....</i>	<i>27</i>
Beoordeling Functioneel Ontwerp.....	27
Beoordeling Implementatie en validatie.	28
Beoordeling Personal Development Report.	29
<i>Leeruitkomsten.</i>	<i>30</i>
Leeruitkomst: Functioneel ontwerp.....	30
Leeruitkomst: Technisch ontwerp:	30
Leeruitkomst: Implementatie:.....	31
Leeruitkomst: Algoritmiek:.....	31
Leeruitkomst: Kwaliteit:.....	31
Leeruitkomst: Professionele vaardigheden.	32
<i>Niveaus van Voortgang</i>	<i>33</i>
Toelichting: Ongedefinieerd	33
Toelichting: Oriënterend	33
Toelichting: Beginnend	33
Toelichting: Geoefend	33
Toelichting: Gevorderd.....	33
<i>Toelichting bij SOLID en GRASP principes.</i>	<i>33</i>
SOLID.....	34
GRASP.	34

PDR evaluaties.

Noot:

Elk deel van dit verslag is gekoppeld aan een specifieke leeruitkomst. De leeruitkomsten worden in detail beschreven in de bijlage en zijn afkomstig uit het lesmateriaal van Fontys. In de “Persoonlijk Development Reports” kan je ze terugvinden kort samengevat als toelichting.

In de leeruitkomsten voor implementatie worden de SOLID- en GRASP-principes genoemd. Omdat deze principes te complex zijn om in één regel uit te leggen, heb ik ze uitgebreid beschreven op een aparte pagina in de bijlage.

Daarnaast heb ik de voortgangsniveaus overgenomen uit het lesmateriaal en opgenomen in de bijlage. Om het verslag overzichtelijk te houden, zijn deze terminologieën niet in de hoofdttekst opgenomen, maar uitsluitend in de bijlages.

Development report: Functioneel Ontwerp.

Toelichting:

Ik documenteer gevalideerde gebruikerseisen en specificaties van een informatiesysteem in een functioneel ontwerp. Dit betekent dat ik vaststel wat gebruikers nodig hebben, waarbij ik prioriteit geef aan wat het meest waardevol is voor de belanghebbenden. Ik beschrijf het verwachte gedrag van het informatiesysteem, waarbij ik uitleg hoe gebruikers met het systeem omgaan, meestal via use-casebeschrijvingen en flowdiagrammen. Ik valideer deze specificaties met acceptatietests om ervoor te zorgen dat alles werkt zoals verwacht.

Een functioneel ontwerp is een compleet en duidelijk document dat het probleem of de kans die het systeem oplost, omschrijft. Het geeft inzicht in de functies die het systeem gaat bieden. In het ontwerp neem ik zaken op zoals de context en het probleem, usecasediagrammen, conceptuele modellen, flowdiagrammen, acceptatietestplannen en wireframes. Zo zorg ik ervoor dat het systeem goed is ontworpen en klaar is voor implementatie.

Eerste Evaluatie

Bij het functioneel ontwerp dien je de gevalideerde gebruikerseisen en specificaties van een informatie systeem te documenteren. De gebruikers eisen zijn geaccepteert door de docenten.

Ik heb een functioneel ontwerp gemaakt wat bestond uit een zelfgeschreven casus, een noun verb analysis, een conceptueel model, een use-case diagram.

Feedback:

Ik heb de volgende feedback mogen ontvangen:

Hi Jan, Een zeer gedegen en uitgebreid document. Het conceptueel model ziet er logisch uit. De vraag is of "Leerkracht" een concept is; deze levert weliswaar activiteiten, maar "heeft" de leerkracht deze ook? Immers zou ik aan een concept (als we dit vertalen naar een class diagram) willen vragen wat alle leeractiviteiten zijn. Dit vraag ik niet perse aan een Leerkracht. Aangezien Leerkracht een actor is, blijft de vraag: is het ook een concept.

Self Assessment:

De volgende aspecten heb ik nog niet volledig uitgewerkt, hierdoor heb ik mijzelf de score van beginnend gegeven. De volgende puntjes mogen op de i:

- Ik heb nog geen acceptatietest scenario's uitgeschreven.
- Ik heb nog geen flowdiagrammen geschreven. Deze zou ik prima kunnen uitwerken bij mijn algoritme.
- Ik heb niet expliciet een context probleemdefinitie geschreven. Ik zal moeten bekijken of mijn casus hieraan voldoet, anders zal ik deze moeten herschrijven.
- Ik had de feedback niet helemaal helder. Later heb ik ook de leerkracht en student als aparte klassen geschreven, welke eigenschappen overerven van een user klasse. Ik zal dit moeten aanpassen.
- In git kan ik wat duidelijker een tweedeling laten zien functioneel en technisch ontwerp. Nu is het 1 grote readme geworden.

Tweede evaluatie.

Ik heb de volgende feedback mogen ontvangen van mijn leerkracht uit mijn eerste Personal Development Report:

“In jouw vorige self-assessment geef je aan het "beginning" niveau aan te tonen. Ik vind het sterk dat je heel expliciet aangeeft wat jij nog denkt te moeten doen. En op basis daarvan kan ik wel meegaan in dat self-assessment. Ik ben echter ook van mening dat er in het FO nog iets meer ontbreekt. Het FO dat ik nu ken, is de versie van sprint 0. Hierin zijn use cases nog amper uitgewerkt (denk aan bijv. use case descriptions). Het zou mij, op basis van het FO uit sprint 0, niet volledig duidelijk zijn wat ik als software engineer nu dien te implementeren.”

Ik heb geprobeerd de feedback te implementeren. De volgende stappen heb ik gezet:

- Er is nu een enkele user klasse.
- Er zijn userstories en usecase descriptions toegevoegd.
- Ik heb de readme wat simpeler opgezet en vooral de diagrammen getoond. Zo is het minder een brei aan tekst geworden. Voor wie graag een brei aan tekst doorneemt zijn de verslagen apart als documentatie opgenomen.

Ik heb het functioneel ontwerp opnieuw ingeleverd en heb daar de volgende feedback op mogen ontvangen van de leerkracht.

“Dank voor een zeer uitgebreid document! Ik zie dat je requirements uitwerkt in zowel user stories als use case descriptions. Mogelijk wat

dubbel; echter zie ik toch dat jij daar een andere inhoud aan geeft, maar lijkt het ook dat user stories andere functionaliteiten zijn dan de use case descriptions (?) Bij de use case description wordt duidelijk wat normal en exceptional (of alternate) flow is. Hier zou je dan ook eenvoudig test cases op kunnen baseren. Nu zijn de test cases gebaseerd op de user stories, die naar mijn mening wat minder duidelijk beschrijven hoe ze werken. Voorbeelden: - De learning streak wordt weergegeven op het dashboard - Spelletjes moeten punten of scores toekennen voor feedback Hoe werken deze concreet? Hoe kan ik testen dat de juiste punten worden toegekend of dat de weergave van een streak al dan niet terecht is? Over het bepalen van die streak hebben we afgelopen maandag ook gesproken: het bepalen hiervan, klinkt als iets algoritmisch; hoe steekt dit algoritme dan in elkaar?"

Bij de beoordeling van de leerkracht is aangegeven dat ik voldoende heb gescoord op Professionaliteit, context en conceptuele weergave, doch is er twijfel bij de scores Functionaliteiten en Acceptatietest. (De beoordeling van het functioneel ontwerp is toegevoegd in de bijlage als bewijs.)

SelfAssesment:

Ik heb mijn functioneel ontwerp uitgebreid, met de benodigde onderdelen zoals voorheen aangegeven. Doch waren er enkele punten van feedback. Er zijn geen twijfels over de professionaliteit, Context of Conceptuele weergave van mijn project. Wel zal ik mij nog moeten richten op de functionaliteiten en de acceptatie test.

In mijn vorige zelf-assessment gaf ik aan dat ik op het “beginning” niveau zat. Na reflectie en op basis van de ontvangen feedback, beoordeel ik mezelf nu op het “proficient” niveau. Ik zie nog steeds dat er verbeterpunten zijn aan mijn FO, maar ik heb significante vooruitgang geboekt.

Ik realiseer me nu dat de versie van het FO die ik tot nu toe heb opgesteld, niet volledig is. De usecase descriptions waren wel aanwezig, maar niet uitgebreid genoeg. Voor de volgende sprint zorg ik ervoor dat deze elementen vollediger uitgewerkt worden. Ik heb geleerd dat duidelijke en gedetailleerde usecase descriptions essentieel zijn voor de implementatie door software engineers. Daarom zal ik hier extra aandacht aan besteden in mijn volgende FO.

De volgende stappen zal ik nog moeten zetten.

- Uitbreidere usecase descriptions.
- Onderscheid tussen userstories en usecase descriptions:

- Zorg ervoor dat de inhoud van de userstories en de usecase descriptions overeenkomt.
 - Vermeld eventueel waar de overeenkomst zit door in de userstories expliciet de usecase te benomen en vica versa.
- Baseer je acceptatietest op usecases.
- Bekijk waar je het algoritme gaat toelichten. Dat mag ook in het functioneel ontwerp.
 - Controleer of usecase en userstory nog overeenkomen met het gedeelte in het functioneel ontwerp.

Derde evaluatie.

Ik heb mij tijdens de laatste maand er naar gekeken of het functioneel ontwerp nog in lijn was met het technisch ontwerp. Daar waar nodig heb ik aanpassingen gemaakt. Naar mijn weten heb ik alle onderdelen toegevoegd. Op aanraden van mijn leerkracht heb ik de acceptatietesten in tabelvorm toegevoegd.

Selfassesment.

Ik heb naar mijn weten tijdens PDR 1 en 2 het grootste gedeelte al opgeleverd. De Testen heb ik nu toegevoegd. Ik kan dit onderdeel nu scoren op advanced proficient.

Development report: Technisch ontwerp.

Toelichting.

Ik vertaal een functioneel ontwerp naar correcte softwareontwerpen en relevante diagrammen die geïmplementeerd kunnen worden. Vervolgens documenteer ik deze ontwerpen samen met de ontwerpbeslissingen en aannames in een technisch ontwerp.

Wanneer ik correcte softwareontwerpen maak, gebruik ik moderne standaarden zoals UML of c4. Bij het maken van ontwerpbeslissingen houd ik rekening met de argumenten die de keuzes ondersteunen. Als er nog onduidelijkheden zijn, maak ik aannames en documenteer ik ze.

Een technisch ontwerp is een compleet en professioneel document dat duidelijk beschrijft hoe het informatiesysteem moet worden geïmplementeerd. Het kan verschillende onderdelen bevatten, zoals UML-klassendiagrammen, UML-packagediagrammen, softwarearchitectuur, persistentiedefinities en interfacespecificaties. Deze details helpen om het systeem op de juiste manier te bouwen.

Eerste Evaluatie.

Ik heb een technisch ontwerp gemaakt dat een UML-klassendiagram bevat. De eerste versie van dit diagram is gebaseerd op de concepten die zijn vastgelegd in het functioneel ontwerp. Het UML-klassendiagram geeft de relaties weer tussen de verschillende klassen en hun functies, waarbij de meest cruciale use cases in aanmerking worden genomen. Het toont hoe de klassen met elkaar in verband staan en welke functies zij vervullen binnen het systeem.

Feedback.

Ik heb de volgende feedback mogen ontvangen:

Hi Jan, Dank voor het document. Splits het document bij voorkeur op naar functioneel ontwerp en technisch ontwerp zodat de feedback bij het betreffende document kan worden geplaatst. De structuur ziet er logisch uit. Hierin mis ik mogelijk de relatie die een student heeft met modules - er zal immers voortgang worden gemaakt op basis van activiteiten die succesvol zijn afgerond?

Self Assessment: Beginning.

Behalve een klassediagram van de domein laag dien ik ook de volgende onderdelen van mijn software een beschrijving en een diagram te bevatten:

- UML Packagediagram

- Softwarearchitectuur
- Persistentiedefinities
- Interfacespecificaties

Tweede evaluatie.

Bij mijn eerste Reflectie op het technisch ontwerp in dit personal development report heb ik de volgende feedback op het technisch ontwerp mogen ontvangen van mijn docent. Mijn Technisch ontwerp is toen gescoord op beginning.

Ook bij het TO geef je duidelijk aan waar het nog aan ontbreekt: denk hier vooral ook nog aan de overwegingen die je maakt: hoe ondersteunt jouw ontwerp een onderhoudbaar systeem?

Ik heb het Technisch ontwerp ingeleverd ter beoordeling en heb de volgende feedback mogen ontvangen van mijn docent.

Als eerste valt mijn oog op het package diagram: mogelijk een prima plaatje maar zonder textuele toelichting, zegt het mij niet gek veel. Ik zie nu vooral dependencies van modules, maar bijvoorbeeld niet in iets meer detail wat de inhoud van de lagen is en bijvoorbeeld ook niet waarom er een shared library bestaat. Class diagram: - Tussen een user en een friendrequest, zie ik een relatie die twee richtingen uitgaat; dit hebben we bij de mission control casus besproken. Mogelijk een expliciete overweging om dit wél te doen, maar dit heeft nogal wat nadelen en ik ben benieuwd hoe je dit ondervangt in jouw implementatie - Waarom zou een User een SetRole methode hebben? Wordt deze niet middels de constructor eenmalig gezet en wijzigt deze daarna niet meer...? - Een methode als Update progress krijgt een string als parameter; maar zou voortgang niet dienen te worden bepaald op basis van leer activiteiten i.p.v de progressie middels een methode te kunnen zetten (encapsulation). En wat is er dan in LearningActivity dat bepaald of er voortgang is gemaakt? - Een User kan meerdere rewards hebben, maar hoe komt hij/zij aan die rewards? volgens mij los je dit probleem in Progress prima op. Database model Een reward heeft meerdere users - ik denk dat dit waar is, maar een user kan toch ook meerdere rewards hebben? Hoe los je deze situatie op in een database model? In termen van algoritmiek hebben we besproken dat het een optie is om meerdere manieren van het berekenen van een streak toe te kunnen

passen. Dit zie ik in deze versie (nog) niet helemaal terug. Ook in het class diagram, zie ik dat uiteindelijk graag terug.

Het technisch ontwerp wordt gescoord op Professionaliteit, Structuur, Algoritmiek, Architectuur, aannames en overwegingen. Bij Professionaliteit heb ik voldaan aan de eisen, echter bij Structuur, Algoritmiek, Architectuur is er twijfel. Het onderdeel Aannames en overwegingen ontbreekt.

Ik heb veel aandacht besteed aan de professionaliteit van mijn documentatie. Ik heb gewerkt met gestandaardiseerde methodes. Zo heb ik voor het maken van diagrammen gebruik gemaakt van plantuml. Diagrammen zien er daardoor altijd haarscherp uit en zijn makkelijk aan te passen. Verschillende iteraties van diagrammen zijn op deze wijze makkelijk uit te wisselen. De eerste evaluatie wees op onvolledige UML-diagrammen en een onduidelijk databaseontwerp. Ik heb dit opgelost door de UML-diagrammen uit te breiden en een ER-diagram toe te voegen. Daarnaast heb ik activity diagrams en waarheidstabellen toegevoegd om de processen en algoritmische complexiteit beter te visualiseren en uit te leggen. De modulaire opbouw van mijn applicatie heb ik willen aantonen door het toevoegen van mijn package diagram.

Self assessment.

De feedback van de leerkracht gaf terecht aan dat het in het technisch ontwerp niet ondersteund was met behulp van een beschrijving en dat het onderdeel “Aannames en overwegingen” ontbrak.

Ondanks dat ik dit onderdeel verder heb uitgebreid met diagrammen vind ik onvoldoende progressie en laat ik de beoordeling staan op beginning.

De volgende onderdelen zal ik moeten toevoegen, danwel uitbreiden.

1. Package Diagram:

- Toevoegen van een duidelijkere toelichting bij het package diagram.
- Verminderen van de dependencies tussen modules waar mogelijk, zoals het verminderen van afhankelijkheden van een shared library.

2. Class Diagram:

- Verduidelijken van de relaties, met name tussen User en Friendrequest.
- Toevoegen van een duidelijke structuur voor de implementatie van business logica, inclusief methodes en attributen.

3. Algoritmische Complexiteit:

- Meer gedetailleerde beschrijving van de constraints en methoden.
- Kijken waar het toevoegen van waarheidstabellen en activity diagrams toegevoegde waarde biedt en waar niet.

- o Waak ervoor dat de samenhang goed beschreven wordt.
- Beschrijven hoe de User wordt geüpdatet op basis van progress in de leeractiviteiten.

4. Overwegingen en Aannames:

- Gedetailleerde onderbouwing van elke ontwerpbeslissing, inclusief de verschillende alternatieven die overwogen zijn.
- Voor- en nadelen van elke beslissing duidelijk beschrijven om de rationale achter elke keuze te verduidelijken.
- Beschrijven hoe je een User met meerdere accounts of interacties beheert.
- Verbeteren van flexibiliteit en schaalbaarheid van het ontwerp.
- Vermelden van het gebruik van dependency injection en waarom dit niet eerder besproken is.
- Uitleggen hoe je het strategy pattern wilt toepassen in je ontwerp.

Derde evaluatie.

Later in de course kregen we dependency inversion uitgelegd. Ik ben dit toe gaan passen en dit heeft mij aardig wat tijd gekost. Later ben ik gebruik gaan maken van een service provider. Deze wijzingen hebben er voor gezorgd dat ik de hele code op zijn kop heb gegoooid. Ik besef dat de adminstration klasse alle verantwoordelijkheden naar zich toe trok gaf veel problemen. Dependency inversion gaf hiervoor oplossing. Ik heb het in PDR2 toegepast door de authenticatie uit de adminstration klasse te trekken. In PDR3 heb ik alles uit de adminstration klasse getrokken, deze opgeheven en voor elke entiteit eigen classes geschreven. Ik heb van deze wijziging gebruik gemaakt om dit uitgebreid te beschrijven in mijn technisch ontwerp. Ik heb beschreven wat S.O.L.I.D. principes zijn en heb expliciet aangegeven waar ik deze toegepast heb. Bij grote wijzingen heb ik deze een eigen kopje gegeven.

Bij mijn LearningStreak algoritme heb ik niet alleen aangegeven wat het aangrijppunt van mijn algoritme is, maar ook hoe het werkt, welke strategy patterns ik gebruik hebt en hoe ik dat dan heb toegepast. Op het einde heb ik nog een expliciet stukje geschreven waarom ik denk dat dit handiger is en welke principes hierbij worden toegepast.

Self Assement.

Ik denk dat ik hier enorme stappen heb gemaakt en alle onderdelen heb toegevoegd die belangrijk zijn voor dit onderdeel.

Development report: Implementatie.

Toelichting:

Bij de implementatie lever ik keer op keer informatiesystemen die robuust en onderhoudbaar zijn. Deze systemen zijn gebaseerd op objectgeoriënteerde (OO) principes en patronen, zoals SOLID, GRASP en design patterns, en volgen een technisch ontwerp. Met "opleveren" bedoel ik dat de software zo wordt gemaakt dat belanghebbenden er meteen gebruik van kunnen maken. "Robuust" betekent dat het systeem bestand is tegen verkeerde gebruikersacties en onverwachte fouten zonder te crashen. Een "onderhoudbaar" ontwerp is flexibel genoeg om toekomstige aanpassingen of nieuwe eisen te kunnen verwerken.

OO-principes en patronen vormen de basis van het ontwerp en helpen bij het maken van systemen die eenvoudig te onderhouden zijn.

Eerste Evaluatie.

ontwerp geïmplementeerd. De leerkracht had aangeven dat ik al een start had kunnen maken met de infrastructure layer en dat heb ik ook gedaan.

Feedback:

Ik heb de volgende feedback mogen ontvangen:

Hi Jan, In de bijgevoegde zip lever je enkel het domain op. Ik heb dus de code gedownload uit git. Ik zie dat je de applicatie hebt opgezet in lagen: ik mis hierin de infrastructuur / data access layer (?) In de classes in het domain, zie ik zaken terug die ingaan tegen object oriented programming zoals het relateren van objecten op basis van ID (Guid). Voorbeeld is een Progress class die een referentie heeft naar een UserID, LearningEntityId, RewardId. In een Student class doe je dit dan wel goed: List<Progress>, List<Reward>. Maar denk dan aan IReadOnlyList. Ook in een class als FriendRequest, maar denk dan aan encapsulation. De class "Acquaintance" snap ik niet helemaal: zou een student niet enkel een lijst van verzonden en ontvangen "FriendRequests" hoeven te hebben? We gaan het nog over overerving hebben maar daarop vooruitlopend: wat de toegevoegde waarde van inheritance in het geval van een Student is, begrijp ik niet. De voortgang die je nu in sprint 0 laat zien is voldoende.

Self Assesment:

Ik heb enkel het domain opgeleverd en nogal wat missers gemaakt zoals hierboven beschreven. Ik scoor dit onderdeel voorlopig op "Oriënting".

Tweede evaluatie.

Van mijn werk dat ik in de maand mei heb opgeleverd, heb ik de volgende feedback mogen ontvangen:

Dank voor de oplevering van de code. Bij deze mijn feedback:
Prima loose coupling van lagen toegepast
In de unit tests zie ik dat je de loose coupling ook gebruikt t.b.v. mocking: top!
Als ik een unit test als "AddUser_ShouldAddUser_WhenValidUser" bekijk, vraag ik af af wat er nu precies getest wordt; in de add methode lijkt direct de repository te worden aangeroepen; anders gezegd: doet deze test nu wat deze beschrijft te doen?
In de User class gebruik je een Initialize methode: op zich een oplossing voor een probleem van code duplication, maar het zou ook met constructor chaining kunnen. In een class als User, wordt Console.WriteLine gebruikt; dat werkt aangezien je een console app hebt, maar werkt dus niet indien het Domain gebruikt wordt door een andere GUI
Je hebt een strategy gemaakt om scores op verschillende wijzen te kunnen berekenen. Dit ziet er naar mijn mening prima uit. Ik vraag me wel af op basis waarvan nu wordt besloten in het systeem welke strategy dient te worden toegepast.
De voortgang die je in de implementatie laat zien, is prima! Het testen van het algoritme is in elk geval nog een prima toevoeging naar mijn mening.

Ik was blij met de ontvangen feedback, maar het betekende wel dat ik nog een hoop werk te doen had. Niet alle code is schoon. Ik had nog steeds `Console.WriteLine`-statements staan. Ook was de administration-klasse nog aanwezig en nog niet geheel verwijderd. Dit wil ik in PDR3 gaan doen.

Ik heb geëxperimenteerd door de functionaliteit voor authenticatie weg te zetten in een aparte klasse genaamd authentication service

Self assesment(mei):

Ik ben nog niet geheel tevreden. Nog niet alle onderdelen zijn af. Wel heb ik flinke stappen kunnen zetten en is de "godklasse" administration bijna eruit. Ik vind het nog onvoldoende om de kwalificatie 'Proficient' neer te zetten, maar ik heb hier wel een flinke vooruitgang laten zien. Ik geef hier de score 'Beginning/Proficient'.

Derde evaluatie.

Ik heb de administration-klasse er in zijn geheel uit kunnen slopen. Ik heb alle entiteiten hun eigen service gegeven en het UML-diagram hierop kunnen aanpassen. Hierdoor is de logica totaal anders en hoef ik geen DTO's meer te gebruiken. Ik heb deze wijzigingen uitgebreid

kunnen toelichten in mijn technisch ontwerp. Waar ik eerst echt moeite had om een test te schrijven, gaat dat nu een stuk makkelijker. Als ik een andere manier van scoren wil, hoef ik alleen een andere `ILearningScoringStrategy` te veranderen.

Selfassessment.

Ik ben erg blij met de behaalde resultaten. Ik heb het gevoel dat ik meer controle heb over het eindresultaat en dat ik veel duidelijker heb waarom ik de dingen doe zoals ik ze doe. Dat geeft een fijn en zelfverzekerd gevoel. Ik kan nog steeds een hoop leren, maar ik ben blij met het geleverde resultaat. Ik heb het idee dat ik nu meer de diepte in kan gaan. Niet alleen hoe iets werkt, maar ook waarom je iets gebruikt.

Ik geef mijzelf hier dan ook de score proficient/advanced.

Development report: Algoritmiek.

Toelichting:

Ik analyseer computationele uitdagingen en implementeer algoritmisch complexe problemen in software. Dit betekent dat ik complexe softwareoperaties opdeel in kleinere stappen en beredeneer hoe ik specifieke problemen automatisch kan oplossen.

Om algoritmische complexiteit toe te voegen, gebruik ik keuzestructuren, herhalingsstructuren, en andere voorwaarden om aan de eisen van belanghebbenden te voldoen. Ik zorg ervoor dat de software voldoet aan de randvoorwaarden en beperkingen die voor het project gelden. Hierdoor kan ik uitdagingen oplossen met goed doordachte en efficiënte algoritmen.

Eerste Evaluatie.

Ik heb de domain layer uitgewerkt en mijn feedback verwerkt. In al mijn enthousiasme ben ik aan het coderen geslagen. Tot mijn verbazing kwam ik na het bestuderen van mijn documentatie dat ik het algoritme niet expliciet beschreven heb. Feedback:

Hi Jan, Een zeer uitgebreid document: ik denk dat je met alle opties de leeruitkomsten kunt aantonen. Je geeft ook duidelijk aan hoe je invulling wilt geven aan het algoritme. Bij de derde optie benoem je expliciet AI: daarbij de opmerking dat we geen AI gaan gebruiken voor het algoritme (ook al zou het een prima toepassing zijn), maar dit zelf dienen te programmeren.

Self Assessment: Oriënterend.

Ik heb meerdere casussen uitgeschreven, waarbij ik gekozen heb voor de applicatie die ik nu heb toegelicht in dit document. Vol goede moed ben ik begonnen met programmeren en heb ik niet meer bij stil gestaan dat ik mijn algoritme beter had moeten toelichten, gezien de feedback. Als actiepunten.

- Vastleggen waar ik algoritmes kan maken.
- Algoritme uitwerken in flowchart.

Tweede evaluatie(mei).

Ik kwam er laat achter dat ik het algoritme niet expliciet beschreven had in de casus die ik geschreven had. Ik heb dit besproken met de leerkracht en we hebben gekeken naar een nieuw algoritme, namelijk het berekenen van leerreeksen. Ik ben hiermee aan de slag gegaan en heb het algoritme uitgewerkt. Ik heb hierop meerdere malen feedback gekregen. Ik bemerkte dat de feedback vooral aangaf dat mijn algoritme te simpel was. Ik kan hiermee instemmen, aangezien het gebaseerd was op een eenvoudig activiteiten diagram, wat een reeks van data's behandelde en hier enkele manipulaties op uitvoerde. Dat ik hierop kritiek (feedback) kreeg, is in mijn ogen dan ook geheel terecht. Wat als je het berekenen van

leerreksen en het geven van scores hierop wilt veranderen? Moet je dan de hele code aanpassen, en waar pas je dat dan aan? Hier had ik meer aandacht voor moeten hebben. Concrete ideeën hierover heb ik al, maar deze heb ik nog niet kunnen uitwerken. Dat komt hopelijk in PDR3.

In het onderdeel kwaliteit heb ik al toegelicht dat ik de administration-klasse verwijderd heb. Deze zorgde ervoor dat ik niet op een mooie manier het algoritme zou kunnen toepassen. Het algoritme zou dan verweven worden met deze klasse en dat wilde ik niet. Ik wil één enkele klasse hebben.

Self Assessment (mei): Beginning/Proficient.

Ik had dit onderdeel graag al op 'Advanced' of 'Proficient' willen scoren, echter ben ik niet tevreden over de structuur van de huidige uitwerking. Ik heb enkel één activitydiagram waarin mijn algoritme is uitgelegd. Maar wat nu als het uitgebreid dient te worden? Er zijn onvoldoende abstracties en te weinig mogelijkheden tot modularisatie. Wanneer ik zaken als deze heb toegepast, kan ik bij PDR3 hier voldoende niveau aantonen. Wel heb ik hier ontwikkeling laten zien waar ik tevreden over ben en heb ik concrete plannen hoe ik dit kan toepassen. Ik hoop dan ook dat ik bij PDR3 hier een 'Proficient' of 'Advanced' kan neerzetten.

Derde evaluatie (juni)

Na de feedback en zelfevaluaties van de afgelopen maanden, heb ik verdere stappen gezet om mijn algoritme te verbeteren. Ik heb uitgebreid gewerkt aan het creëren van meer abstracties en mogelijkheden voor modularisatie en testen. Dit heb ik gedaan door het algoritme op te splitsen in meerdere klassen, interfaces, strategieën en functies, waardoor het eenvoudiger is om delen van het algoritme aan te passen zonder de hele codebasis te moeten wijzigen.

Vooruitgang sinds de tweede evaluatie:

- **Abstracties en Modularisatie**
- **LearningStreakService Klasse**
- **Gedetailleerde Documentatie**

Zelfbeoordeling (juni): Proficient

Ik ben tevreden met de vooruitgang die ik heb geboekt sinds de laatste evaluatie. Door de verbeterde structuur en documentatie van mijn algoritme, kan ik nu met meer vertrouwen zeggen dat ik op een 'Proficient' niveau werk. Hoewel er altijd ruimte is voor verdere verbetering, voel ik dat mijn huidige implementatie robuust en flexibel genoeg is om aan de eisen van complexe projecten te voldoen.

Ik ben van plan om deze aanpak voort te zetten en verdere feedback te gebruiken om mijn vaardigheden verder te ontwikkelen. Ik hoop dat ook de geleerde technieken vaker toe te kunnen passen in mijn werk en nieuwe patronen mij toe te eisen en toe te passen.

Development report: Kwaliteit.

Toelichting:

Ik verbeter en toon de kwaliteit van mijn software continu aan, gebruikmakend van standaard technieken en hulpmiddelen.

Dit doe ik door op een iteratieve manier te werken, zonder dat bestaande functionaliteit wordt verstoord, en door veranderingen bij te houden. Om te verbeteren, gebruik ik standaardhulpmiddelen om de kwaliteit van mijn code te bewaken. Ik test de programmacode voor het gewenste gebruik en voor verwachte en onverwachte fouten. Deze tests voer ik uit in verschillende fasen van het project.

Voor het aantonen van kwaliteit maak ik gebruik van standaardtechnieken zoals versiebeheersystemen, acceptatietests en unit-tests. Hierdoor zorg ik ervoor dat de software betrouwbaar en van hoge kwaliteit blijft.

Eerste Evaluatie

Vanaf het begin heb ik geprobeerd mij te richten op kwaliteit. Ik heb alle stappen waar mogelijk en binnen mijn kunnen gedocumenteerd, aldoende ik later kan controleren of mijn applicatie aan de vereisten voldoet.

Ik heb effectief gebruik gemaakt van GIT en push mijn code naar gitlab en github.

Self Assessment eerste evaluatie: Beginnend.

Ik moet nog gebruik maken van acceptatie testen, derhalve heb ik dit onderdeel gescoord op beginnend. Volgende zaken wil ik mij op gaan richten:

- Bepalen van 4 belangrijkste usecases.
- Bepalen van de betrokken classes.
- Minimaal 5 a 10 acceptatietesten schrijven.

Tweede evaluatie.

Kwaliteit bij mij op het werk is voornamelijk een onderwerp van discussie. Derhalve had ik enige angst bij dit onderwerp. Het niet implementeren van unittesten en onduidelijkheden bij usertesten/gebruikerstesten levert een hoop problemen en stress op het werk.

Het functioneel ontwerp is een goede aanzet, maar het technisch ontwerp geeft meer details en je hebt het gevoel wat richting te hebben. Privé speelden er ook een aantal zaken mee, wat het lastig maakte om de juiste focus op het project te krijgen.

Ik heb een eerste aanzet gemaakt voor de documentatie en feedback gevraagd. De unittesten en acceptatietesten waren nog niet gedaan. De feedback hierop was aanzienlijk; ik zag even de bomen door het bos niet meer en heb ervoor gekozen om de feedback eerst

uit te werken. De cohesie met de use cases en user stories was niet geheel logisch, en ook het verband met de acceptatietesten was niet goed aangezet. In deze periode heb ik me dan ook gefocust op de functionaliteiten. Bij de user stories heb ik expliciet aangegeven welke use cases daarbij horen en vice versa.

Ik heb unittesten geschreven en kwam erachter dat alles gebundeld was in de administration-klasse. Dat was een soort god-klasse geworden waarin alles was gebundeld. Je had ook altijd deze klasse nodig om iets te testen. Dit was niet echt makkelijk en in mijn technisch ontwerp ben ik hier dieper op ingegaan waarom dit niet handig is.

Uiteindelijk heb ik ervoor gekozen om deze klasse eruit te slopen. Een god-klasse claimt alle verantwoordelijkheid naar zich toe en zat me enorm in de weg. Dit kost tijd en hierdoor heb ik in PDR2 nog steeds geen testen goed uit kunnen werken.

Self Assessment tweede evaluatie (mei): Beginning / Proficient.

Ik moet nog steeds gebruik maken van acceptatietesten. Derhalve kan ik mezelf hier nog niet voldoende op scoren. Wel ben ik tot het inzicht gekomen dat alle verantwoordelijkheid in een enkele klasse niet wenselijk is. Het was lastig om te testen; als er iets fout ging, was het lastig troubleshooten in een spaghetti aan code. Nu heb ik bijvoorbeeld een authenticatieservice kunnen schrijven, die ik overal kan injecteren waar ik hem nodig heb. De geauthenticeerde gebruiker is overal bij de hand, en kleine stukjes code kan ik testen of repareren als er bugs zijn.

Zoals benoemd heb ik nog geen acceptatietesten en uitgevoerde gebruikerstesten. Derhalve kan ik mezelf nog niet scoren op advanced. Zelf zie ik mezelf nu tussen beginning en proficient. Ik zie dat er iets niet goed was in mijn ontwerp en heb hier actie op ondernomen door de code te herschrijven. Ik heb hiervoor nieuwe concepten zoals dependency inversion moeten toepassen. Daarom scoor ik mezelf op beginning/proficient.

Development report: Professional Skills.

Toelichting:

Naast de leerdoelen van het semester moet ik ook mijn professionele vaardigheden aantonen. Hoewel er geen expliciete lestijd aan wordt besteed, kunnen de activiteiten tijdens de lessen als bewijs dienen. De professionele leerdoelen zijn als volgt:

Ik werk samen, communiceer op een constructieve manier, en onderbouw mijn keuzes op basis van ontvangen feedback, op een duidelijke en professionele wijze.

Bij samenwerking werk ik samen met docenten en/of medestudenten aan projecten en neem ik initiatief om processen te verbeteren. Constructieve communicatie betekent dat ik zinvolle gesprekken voer met belanghebbenden, zoals docenten en medestudenten, en dat ik producten en documentatie oplever. Bij het onderbouwen van keuzes gebruik ik betrouwbare en relevante bronnen. Ik controleer of deze bronnen betrouwbaar zijn en of ze passen bij het project.

Om helder en professioneel te zijn, zorg ik ervoor dat mijn documentatie compleet maar compact is, vrij van spelfouten, en geschikt voor de beoogde belanghebbenden. Zo toon ik aan dat ik professioneel kan werken en effectief kan communiceren.

Eerste Evaluatie

Ik probeer om de 1 a 2 weken een gesprek aan te gaan met de leerkracht voor feedback. Tevens spreek ik regelmatig met mijn klasgenoten om te informeren hoe zij staan met hun opdracht. Ik heb geprobeerd om alle informatie tot mij te nemen.

Conceptuele Onduidelijkheid Oplossen:

- Bepaal of “Leerkracht” een concept of een actor is en pas het conceptuele model hierop aan.
- Indien nodig, herstructureer het classdiagram om deze rol te verduidelijken.

Acceptatietests Schrijven:

- Ontwikkel acceptatietestscenario's die de verwachte functionaliteit van het systeem valideren.

Flowdiagrammen Toevoegen Bij Algoritme:

- Maak flowdiagrammen om gebruikersinteracties en systeemlogica te beschrijven.

Context en Probleemdefinitie Herzien:

- Voeg een duidelijke context en probleemdefinitie toe.
- Controleer of de bestaande casus hieraan voldoet; anders herschrijven.

Classdefinitie Aanpassen:

- Definieer de leerkracht en de student als een user klasse.

Git-structuur Verbeteren:

- Maak een duidelijke scheiding tussen functioneel en technisch ontwerp in de Gitrepository.
- Organiseer de README zodat het overzichtelijker is.

Self Assement: Orienting/Beginning.

Ik heb al aardige stappen gezet. Toch nog onvoldoende om alle leeruitkomsten aan te tonen. Derhalve heb ik mij gescoord op Orienting / beginning.

Tweede evaluatie.

Op mijn eerste PDR heb ik de volgende feedback mogen ontvangen.

Professionaliteit: je vraagt prima om feedback, bent actief in de les, gaat aan de slag met oefenmateriaal: top! De stap die je hier nog kunt/ dient te zetten, is de onderbouwing van keuzes. Zeker nu we de laatste weken over onderhoudbaarheid van code hebben gesproken, vraagt dit om toelichting in zowel TO als mondeling.

Deze heb feedback heb ik ter harte genomen. En ik heb getracht mijn keuze te onderbouwen.

Op mijn funtioneel ontwerp heb ik de volgende feedback mogen ontvangen.

Functioneel Ontwerp

Criteria	Beoordelingen		
Professionaliteit langere beschrijving weergeven	Ja 	Twijfel Het werk is wellicht 1 geheel en/of wekt een professionele indruk, maar het verhaal is moeilijk te doorgronden voor de lezer.	Nee Het werk is niet 1 geheel of loopt niet van kop tot staart.

En bij het technisch ontwerp een zelfde score.

Technisch Ontwerp

Criteria	Beoordelingen		
Professionaliteit langere beschrijving weergeven	Ja 	Twijfel Het werk is wellicht 1 geheel en/of wekt een professionele indruk, maar het verhaal is moeilijk te doorgronden voor de lezer.	Nee Het werk is niet 1 geheel of loopt niet van kop tot staart.

Ik heb concrete plannen om mijn keuze te onderbouwen en vast te leggen in de gewenste documentatie.

Self Assessment: Beginning / Proficient.

Ik voorzie hier voorlopig geen problemen. Ik kan mijn documentatie op orde krijgen. Ik kan mijn gedachten helder uiteenzetten. Tijdens PDR 1 had ik geen idee hoe ik de kwaliteit kon waarborgen. Nu ik de code aan het omgooien ben, heb ik hier meer vertrouwen in gekregen. Ik heb meer manieren waarop ik een probleem kan aanpakken. Het is een kwestie van uitvoeren, uitproberen en tweakken. Meer manieren om een probleem aan te kunnen pakken betekent dat ik voor verschillende situaties verschillende oplossingen kan kiezen. Ik beoordeel mij voorlopig op Beginning/Proficient.

Derde Evaluatie.

Ik had enige angst tijdens PDR 1 of ik wel de gewenste resultaten kon behalen. Tijdens de eerste 8 weken van de cursus werd vooral herhaald wat eerder besproken was bij een andere module. Ik had ook geen idee toen of ik enige verdieping kon aanbrengen in mijn project. Privé was ik enigszins afgeleid, het testen was erg lastig, en ik liep een beetje achter. Ik heb aangegeven dat ik tegen mijn grenzen aanliep. Hierdoor had ik tijdens PDR 1 de infrastructuur af en bij PDR 2 had ik nog niet echt een goede concrete invulling van mijn algoritme die naar mijn zin was. Ik heb besloten om regelmatig, danwel wekelijks feedback te vragen. Wel zag ik bij PDR 2 dat ik reverse dependency kon toepassen en ik zag in dat het soms beter is om voor bepaalde zaken een nieuwe klasse te maken middels GRASP-principes. Het testen liep hierdoor makkelijker en in PDR 3 heb ik bijna mijn volledige code om kunnen gooien naar een bruikbaar geheel. Het blijven communiceren hierover is cruciaal geweest. Zonder een gesprek had ik zeker bepaalde punten over het hoofd gezien. Nu ik op het einde ben gekomen, heb ik een beter begrip van de beste praktijken en technieken om mijn project effectief te voltooien. Ik voel me veel zekerder over mijn capaciteiten en de kwaliteit van mijn werk.

Selfassessment.

Waar ik eerst het gevoel had dat ik achterliep en niet echt helder had waarom bepaalde zaken op een bepaalde manier zouden moeten gaan, heb ik voor mijzelf een weg gevonden om in mijn project mijn keuzes te verantwoorden en te onderbouwen. Je ziet dit terug in de verantwoording van mijn technisch ontwerp en het regelmatig vragen om feedback. Ik scoor mij hier dan ook op proficient/advanced.

Conclusie Report.

TerugBlik.

Na een grondige analyse van alle evaluaties in dit document, kan ik de volgende conclusies trekken over mijn ontwikkeling tijdens dit semester:

Functioneel Ontwerp

Mijn functioneel ontwerp heeft significante verbeteringen ondergaan. Aanvankelijk waren er veel ontbrekende elementen zoals acceptatietestscenario's en flowdiagrammen, en was er onduidelijkheid over conceptuele modellen. Door gerichte feedback van mijn docenten heb ik deze punten aangepakt. Ik heb nu uitgebreide use-casebeschrijvingen, duidelijke context- en probleemdefinities, en een betere documentatiestructuur. Dit heeft mijn functioneel ontwerp versterkt en meer waardevol gemaakt voor de belanghebbenden.

Technisch Ontwerp

Het technisch ontwerp begon met een basis UML-klassendiagram en beperkte documentatie van ontwerpbeslissingen. Door voortdurende feedback en reflectie heb ik mijn technisch ontwerp aanzienlijk verbeterd. Ik heb gedetailleerde UML-diagrammen toegevoegd, waaronder packagediagrammen en ER-diagrammen, en uitgebreid aandacht besteed aan softwarearchitectuur en persistentiedefinities. Het integreren van S.O.L.I.D. en GRASP-principes heeft de onderhoudbaarheid en flexibiliteit van mijn ontwerp vergroot.

Implementatie

Tijdens de implementatiefase heb ik mijn vaardigheden in het opleveren van robuuste en onderhoudbare systemen aanzienlijk verbeterd. Aanvankelijk had ik te maken met een overmatig complexe administration-klasse die alle verantwoordelijkheden naar zich toe trok. Door het toepassen van dependency inversion en het uitsplitsen van functionaliteit in aparte services, heb ik de codebase flexibeler en beter testbaar gemaakt. Mijn ervaring met unittesten en het schrijven van duidelijke en onderhoudbare code is sterk gegroeid.

Algoritmie

Mijn aanpak van algoritmische uitdagingen heeft ook een aanzienlijke evolutie doorgemaakt. Waar ik aanvankelijk simplistische oplossingen toepaste, ben ik nu in staat om complexere en efficiëntere algoritmen te ontwerpen en implementeren. Door het gebruik van de LearningStreakService-klasse en het toepassen van verschillende strategiepatronen, heb ik de modulariteit en uitbreidbaarheid van mijn algoritmen verbeterd.

Kwaliteit

Het bewaken en verbeteren van de kwaliteit van mijn software is een continu proces geweest. In het begin had ik nog geen acceptatietesten en waren mijn unittesten beperkt. Naarmate het semester vorderde, heb ik meer nadruk gelegd op het schrijven van uitgebreide tests en het gebruik van tools om de codekwaliteit te bewaken. Dit heeft geleid tot een meer betrouwbare en robuuste codebase.

Professionele Vaardigheden

Mijn professionele vaardigheden zijn gedurende dit semester aanzienlijk verbeterd. Door regelmatig feedback te vragen, actief deel te nemen aan lessen en constructieve gesprekken te voeren met docenten en medestudenten, heb ik mijn communicatievaardigheden en vermogen om samen te werken versterkt. Ik heb geleerd om keuzes beter te onderbouwen en betrouwbare bronnen te gebruiken voor mijn beslissingen.

Samenvattende Conclusie

Over het geheel genomen heb ik aanzienlijke vooruitgang geboekt in alle leeruitkomsten van het semester. Door voortdurende reflectie, feedback en een actieve aanpak van verbeterpunten, ben ik in staat geweest om mijn vaardigheden als programmeur te verdiepen en te verbeteren. Hoewel er altijd ruimte is voor verdere groei, voel ik me nu veel zelfverzekerder in mijn capaciteiten en ben ik goed voorbereid op toekomstige uitdagingen. Mijn zelfbeoordeling is geëvolueerd van een 'beginnend' naar een 'proficiënt' niveau, met uitzicht op verdere groei naar 'advanced' in de nabije toekomst.

Concrete Voorbeelden van Vooruitgang:

Herstructurering van de Administration-klasse:

Door de administration-klasse te verwijderen en functionaliteiten uit te splitsen in aparte services zoals de authenticatieservice, heb ik de complexiteit verminderd en de testbaarheid van de code verbeterd. Dit heeft geresulteerd in een meer modulair en onderhoudbaar systeem.

Gebruik van Dependency Inversion:

Het toepassen van dependency inversion heeft ervoor gezorgd dat mijn code minder afhankelijk is van specifieke implementaties, wat de flexibiliteit en uitbreidbaarheid heeft vergroot. Hierdoor kon ik eenvoudiger nieuwe functionaliteiten toevoegen zonder bestaande code te breken.

Uitbreiding van Acceptatietesten:

Het schrijven en implementeren van gedetailleerde acceptatietesten heeft niet alleen de kwaliteit van mijn software verbeterd, maar ook mijn begrip van de vereisten en de

robuustheid van mijn applicatie vergroot. Deze tests hebben aangetoond dat de software correct functioneert onder verschillende omstandigheden en gebruikersscenario's.

Implementatie van de LearningStreakService-klasse:

Door de introductie van de LearningStreakService-klasse en het gebruik van strategiepatronen heb ik de modulariteit en flexibiliteit van mijn algoritmen aanzienlijk verbeterd. Dit heeft het mogelijk gemaakt om verschillende scoringsmethoden eenvoudig te implementeren en aan te passen, afhankelijk van de behoeften van het project.

Gedetailleerde Documentatie:

Door consistent gedetailleerde documentatie bij te houden, inclusief UML-diagrammen, flowcharts en ER-diagrammen, heb ik ervoor gezorgd dat mijn ontwerpen duidelijk en begrijpelijk zijn voor andere ontwikkelaars. Dit heeft de samenwerking vergemakkelijkt en de kwaliteit van het project verhoogd.






Door deze concrete stappen en verbeteringen heb ik een solide basis gelegd voor verdere ontwikkeling en succes in toekomstige projecten.

Bijlage.

Beoordelingen.





Beoordeling Functioneel Ontwerp.

Functioneel Ontwerp

Criteria	Beoordelingen		
Professionaliteit langere beschrijving weergeven	Ja 	Twijfel Het werk is wellicht 1 geheel en/of wekt een professionele indruk, maar het verhaal is moeilijk te doorgronden voor de lezer.	Nee Het werk is niet 1 geheel of loopt niet van kop tot staart.
Context langere beschrijving weergeven	Ja 	Twijfel De theorie lijkt goed begrepen, maar de uitwerking is incompleet.	Nee Is afwezig, of de theorie is niet goed begrepen.
Functionaliteiten langere beschrijving weergeven	Ja 	Twijfel De theorie lijkt goed begrepen maar de uitwerking is incorrect of onvolledig gezien de context. 	Nee Is afwezig of de theorie is niet goed begrepen.
Conceptuele weergave langere beschrijving weergeven	Ja 	Twijfel De theorie lijkt goed begrepen maar de uitwerking is incorrect of onvolledig gezien de context.	Nee Is afwezig of de theorie is niet goed begrepen.
Acceptatietest langere beschrijving weergeven	Ja 	Twijfel De theorie lijkt goed begrepen maar de uitwerking is incorrect of onvolledig gezien de context. 	Nee Is afwezig of de theorie is niet goed begrepen.

Beoordeling Implementatie en validatie.

Implementatie & Validatie

Criteria	Beoordelingen		
OO-Principes langere beschrijving weergeven	Ja	Twijfel De OO-principes worden wel toegepast, maar niet gepast of de code is moeilijk leesbaar/begrijpbaar/onderhoudbaar. 	Nee Er worden geen OO-principes toegepast of het is niet duidelijk waar ze toegepast worden. De code is moeilijk leesbaar/begrijpbaar/onderhoudbaar.
Modulariteit langere beschrijving weergeven	Ja 	Twijfel De applicatie is wel verdeeld in modules/componenten maar de verdeling is incorrect of niet losjes gekoppeld.	Nee De applicatie is niet in modules/componenten verdeeld of het is niet duidelijk hoe het gedaan is.
Unittesten langere beschrijving weergeven	Ja	Twijfel 	Nee
Acceptatietestrapport langere beschrijving weergeven	Ja	Twijfel Het plan is niet goed uitgevoerd en/of de conclusie van het rapport is onduidelijk	Nee 

Beoordeling Personal Development Report.

Personal Development Report

Criteria	Beoordelingen				
🌀 1 Functional design langere beschrijving weergeven	Advanced	Proficient	Beginning 	Orienting	Undefined
🌀 2 Technical design langere beschrijving weergeven	Advanced	Proficient	Beginning 	Orienting	Undefined
🌀 3 Implementation langere beschrijving weergeven	Advanced	Proficient	Beginning	Orienting 	Undefined
🌀 4 Algorithmics langere beschrijving weergeven	Advanced	Proficient	Beginning	Orienting 	Undefined
🌀 5 Quality langere beschrijving weergeven	Advanced	Proficient	Beginning	Orienting 	Undefined
🌀 6 Professionalism langere beschrijving weergeven	Advanced	Proficient	Beginning 	Orienting	Undefined

Leeruitkomsten.

Leeruitkomst: Functioneel ontwerp

*Je documenteert gevalideerde gebruikerseisen en specificaties van een informatiesysteem in een functioneel ontwerp. **Gevalideerde gebruikerseisen:***

De gebruikerseisen zijn geaccepteerd door de belanghebbenden en de hoogste prioriteit is gegeven aan de eisen die de meeste waarde opleveren voor deze belanghebbenden.

Specificaties:

Het verwachte gedrag van het informatiesysteem, gedefinieerd in termen van interactie tussen de gebruiker en het systeem en gespecificeerd in algoritmische processen zoals use-caseomschrijvingen en flowdiagrammen. Specificaties worden gevalideerd met behulp van acceptatietestscenario's.

Functioneel ontwerp:

Een professioneel, compleet en verzorgd document dat het probleemdomein of de kans van het te ontwikkelen informatiesysteem afbakt en inzicht geeft in de functionaliteiten die dit systeem gaat leveren. Als onderdelen van het functioneel ontwerp valt te denken aan:

- Context/Probleemdefinitie
- Use-casediagram
- Conceptueel model
- Flowdiagram(men)
- Acceptatietestplan • Wireframes, enz.

Leeruitkomst: Technisch ontwerp:

Je vertaalt een functioneel ontwerp naar correcte softwareontwerpen en relevante diagrammen die geïmplementeerd kunnen worden, en documenteert deze samen met de ontwerpbeslissingen en aannames in een technisch ontwerp.

Correcte softwareontwerpen:

Er wordt gebruik gemaakt van moderne ontwerpstandaarden in de softwareontwikkeling, zoals UML en/of c4. Ontwerpbeslissingen en aannames: De argumenten waarop beslissingen genomen worden in de ontwerpfase van het project en wanneer aspecten in deze fase nog niet eenduidig zijn, worden er gedocumenteerde aannames gemaakt.

Technisch ontwerp:

Een professioneel, compleet en verzorgd document dat op eenduidige wijze beschrijft hoe het informatiesysteem geïmplementeerd dient te worden. Als onderdelen van het technisch ontwerp valt te denken aan:

- UML Klassendiagram
- UML Packagediagram
- Softwarearchitectuur
- Persistentiedefinities
- Interfacespecificaties, enz.

Leeruitkomst: Implementatie:

Je implementeert en levert herhaaldelijk robuuste en onderhoudbare informatiesystemen op, welke gebaseerd zijn op OO-principes en patronen en het directe gevolg zijn van een technisch ontwerp.

Opleveren:

De software is op een dusdanige wijze beschikbaar gesteld dat de belanghebbenden er gebruik van kunnen maken.

Robuust:

Het informatiesysteem gaat correct om met verkeerde gebruikersacties en onverwachte fouten en crasht daardoor niet.

Onderhoudbaar:

Een ontwerp moet klaar zijn voor toekomstige nieuwe eisen of aanpassingen aan bestaande specificaties. OO-principes en patronen: Er is gebruik gemaakt van gangbare OO-principes en patronen om het softwaresysteem te ontwerpen en tevens zijn de ontwerpbeslissingen en aannames toegelicht en onderbouwd. Te denken valt aan:

- SOLID
- GRASP
- Design Patterns
- enz.

Leeruitkomst: Algoritmiek:

Je analyseert en redeneert over computationele uitdagingen en implementeert algoritmisch complexe problemen in software.

Computationele uitdagingen:

Er is decompositie toegepast op complexere softwareoperaties en de benodigde stappen zijn beredeneerd en onderbouwd, teneinde het geautomatiseerd oplossen van specifieke problemen.

Algoritmische complexiteit: De geïmplementeerde software heeft voldoende algoritmische complexiteit zoals keuzestructuren en herhalingsstructuren en bevat voorwaarden en beperkingen zodanig dat er rekening gehouden wordt met de randvoorwaarden van de belanghebbenden.

Leeruitkomst: Kwaliteit:

Je verbetert en toont de kwaliteit van je software continue aan, gebruikmakend van standaard technieken en hulpmiddelen.

Continue:

Er wordt op een iteratieve wijze gewerkt zonder bestaande functionaliteit te verstoren en waarbij veranderingen worden bijgehouden.

Verbeteren:

Er wordt gebruik gemaakt van standaard hulpmiddelen en technieken om de kwaliteit van de code te bewaken en te verbeteren.

Aantonen:

De programmacode moet getest worden voor zowel het gewenste gebruikt, als verwachte en onverwachte foutsituaties. Deze testen moeten in meerdere fasen van het project uitgevoerd kunnen worden.

Standaard technieken en hulpmiddelen: Er wordt bijvoorbeeld gebruik gemaakt een versiebeheersysteem, acceptatietests en unit-tests.

Leeruitkomst: Professionele vaardigheden.

Daarnaast dienen de volgende onderdelen van professionele vaardigheden in dit semester te worden aangetoond. Het zijn geen leerdoelen van het semester in die zin dat er geen expliciete onderwijstijd en aandacht aan wordt besteed gedurende de lessen.

Desalniettemin zullen de nodige activiteiten van het onderwijs als bewijslast kunnen dienen voor het aantonen ervan. De professionele leerdoelen zijn als volgt gedefinieerd:

Je werkt samen, communiceert op constructieve wijze en je baseert en onderbouwt je keuzes op ontvangen feedback op een heldere en professionele wijze.

Samenwerken:

Je betreft je docent(en) en/of werkt samen met medestudenten aan een project en neemt initiatieven om het proces te verbeteren.

Communiceren op constructieve wijze:

Je levert (deel)producten en documentatie op en initieert zinvolle gesprekken met de belanghebbenden (docenten/medestudenten).

Onderbouwen:

Je geeft betrouwbare en relevante bronnen voor alle beslissingen. Je beoordeelt bronnen op hun betrouwbaarheid en relevantie voor het project.

Helder en professioneel:

Je documentatie is compleet maar compact, niet-triviaal, gecontroleerd op spelfouten en toepasselijk voor de belanghebbende(n) waar deze voor bedoeld is.

Niveaus van Voortgang

Toelichting: Ongedefinieerd

Beschrijving: Je hebt nog geen activiteiten ondernomen om de leeruitkomst te demonstreren. Dit niveau geeft aan dat je nog moet beginnen met het verkennen of aanpakken van de specifieke leerdoelen.

Toelichting: Oriënterend

Beschrijving: Je hebt een start gemaakt en de mogelijkheden verkend om de leeruitkomst te demonstreren. Op dit niveau heb je basiskennis opgedaan en begin je te begrijpen wat nodig is om de leeruitkomst te bereiken, maar je hebt nog geen concrete stappen gezet die bijdragen aan een volledige realisatie.

Toelichting: Beginnend

Beschrijving: Je hebt de eerste stappen genomen en deze uitgevoerd, wat bijdraagt aan het demonstreren van de leeruitkomst. Dit niveau betekent dat je actief begonnen bent met het toepassen van je kennis en vaardigheden op praktische of theoretische manieren die rechtstreeks verband houden met het leerdoel.

Toelichting: Geoefend

Beschrijving: Je hebt meerdere keren aangetoond dat je een basis hebt gecreëerd om de leeruitkomst te demonstreren. Je zal de leeruitkomst op een voldoende niveau demonstreren, als je jouw ontwikkeling op deze manier voortzet. Dit niveau toont aan dat je consistent de vaardigheden of kennis toepast en begint te werken aan het verfijnen en verbeteren van je begrip en uitvoering.

Toelichting: Gevorderd

Beschrijving: Je hebt meerdere keren aangetoond dat je aan deze leeruitkomst hebt gewerkt met goede resultaten. Je hebt boven verwachtingen gepresteerd en bent gericht op continue verbetering. Je zal de leeruitkomst op een meer dan voldoende niveau demonstreren, als je jouw ontwikkeling in deze richting voortzet. Op dit niveau heb je niet alleen de verwachtingen overtroffen, maar werk je ook aan verdere optimalisatie en innovatie binnen het leerdoel.

Toelichting bij SOLID en GRASP principes.

SOLID en GRASP zijn leidende principes voor objectgeoriënteerd (OO) ontwerpen en ontwikkelen van software. Ze helpen bij het creëren van robuuste, onderhoudbare en schaalbare systemen. Samen omvatten ze 14 ontwerpprincipes, wat best veel kan lijken. Daarom is dit deel toegevoegd voor de leesbaarheid, om elk principe beter te kunnen begrijpen. Hieronder vind je een toelichting op de afzonderlijke principes en hun rol in het ontwerp van software.

SOLID.

Dit is een acroniem voor vijf principes van objectgeoriënteerd design. Ze bevorderen een flexibel en onderhoudbaar ontwerp:

1. **Single Responsibility Principle (SRP):** Een klasse moet slechts één verantwoordelijkheid hebben of één reden om te veranderen.
2. **Open/Closed Principle (OCP):** Software-entiteiten moeten open zijn voor uitbreiding, maar gesloten voor wijzigingen.
3. **Liskov Substitution Principle (LSP):** Subklassen moeten in staat zijn om hun basisklasse te vervangen zonder de correctheid van het programma te beïnvloeden.
4. **Interface Segregation Principle (ISP):** Het is beter om veel specifieke interfaces te hebben dan één grote, algemene interface.
5. **Dependency Inversion Principle (DIP):** Hoog niveau modules moeten niet afhankelijk zijn van laag niveau modules; beide moeten afhankelijk zijn van abstracties.

GRASP.

Dit staat voor "General Responsibility Assignment Software Patterns." GRASP biedt negen ontwerpprincipes om verantwoordelijkheden aan klassen toe te wijzen in een objectgeoriënteerd ontwerp:

1. **Information Expert:** Wijs verantwoordelijkheden toe aan de klasse die de meeste kennis heeft.
2. **Creator:** Een klasse moet verantwoordelijk zijn voor het maken van objecten die ze beheert of gebruikt.
3. **Controller:** Wijs de verantwoordelijkheid voor het beheren van systeemgebeurtenissen toe aan een controller-klasse.
4. **Low Coupling:** Minimaliseer afhankelijkheden tussen klassen.
5. **High Cohesion:** Zorg ervoor dat klassen zich richten op een duidelijk doel, zonder te veel uiteenlopende verantwoordelijkheden.
6. **Polymorphism:** Gebruik polymorfisme om afhankelijkheid van specifieke implementaties te verminderen.
7. **Pure Fabrication:** Wanneer een verantwoordelijkheid nergens logisch lijkt te passen, creëer dan een nieuwe klasse ervoor.
8. **Indirection:** Gebruik indirectie om afhankelijkheden te verminderen.

1. **Protected Variations:** Bescherm tegen wijzigingen door abstracties te gebruiken.

Zowel SOLID als GRASP helpen om softwareontwerpen te maken die gemakkelijk te onderhouden en uit te breiden zijn, terwijl ze tegelijkertijd de complexiteit verminderen.