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](#_Toc13814)

[PDR evaluaties 4](#_Toc13815)

[Development report: Functioneel Ontwerp 5](#_Toc13816)

[Toelichting: 5](#_Toc13817)

[Eerste Evaluatie 5](#_Toc13818)

[Development report: Technisch ontwerp 7](#_Toc13819)

[Toelichting 7](#_Toc13820)

[Eerste Evaluatie 7](#_Toc13821)

[Development report: Implementatie 9](#_Toc13822)

[Toelichting: 9](#_Toc13823)

[Eerste Evaluatie 9](#_Toc13824)

[Development report: Algoritmiek 10](#_Toc13825)

[Toelichting: 10](#_Toc13826)

[Eerste Evaluatie 10](#_Toc13827)

[Development report: Kwaliteit 11](#_Toc13828)

[Toelichting: 11](#_Toc13829)

[Eerste Evaluatie 11](#_Toc13830)

[Development report: Professional Skills 12](#_Toc13831)

[Toelichting: 12](#_Toc13832)

[Eerste Evaluatie 12](#_Toc13833)

[Bijlage 14](#_Toc13834)

[Leeruitkomsten 15](#_Toc13835)

[Leeruitkomst: Functioneel ontwerp 15](#_Toc13836)

[Leeruitkomst: Technisch ontwerp: 15](#_Toc13837)

[Leeruitkomst: Implementatie: 16](#_Toc13838)

[Leeruitkomst: Algoritmiek: 16](#_Toc13839)

[Leeruitkomst: Kwaliteit: 16](#_Toc13840)

[Leeruitkomst: Professionele vaardigheden 17](#_Toc13841)

[Niveaus van Voortgang 18](#_Toc13842)

[Toelichting: Ongedefinieerd 18](#_Toc13843)

[Toelichting: Oriënterend 18](#_Toc13844)

[Toelichting: Beginnend 18](#_Toc13845)

[Toelichting: Geoefend 18](#_Toc13846)

[Toelichting: Gevorderd 18](#_Toc13847)

[Toelichting bij SOLID en GRASP principes 19](#_Toc13848)

[SOLID 19](#_Toc13849)

[GRASP 19](#_Toc13850)

# 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 hoofdtekst 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*

*welliswaar 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 week 8: Beginning.*

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 klasses 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 Evaluatie

## 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*

*betreLende 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 week 8: 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

## 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.

Bij de eerste sprint heb ik de domain layer opgeleverd. Ik heb alle klasses uit het technisch ontwerp geimplementeerd. 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.*

## 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 e‘iciënte 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 wil 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 week 8: Orienterend.*

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 actiepunt.

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

## 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 e‘ectief gebruik gemaakt van GIT en push mijn code naar gitlab en github.

*Self Assessment week 8: Beginnend.*

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

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

## 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 e‘ectief 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.

Conclusie Report.

Eerste Evaluatie.

Ik heb een functioneel ontwerp gemaakt waarin ik de gebruikerseisen en specificaties van een informatiesysteem documenteer. Het ontwerp bevat een casus, een noun-verbanalyse, een conceptueel model en een use-case diagram. In de feedback die ik heb ontvangen, wordt aangegeven dat het conceptueel model logisch is, maar er is onduidelijkheid over de rol van de "Leerkracht". Het lijkt erop dat ik een duidelijkere definitie nodig heb van wat een concept is en hoe een actor daarin past.

Daarnaast merk ik dat ik nog geen acceptatietestscenario's heb geschreven. Dit is een belangrijk onderdeel van een functioneel ontwerp, dus ik moet eraan werken. Ook ontbreken er nog flowdiagrammen; ik zal deze moeten toevoegen om de interactie met het systeem te verduidelijken.

Een ander punt is de context en probleemdefinitie. Mijn casus moet duidelijker omschrijven welk probleem wordt opgelost. Als dit niet het geval is, zal ik een aparte sectie moeten maken die deze context en probleemdefinitie uitwerkt.

Wat betreft mijn self-assessment: ik heb mezelf als 'beginnend' beoordeeld omdat er nog verbeterpunten zijn. Zo moet ik de leerkracht en de student als aparte klassen definiëren, en deze eventueel aanpassen zodat ze eigenschappen overerven van een algemene userklasse. Ook moet ik in Git een duidelijkere scheiding maken tussen het functioneel en technisch ontwerp, omdat het nu één grote README is geworden.

Met deze feedback heb ik een duidelijk actieplan om mijn functioneel ontwerp te verbeteren. Ik zal de ontbrekende elementen toevoegen en de feedback verwerken om een compleet en duidelijk document te maken.

# Bijlage.

## 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 afbakent 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 betrekt 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 overtroLen, 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.