Help, een error!
Leerlingen ondersteunen bij het zelfstandig oplossen van programmeerfouten

Verslag van Onderzoek van Onderwijs (10 EC)
Voor het vak Informatica

Arthur Rump
s2010720

maandag 12 augustus 2024

Inhoudsopgave

Hoofdstuk 1

Inleiding

Leerlingen maken fouten. Dat is goed, want van fouten leer je. Maar bij het programmeren hebben fouten ineens een iets andere betekenis: waar een essay door een enkele spelfout niet ineens onleesbaar wordt, weigert een computer wel volledig om een programma met een kleine typefout uit te voeren. Daardoor kun je niet even uitproberen of je überhaupt in de juiste richting werkt, voordat je netjes alle puntjes op de i hebt gezet. Het is dus belangrijk dat leerlingen weten hoe ze snel een fout kunnen oplossen, zodat ze niet vast komen te zitten in onbenullige foutjes zelfs als ze eigenlijk wel snappen waar ze mee bezig zijn.

In programmeerlessen wordt echter zelden op een structurele manier aandacht besteed aan het oplossen van fouten (McCauley et al., 2008; Michaeli & Romeike, 2019). Uitleg over de betekenis van een foutmelding, of ondersteuning bij het vinden van een fout in een programma, geeft een docent vaak één op één, op het moment dat een leerling er behoefte aan heeft: het moment dat ze vastlopen op een fout. Dat is natuurlijk prima, maar er is niet in iedere situatie direct een docent aanwezig om die hulp te bieden. Daarbij helpt het niet mee dat het vinden van de oorzaak van een fout soms best lastig kan zijn. Bij veel fouten krijgt een leerling wel een foutmelding te zien, maar die zijn niet altijd begrijpbaar (Becker et al., 2019). Hierdoor besteden docenten in reguliere programmeerlessen veel tijd aan het individueel helpen van leerlingen (Michaeli & Romeike, 2019).

Bij Co-Teach Informatica wordt het vak via online materiaal aangeboden op scholen die geen vakdocent hebben. De inhoudelijke ondersteuning verloopt vanuit een regionaal vaksteunpunt, waar studentassistenten vragen van leerlingen via email beantwoorden en ingeleverde opdrachten beoordelen. In klas 4 havo/vwo gaan de leerlingen ook aan de slag met programmeren, maar er is dus geen vakdocent in het lokaal die leerlingen inhoudelijk kan ondersteunen bij het oplossen van de fouten die ze tegenkomen.

Het doel van dit onderzoek is om leerlingen die via Co-Teach het vak informatica volgen, de kennis en vaardigheden te geven die ze nodig hebben om meer fouten zelfstandig op te kunnen lossen. Daarvoor willen we uitzoeken op welke fouten de leerlingen vastlopen, welke kennis ze nodig hebben om die fouten op te kunnen lossen, en hoe we die in het materiaal kunnen aanbieden, zodat leerlingen sneller zelf de fouten die ze maken op kunnen lossen. De hoofdvraag luidt: *Hoe kunnen we leerlingen in klas 4 (havo/vwo) die voor het eerst programmeren voldoende toerusten om zelfstandig veelvoorkomende programmeerfouten op te lossen?* Op basis van het theoretisch kader in hoofdstuk 2 splitsen we die verder uit in drie deelvragen.

Hoofdstuk 2

Theoretisch kader

Bij het programmeren komt het oplossen van problemen zo vaak voor dat er een speciale vakterm voor is: *debuggen*, oftewel een programma ontdoen van fouten (bugs). Li et al. (2019) beschrijven vijf soorten kennis die nodig zijn om te kunnen debuggen:

- *Strategische kennis*: hoe kun je effectief debuggen, welke stappen moet je daarbij zetten? Hierbij zijn twee subcategorieën te onderscheiden:
  - Globale proceskennis, die in elke situatie van toepassing is. Dit zijn de stappen die bij elke soort fout doorlopen worden: de uitkomsten analyseren, de oorzaak van de fout vinden en een goede oplossing bedenken en implementeren. In de praktijk is dit een variant van de wetenschappelijke methode, met voor elke stap een of meerdere hypothesen om te toetsen (Perscheid et al., 2016; Spinellis, 2018).
  - Kennis van lokale strategieën, die in bepaalde gevallen gebruikt kunnen worden. Een voorbeeld is het toevoegen van print-statements aan het programma, om te kunnen volgen wat er tijdens het uitvoeren gebeurt.
- *Procedurele kennis*: hoe je de stappen van het proces en verschillende strategieën uit moet voeren. Hieronder valt bijvoorbeeld hoe je een debugger kunt gebruiken.
- *Domeinkennis*: kennis van de programmeertaal en hoe programma's uitgevoerd worden.
- *Systeemkennis*: kennis van het programma en de code. Dit gaat zowel over het overzicht over het gehele programma (wat gebeurt waar), als ook de functionaliteit (hoe iets gebeurt).
- *Ervaring*: op basis van ervaring kun je sneller inschatten waar een fout vandaan komt, of wat in een bepaalde situatie de beste strategie is.

Voor het krijgen van een overzicht in een programma, moet een leerling over alle onderdelen van het programma tegelijkertijd kunnen nadenken. Alle onderdelen moeten dus tegelijkertijd in het werkgeheugen van een leerling aanwezig zijn. We weten dat de capaciteit van het werkgeheugen bij mensen beperkt is, dus de enige manier waarop het mogelijk is om alle onderdelen van een programma erbij te houden, is door kleinere onderdelen samen te zien als één deelprogramma en het geheel van het programma te overzien als een compositie van deelprogramma's. Als je overzicht in een programma met een loop wilt hebben, kun je die loop bijvoorbeeld onthouden als "een loop over de getallen 1 t/m 10" in plaats van alle afzonderlijke onderdelen van de loop: de initialisatie van de teller, de controle van de huidige waarde van de teller, het verhogen van de teller etc. Dit principe wordt *chunking* genoemd (Hermans, 2021; Sorva, 2012), en is afhankelijk van de kennis (in dit geval over programmeren) in het langetermijngeheugen. Daardoor is dus de mate waarin het voor leerlingen mogelijk is om systeemkennis te vergaren afhankelijk van de domeinkennis.

Murphy et al. (2008) deden een studie naar het debuggedrag van beginnende programmeurs, en doen een aantal aanbevelingen voor het programmeeronderwijs. Allereerst zien ze het belang van

ervaring (zie ook het laatste punt van Li et al. (2019)), en bevelen daarom aan om deze ervaring als het ware met leerlingen te delen in de vorm van heuristieken of regels voor het toepassen van een bepaalde strategie in een situatie. Zo kunnen we leerlingen bijvoorbeeld leren dat "als je een bug niet kunt vinden op basis van de input en output, dan moet je print-statements toevoegen." Ze doen geen aanbevelingen voor concrete regels, maar geven slechts enkele voorbeelden.

Murphy et al. (2008) merken ook op dat de moeilijkheden die leerlingen ervaren bij debuggen lijken voort te komen uit een gebrek aan *meta-cognitieve vaardigheden*: leerlingen hebben niet door wanneer ze vastzitten en iets anders moeten proberen, of dat ze teveel uit het hoofd proberen te doen en beter een paar aantekeningen kunnen maken. Ze bevelen dan ook aan om in lesmateriaal aandacht te besteden aan de meta-cognitieve factoren, bijvoorbeeld door het stellen van "zelf-vragen": "Wat kan ik nog meer proberen?", "Is dit te veel om uit het hoofd te doen?", en "Wat zijn andere mogelijke oorzaken van deze bug?"

Volgens Michaeli & Romeike (2019) zijn de foutmeldingen die een programmeur van de compiler krijgt voor beginners een groot obstakel. Dit speelt vooral bij leerlingen die nog helemaal aan het begin van hun leerproces staan, op universitair niveau speelt dit probleem minder (Yen et al., 2012). Michaeli & Romeike (2019) beschrijven ook dat het debugproces afhangt van het type fout en of er een foutmelding getoond wordt. In de praktijk wordt dit vaak door de docent één op één uitgelegd op het moment dat een leerling hier tegenaan loopt. Ook zij noemen ervaring en heuristieken als een belangrijke vorm van kennis en zien dat sommige docenten dit proberen mee te nemen in hun lessen, door uit te leggen wat een oorzaak kan zijn van een bepaalde foutmelding of onverwacht gedrag in een programma. Foutmeldingen zijn een belangrijke bron van informatie om zo'n heuristiek op toe te passen, en kunnen dus ook als een startpunt dienen om leerlingen met verschillende heuristieken bekend te maken.

Op het gebied van foutmeldingen is het algehele beeld dat ze niet zeer behulpzaam worden gevonden (Becker et al., 2019). Diezelfde studie noteert ook dat leerlingen de foutmeldingen wel lezen, maar dat ze vervolgens niet goed weten wat het betekent of wat ze ermee moeten doen. We weten ook dat bij professionele programmeurs een foutmelding een van de meest voorkomende aanleidingen voor een zoekopdracht op internet is (Xia et al., 2017). De informatie op internet is echter, net als de foutmeldingen zelf, vaak alleen in het Engels beschikbaar. Becker et al. (2019) geven een overzicht van verschillende manieren waarop in de literatuur is geprobeerd om foutmeldingen te verbeteren. Qua techniek en presentatie zijn er verschillende mogelijkheden, maar ze kijken vooral naar tekstuele aanvullingen op of vervangingen van de foutmeldingen die in de editor worden weergegeven. Daar zien ze 9 aspecten waar de verschillende interventies in de literatuur in proberen te verbeteren:

- *Leesbaarheid*: in veel interventies worden de foutmeldingen herschreven in simpelere taal en met minder jargon. De taal van de foutmelding zou hierbij ook mee kunnen spelen.
- *Cognitive load*: door relevante informatie in de buurt van de code te zetten, en door de informatie op één plek te geven, is de druk op het werkgeheugen lager en is het makkelijker alle informatie te verwerken.
- *Context*: in de foutmelding moet duidelijk worden op welke plek in de code de fout optreedt, welke variabelen of functies erbij betrokken zijn etc. Enkele interventies proberen met diagrammen van de code aan te geven waar de fout optreedt.
- *Toon*: de foutmelding moet een positieve toon hebben en niet de schuld aan de programmeur geven.
- *Gebruik van voorbeelden*: een aantal studies gebruiken succesvol voorbeelden van correcte code en code met een fout om de foutmelding te verduidelijken, maar bij een andere studie raakten de programmeurs juist verward omdat ze dachten dat het voorbeeld hun code was.
- *Geven van suggesties en hints*: een suggestie kan veel helpen, maar het is wel van belang dat die suggestie dan ook echt correct is. Er moet een duidelijk onderscheid zijn tussen

voorbeelden en suggesties.
- *Interactiviteit*: in sommige ontwerpen krijgt de programmeur de mogelijkheid om meer informatie of suggesties op te vragen bij een foutmelding. (Dat helpt dan ook bij het verlagen van de eerste cognitive load.) Een andere mogelijkheid is om de gegeven informatie af te laten hangen van het niveau van de programmeur, en leerlingen zo stapsgewijs aan de echte foutmeldingen laten wennen.
- *Scaffolding*: foutmeldingen moeten als ondersteuning dienen bij het leren programmeren. Dat kan door verbindingen te leggen: tussen de foutmelding en waarom die getoond wordt, tussen de foutmelding en de code, tussen terminologie en de concrete stukjes code en tussen de foutmelding en mogelijke oplossingen.
- *Gebruik van logische argumentatie*: in enkele recente studies is geprobeerd om de foutmeldingen van logische argumentatie te voorzien. Het idee is dat door elke stelling te voorzien van argumenten, de programmeur beter begrijpt waarom een foutmelding optreedt.

Wat er aan foutmeldingen verbeterd kan worden en waar bij materiaal met betrekking op debuggen op gelet moet worden, komt dus uit de literatuur wel naar voren. Welke fouten en foutmeldingen leerlingen maken en problemen mee hebben is echter minder duidelijk, vooral omdat dat sterk afhangt van de situatie. Veel onderzoek is gedaan in eerstejaars vakken op de universiteit (bijv. Hartz, 2012; Smith & Rixner, 2019), waar natuurlijk veel meer diepgang mogelijk is dan op de middelbare school. Veel van de moeilijkere onderwerpen komen in het curriculum van de middelbare school simpelweg niet aan bod. Daarnaast is de literatuur vaak in meer algemene termen gegoten, om zo breed mogelijk van toepassing te zijn. Voor een ontwerp dat leerlingen ondersteunt bij het oplossen van specifieke fouten, willen we echter in meer detail weten tegen welke fouten de leerlingen daadwerkelijk aan lopen, zodat we direct aan kunnen sluiten op de problemen waar leerlingen in de praktijk tegenaan lopen.

In de rest van dit onderzoek zal de focus daardoor vooral liggen op de vraag waar leerlingen in onze specifieke situatie tegenaan lopen, en welke soorten kennis (uit Li et al., 2019) de leerlingen waarschijnlijk missen waardoor ze bij die fouten vastlopen. Met die kennis kunnen we de leerlingen ondersteunen bij het zelfstandig oplossen van deze fouten. Zo komen we tot drie deelvragen:

1. *Bij welke soorten fouten lopen leerlingen vast of vragen ze hulp?*

   Met de soort fout bedoelen we een groepering van fouten die gemaakt worden om dezelfde reden, bijvoorbeeld verschillende fouten in de syntax die ontstaan doordat leerlingen de syntax niet goed genoeg kennen, of de verschillende fouten die kunnen optreden door een off-by-one error. De categorieën van fouten worden bepaald aan de hand van de verkregen data.

2. *Welke kennis hebben leerlingen nodig om die lastige fouten (zoals in (1) gevonden) op te kunnen lossen?*

   Om leerlingen te ondersteunen bij het oplossen van fouten, moeten we weten welke kennis leerlingen nodig hebben om een fout op te lossen. Hierbij gebruiken we de soorten kennis zoals beschreven door Li et al. (2019) als leidraad. Deze soorten hoeven niet per se overeen te komen met de groeperingen uit de eerste deelvraag: als een fout gemaakt wordt door een gebrek aan kennis, betekent dat niet per se dat die kennis volstaat om de fout in het programma te vinden, of zelfs nodig is om de fout op te kunnen lossen. Als een leerling bijvoorbeeld het verschil tussen `<` en `<=` is vergeten en de verkeerde heeft gekozen, kan die door het toepassen van debugstrategieën erachter komen dat de fout in die vergelijking zit, zonder op dat moment te weten welke operator het wel zou moeten zijn. Deze fout zou dan zelfs op te lossen zijn zonder die kennis expliciet te vergaren, simpelweg door de andere optie te proberen.

3. *Wat kunnen we in het lesmateriaal verbeteren om leerlingen meer fouten zelfstandig op te laten lossen?*

   Om de leerlingen te ondersteunen bij het zelfstandig oplossen van deze fouten, gebruiken we de antwoorden op de eerste twee deelvragen om het lesmateriaal te verbeteren. We kijken in dit onderzoek naar een verbetering van of toevoeging aan het lesmateriaal, omdat dit de primaire manier is waarop leerlingen met Co-Teach in aanraking komen. Een meer technische aanpak (bijvoorbeeld verbetering van de foutmeldingen die leerlingen te zien krijgen) is binnen het project niet haalbaar.

Hoofdstuk 3

Methode en instrumenten

In dit onderzoek volgen we de innovatiecyclus zoals beschreven door Van der Donk & Van Lanen (2020). De dataverzameling en analyse wordt in die cyclus gevolgd door een ontwerpfase, waarna er een nieuwe reeks dataverzameling en analyse plaatsvindt voor de evaluatie van dat ontwerp. De resultaten daaruit kunnen dienen voor een nieuw ontwerp en bijbehorende evaluatie, maar in dit onderzoek houden we het bij één ontwerpfase en dus twee fases van dataverzameling en analyse: het vooronderzoek en de evaluatie van het ontwerp.

In dit hoofdstuk beschrijven we de algemene procedure over deze drie fases aan de hand van de deelvragen in sectie 3.1. In secties 3.2 tot 3.5 geven we respectievelijk meer toelichting op de respondenten, instrumenten, procedure van dataverzameling en analyse. De rest van dit verslag volgt de drie fasen: in hoofdstuk 4 beschrijven we de resultaten uit het vooronderzoek, in hoofdstuk 5 beschrijven we de ontwerpkeuzes die daaruit voortkomen en introduceren we het ontwerp en in hoofdstuk 6 bespreken we de evaluatie van het ontwerp.

3.1 Procedure

De drie deelvragen in dit onderzoek horen bij verschillende fases: de eerste twee vragen worden beantwoord in het vooronderzoek en de derde vraag is de ontwerpvraag, die we beantwoorden in de ontwerpfase en de evaluatie. Hier beschrijven we per deelvraag kort de methode die we hebben gehanteerd om die vraag te beantwoorden.

De uitvoering van het onderzoek vond plaats in 2022 en 2024, waarbij het grootste deel van het voorzonderzoek in 2022 is afgerond, gevolgd door een aanvulling daarop en de ontwerpfase in 2024.

3.1.1 Bij welke soorten fouten lopen leerlingen vast of vragen ze hulp?

De meest logische methode om deze vraag te beantwoorden is direct bij leerlingen kijken waar ze tegenaan lopen. Om een grote groep leerlingen te kunnen volgen, hebben we ervoor gekozen om leerlingen een logboek te laten bijhouden met de fouten waar ze gedurende een les op vastlopen. Het was de bedoeling om het logboek te laten invullen door enkele tientallen leerlingen die informatica volgen via Co-Teach, maar dat is uiteindelijk niet gelukt door de planning in 2022. Wel hebben we het logboek nog laten invullen door twee klassen die informatica volgen bij een vakdocent, waarbij 9 leerlingen hebben deelgenomen.

Daarnaast hebben we interviews gehouden met vakdocenten en studentassistenten van de Co-Teach vaksteunpunten. We hebben in 2022 de vakdocenten gevraagd naar hun ervaringen met de fouten die zij leerlingen zien maken tijdens hun lessen en naar de vragen die zij van leerlingen

krijgen. De vaksteunpunten verzorgen de inhoudelijke begeleiding en beoordeling bij Co-Teach, dus we hebben hen gevraagd naar de vragen die leerlingen stellen en de fouten die ze in de ingeleverde opdrachten terugzien.

Bij de eerste ronde interviews in 2022 was het Co-Teach project nog in een opstartfase en konden sommige vaksteunpunten nog weinig input leveren. Daarom hebben we twee jaar later in 2024 opnieuw interviews gehouden met een aantal assistenten om te zien hoe de situatie in die twee jaar veranderd was.

3.1.2 Welke kennis hebben leerlingen nodig om die lastige fouten (zoals in (1) gevonden) op te kunnen lossen?

Om uit te vinden wat leerlingen nodig hebben om een fout op te lossen, vroegen we de vakdocenten en studentassistenten bij de interviews ook hoe zij leerlingen helpen om een fout op te lossen. Met name vanuit de vakdocenten krijgen we zo een overzicht van wat zij hun leerlingen aanreiken zodat die een fout op kunnen lossen. Daarnaast vroegen we ook waaraan het zou kunnen liggen dat leerlingen een fout niet zelf op kunnen lossen, wat kan duiden op bepaalde kennis die bij leerlingen ontbreekt.

Bij deze vraag zijn de beperkingen van interviews het grootst: we nemen aan dat vakdocenten en vaksteunpuntassistenten leerlingen op een zinvolle manier helpen, maar dat is natuurlijk geen garantie. Om de resultaten enigszins te valideren, vergelijken we de uitkomsten met de literatuur uit hoofdstuk 2 en aanvullend literatuuronderzoek.

3.1.3 Wat kunnen we in het lesmateriaal verbeteren om leerlingen meer fouten zelfstandig op te laten lossen?

De antwoorden op de vorige twee vragen hebben we gebruikt als startpunt voor een aanvullend literatuuronderzoek naar mogelijke interventies in het lesmateriaal. Daaruit hebben we een aantal ontwerprichtlijnen gehaald en een ontwerp voor lesmateriaal gemaakt. In verband met de beperkte tijd, hebben we slechts een prototype voor een enkele paragraaf van het lesmateriaal uitgewerkt en die geëvalueerd in een expertreview met vakdocenten. Deze laatste vraag wordt in dit onderzoek dus niet sluitend beantwoord; we geven slechts een mogelijke antwoordrichting die in vervolgonderzoek verder getoetst kan worden.

3.2 Respondenten

Om de vragen te beantwoorden, maakten we gebruik van drie groepen respondenten: leerlingen die informatica volgen, vakdocenten informatica en studentassistenten bij de vaksteunpunten van Co-Teach.

3.2.1 Vakdocenten informatica

Dit zijn docenten die informatica geven op een middelbare school. Ze zijn dus niet per se verbonden aan Co-Teach, maar hebben wel ervaring met lesgeven in programmeren aan middelbare scholieren. We selecteerden deze docenten op basis van ervaring, inzicht in de situatie bij Co-Teach en praktijkkennis van programmeren. In het vooronderzoek in 2022 hebben we drie docenten geïnterviewd over de fouten die zij hun leerlingen zien maken. Deze docenten waren allemaal bekend met Co-Teach en waren qua ervaring zeer divers: aan het ene uiterste een beginnend docent die net een jaar les gaf en aan de andere kant een docent die op het punt stond met pensioen te gaan. De derde zat daartussenin. Alle drie hadden ervaring met programmeerlessen in Python.

Voor de evaluatie in 2024 hebben we twee andere vakdocenten als expert geïnterviewd: de een met veel eigen praktijkervaring als programmeur en een jaar leservaring waarin die vier keer een introductiemodule programmeren met Python heeft verzorgd; de ander een vakdocent met veel leservaring in informatica en programmeren, die zich het laatste jaar meer met de keuzemodules (waaronder programmeren) heeft beziggehouden. Beide hebben ook ervaring met leerlingen op afstand begeleiden. De docenten geven beide les op dezelfde school en hun ervaring is dus grotendeels met hetzelfde lesmateriaal.

3.2.2 Assistenten van de vaksteunpunten

Dit zijn universitair studenten die als studentassistent bij een van de drie vaksteunpunten werken, in Twente, Utrecht, Amsterdam (in de eerste ronde in 2022) of Delft (in de tweede ronde in 2024). Daardoor hebben zij een breed inzicht in het werk dat de leerlingen leveren: zij zijn het eerste aanspreekpunt voor vragen en beoordelen de ingeleverde opdrachten. We hebben in iedere ronde van elk van de drie regionale vaksteunpunten één assistent gesproken voor dit onderzoek. Bij de vaksteunpunten werken studentassistenten meestal in een kort dienstverband, dus bij de tweede ronde waren dit allemaal nieuwe assistenten. In totaal hebben we dus zes verschillende assistenten gesproken. De regionale spreiding is hier belangrijk om een zo breed mogelijk beeld te krijgen van hun ervaringen. De lokale universiteit is belangrijk voor de organisatie, dus er kunnen kleine regionale verschillen zijn in hoe dingen worden aangepakt en die invloed hebben op hoe leerlingen vragen stellen.

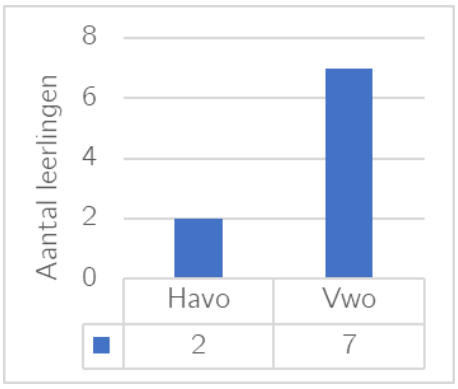
3.2.3 Leerlingen die informatica volgen

Dit zijn leerlingen uit klas 4, havo of vwo, die het vak informatica volgen op de middelbare school. We hebben getracht zowel leerlingen op vwo- als op havo-niveau bij het onderzoek te betrekken, en binnen die groepen zowel de zwakkere als de sterkere leerlingen.

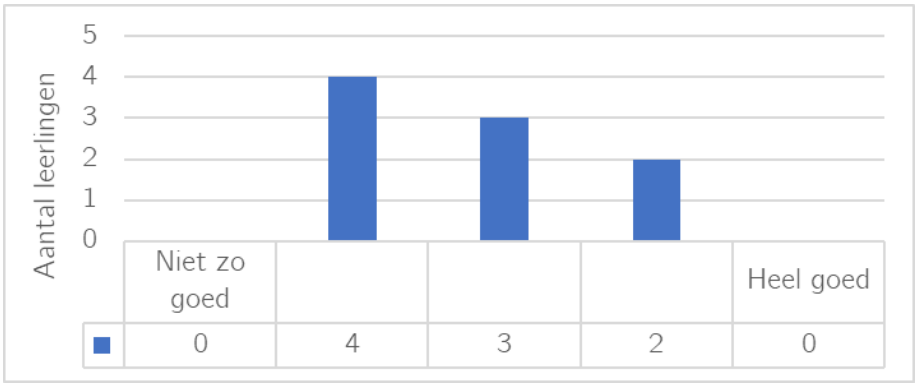
Het is helaas niet haalbaar gebleken om dit onderzoek uit te voeren met leerlingen die informatica volgen via Co-Teach, door uiteenlopende planningen. In plaats daarvan vroegen we leerlingen die informatica volgen in een normale situatie met vakdocent om deel te nemen aan het onderzoek. Zij volgden een vergelijkbaar curriculum, met als grootste verschil dat ze tijdens de les de mogelijkheid hebben om vragen te stellen aan een docent met vakkennis. In de les dat het logboek werd ingevuld, probeerden we dit verschil weg te nemen door de docent te vragen tijdens die les geen vragen te beantwoorden.

We vroegen de leerlingen naar hun schoolniveau (havo of vwo) en een zelfinschatting van hun vaardigheden en voorgaande ervaring in het programmeren. Idealiter was onze groep respondenten op deze drie eigenschappen representatief voor alle leerlingen die informatica volgen via Co-Teach. Wat betreft het schoolniveau hadden we daar invloed op door van beide niveaus evenveel klassen uit te nodigen. De beide andere eigenschappen zijn naar alle waarschijnlijkheid binnen de gehele populatie normaal verdeeld, dus verwachten we dat ook bij de respondenten. Het belangrijkste is echter dat zowel de leerlingen die het programmeren makkelijk afgaat als de leerlingen die er moeite mee hebben vertegenwoordigd waren, ook al was de verdeling misschien een beetje anders.

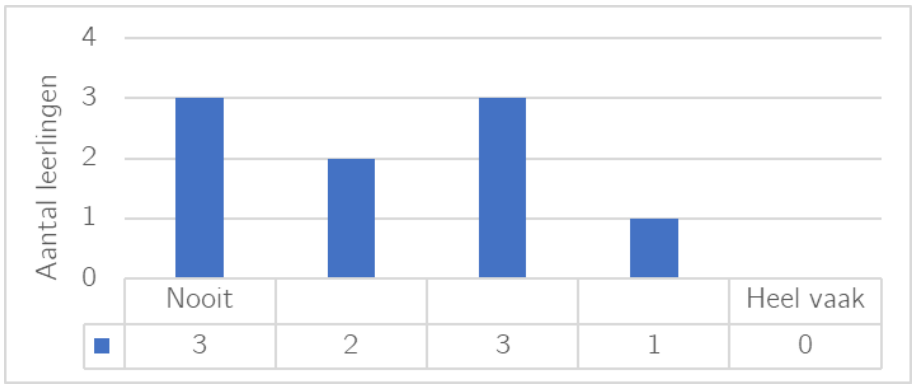
Uiteindelijk hebben in 2022 negen leerlingen uit twee klassen het logboek ingevuld, in twee opeenvolgende lessen. Twee van de deelnemers zaten op de havo, en zeven op het vwo (zie figuur 3.1). De leerlingen schatten zich op 2 tot 4 punten als het aankomt op hun programmeervaardigheid, waarbij 1 staat voor “niet zo goed in programmeren” en 5 voor “heel goed” (zie figuur 3.2). Gemiddeld schatten de leerlingen hun eerdere ervaring op 2.9, waarbij 1 staat voor “nooit eerder geprogrammeerd” en 5 voor “heel vaak”, verspreid over 1 tot 4 punten (zie figuur 3.3). Gezien de zeer beperkte grootte van de populatie, valt hier verder weinig uit af te leiden.



Figuur 3.1: Verdeling van respondenten over schoolniveau's



Figuur 3.2: Verdeling van respondenten over zelfingeschatte programmeervaardigheid



Figuur 3.3: Verdeling van respondenten over eerdere programmeerervaring

3.3 Instrumenten

We gebruikten in dit onderzoek vijf verschillende instrumenten. Deze vijf instrumenten geven samen een antwoord op de drie deelvragen (zoals beschreven in sectie 3.1) en daarmee een antwoord op de onderzoeksvraag. Secties 3.3.1 tot 3.3.3 beschrijven de drie instrumenten van het vooronderzoek, sectie 3.3.4 beschrijft het aanvullende literatuuronderzoek in aanloop naar het en sectie 3.3.5 beschrijft het instrument voor de evaluatie van de ontwerpfase. In deze sectie geven we een algemene beschrijving van de instrumenten, meer details over de uitvoering en dataverzameling voor elk instrument zijn te vinden in sectie 3.4.

3.3.1 Interview met vakdocenten

In deze interviews met ervaren vakdocenten wilden we een overzicht krijgen van het speelveld: wat zijn de fouten waar zij leerlingen vaak bij moeten helpen? Hoe helpen ze leerlingen om die fouten op te lossen? Aan welke kennis of vaardigheden ontbreekt het bij leerlingen dat ze deze fouten niet of moeilijk op kunnen lossen? En hoeveel komt dat eigenlijk voor?

De uitgewerkte interviewleidraad is te vinden in appendix A.1. We hebben gekozen voor een omgekeerde trechterstructuur van het interview (Van der Donk & Van Lanen, 2020): we beginnen met vragen over de specifieke situatie dat een leerling vastloopt op een foutmelding en een vraag stelt. Door naar een voorbeeld te vragen proberen we gedetailleerde antwoorden te krijgen over de fout waar de leerling mee zat en hoe de docent kon helpen om de fout op te lossen. Vervolgens kijken we breder naar de fouten die leerlingen maken en proberen via overeenkomsten en verschillen tot een aantal categorieën te komen. Voor het overzicht van het speelveld vragen we ook hoe vaak het voorkomt dat een leerling vastloopt op een fout.

Tot slot vliegen we het onderwerp nog vanuit een andere hoek aan: besteedt de docent ook klassikaal aandacht aan het oplossen van fouten? Dit kan namelijk invloed hebben op de hulp die leerlingen één-op-één nodig hebben, bijvoorbeeld omdat ze geleerd hebben hoe een bepaalde fout opgelost kan worden. Het zou ook kunnen dat een klassikale uitleg over foutmeldingen in algemenere termen plaatsvindt in plaats van uitleg één-op-één bij een specifieke fout. In dat geval kunnen we hier iets leren over een mogelijke categorisering van fouten, of over de kennis die de docent in het algemeen nodig acht om een fout op te lossen.

3.3.2 Interview met vaksteunpunten

Alle vakinhoudelijke begeleiding loopt in Co-Teach via de vaksteunpunten, dus de assistenten die daar werken zouden een goed beeld moeten krijgen van de fouten waar leerlingen op vastlopen. Idealiter krijgen ze in die gevallen vragen van leerlingen, maar het kan ook zo zijn dat ze die fouten in ingeleverde opdrachten tegenkomen.

We zouden de e-mails met vragen die leerlingen stuurden en de ingeleverde opdrachten kunnen analyseren op fouten, maar deze gegevens zijn natuurlijk niet anoniem en alle leerlingen om toestemming vragen om hun e-mails voor dit onderzoek te gebruiken, is niet haalbaar. De assistenten bij de vaksteunpunten hebben door hun werk al een redelijk overzicht van alle beschikbare informatie, dus kozen we ervoor om hen te interviewen. De volgende zaken wilden we te weten komen:

- Hoeveel vragen stellen leerlingen? Hoe is de verdeling: zijn er een paar leerlingen die bijna alle vragen stellen, of stelt iedereen af en toe een vraag? Stellen ze vragen als ze een fout hebben waar ze niet uitkomen, of alleen als ze bijvoorbeeld de uitleg niet begrijpen?
- Over welke fouten stellen leerlingen vragen? Zijn er bepaalde foutmeldingen die terugkomen?

- Met welke antwoorden op die vragen zijn leerlingen het best geholpen? Wat leg je uit als je een bepaalde vraag krijgt?
- Welke soorten fouten maken leerlingen in de ingeleverde opdrachten? Welke kennis denk je dat er bij leerlingen ontbreekt waardoor ze deze fout maken of niet op kunnen lossen?

De uitgewerkte interviewleidraad is te vinden in appendix A.2. We beginnen het interview met wat algemene vragen om een beeld te krijgen van de situatie: hoeveel leerlingen zijn er eigenlijk die vragen kunnen stellen, en in hoeverre is het onderwerp al afgerond op de scholen? In de tweede ronde hebben we op dit punt ook enkele vragen toegevoegd over de begeleiding en contact met leerlingen in het algemeen, naar aanleiding van de antwoorden uit de eerste ronde.

Vervolgens vragen we naar de vragen die leerlingen stellen in het algemeen, om een beeld te krijgen van de hoeveelheid vragen die gesteld worden en hoeveel daarvan betrekking hebben op programmeerfouten. Daarna kijken we naar de vragen die leerlingen stellen met betrekking tot foutmeldingen, gevolgd door vragen over programma's die niet werken als verwacht. Bij beide onderdelen vragen we hoe de assistent leerlingen hielp in verschillende situaties, welke kennis de leerlingen misten om de fout op te lossen, en proberen we te sturen naar een categorisering. We sluiten af met de fouten die nog in opdrachten te vinden zijn, en stellen daar dezelfde vragen over.

De volgorde van de vragen in het interview volgt ongeveer de volgorde waarin leerlingen verschillende fouten tegenkomen: je loopt vast op foutmeldingen voordat je kunt zien of het programma überhaupt werkt, en hopelijk zijn de meeste fouten opgelost voordat het programma ingeleverd wordt. Daarnaast zijn foutmeldingen makkelijker te categoriseren dan fouten waarbij het programma onverwacht gedrag vertoond, waardoor de deelnemers bij het bespreken van foutmeldingen een duidelijker idee krijgen van een nuttige categorisering, die dan ook bij de andere fouten toegepast kan worden.

3.3.3 Logboek

Voor direct inzicht in de fouten die leerlingen als moeilijk ervaren, kunnen we natuurlijk de leerlingen zelf vragen. Om de ervaringen van zoveel mogelijk leerlingen mee te kunnen nemen, maakten we hiervoor gebruik van een logboek. In een online omgeving vragen we leerlingen om gedurende één les bij te houden welke fouten en foutmeldingen ze tegenkwamen waarvan ze niet direct weten hoe ze erop moesten reageren. Dit kan bijvoorbeeld het geval zijn als het programma niet werkt zoals verwacht en de leerling niet begrijpt waarom, of omdat er een foutmelding getoond wordt waarvan de leerling niet begrijpt wat die betekent.

Vooraf aan het logboek stellen we enkele vragen over de leerling, om te kunnen controleren dat de gewenste spreiding in eigenschappen bij de respondenten aanwezig is. We vragen leerlingen daarvoor naar hun schoolniveau (havo of vwo) en een zelfinschatting van hun vaardigheden en voorgaande ervaring in het programmeren.

In het logboek vragen we leerlingen te noteren welke fouten ze tegenkomen: wat doet het programma en wat had je eigenlijk verwacht? Of wat is de foutmelding die je krijgt? Daarnaast vragen we leerlingen om in één zin op te schrijven wat ze gaan doen om de fout op te lossen.

Deze vragen kunnen allemaal beantwoord worden op het moment dat een leerling vastloopt op een fout, waardoor we hopen dat het invullen van het logboek een minimale onderbreking van hun werk is. We vragen ze immers alleen iets in te vullen als ze toch al even niet wisten hoe ze verder konden. We vragen leerlingen niet of het gelukt is om een fout op te lossen, of hoeveel moeite dat kostte, omdat de leerling op dat moment juist wel weer aan de slag kon. Het invullen van het logboek zou op dat moment een grote onderbreking zijn geweest.

Als afsluiting stellen we nog wel enkele vragen over in hoeverre leerlingen de fouten die ze hebben

opgeschreven konden oplossen en hoeveel tijd ze dat heeft gekost. Dit instrument heeft daarmee praktisch de vorm van een enquête, die gedurende een wat langere periode wordt afgenomen. Een overzicht van alle vragen is te vinden in appendix A.3.

3.3.4 Aanvullend literatuuronderzoek

Op basis van de resultaten uit het vooronderzoek (de drie hiervoor beschreven instrumenten) verrichten we aanvullend literatuuronderzoek naar mogelijke interventies in het lesmateriaal die leerlingen kunnen ondersteunen bij het zelfstandig oplossen van programmeerfouten. Hiervoor kijken we specifiek naar één van de basisvoorwaarden die uit het vooronderzoek naar voren kwam: zelfvertrouwen bij leerlingen dat ze daadwerkelijk in staat zijn om een fout zelf op te lossen. In de literatuur wordt dan gesproken over *self-efficacy*, dus dat is de voornaamste zoekterm voor dit onderzoek.

3.3.5 Expertreview

Uit de resultaten van het vooronderzoek en aanvullend literatuuronderzoek hebben we aantal ontwerpeisen opgesteld en daarmee een voorstel voor een ontwerp gemaakt dat leerlingen zou kunnen ondersteunen bij het zelfstandig oplossen van programmeerfouten. Dit ontwerp is niet volledig uitgewerkt en dus niet geschikt om met leerlingen te testen, maar om het voorstel toch van enige validatie te voorzien hebben we twee vakdocenten gevraagd voor een expertreview. Het doel van deze reviews was om te begrijpen in hoeverre het ontwerpvoorstel een goed antwoord op de derde onderzoeksvraag geeft.

De deelnemers aan de expertreview krijgen van tevoren het ontwerpvoorstel opgestuurd om alvast te bekijken, maar de daadwerkelijke review vind mondeling plaats om door te kunnen vragen op de gegeven antwoorden en zo een uitgebreider beeld te krijgen. Met de review wilden we twee zaken in meer detail te weten komen:

- Helpt dit materiaal om het zelfvertrouwen van leerlingen in het oplossen van programmeerfouten te vergroten?
- Welke kennis missen leerlingen nog om een fout op te lossen op basis van dit materiaal?

De uitgewerkte interviewleidraad is te vinden in appendix A.4. We beginnen het interview met een introductie van het doel: leerlingen ondersteunen bij het zelfstandig oplossen van programmeerfouten. Om de respondenten verder niet te beïnvloeden met de gemaakte keuzes, vragen we eerst naar een algemene eerste indruk. Zouden de docenten dergelijk materiaal zelf willen gebruiken? Wat zouden ze aanpassen als ze het zouden overnemen? Daarbij proberen we wel om de docenten meer naar de inhoudelijke kant van het materiaal te sturen dan naar de vorm.

Vervolgens lichten we de gemaakte ontwerpkeuzes toe, om te kunnen vragen naar hun inschatting van de effectiviteit van die keuzes. Helpt het materiaal bij het zelfvertrouwen van leerlingen? Is het niveau van het materiaal goed? Leiden de oefeningen tot succeservaringen? Geeft het leerlingen de benodigde kennis? Ook hier beginnen we wat algemener en vragen vervolgens in meer detail door op basis van de theoretische achtergrond. Daarmee laten we ruimte voor originele invalshoeken, maar kunnen we ook tot in detail onze ontwerpkeuzes toetsen.

3.4 Dataverzameling

De dataverzameling met behulp van deze instrumenten vond plaats in 2022 en 2024. In deze sectie beschrijven we per instrument hoe de dataverzameling is uitgevoerd.

3.4.1 Interviews met vakdocenten

We hebben drie interviews met een vakdocent gehouden, allen in 2022. Eén interview vond plaats op de Universiteit Twente, de andere twee online via Microsoft Teams. De interviews werden opgenomen en duurden ongeveer 20 tot 30 minuten. Deze opnames zijn uitgewerkt tot een anoniem transcript en worden vernietigd aan het eind van het onderzoek. Voor het interview is mondeling toestemming gegeven voor het gebruik van de opname voor het onderzoek en het opnemen van anonieme citaten in het verslag, nadat de informed consent door de interviewer was toegelicht. De respondenten hebben deze informatie ook in geschreven tekst ontvangen.

3.4.2 Interviews met vaksteunpunten

We hebben twee rondes van deze interviews gehouden: de eerste in 2022 en de tweede twee jaar later in 2024. In beide rondes spraken we van iedere deelnemende universiteit één vaksteunpuntassistent, om een zo volledig mogelijk beeld te krijgen. Het interview zou idealiter afgenomen worden op het moment dat het onderwerp programmeren net is afgerond, maar door de verschillende planningen was dat niet altijd het geval. De lessen waren wel al afgerond, maar nog niet alle leerlingen hadden alle opdrachten ingeleverd. Wel hadden alle studentassistenten die we spraken al een deel van de opdrachten nagekeken.

Alle interviews vonden online plaats via Microsoft Teams en werden opgenomen. Een interview duurde ongeveer 15 minuten. De opnames zijn uitgewerkt tot een anoniem transcript en worden vernietigd aan het eind van het onderzoek. Aan het begin van het interview is mondeling toestemming gegeven voor het gebruik van de opname voor het onderzoek en het opnemen van anonieme citaten in het verslag, nadat de informed consent door de interviewer was toegelicht. De respondenten hebben deze informatie ook in geschreven tekst ontvangen.

3.4.3 Logboek

Het logboek kan volledig anoniem worden ingevuld, waardoor er geen uitgebreide consentprocedure met toestemming van ouders nodig was. We informeerden leerlingen voorafgaand over het doel van het onderzoek en hoe de ingevulde gegevens gebruikt worden. Mocht de leerling hiermee niet akkoord zijn gegaan, kon die simpelweg de website sluiten en niet deelnemen aan het onderzoek. De gegevens die leerlingen invulden werden gedurende 3 uur gekoppeld aan hun computer. Tot die tijd hadden ze de mogelijkheid om hun gegevens aan te passen, of hun toestemming in te trekken en alles te verwijderen. Na 3 uur werden de gegevens van de computer van de leerling verwijderd, en daarmee volledig geanonimiseerd.

Leerlingen vulden het logboek in op de daarvoor ontwikkelde website. Zodra de leerling een vraag beantwoorde, werd het antwoord ook op de server opgeslagen. Zo konden we voorkomen dat leerlingen aan het eind van de les kunnen vergeten om hun logboek in te sturen. De antwoorden op de startvragen en de ingevoerde logboekregels werden gekoppeld aan een willekeurig gegenereerde code, om mogelijke verbanden daarin te kunnen zien en om leerlingen de mogelijkheid te geven binnen 3 uur na het starten van het logboek hun toestemming in te trekken. Na 3 uur werd deze code verwijderd van de computer van de leerling, waarmee de antwoorden automatisch geanonimiseerd werden.

Het was helaas niet mogelijk om het logboek in te laten vullen door leerlingen die zelf les krijgen via Co-Teach. In plaats daarvan vroegen we leerlingen die volgens een vergelijkbaar curriculum les krijgen van een vakdocent om het logboek in te vullen. Dit is in 2022 gelukt bij twee klassen van dezelfde docent, met in totaal 9 leerlingen. (Het waren kleine klassen, en ook nog veel afwezigen.) We vroegen de docent om tijdens die les geen vragen van de leerlingen over programmeren te beantwoorden, zodat de situatie in die les vergelijkbaar was met de situatie van Co-Teach. Uit de literatuur blijkt dat de zelfstandigheid van leerlingen invloed heeft op

hun vermogen om fouten op te lossen (Michaeli & Romeike, 2019) en ook uit interviews met vakdocenten kwam dat naar voren. Omdat de leerlingen in deze les zelfstandiger waren dan in andere lessen, vroegen we ze aan het eind van de les of ze nu anders omgingen met fouten dan in andere lessen. Daarmee hoopten we inzicht te krijgen in het effect van zelfstandigheid. Dat is van belang omdat de mate van zelfstandigheid het grootste verschil is tussen lessen via Co-Teach en reguliere informaticalessen.

3.4.4 Aanvullend literatuuronderzoek

Het aanvullend literatuuronderzoek vond plaats in het voorjaar van 2024. De voornaamste zoekterm voor dit onderzoek was *self-efficacy*, in verband tot programmeren, lesmateriaal en interventies. Voor algemene resultaten hebben we gebruik gemaakt van de zoekmachines Scopus en Google Scholar, voor meer vakgerichte resultaten hebben we ook gezocht in de ACM Digital Library.

3.4.5 Expertreview

Voor de expertreview hebben we interviews gehouden met twee vakdocenten (anderen dan we tijdens de interviews hebben gesproken). De twee deelnemers kregen van tevoren het ontwerpvoorstel opgestuurd om alvast te bekijken, maar de daadwerkelijke review vond mondeling plaats.

De interviews vonden online plaats via Microsoft Teams en werden opgenomen. Een interview duurde ongeveer 15 minuten. De opnames zijn uitgewerkt tot een anoniem transcript en worden vernietigd aan het eind van het onderzoek. Bij een van de interviews is de opname en Teams mislukt en is de analyse gebaseerd op aantekeningen die direct na afloop zijn gemaakt. Aan het begin van het interview is mondeling toestemming gegeven voor het gebruik van de opname voor het onderzoek en het opnemen van anonieme citaten in het verslag, nadat de informed consent door de interviewer was toegelicht. De respondenten hebben deze informatie ook in geschreven tekst ontvangen.

3.5 Analyse

De resultaten uit de dataverzameling hebben we vervolgens geanalyseerd om tot bruikbare informatie te komen. Hieronder beschrijven we per type instrument de gebruikte analysemethode.

3.5.1 Analyse van de interviews

Voor de interviews met vakdocenten en vaksteunpunten en de expertreview-interviews gebruikten we eenzelfde analysemethode. Alle interviews werden opgenomen en verwerkt tot een transcript, dat gebruikt is voor de verdere analyse. Het transcript van elk interview hebben we thematisch gecodeerd, met de vragen uit de beschrijving van het interview als thema's (Van der Donk & Van Lanen, 2020).[1] We kozen voor thematische codering in plaats van een open codering om de nuance in elk antwoord op de vragen te kunnen vangen, en zo een betere vergelijking te kunnen maken. De verschillende antwoorden hebben we samengevoegd tot een horizontale vergelijking per vraag,[2] waaruit we uiteindelijk een samenvatting als antwoord op de vraag hebben gedestilleerd.

[1] De thematische coderingen van elk interview zijn te vinden in appendix B.

[2] De horizontale vergelijkingen zijn te vinden in tabellen 4.1 tot 4.9 en 6.1 tot 6.3.

3.5.2 Analyse van het logboek

Allereerst hebben we bij het logboek beschrijvende statistiek toegepast om de onderzoekspopulatie te beschrijven: hoeveel leerlingen hebben het logboek ingevuld, hoeveel fouten hebben ze daarbij genoteerd en hoe is de verdeling van leerlingen in schoolniveau, zelf ingeschatte vaardigheden en ervaring. De resultaten hiervan zijn te vinden in sectie 3.2. Bij de ingevulde fouten hebben we gekeken naar het type fout en de waarschijnlijke aanleiding voor die fout, om te zien waar deze fouten vandaan komen. De ingevulde fouten zijn te vinden in sectie 4.3.

3.5.3 Literatuuronderzoek, ontwerp en evaluatie

In het vooronderzoek hebben we met interviews en het logboek uitgezocht wat de fouten zijn waar leerlingen op vastlopen en wat ze nodig hebben om die fouten wel zelfstandig op te kunnen lossen. Met die resultaten hebben we aanvullend literatuuronderzoek gedaan naar mogelijke interventies in het lesmateriaal om leerlingen daarin te ondersteunen, waaruit we een aantal ontwerpeisen hebben gedestilleerd. Op basis daarvan hebben we een prototype voor een ontwerp gemaakt, dat een mogelijk antwoord geeft op de derde deelvraag: *wat kunnen we in het lesmateriaal verbeteren om leerlingen meer fouten zelfstandig op te laten lossen?* Om de antwoorden op beide vragen te valideren maakten we gebruik van het laatste instrument: de expertreview, in de vorm van interviews waarin het ontwerp besproken wordt. Die interviews hebben we geanalyseerd zoals beschreven in sectie 3.5.1.

Hoofdstuk 4

Resultaten en conclusies vooronderzoek

In dit hoofdstuk bespreken we de resultaten van het vooronderzoek, dus uit de interviews en het logboek: in secties 4.1 tot 4.3 eerst per instrument en in sectie 4.4 samengevat per onderzoeksvraag. Bij de interviews maken we gebruik van de vragen die in sectie 3.3 beschreven zijn, plaatsen we de antwoorden in een horizontale vergelijking en geven tot slot een samenvatting van de antwoorden per vraag. Voor het logboek presenteren we de ingevulde antwoorden en geven vervolgens een samenvatting. Een volledige thematische codering van ieder interview is te vinden in appendix B.

4.1 Interviews met vakdocenten

In deze sectie bespreken we de resultaten uit de interviews met vakdocenten, zoals beschreven in sectie 3.3.1. De resultaten zijn opgesplitst aan de hand van de vier vragen die we met deze interviews wilden beantwoorden. In elke sectie staat een tabel met een horizontale vergelijking van de antwoorden uit elk van de drie interviews, gevolgd door een samenvatting van die antwoorden. Een volledige thematisch codering van elk van de interviews is te vinden in appendix B.1.

4.1.1 Wat zijn de fouten waarbij je leerlingen vaak moet helpen?

Tabel 4.1: Wat zijn de fouten waarbij je leerlingen vaak moet helpen?

Docent 1	Docent 2	Docent 3
80% syntaxfouten, vooral aan het begin en bij havo meer dan vwo	Syntaxfouten, bij de havo meer dan vwo	80-90% syntaxfouten
Bij lijsten: indexfouten		
Algoritmische fouten, opgemerkt door een falende test	Fouten in opzet van constructie: welke variabelen, loops, condities	Logica niet helemaal helder: fouten in variabelen, condities

Docent 1	Docent 2	Docent 3
	Veel verschil: een deel loopt al vast op syntactische fouten, een ander deel lukt dat wel maar loopt meer vast op semantische fouten in de opbouw	

Alle drie de docenten gaven aan dat syntaxfouten toch het meeste voorkomen bij hun leerlingen en twee docenten gaven daar zelfs een aandeel van 80% tot wel 90% aan. Leerlingen leken na verloop van tijd wel aan de syntax te wennen, waardoor de hoeveelheid fouten afnam, hoewel nieuw geïntroduceerde syntax ook weer tot nieuwe syntaxfouten leidde. Fouten zoals het verkeerd schrijven van een variabelenaam (bijvoorbeeld met/zonder hoofdletter) schaarden de docenten hier ook onder.

De tweede categorie fouten die de docenten noemden was verwarring in de logica van het programma. Dat had over het algemeen tot resultaat dat het programma niet deed wat de leerling verwacht had, maar een van de docenten gebruikte ook geautomatiseerde tests om deze fouten aan het licht te brengen. Vaak ging het bij deze fouten om een variabele die hergebruikt werd, een conditie die niet helemaal klopte of een opbouw die zo gegroeid was dat het onduidelijk werd wat het programma zou moeten doen.

Een docent plaatste nog een belangrijke kanttekening: er zit veel verschil tussen leerlingen. Sommige leerlingen liepen vast op syntaxfouten, maar er waren er ook die daar geen last van hebben en meer vastliepen op een verwarring in de logica. De andere docenten gaven ook aan dat syntaxfouten bij de havo vaak een groter probleem waren dan bij het vwo. Dat was natuurlijk geen strakke scheiding: ook op de havo waren er leerlingen die goed wisten wat ze aan het doen waren en ook op het vwo waren er leerlingen die veel moeite hadden met syntaxfouten. Alle docenten benoemden dit verschil echter wel op enig moment in het interview.

4.1.2 Hoe help je leerlingen bij het oplossen van een fout?

Tabel 4.2: Hoe help je leerlingen bij het oplossen van een fout?

Docent 1	Docent 2	Docent 3
Voorbeeldcode geven om op voort te bouwen	Met een groep zwakkere leerlingen: samen de basis leggen voor een opdracht	
Syntax explicieter benadrukken bij de uitleg		
Helpen bij het maken van de denkstappen vanaf de fout naar een oplossing, totdat de leerling doorheeft hoe die oplossing eruit ziet		Terug naar de basis: wat is het doel en hoe pakken we dat aan

Docent 1	Docent 2	Docent 3
	Code stapje voor stapje doorlopen, eventueel met print-statements om te ontdekken waar de fout plaatsvindt	
Foutmelding voorlezen en eventueel uitleggen		Aanwijzingen geven over de locatie van de fout
		Inschatten waar de leerling staat en daarop gericht uitleg geven
		Leerlingen online naar materiaal laten zoeken

Twee docenten geven aan dat ze proberen fouten te voorkomen door de leerlingen een stevigere basis te geven: een van de docenten geeft leerlingen voorbeeldcode om op voort te bouwen. Bij de eerste opdrachten kunnen leerlingen die code dan steeds aanpassen aan de eisen van de opdracht, zodat ze minder snel syntaxfouten maken in de nieuwe syntax. Deze docent gaf ook aan de syntax meer te benadrukken bij de uitleg dan voorheen door expliciet aandacht te geven aan de verschillende onderdelen van de code, in plaats van alleen het concept uit te leggen. Een andere docent neemt bij grotere opdrachten een groep met zwakkere leerlingen apart om samen een beginnetje te maken met de opdracht. Daarbij ligt de focus vooral op de aanpak van het probleem, waardoor leerlingen van tevoren een duidelijker idee hebben van hoe het programma kan werken.

Alle drie de docenten geven aan leerlingen te helpen met het denkproces bij het oplossen van een fout: dat kan zijn door concreet naar de fout te kijken en kleine stapjes te nemen naar een oplossing totdat de leerling doorheeft wat die oplossing kan zijn; door de code stapje voor stapje door te nemen om ontdekken waar de fout precies plaatsvindt; of door een stap terug te nemen naar het doel en leerlingen meer top-down te laten kijken naar hoe het programma zou moeten werken. Welke manier het meest van toepassing is, hangt natuurlijk af van de specifieke situatie: als een leerling een heel onlogische structuur voor het programma heeft gekozen, is het handiger om vanaf het begin te kijken wat nu eigenlijk het doel was. Bij een klein foutje in een bepaalde conditie is het doorlopen van de code een goede manier om hulp te bieden, omdat leerlingen dan kunnen ontdekken waar het programma de verkeerde kant op gaat.

Het is dus van belang om te kijken naar het type fout dat de leerling maakt en daar de uitleg op aan te passen. Eén docent noemt ook dat het van belang is om het niveau van de leerling in te schatten om zo op het juiste niveau uitleg te geven. Bij programmeren bouwen veel concepten op elkaar door, dus is het van belang om alle onderliggende concepten te begrijpen. Een belangrijke taak van de docent is om uit te vinden waar de leerling de mist in gaat en vanaf dat punt bij te sturen. De docent verwacht ook dat dit een van de moeilijkste aspecten is om in Co-Teach op te vangen.

Tot slot geven twee docenten aan leerlingen te helpen met het lezen van foutmeldingen: door uit te leggen wat het betekent en door te wijzen op de locatie in de code waar de fout plaatsvindt. Vooral bij syntaxerrors is dat laatste van belang, omdat de foutmelding niet altijd de juiste locatie aangeeft. Verder laat een van de docenten leerlingen ook online zoeken naar materiaal, als de uitleg in de methode voor hen niet duidelijk was.

4.1.3 Waar ligt het aan dat leerlingen een fout niet kunnen oplossen?

Tabel 4.3: Waar ligt het aan dat leerlingen een fout niet kunnen oplossen?

Docent 1	Docent 2	Docent 3
Leerlingen hebben op basis van de foutmelding niet door wat er fout is (in het geval van een indexfout)		
Leerlingen lezen de foutmelding niet, gaan in paniekstand	Gebrek aan zelfvertrouwen en/of doorzettingsvermogen: leerlingen die iets minder snel opgeven bij een fout, zijn beter in staat een fout op te lossen	
		Leerlingen gaan te snel en raken overmoedig, omdat de uitleg heel makkelijk lijkt. Ze onderschatten de precisie die vereist is en nemen daarom niet de tijd om te analyseren wat er fout is gegaan
Te weinig druk om het zelf op te lossen: de docent is in de buurt om te helpen	De docent wil te snel leerlingen helpen	
Leerlingen weten niet waar ze op moeten letten, wat de oorzaak van een fout is	Leerlingen moeten nog wennen aan de syntax	
	Gebrek aan analytisch, abstract denkvermogen: opdelen van groot probleem in kleinere problemen	Leerlingen hebben niet helder wat ze aan het doen zijn: bij een gestructureerde opdracht kunnen ze wel de stappen volgen, maar bij een vrijere opdracht weten ze niet waar te beginnen

Een thema dat bij alle drie de docenten naar voren komt, is zelfvertrouwen. Twee docenten noemen een gebrek aan zelfvertrouwen bij leerlingen: als ze een fout tegenkomen hebben ze niet het idee dat ze die zelf op kunnen lossen en dus geven ze op. Voor sommigen is dat een paniekreactie bij het krijgen van een foutmelding. De derde docent gaf aan dat het ook kan komen doordat leerlingen overmoedig raken: als de docent het voordoet, is het vrij simpel en het principe is helder, maar als leerlingen het zelf moeten doen vallen ze over alle details en blijkt het toch moeilijker dan gedacht. Dat kan ook een klap zijn voor het zelfvertrouwen. Twee docenten gaven ook aan leerlingen graag te willen helpen, waardoor leerlingen misschien niet de kans krijgen om zelf met een fout te worstelen en gewend raken aan directe hulp.

Verder zien de docenten een gebrek aan kennis: twee merken op dat leerlingen nog moeten wennen aan de syntax en daardoor niet opmerken waar een syntaxfout zit. Naarmate de leerlingen de

syntax beter kennen, zien ze een fout ook makkelijker. Bij fouten waarbij het programma niet werkt als verwacht is het voor leerlingen soms moeilijk om tot een oplossing te komen omdat hun analytisch en abstract denkvermogen nog niet voldoende ontwikkeld is. Eén van de docenten gaf ook aan dat ze stappen niet helder hebben: bij een gestructureerde opdracht kunnen leerlingen de stappen volgen en fouten oplossen in die kleinere stappen, maar bij vrijere opdrachten weten ze niet waar te beginnen.

Eén docent noemt ook nog dat gegeven foutmeldingen niet altijd duidelijk zijn voor de leerlingen, bijvoorbeeld bij een "index out of bounds" error. Met name de frase "out of bounds" is voor veel leerlingen onbekend, waardoor ze geen idee hebben wat de foutmelding betekent. Zodra de docent dat uitlegde, waren veel leerlingen wel in staat om de fout op te lossen.

4.1.4 Hoe vaak lopen leerlingen vast?

Tabel 4.4: Hoe vaak lopen leerlingen vast?

Docent 1	Docent 2	Docent 3
20% loopt eens per tien minuten vast en komt geen stap verder	Een deel loopt continue vast, elke keer als de docent langskomt	
Gemiddeld eens per kwartier of 20 minuten		Gemiddeld één à twee keer per les is een klein 'tikje' nodig

Een deel van de leerlingen loopt vrijwel continue vast, elke keer dat de docent langskomt kunnen ze hulp gebruiken. Gemiddeld lopen leerlingen een paar keer per les vast.

4.2 Interviews met vaksteunpunten

In deze sectie bespreken we de resultaten uit de interviews met assistenten van de vaksteunpunten, zoals beschreven in sectie 3.3.2. De resultaten zijn opgesplitst aan de hand van de vier vragen die we met deze interviews wilden beantwoorden, gevolgd door een sectie met de vragen die tijdens het interview naar boven kwamen. In elke sectie staat een tabel met een horizontale vergelijking van de antwoorden uit elk van de drie interviews, gevolgd door een samenvatting van die antwoorden. Een volledige thematische codering van elk van de interviews is te vinden in appendix B.2.

4.2.1 Hoeveel vragen stellen leerlingen?

Tabel 4.5: Hoeveel vragen stellen leerlingen over programmeren (gedurende die module)?

2022			2024		
Vaksteunpunt			Vaksteunpunt		
1	2	3	1	2	3
Geen vragen uit 20 leerlingen	Twee vragen door één leerling, uit 40 à 45 leerlingen	Redelijk weinig vragen	Twee vragen in totaal over programmeren	Weinig vragen	Weinig vragen, zou veel meer mogen
			Over programmeren veel minder dan over andere onderwerpen	Vergelijkbaar met andere onderwerpen	

Leerlingen stellen weinig tot geen vragen over programmeren. Bij sommige vakststeunpunten ook aanzienlijk minder dan over andere onderwerpen, bij anderen is het algehele beeld dat er weinig vragen binnenkomen. Een van de assistenten hoorde van de co-teachers wel dat de leerlingen het onderwerp heel lastig vonden, maar ook daar kwamen weinig vragen uit. In de vergelijking tussen 2022 en 2024 lijkt er weinig verschil te zitten in de hoeveelheid vragen die gesteld worden, waarbij ook rekening gehouden moet worden met dat er nu meer leerlingen zijn.

4.2.2 Over welke fouten stellen leerlingen vragen?

Tabel 4.6: Over welke fouten stellen leerlingen vragen?

2022			2024		
Vaksteunpunt			Vaksteunpunt		
1	2	3	1	2	3
	Het programma werkt niet als verwacht en de leerling komt er niet uit, soms omdat ze niet alle stappen in de opdracht meenemen				

2022			2024		
Vaksteunpunt			Vaksteunpunt		
1	2	3	1	2	3
		Grootste deel syntax errors of fouten in variabelenamen	Een syntaxfout met een dubbele punt in een while loop		
					Vrijwel geen vragen over foutmeldingen
			Alle code, inclusief imports etc., binnen de oneindige programmaloop	Verkeerde import, daardoor een error dat een functie niet bestaat	
	Geen fout: vraag om advies/uitleg over een constructie	Geen fout: vraag over hoe functies werken		Geen fout: vragen over het platform, meer organisatorisch	Geen fout: opdracht niet goed begrepen/gelezen

Bij een van de vaksteunpunten ging de enige vraag gerelateerd aan een fout over een programma dat niet werkte zoals verwacht, wat door de anderen niet genoemd werd. Twee vaksteunpunten gaven aan wel eens vragen over fouten in de syntax en variabelenamen te krijgen, maar een derde gaf juist expliciet aan dat dat bijna niet voorkwam. Twee kregen ook vragen over meer code-organisatorische zaken, zoals imports binnen de hoofdloop van het programma of een fout bij het importeren van functies. Vier vaksteunpunten kregen ook vragen die niet over fouten gingen, bijvoorbeeld een vraag om uitleg, meer organisatorische vragen of dat de leerling de opdracht niet goed begreep. Bij één vaksteunpunt kwamen helemaal geen vragen binnen.

4.2.3 Hoe help je leerlingen die een bepaalde vraag hebben?

Tabel 4.7: Hoe help je leerlingen die een bepaalde vraag hebben?

2022			2024		
Vaksteunpunt			Vaksteunpunt		
1	2	3	1	2	3
	Hints geven, vaak op basis van wat al in de opdracht staat, maar wat de leerling nog niet gebruikt had	Hints om leerlingen in de juiste richting te sturen, wijzen op plaats in de code waar ze naar moeten kijken	Uitleg geven over waarom het fout gaat		Sturen in de richting van het antwoord, op basis van de uitwerkingen

De meeste assistenten die vragen hebben gekregen gaven aan leerlingen hints te geven om ze de juiste kant op te sturen. Soms op basis van stappen in de opdracht die de leerling over het hoofd had gezien of door ze naar een bepaald punt in de code te sturen waar de fout opgelost kon worden.

4.2.4 Welke soorten fouten maken leerlingen in de ingeleverde opdrachten?

Tabel 4.8: Welke soorten fouten maken leerlingen in de ingeleverde opdrachten?

2022			2024		
Vaksteunpunt			Vaksteunpunt		
1	2	3	1	2	3
		Vooral fouten met functies, waardoor het programma niet werkt als verwacht			
		Paar verkeerde loopconstructies, waardoor het programma niet werkt als verwacht		Duidelijk geen idee wat een if-statement is	

2022			2024		
Vaksteunpunt			Vaksteunpunt		
1	2	3	1	2	3
Verwarring over het combineren van if-statements, waardoor het programma niet werkt zoals verwacht			Fouten in het combineren van onderdelen in een groter programma	Logische fouten bij het combineren van verschillende functionaliteiten	
			Meerdere variabelen met dezelfde naam, waardoor het programma niet werkt als verwacht		
	Eén opdracht met een fout: waarschijnlijk een import error of een name error		Heel veel syntaxfouten		Van sommige onderdelen lijkt de syntax lastig

Bij alle vaksteunpunten kwamen wel fouten voor in de ingeleverde opdrachten. Daarbij ging het voornamelijk om logische fouten, bijvoorbeeld in functies, bij loop-constructies en if-statements. Drie assistenten noemden vooral het combineren van verschillende onderdelen in een groter programma waarop het voor een aantal leerlingen misliep. Bij drie vaksteunpunten kwamen ook programma's met syntaxfouten binnen, maar de mate waarin verschilt: bij de ene was het slechts één programma, bij een ander juist opvallend veel.

4.2.5 Overige observaties

Tabel 4.9: Overige observaties

2022			2024		
Vaksteunpunt			Vaksteunpunt		
1	2	3	1	2	3
	Ingeleverde programma's met een duidelijk fout vermoedelijk doordat de leerling overmoedig was en niet heeft getest	Programma met fout ingeleverd omdat antwoord op een vraag ook lang duurt			
Geen vragen doordat de co-teacher zelf ook redelijk kon helpen			Co-teachers kunnen bij de theorie best veel helpen, als ze er zich een beetje in verdiepen	Sommige co-teachers kunnen leerlingen veel zelf helpen, anderen doen dat minder	Hulp door de co-teachers verschilt van heel veel tot geen
			Veel online te vinden en leerlingen kunnen ook ChatGPT gebruiken	Leerlingen redden zich zedelijk met online bronnen en ChatGPT	Leerlingen vragen eerder aan ChatGPT dan bij het vaksteunpunt
	Hoge drempel om vragen te stellen voor leerlingen via email en aan een onbekende	Geen vragen doordat het een tijdje duurt voor je antwoord hebt, dan zoeken ze het zelf wel uit	De drempel om te mailen is hoog, vooral voor pubers	Het helpt als een co-teacher aanmoedigt om vragen te mailen, zou ook in het platform genoemd kunnen worden	
				Eenmaal over de drempel heen is het makkelijker	Zodra ze een vraag gesteld hebben, stellen ze er meer

2022			2024		
Vaksteunpunt			Vaksteunpunt		
1	2	3	1	2	3
			Bij een moeilijke module als programmeren stellen leerlingen minder snel vragen		Het onderwerp is “overwhelming”
			Te weinig aandacht voor de syntax		

Tijdens de interviews bleek natuurlijk dat er weinig vragen bij de vaksteunpunten binnenkwamen en dat leerlingen ook duidelijk foutieve programma's inleverden. De assistenten hadden wel ideeën bij de volgende vragen:

4.2.5.1 Waarom stellen leerlingen weinig vragen?

Het verschilt per school heel erg in hoeverre de co-teacher zelf kennis heeft van programmeren en de leerlingen daarmee kan helpen. Op scholen waar dat het geval is, worden duidelijk minder vragen bij het vaksteunpunt gesteld. Vier van de assistenten geven ook aan dat ze denken dat de drempel om te mailen hoog is, vooral voor pubers. Twee geven ook aan dat leerlingen die eenmaal een vraag gesteld hebben daarna sneller nog een vraag stellen, dus het lijkt inderdaad een drempel om te nemen. Eén assistent merkt op dat meiden er minder moeite mee lijken te hebben om de eerste vraag te stellen. Ook geven twee assistenten aan dat het een moeilijk onderwerp is, en één koppelt dat expliciet aan dat er hier minder vragen over gesteld worden. Het vermoeden is dat leerlingen al snel achter lopen en dat niet toe willen geven.

In 2024 is er duidelijk een nieuwe reden bij gekomen: ChatGPT. Alle drie de vaksteunpuntassistenten die we in 2024 hebben gesproken denken dat veel leerlingen gebruik maken van ChatGPT om hulp te krijgen. Sommigen hebben daar wel hun twijfels bij omdat het niet per se duidelijk is hoe ver die “hulp” gaat, maar ze zien het overwegend wel positief dat leerlingen er wel snel door op weg geholpen kunnen worden.

4.2.5.2 Waarom leveren leerlingen een programma met (duidelijke) fouten in?

In het geval van het programma met een foutmelding vermoedt de assistent dat de leerling overmoedig was geworden door eerdere oefeningen die goed gingen en daardoor het programma niet heeft getest. Een andere assistent gaf aan dat leerlingen waarschijnlijk geen zin hadden om te wachten tot ze antwoord hadden gekregen op een vraag en dus hun programma maar gewoon inleverden, ook al werkte het niet helemaal.

4.2.5.3 Wat zou je veranderen?

Enkele assistenten hadden wel ideeën over aanpassingen aan het programma. Een van de assistenten merkt op dat het wel lijkt te helpen als de co-teachers leerlingen erop wijzen dat ze

vragen kunnen stellen en ze aanmoedigen om dat ook te doen. Een andere assistent heeft op dat punt een aanbeveling: zet in het platform en in het materiaal ook af en toe “Kom je er niet uit? Vraag het!” om leerlingen aan te moedigen daar ook gebruik van te maken.

Eén assistent merkte ook op dat syntax toch wel heel belangrijk is bij programmeren, maar dat er in het programma te weinig aandacht aan wordt besteed. Vooral meer kleine opdrachten zouden volgens hen kunnen helpen, want nu is de sprong naar een grote opdracht soms erg groot.

4.3 Logboek

In totaal hebben twee leerlingen (van de negen respondenten) drie fouten aan het logboek toegevoegd. Deze twee leerlingen volgden informatica beide op vwo-niveau, met gemiddelde vaardigheid en geen vorige ervaring. Zij vulden deze drie fouten in:

- ```
    File "main.py", line 22
      while cijfer < 1.0 or cijfer > 10.0:
                                          ^
  IndentationError: unindent does not match any outer indentation level
  ```

  Vervolgactie: De code nog eens doorlezen.

- ```
   File "main.py", line 2
       cijfer = float(input("Voer het cijfer in:"))
     ^
  SyntaxError: invalid character in identifier
  ```

  Vervolgactie: klasgenoot vragen en rustig opnieuw door lezen

- ```
  Traceback (most recent call last):
    File "main.py", line 39, in <module>
      gemiddelde = round((cijfer1 + cijfer2) / 2, 1)
  TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'
  ```

  Vervolgactie: De docent om hulp vragen.

Hier zien we twee fouten in de syntax (`IndentationError` is een specifieke soort `SyntaxError`) en (vermoedelijk) een fout in de logica die te maken heeft met datatypes. Zonder verdere context is echter niet precies te achterhalen wat bij die laatste de aanleiding voor de fout is. Er is geen fout waarbij het programma niet werkte zoals verwacht aan het logboek toegevoegd door een van de leerlingen.

Eén leerling heeft de afsluitingsvragen nog beantwoord. Dit was een andere leerling dan de twee die fouten aan het logboek hebben toegevoegd. Hierdoor waren de vragen eigenlijk niet van toepassing op deze leerling, omdat ze verwijzen naar fouten in het logboek. Daarom laten we de antwoorden buiten beschouwing.

4.4 Antwoorden op vragen

Op basis van deze resultaten kunnen we nu de eerste twee deelvragen beantwoorden:

4.4.1 Bij welke soorten fouten lopen leerlingen vast of vragen ze hulp?

Op deze vraag gaven de vakdocenten unaniem antwoord: voor een groot deel zijn dit syntaxfouten. Ook in het logboek zien we enkele syntaxfouten terug. Bij de vaksteunpunten is er echter een ander beeld: slechts één van de assistenten geeft aan veel vragen over syntaxfouten te krijgen bij
  ```

de rest speelt dat nauwelijks een rol. Ook in de ingeleverde opdrachten zitten bij de meesten nauwelijks syntaxfouten. Dat betekent dat leerlingen die fouten zelfstandig opgelost hebben (of ze nooit gemaakt hebben, maar dat is onwaarschijnlijk), of dat de leerlingen die vastlopen op een fout de opdracht niet inleveren. In 2022 was er bij alle vaksteunpunten inderdaad sprake van te laat en niet ingeleverde opdrachten, maar het merendeel van de leerlingen is het wel gelukt om iets in te leveren en zij hebben dus die fouten zelf opgelost. De niet-ingeleverde opdrachten kunnen ook niet per se hieraan toegeschreven worden, omdat het project nog in de pilot-fase zat en er qua organisatie ook nog veel geleerd werd. In 2024 was het beeld over ingeleverde opdrachten nog niet duidelijk.

Het tweede type fouten waarbij leerlingen de vakdocenten om hulp vragen is als een programma niet werkt zoals zij verwacht hadden. Dit heeft er vaak mee te maken dat een leerling een constructie niet goed heeft toegepast, meestal door een misconceptie over hoe die constructie werkt. Om deze fouten te kunnen maken, moet het programma wel geaccepteerd worden door de compiler en dus geen foutmeldingen geven. De leerlingen die deze fouten maken zijn daardoor vaak al wat verder in de stof en werken ook aan complexere opdrachten. Dat is ook terug te zien in de opdrachten die bij de vaksteunpunten worden nagekeken: hoewel daar geen syntaxfouten gemaakt worden, komen fouten waarbij het programma niet werkt zoals verwacht wel regelmatig voor. In het logboek heeft geen van de 9 leerlingen een dergelijke fout ingevoerd.

4.4.2 Welke kennis hebben leerlingen nodig om die lastige fouten (zoals in (1) gevonden) op te kunnen lossen?

Uit de interviews blijkt dat vakdocenten twee belangrijke voorwaarden zien voor het zelfstandig oplossen van een fout: zelfvertrouwen en noodzaak. Twee docenten merken op dat leerlingen soms in paniek raken of opgeven bij een fout, wat dus duidt op een gebrek aan zelfvertrouwen bij leerlingen die een fout niet zelf op kunnen lossen. Dezelfde twee geven ook aan dat ze leerlingen snel helpen, waardoor er weinig druk is voor leerlingen om een fout zelf op te lossen. Dat zou kunnen verklaren dat er bij Co-Teach weinig problematische syntaxfouten lijken te zijn: omdat er geen directe hulp beschikbaar is, voelen de leerlingen de noodzaak om het zelf op te lossen, en dat lukt dan ook.

Tegelijkertijd kan een te groot zelfvertrouwen (overmoed) ook tot fouten leiden: een vakdocent geeft aan dat leerlingen daardoor niet altijd de tijd nemen om te analyseren wat er fout is gegaan. Een van de vaksteunpuntassisstenten vermoedde ook dat dit het geval was bij een ingeleverd programma dat een foutmelding gaf: waarschijnlijk had deze leerling het programma niet getest.

Naast deze voorwaarden, is er ook kennis die sommige leerlingen niet hebben, waardoor ze een fout niet op kunnen lossen: twee docenten merken op dat leerlingen nog moeten wennen aan de syntax, wat duidt op een gebrek aan domeinkennis. Twee docenten vertelden ook dat ze leerlingen vaak helpen met het lezen van een foutmelding of aanwijzingen geven over waar de fout in de code zit (bijvoorbeeld een regel voor de regel die in de foutmelding wordt aangewezen). Dat hebben die docenten uit ervaring geleerd en die kennis ontbreekt dus bij leerlingen: zij hebben geen heuristieke kennis over de waarschijnlijke oorzaak van een fout.

Docenten helpen leerlingen ook met het vergaren van systeemkennis, zowel globaal door bijvoorbeeld stap voor stap de code door te nemen of door een stap terug te nemen en de opdracht te analyseren, als ook lokaal, bijvoorbeeld door print statements toe te voegen om te zien hoe het programma werkt. Bij de vaksteunpunten wordt op een vergelijkbare manier hulp geboden. Dit vergt analytisch en abstract denkvermogen, maar ook strategische kennis van het debugproces en lokale technieken zoals het toevoegen van print-statements.

Als laatste merkte een van de docenten ook op dat het werk van een docent ook bestaat uit het inschatten van de hulp die een leerling nodig heeft. Is het een gebrek aan zelfvertrouwen,

aan domeinkennis of meer aan strategische kennis? En in het geval van kennis: wat weet de leerling wel, welke misconcepties zijn er misschien en waar kun je op voort bouwen in de uitleg? Als leerlingen zelf hun eigen hulp moeten organiseren, moeten ze daar ook zelf achter komen, waar veel metacognitieve kennis voor nodig is. Een aantal vaksteunpuntassistenten ziet dat ook terug: als de co-teacher de leerlingen ondersteunt en ze aanmoedigt om vragen te stellen aan het vaksteunpunt als ze er niet uitkomen, dan komen er ook meer vragen van die leerlingen. Op dit vlak zouden ook co-teachers zonder vakinhoudelijke kennis een rol kunnen spelen.

Het kan dus zijn dat leerlingen onvoldoende zelfvertrouwen of noodzaak hebben om een fout op te lossen, of een gebrek hebben aan domeinkennis, ervaring of strategische kennis (om daarmee systeemkennis te kunnen vergaren). Als leerlingen een gebrek daaraan zelf op moeten lossen, hebben ze ook metacognitieve kennis nodig.

Hoofdstuk 5

Ontwerp en onderbouwing

Op basis van het vooronderzoek introduceren we in dit hoofdstuk een voorstel voor een ontwerp dat leerlingen beter zou kunnen ondersteunen bij het zelfstandig oplossen van programmeerfouten. In sectie 5.1 breiden we eerst het theoretisch kader verder uit op basis van de thema's die in het vooronderzoek naar voren zijn gekomen. Sectie 5.2 introduceert vervolgens het ontwerp. Het uitgewerkte ontwerpvoorstel is te vinden in appendix C en online[1].

5.1 Theoretische achtergrond

Uit de interviews met vakdocenten blijkt dat er twee belangrijke voorwaarden zijn voor het zelfstandig op kunnen lossen van programmeerfouten, buiten de kennis die daar ook voor is vereist: zelfvertrouwen en noodzaak. Leerlingen moeten er vertrouwen in hebben dat ze een fout op kunnen lossen, anders is er een kans dat ze het al bij voorbaat opgeven. Uit onderzoek blijkt ook dat zelfvertrouwen een belangrijke voorspeller is van succes (Lishinski et al., 2016; Sharmin et al., 2019). In een les met een vakdocent kunnen leerlingen makkelijk om hulp vragen en de vakdocenten zien dat leerlingen daar ook veel gebruik van maken. Er is dan dus niet de noodzaak om het zelf op te lossen. Dit effect wordt ook in ander onderzoek gevonden (Ko et al., 2019).

De noodzaak is bij Co-Teach al groter, inherent aan de situatie: er is geen vakdocent die hulp kan bieden en een antwoord van een vaksteunpunt kan even op zich laten wachten. In het lesmateriaal is het dus belangrijk dat we leerlingen helpen bij het krijgen van zelfvertrouwen. In de literatuur wordt dan gesproken van *self-efficacy*, dat vaak wordt omschreven als zelfvertrouwen gericht op een specifieke taak. Er worden in het algemeen vier "bronnen" voor self-efficacy onderscheiden (Artino, 2012):

- Actieve ervaringen van meesterschap: succeservaringen zorgen ervoor dat men meer vertrouwen krijgt. Andersom werkt het ook: herhaald falen verlaagt het vertrouwen.
- Plaatsvervangende ervaringen: observatie van de successen en mislukkingen van anderen.
- Vormen van overreding, verbaal of anderszins.
- Fysieke en affectieve toestand: de ervaring van stress bij het uitvoeren van een taak leidt tot minder vertrouwen.

De eerste van deze vier is verreweg de meest effectieve, waarschijnlijk omdat het de meest directe manier is (Artino, 2012; Lishinski et al., 2016). Hushman & Marley (2015) tonen aan dat leerlingen die de stof tot zich krijgen via geleide instructie daarna een hogere self-efficacy rapporteren dan leerlingen die direct of minimaal zijn geïnstrueerd. Bij de geleide instructie

[1] Zie https://arthurrump.github.io/HelpEenError/.

stelde de docent leidende vragen, maar moesten de leerlingen zelf de juiste uitleg geven. Daarmee is het een tussenvorm tussen de directe instructie, waarin de docent uitleg gaf met uitgewerkte voorbeelden, en minimale instructie, waarbij de leerlingen helemaal vrij werden gelaten om de stof zelf te ontdekken.

Er zijn diverse onderzoeken gedaan naar het bevorderen van self-efficacy bij beginnend programmeurs: Zingaro (2014) laten bijvoorbeeld zien dat het gebruik van *peer instruction* een positieve impact heeft op de self-efficacy van studenten in CS1. Peer instruction heeft in dit onderzoek een vorm die veel lijkt op denken-delen-uitwisselen. Verder vinden Sharmin et al. (2019) een klein, niet significant, effect door het gebruik van meer open opdrachten, maar ze vermoeden dat het op grotere schaal wel significant zou kunnen zijn; tonen Lishinski & Yadav (2021) aan dat regelmatige zelfreflectie een positief effect op self-efficacy heeft; en suggereren Chapin & Bowen (2023) dat leerlingen programmeeroefeningen eerst op een whiteboard laten uitwerken een positief effect heeft op self-efficacy. Afgezien van de meer open opdrachten (maar die hadden geen significant effect) zijn deze interventies echter niet in lesmateriaal voor de context van Co-Teach toe te passen.

Naast noodzaak en zelfvertrouwen hebben leerlingen natuurlijk ook kennis nodig om fouten op te lossen: zowel vakdocenten als de vaksteunpuntassistenten helpen leerlingen bijvoorbeeld met het begrijpen van een foutmelding (domeinkennis) en bij het doorlopen van een debugstrategie om een logische fout op te lossen (strategische kennis). Vooral dat laatste blijkt lastig aan te leren. Ko et al. (2019) en Whalley et al. (2021) laten bijvoorbeeld zien dat het expliciet aanleren van strategieën voor programmeren en debuggen moeizaam verloopt. Leerlingen zien vaak wel het voordeel van een strategie, maar niet genoeg om die ook echt te gebruiken. De meesten (bijna de helft) geven de voorkeur aan snel kleine veranderingen proberen dan de meer gestructureerde debugstrategie die de onderzoekers hen hebben aangereikt.

5.2 Ontwerpvoorstel

Om het zelfvertrouwen bij leerlingen te bevorderen is het dus een goed idee om leerlingen succeservaringen te bieden met het oplossen van programmeerfouten. Dat betekent dat leerlingen fouten moeten oplossen en dat de oefeningen waarin ze dat doen makkelijk genoeg moeten zijn om een succeservaring vrijwel te garanderen, anders is er een risico dat dit averechts werkt. Een tweede eis voor het ontwerp is dat we zo min mogelijk willen aanpassen in het lesmateriaal. Voor het onderzoek is dat nuttig om beter het effect van een enkele interventie in te schatten en voor de praktijk is het haalbaarder om de interventie daadwerkelijk toe te passen.

Het ontwerpvoorstel is een hoofdstuk over if-statements, dat dezelfde lijn hanteert als Fundament Informatica. De teksten en oefeningen zijn origineel, maar inhoudelijk is dit ontwerp vergelijkbaar: we introduceren if-statements met een aantal voorbeelden en beschrijven de syntax. Vervolgens voegen we daar de `else` constructie aan toe en is er een oefening in het schrijven van een programma dat beide concepten gebruikt. Tot slot introduceren we de `elif` constructie met een voorbeeld en volgen er een aantal oefeningen waarin leerlingen programma's schrijven met die constructies.

Op het gebied van syntaxfouten, kunnen we leerlingen succeservaringen meegeven door ze bij elk onderwerp meteen te laten oefenen met het oplossen van fouten die daarbij komen kijken. Bij elk stukje syntax zijn dat bijbehorende `SynaxError`s en `IndentationError`s, bij de introductie van variabelen `NameError`s, bij het verkrijgen van input `TypeError`s, bij lijsten de `IndexError`s etc.

We kiezen ervoor om de errors niet uit te leggen, maar te introduceren met een oefening, zodat leerlingen ook echt de ervaring krijgen met het oplossen van een dergelijke error. Door de oefeningen direct te laten volgen op de introductie van de syntax, verwachten we dat leerlingen

wel in staat zullen zijn de fouten op te lossen, ook als ze de foutmelding niet direct begrijpen, bijvoorbeeld door de syntax te vergelijken met de eerder gegeven voorbeelden. Zie figuur 5.1 voor een voorbeeld van een dergelijke oefening. In het ontwerp hebben we dit soort oefeningen toegevoegd na iedere introductie van een nieuw stukje syntax, dus na de eerste introductie van if-statements, na de introductie van `else` en na de introductie van `elif`.

Om te zorgen dat deze oefeningen tot een succeservaring leiden, mogen ze niet te ingewikkeld zijn. Bij programmeeronderwijs hebben we soms de neiging om bij oefeningen meteen de strikvragen en randgevallen aan te bieden, waardoor voor veel leerlingen de eerste ervaring niet succesvol is (Lishinski et al., 2016). Daarom beginnen we in dit ontwerp de oefening met fouten waarbij de foutmelding de beste uitleg geeft, voordat we de meer onduidelijke foutmeldingen tonen. Indien nodig voegen we ook een hint toe als relevante uitleg in een ander hoofdstuk staat, zoals bij een oefening waarbij `=` en `==` door elkaar gehaald worden.

Oefening

Alice heeft de volgende stukjes code met een if-statement geschreven, maar ze krijgt steeds een foutmelding. Kijk goed naar de voorbeelden hierboven en verbeter haar code:

```
getal = 9
if getal >= 10
    print("Dat getal heeft twee of meer cijfers.")
```

```
  Cell In[4], line 2
    if getal >= 10
                  ^
SyntaxError: expected ':'
```

Figuur 5.1: Voorbeeld van een oefening over syntaxfouten bij een if-statement.

Naast de succeservaringen die de leerlingen hopelijk met deze oefeningen opdoen, krijgen ze ook meteen de domeinkennis over de betekenis van foutmeldingen aangereikt: ze hebben een concreet voorbeeld van wat er fout is bij een bepaalde foutmelding. Doordat ze zelf de fout oplossen, doen ze ook gelijk gerichte ervaring op, een andere belangrijke vorm van kennis bij het oplossen van fouten (zie hoofdstuk 2).

Om leerlingen ook de ervaring te geven met het oplossen van logische fouten hanteren we een vergelijkbare aanpak met oefeningen die hopelijk tot een succeservaring leiden. Ook hier kiezen we ervoor om leerlingen vooral te laten oefenen, in plaats van uit te leggen wat een goede debugstrategie is. Lokale strategieën kunnen met tips en hints in de oefening worden meegegeven, zodat leerlingen die in de praktijk leren gebruiken. Zie figuur 5.2 voor een voorbeeld. In die oefening wordt met een tip een concrete lokale strategie aangerijkt, die bij die oefening nuttig is. In het ontwerp hebben we drie van deze oefeningen toegevoegd: één na de introductie (en de oefeningen met syntaxfouten) van `else` en twee na de introductie van `elif`.

Voor het aanleren van globale proceskennis en debugstrategieën zouden oefeningen met leidende subvragen gebruikt kunnen worden, om leerlingen te sturen in de manier waarop ze het probleem aanpakken. Dat maakt een ingewikkelde oefening makkelijker omdat leerlingen meer richting de oplossing begeleid worden en tegelijkertijd maken ze dan kennis met een gerichte debugstrategie. In het ontwerp hebben we dergelijke oefeningen echter nog niet meegenomen, om de veranderingen ten opzichte van het bestaande materiaal eerst kleiner te houden.

Het volledige ontwerp is te vinden in appendix C.

Oefening

Eve heeft een programma geschreven dat de gebruiker groet afhankelijk van de tijd: goedemorgen tussen 6:00 en 12:00, goedemiddag tussen 12:00 en 18:00, goedenavond tussen 18:00 en 0:00 en goedenacht tussen 0:00 en 6:00. Ze krijgt geen foutmeldingen, maar het programma werkt toch niet helemaal zoals Eve bedoeld had. Om 6:15 print het programma bijvoorbeeld “Goedenavond!”, terwijl het dan natuurlijk ochtend is. Dat gebeurt ook om bijvoorbeeld 12:43 (in de middag) en 0:26 ('s nachts). Verbeter het programma zodat het wel voldoet aan de eisen.

Tip: vervang de regel `uur = datetime.now().hour` door bijvoorbeeld `uur = 6`, om te kijken welke groet geprint wordt om 6 uur 's ochtends.

```
from datetime import datetime

uur = datetime.now().hour

if uur > 0 and uur < 6:
    print("Goedenacht!")
elif uur > 6 and uur < 12:
    print("Goedemorgen!")
elif uur > 12 and uur < 18:
    print("Goedemiddag!")
else:
    print("Goedenavond!")
```

```
Goedemorgen!
```

Hint

Wat is het verschil tussen `>` en `>=` en tussen `<` en `<=`?

Figuur 5.2: Voorbeeld van een oefening waarin de leerling een logische fout moet oplossen.

Hoofdstuk 6

Resultaten evaluatie

Met het ontwerp uit het vorige hoofdstuk hebben we een mogelijk antwoord gegeven op de derde deelvraag: *wat kunnen we in het lesmateriaal verbeteren om leerlingen meer fouten zelfstandig op te laten lossen?* Om te evalueren of het een goed antwoord op die vraag is, hebben we het ontwerp voorgelegd aan twee ervaren vakdocenten in een expertreview, zoals beschreven in sectie 3.3.5. In dit hoofdstuk bespreken we de resultaten van die expertreview en geven we een antwoord op die derde deelvraag. Een volledige thematische codering van beide interviews is te vinden in appendix B.3.

6.1 Expertreview

6.1.1 Helpt dit materiaal om het zelfvertrouwen van leerlingen in het oplossen van programmeerfouten te vergroten?

Tabel 6.1: Helpen de oefeningen met syntaxfouten om het zelfvertrouwen van leerlingen te vergroten?

Docent 1	Docent 2
Deze oefening werkt, ze gaan hier iets van leren	Helpt met herkenning, leerlingen hebben fouten in ieder geval eerder gezien
Overzichtelijk en beperkt, daarmee grote kans op succes	Gaat ze waarschijnlijk lukken
Vergt een besef dat details belangrijk zijn	

Tabel 6.2: Helpen de oefeningen met logische fouten om het zelfvertrouwen van leerlingen te vergroten?

Docent 1	Docent 2
Het is een begrijpend lezen oefening	Het is meer begrijpend lezen dan programmeren
Extreme resultaten: sommigen zien het meteen, anderen helemaal niet	

Docent 1	Docent 2
Mist iets constructiefs: te weinig opbouw, te ingewikkeld met een te simpel antwoord	Te grote stap, te weinig handvatten

Over de oefeningen met syntaxfouten zijn beide vakdocenten positief: ze verwachten dat de meeste leerlingen wel in staat zijn om deze oefening succesvol te volbrengen en dat ze er iets van leren. Een van de docenten merkte wel op dat het van leerlingen vereist dat ze beseffen hoe belangrijk de kleine details zijn, dus daar moeten ze wel op voorbereid worden.

De oefeningen met logische fouten gingen volgens beide docenten te snel: de stap van de syntaxfouten naar een logische fout is groot en leerlingen zouden hier meer begeleiding moeten krijgen. Met name bij de eerste van deze oefeningen mist iets constructiefs en in zekere zin is het antwoord te simpel voor hoe ingewikkeld de opdracht is en kan het leiden tot een oppervlakkige aanpak: in een klassikale setting roept één leerling het juiste antwoord (`!=` moet `==` zijn) en voor de rest is het leereffect weg. Een van de docenten merkte ook op dat vooral leerlingen die minder scherp zijn bij deze oefening lang vast kunnen zitten, wat dus niet helpt bij het bevorderen van zelfvertrouwen, juist voor de groep die het nodig heeft. Beide docenten merken ook op dat de oefening ook veel met begrijpend lezen te maken heeft, volgens één zelfs meer dan met programmeren.

6.1.2 Welke kennis missen leerlingen nog om een fout op te lossen op basis van dit materiaal?

Tabel 6.3: Welke kennis missen leerlingen nog om een fout op te lossen op basis van dit materiaal?

Docent 1	Docent 2
Lijn naar de echte wereld: deze syntaxoefeningen zijn invuloefeningen, de volgende stap ontbreekt	
Begrijpend code lezen	
Concrete globale probleemoplossingsstrategieën	
	Lokale debugstrategieën, bijvoorbeeld letten op de keuzepunten bij het debuggen van een logische fout
	Opmaak van code voor betere leesbaarheid

Over de kennis die leerlingen niet voldoende wordt aangereikt met dit materiaal zijn beide docenten het op een belangrijk punt eens: strategische kennis. De één benoemt het ontbreken van een globale probleemoplossingsstrategie (en noemt daarbij ook dat die wel concreet moeten zijn, anders haken leerlingen af), de ander komt met een suggestie voor lokale debugstrategieën die leerlingen bij bepaalde oefeningen kunnen helpen. Verder wordt opgemerkt dat de lijn van de oefeningen met syntaxfouten naar het 'echte programmeren' ontbreekt: wat doe je bijvoorbeeld met een foutmelding die je nog niet eerder hebt gezien? Ook "begrijpend code lezen" en nette code schrijven om de leesbaarheid te bevorderen werden genoemd als vaardigheden die leerlingen kunnen helpen bij het oplossen van fouten en waar we leerlingen dus in zouden moeten trainen.

6.1.3 Overige observaties

Tabel 6.4

Docent 1	Docent 2
Een meer technische oplossing: minder syntax-gevoelige taal, bijvoorbeeld Hedy (Hermans, 2020)	
Bij wiskunde hebben we A t/m D, met als belangrijk onderscheid hoe goed een leerling ertegen kan om in een probleem vast te zitten; dat is op programmeren ook van toepassing	
Motivatie bouwt op autonomie, relatie en competentie; relatie is lastig in de Co-Teach context	

Een van de docenten had ook nog een aantal andere observaties, die niet direct bij een van de vragen voor de expertreview passen, maar die wel relevant zijn om te vermelden. Op het gebied van syntaxfouten zijn er ook meer technische oplossingen mogelijk, bijvoorbeeld een programmeertaal die minder gevoelig is voor syntaxfouten. De docent haalt hier Hedy (Hermans, 2020) aan als voorbeeld: voor beginnend programmeurs is de syntax van die taal veel eenvoudiger en is de kans op syntaxfouten dus veel kleiner. Tegelijkertijd gaat de programmeertaal "bijna een socratische dialoog aan" om meer van de syntax te introduceren op moment dat het nodig is.

De docent maakt ook een vergelijking met wiskunde A t/m D, met als belangrijkste verschil hoe lang een leerling er tegen kan om op een probleem vast te zitten. Bij wiskunde B is die tolerantie hoger dan bij wiskunde A. Voor programmeren is dat ook een belangrijk verschil, dat we bijvoorbeeld kunnen hanteren om een duidelijkere progressie te maken van oefeningen waarbij het probleem relatief snel opgelost kan worden naar oefeningen waar de leerling iets langer op vast mag zitten.

Tot slot uitte deze docent hun zorg over een aspect van het Co-Teach programma. Die haalt de zelfdeterminatie theorie aan, met dus autonomie, relatie en competentie als peilers voor motivatie. Met name op het gebied van relatie ziet die de constructie van Co-Teach als mogelijk problematisch. De vaksteunpuntassistenten hebben dat zelf niet zo expliciet genoemd, maar het sluit wel aan op de drempel die zij zien voor leerlingen om vragen te stellen.

6.2 Wat kunnen we in het lesmateriaal verbeteren om leerlingen meer fouten zelfstandig op te laten lossen?

Oefenen met het oplossen van fouten lijkt een goed idee te zijn. Voor syntaxfouten lijken we daar ook een goede vorm voor te hebben gevonden: direct volgend op de introductie van de syntax en opbouwend met de meest duidelijke foutmeldingen als eerste, kunnen leerlingen oefenen met het oplossen van syntaxfouten op een manier die ze waarschijnlijk ook het zelfvertrouwen geeft dat ze zo'n fout ook kunnen oplossen als ze die in het wild tegenkomen. Wel kunnen we nog meer aandacht besteden aan die transitie naar "het wild," om het zelfvertrouwen wat dat betreft nog sterker te bestendigen.

Ook op het gebied van logische fouten is oefening een goed idee, maar is er wel meer sturing en begeleiding nodig: oefeningen waarin de stappen van een globale probleemoplossingsstrategie

verwerkt zijn en meer handvatten in de vorm van lokale strategieën bij specifieke oefeningen zijn hier belangrijk om meer leerlingen een succeservaring te geven die ze nodig hebben. Om ook hier de transitie naar het oplossen van fouten in nieuwe situaties te versterken, kunnen we meer aandacht besteden aan meer algemene vaardigheden: begrijpend code lezen helpt leerlingen bij het oplossen van logische fouten en het hanteren van vuistregels voor de opmaak bij het schrijven van code kan dan weer helpen om het lezen makkelijker te maken. Ook daar kunnen we in oefeningen aandacht aan besteden, om leerlingen die vaardigheden bij te brengen.

Hoofdstuk 7

Conclusie

Hoe kunnen we leerlingen voldoende toerusten om zelfstandig programmeerfouten op te lossen? Dat is de vraag die we in dit onderzoek hebben gepoogd te beantwoorden. Welke fouten zijn dat? Uit interviews met vakdocenten hebben we geleerd dat leerlingen vaak problemen hebben met syntaxfouten. En als ze de syntax onder de knie hebben, dan kunnen ze vast komen te zitten op een logische fout. Bij de vaksteunpunten van Co-Teach zagen ze in vergelijking weinig syntaxfouten: leerlingen lijken ze dus wel zelfstandig op te kunnen lossen als die noodzaak er is, of ze vinden ergens anders de hulp die ze nodig hebben (bijvoorbeeld in ChatGPT). Uit de opdrachten blijkt echter niet altijd dat leerlingen alle fouten weten op te lossen, dus het kan ook zo zijn dat sommige leerlingen zich erbij neerleggen dat het niet lukt.

Wat hebben leerlingen dan nodig om fouten op te lossen? De geïnterviewde vakdocenten zagen in ieder geval twee voorwaarden: noodzaak en zelfvertrouwen, naast natuurlijk de benodigde domeinkennis (bijvoorbeeld de betekenis van foutmeldingen) en strategische kennis (om succesvol het debugproces te doorlopen). De noodzaak is bij Co-Teach al inherent hoger dan bij reguliere informaticalessen. Wat kunnen we dan in het lesmateriaal doen om het zelfvertrouwen van leerlingen in het oplossen van fouten te bevorderen?

Het blijkt dat succeservaringen de beste aanjagers van zelfvertrouwen zijn, dus moeten we zorgen dat leerlingen successen met het oplossen van fouten ervaren als ze het lesmateriaal doorwerken. Dat betekent dat leerlingen moeten oefenen met het oplossen van fouten en dat die oefeningen eenvoudig genoeg moeten zijn om vrijwel alle leerlingen een succeservaring te geven.

We hebben een poging gedaan om die ideeën te verwerken in een paragraaf over if-statements en die voorgelegd aan twee ervaren vakdocenten in een expertreview. Daaruit blijkt dat ons voorstel voor oefeningen waarin leerlingen een syntaxfout krijgen om op te lossen goede kans van slagen heeft. Om te zorgen dat leerlingen de focus hebben op details in de syntax, volgt die oefening direct op de introductie van die syntax voor een bepaalde constructie. De experts verwachten dat leerlingen iets van deze oefeningen leren en waarschijnlijk een succeservaring zullen hebben.

Op het gebied van logische fouten bleken onze oefeningen te weinig houvast te bieden om met grote waarschijnlijkheid tot een succeservaring te komen. In deze oefeningen zouden leerlingen, zeker in eerste instantie, meer bij de hand genomen moeten worden bij het doorlopen van het debugproces en bij het gebruik van lokale debugstrategieën. Daarmee kunnen ze zich ook al doende die strategische kennis eigen maken.

Met die kennis en met meer zelfvertrouwen zouden leerlingen dan beter in staat moeten zijn om zelfstandig programmeerfouten op te lossen.

7.1 Beperkingen

Die conclusie staat, maar er zijn natuurlijk wel enkele beperkingen om rekening mee te houden. Een daarvan is inherent aan het houden van interviews in een één-persoons-onderzoek met beperkte tijd: de hoeveelheid respondenten is klein. Bij de vakdocenten hebben we geprobeerd dat enigszins op te vangen door een diverse groep respondenten te vragen, maar het blijft natuurlijk een kleine groep. Bij de vaksteunpunten is de pool sowieso klein, dus is het fijn dat we in beide series ongeveer de helft hebben kunnen spreken, verspreid over alle regio's om ook daar de diverse invalshoeken mee te pakken.

Het beeld dat de vaksteunpuntassistenten kunnen geven blijft echter beperkt: tijdens de interviews gaven zij regelmatig aan niet altijd een goed beeld te hebben van hoe het er in de klas aan toe gaat en van sommige scholen in het algemeen weinig te horen tot er iets ingeleverd moet worden. Het beeld dat wij kregen uit deze interviews was heel verschillend: de ene vertelde ons dat er heel veel syntaxfouten in ingeleverde opdrachten zaten, terwijl dat bij anderen juist niet het geval was. Dan kan er aan liggen dat er inderdaad grote verschillen zijn tussen regio's, maar het kan ook liggen aan de verschillende persoonlijke perspectieven die de assistenten natuurlijk hebben. Waarschijnlijk speelt beide mee, maar in hoeverre kunnen we hier moeilijk zeggen. Al met al is dus ook ons beeld van de Co-Teach-praktijk beperkt.

Naast de interviews hadden we als instrument voor het verzamelen van fouten die leerlingen maken het logboek. Dit hebben we uiteindelijk alleen kunnen uitvoeren met een kleine klas, maar ook daar bleek al dat het geen bijzonder goed instrument is. De enkele fouten die een paar leerlingen wel hadden opgeschreven waren allemaal foutmeldingen, wat er op zou kunnen duiden dat leerlingen logische programmeerfouten niet met fouten of errors associëren, waardoor ze die niet hebben opgeschreven. Een ander teken dat het instrument weinig succesvol was, is dat slechts één van de negen leerlingen de afsluitende vragen heeft beantwoord en de rest dus al voor het einde van de les vergeten was dat ze geacht werden een logboek bij te houden. Misschien dat het in een andere vorm kan werken, maar in deze vorm is het geen zinvol instrument.

Bij de evaluatie hebben we alleen gebruik gemaakt van een expertreview om een eerste indicatie te geven. Hier gelden dezelfde beperkingen als bij de interviews, met als toevoeging dat beide respondenten op dezelfde school lesgeven en met hetzelfde lesmateriaal ervaring hebben. Verder is het ontwerp dus niet daadwerkelijk getest met leerlingen en zijn onze conclusies over de invloed op het zelfvertrouwen van leerlingen en hun vaardigheden in het oplossen van fouten daarmee niet meer dan hypotheses om in vervolgonderzoek daadwerkelijk te testen.

Tot slot heeft dit onderzoek een belangrijke beperking in de doelstelling: is zelfstandigheid de heilige graal? Nee, het is natuurlijk ook van belang dat leerlingen hulp kunnen krijgen als ze die nodig hebben, hoe goed het lesmateriaal ze ook ondersteunt. Hoewel het eigenlijk geen deel van de onderzoeksvraag was, hebben we op dit gebied wel enige resultaten gekregen uit de interviews, met als belangrijkste conclusie: de drempel om hulp te krijgen lijkt nu bij Co-Teach nog te hoog te liggen voor veel leerlingen. Meerdere vaksteunpuntassistenten merkten dat als leerlingen eenmaal contact hebben gehad, de drempel lager wordt en ze sneller nog een vraag durven te stellen, dus die drempel lijkt voort te komen uit een gebrek aan verbinding. Die zou al kunnen worden bevorderd door leerlingen meer aan te moedigen contact op te nemen en zo zelf die verbinding te leggen. De co-teachers op school spelen daar een belangrijke rol in, maar ook vanuit het platform kan het vaker benoemd worden. Een vaksteunpuntassistent geeft als suggestie: "Misschien ook een paar keer in het platform zelf gewoon neerzetten: 'Kom je er niet uit? Vraag het!' " Het lijkt mij goed om ook verder te onderzoeken hoe de verbinding tussen leerlingen en het vaksteunpunt al op voorhand versterkt kan worden, zonder dat de eerste stap bij de leerling ligt, zodat de drempel om hulp te vragen lager wordt.

Naast de ondersteuning door het vaksteunpunt, kan de begeleiding van een co-teacher zonder

vakkennis in het leerproces ook veel ondersteuning bieden. Een van de taken die de geïnterviewde vakdocenten voor zichzelf zien is het achterhalen wat een leerling nodig heeft om verder te komen. Dat is lang niet altijd een probleem met vakkennis en zelfs dan zou een co-teacher wel kunnen helpen bij het formuleren van een vraag om naar het vaksteunpunt te sturen. Ik weet niet in hoeverre dat op dit moment de praktijk is (en waarschijnlijk verschilt dat per school), maar het lijkt mij een belangrijk aspect om ook in te investeren.

Leren programmeren is lastig genoeg, dus uiteindelijk hebben we alles nodig: goed lesmateriaal én goede menselijke hulp.

Referenties

Artino, A. R., Jr. (2012). Academic self-efficacy: from educational theory to instructional practice. *Perspectives on Medical Education*, *1*(2), 76–85. https://doi.org/10.1007/s40037-012-0012-5

Becker, B. A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D. J., Harrington, B., Kamil, A., Karkare, A., McDonald, C., Osera, P.-M., Pearce, J. L., & Prather, J. (2019, december 18). Compiler Error Messages Considered Unhelpful. *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*. https://doi.org/10.1145/3344429.3372508

Chapin, J., & Bowen, B. (2023, december). Whiteboarding: A Tool to Improve CS1 Student Self-Efficacy. *Proceedings of the ACM Conference on Global Computing Education Vol 1*. https://doi.org/10.1145/3576882.3617925

Hartz, A. J. (2012). *CAT-SOOP: A Tool for Automatic Collection and Assessment of Homework Exercises* [Mathesis, Massachusetts Institute of Technology]. https://dspace.mit.edu/bitstream/handle/1721.1/77086/825763362-MIT.pdf

Hermans, F. (2020, augustus). Hedy: A Gradual Language for Programming Education. *Proceedings of the 2020 ACM Conference on International Computing Education Research*. https://doi.org/10.1145/3372782.3406262

Hermans, F. (2021). *The Programmer's Brain: What Every Programmer Needs to Know about Cognition*. Manning.

Hushman, C. J., & Marley, S. C. (2015). Guided Instruction Improves Elementary Student Learning and Self-Efficacy in Science. *The Journal of Educational Research*, *108*(5), 371–381. https://doi.org/10.1080/00220671.2014.899958

Ko, A. J., LaToza, T. D., Hull, S., Ko, E. A., Kwok, W., Quichocho, J., Akkaraju, H., & Pandit, R. (2019, februari). Teaching Explicit Programming Strategies to Adolescents. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3287324.3287371

Li, C., Chan, E., Denny, P., Luxton-Reilly, A., & Tempero, E. (2019). Towards a Framework for Teaching Debugging. *Proceedings of the Twenty-First Australasian Computing Education Conference on - ACE '19*. https://doi.org/10.1145/3286960.3286970

Lishinski, A., & Yadav, A. (2021). Self-evaluation Interventions: Impact on Self-efficacy and Performance in Introductory Programming. *ACM Transactions on Computing Education*, *21*(3), 1–28. https://doi.org/10.1145/3447378

Lishinski, A., Yadav, A., Good, J., & Enbody, R. (2016, augustus). Learning to Program: Gender Differences and Interactive Effects of Students' Motivation, Goals, and Self-Efficacy on Performance. *Proceedings of the 2016 ACM Conference on International Computing Education Research*. https://doi.org/10.1145/2960310.2960329

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: a review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67–92. https://doi.org/10.1080/08993400802114581

Michaeli, T., & Romeike, R. (2019, april). Current Status and Perspectives of Debugging in the

K12 Classroom: A Qualitative Study. *2019 IEEE Global Engineering Education Conference (EDUCON)*. https://doi.org/10.1109/educon.2019.8725282

Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: the good, the bad, and the quirky - a qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin*, *40*(1), 163–167. https://doi.org/10.1145/1352322.1352191

Perscheid, M., Siegmund, B., Taeumel, M., & Hirschfeld, R. (2016). Studying the advancement in debugging practice of professional software developers. *Software Quality Journal*, *25*(1), 83–110. https://doi.org/10.1007/s11219-015-9294-2

Sharmin, S., Zingaro, D., Zhang, L., & Brett, C. (2019, mei). Impact of Open-Ended Assignments on Student Self-Efficacy in CS1. *Proceedings of the ACM Conference on Global Computing Education*. https://doi.org/10.1145/3300115.3309532

Smith, R., & Rixner, S. (2019, februari). The Error Landscape. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3287324.3287394

Sorva, J. (2012). *Visual program simulation in introductory programming education* [Phdthesis, Aalto University; Aalto Univ. School of Science]. https://aaltodoc.aalto.fi/bitstream/handle/123456789/3534/isbn9789526046266.pdf

Spinellis, D. (2018). Modern debugging. *Communications of the ACM*, *61*(11), 124–134. https://doi.org/10.1145/3186278

Van der Donk, C., & Van Lanen, B. (2020). *Praktijkonderzoek in de school*. Coutinho.

Whalley, J., Settle, A., & Luxton-Reilly, A. (2021, februari). Analysis of a Process for Introductory Debugging. *Australasian Computing Education Conference*. https://doi.org/10.1145/3441636.3442300

Xia, X., Bao, L., Lo, D., Kochhar, P. S., Hassan, A. E., & Xing, Z. (2017). What do developers search for on the web? *Empirical Software Engineering*, *22*(6), 3149–3185. https://doi.org/10.1007/s10664-017-9514-4

Yen, C.-Z., Wu, P.-H., & Lin, C.-F. (2012). Analysis of Experts' and Novices' Thinking Process in Program Debugging. In *Communications in Computer and Information Science* (pp. 122–134). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-31398-1_12

Zingaro, D. (2014, maart). Peer instruction contributes to self-efficacy in CS1. *Proceedings of the 45th ACM technical symposium on Computer science education*. https://doi.org/10.1145/2538862.2538878

Bijlage A

Instrumenten

A.1 Interviewleidraad vakdocenten

Het interview bestaat uit drie delen, waarbij alle inhoudelijke vragen binnen de kern vallen (Van der Donk & Van Lanen, 2020). Tekst die de interviewer als uitleg geeft staat *schuingedrukt*, de te stellen vragen staan in een normaal font. Eventuele verduidelijkingen staan [tussen blokhaken]. De vragen staan in geneste lijsten: de vragen die onder een andere vraag vallen zijn vervolgvragen op de antwoorden van de bovenliggende vraag. Naarmate de inspringing groter is, is er meer vrijheid om in te spelen op de gegeven antwoorden, maar de vragen geven wel een indicatie van de richting die het gesprek moet krijgen.

Inleiding

Intro

- *Bedankt dat je mee wilt doen aan dit interview.*
- *Ik wil je graag wat vragen stellen over de fouten die leerlingen in hun programma's maken bij het programmeren.*
- *Dit is voor een onderzoek voor Co-Teach, waarbij informatica op scholen wordt aangeboden die geen vakdocent voor informatica hebben. Voor programmeren gebruiken ze online materiaal en filmpjes.*
- *We weten dat het oplossen van fouten over het algemeen in één-op-één hulp door een vakdocent gebeurt, wat in dit geval dus niet mogelijk is.*
- *Het doel is om leerlingen toe te rusten om meer fouten zelfstandig op te kunnen lossen.*
- *Daarvoor wil ik eerst weten waar leerlingen met een vergelijkbaar curriculum tegenaan lopen, dus daar wil ik het nu met jou over hebben.*
- Heb je nog vragen over het onderzoek?

Organisatorisch

- *Je hebt een informatieblad gekregen met informatie over het onderzoek, ik zal nog even kort de belangrijke punten aanstippen:*
- *Ik verwacht dat we ongeveer 10 tot 15 minuten nodig hebben voor het hele interview.*
- *Je mag op elk moment stoppen met het interview, of ervoor kiezen om een vraag niet te beantwoorden. Als je tijdens het onderzoek stopt met je deelname, worden alle gegevens die we hebben verzameld verwijderd. Je bent tot niets verplicht.*
- *Ik wil dit interview graag opnemen. De opname wordt uitgewerkt tot een transcript voor gebruik in dit onderzoek en wordt aan het eind van het onderzoek vernietigd.*
- Heb je hier nog vragen over?

- *Dan start ik de opname, en mag je nog even officieel toestemming geven.*

Toestemming

- Ben je voldoende geïnformeerd over het onderzoek door middel van een separaat informatieblad? Heb je het informatieblad gelezen en de mogelijkheid gehad om vragen te stellen? Zijn die vragen voldoende beantwoord?
- Neem je vrijwillig deel aan dit onderzoek? Is er geen expliciete of impliciete dwang voor jou om aan dit onderzoek deel te nemen? Je hoeft een vraag niet te beantwoorden als je dat niet wil. Is het duidelijk dat je deelname aan het onderzoek op elk moment, zonder opgaaf van reden, kan beëindigen?
- Geef je toestemming om de gegevens die gedurende het onderzoek bij jou worden verzameld te verwerken zoals is opgenomen in het informatieblad?
- Geef je toestemming om tijdens het interview opnames te maken en je antwoorden uit te werken in een transcript?
- Geef je toestemming om je antwoorden te gebruiken voor (anonieme) quotes in de onderzoekspublicaties? (*Niet verplicht.*)

Kern

Algemeen

- *In dit onderzoek richten we ons op leerlingen in klas 4,* in welke klas ga jij over het algemeen aan de slag met programmeren?
- Welke programmeertaal gebruik je daarbij?
- Hoeveel tijd besteed je aan het onderwerp? Hoe diep ga je op de stof in?

We gaan het dus hebben over de fouten die leerlingen maken in hun programma's. Dat kunnen fouten zijn waardoor er een foutmelding verschijnt, maar ook fouten waardoor het programma iets anders doet dan de leerling verwacht had. We zijn vooral geïnteresseerd in de fouten waardoor leerlingen vastlopen, en waarvan ze zelf dus ook al zien dat de fout bestaat. Het kan ook zijn dat een leerling de opdracht niet begrepen heeft, en daardoor een programma schrijft dat niet doet wat de opdracht bedoeld had, maar dat valt niet onder de fouten waar we in dit onderzoek naar kijken.

Over fouten

- *Scenario: de leerlingen zijn aan het werk met een programmeeropdracht, jij loopt rond om te helpen. Een leerling steekt hun hand op (of je spreekt ze aan omdat ze met de handen in het haar naar hun scherm staren).*
- Welke fouten zie je dan als je naar het programma van de leerling kijkt?
  - Kun je een voorbeeld geven?
    - Hoe help je die leerling om de fout op te lossen?
    - Waarom lukt het de leerling niet om de fout zelf op te lossen? Welke kennis of vaardigheden missen ze?
- Zie je vaak dezelfde soort fouten?
  - Wat zijn andere fouten die je veel ziet?
    - Hoe help je leerlingen om deze fouten op te lossen?
  - Is de hulp die je biedt verschillend voor verschillende fouten, of zit er veel overlap in?
    - Wat is die overlap? Waar zitten de verschillen?
- Hoe vaak doet dit scenario zich voor: lopen leerlingen vaak vast op een fout in hun programma?
- Besteed je klassikaal aandacht aan het oplossen van fouten?

  - Zo ja: wat houdt dat in jouw geval concreet in? Welk doel wil je daarmee bereiken?

Slot

- Wil je nog iets kwijt dat we niet besproken hebben?
- *Om je een idee te geven wat we nu verder gaan doen in dit onderzoek: ik ga zometeen eerst dit interview uitschrijven en met de andere interviews samen kijken of we tot een overzicht van fouten en oplossingen komen. Dan ga ik leerlingen vragen om de fouten die ze tegenkomen bij te houden en uiteindelijk een ontwerp maken van lesmateriaal om leerlingen te ondersteunen bij het oplossen van fouten.*
- Zou je mee willen doen aan een expertreview van dat ontwerp? *Je zit nergens aan vast, dus voel je niet verplicht als het tegen die tijd toch niet uitkomt.*
- *Bedankt!*

A.2 Interviewleidraad vaksteunpunten

De structuur van dit interview is vergelijkbaar met het interview met de vakdocenten. De vragen gemarkeerd met (2) zijn toegevoegd in de tweede serie interviews in 2024, op basis van de resultaten uit de eerste serie in 2022.

Inleiding

Intro

- *Bedankt dat je mee wilt doen aan dit interview.*
- *Dit onderzoek gaat over de fouten die leerlingen maken bij het programmeren.*
- *We weten dat het oplossen van fouten over het algemeen in één-op-één hulp door een vakdocent gebeurd, wat in het geval van Co-Teach natuurlijk niet in de klas mogelijk is.*
- *Het doel is om leerlingen toe te rusten om meer fouten zelfstandig op te kunnen lossen.*
- *Daarvoor wil ik weten op welke fouten leerlingen nu vastlopen.*
- *Jullie zijn een aanspreekpunt voor leerlingen als ze vragen hebben, dus wil ik het in dit interview met je hebben over de fouten die je leerlingen ziet maken bij de opdrachten en de vragen die ze stellen.*
- Heb je nog vragen over het onderzoek?

Organisatorisch

- *Je hebt een informatieblad gekregen met informatie over het onderzoek, ik zal nog even kort de belangrijke punten aanstippen:*
- *Ik verwacht dat we ongeveer 20 minuten nodig hebben voor het hele interview.*
- *Je mag op elk moment stoppen met het interview, of ervoor kiezen om een vraag niet te beantwoorden. Als je tijdens het onderzoek stopt met je deelname, worden alle gegevens die we hebben verzameld verwijderd. Je bent tot niets verplicht.*
- *Ik wil dit interview graag opnemen. De opname wordt uitgewerkt tot een transcript voor gebruik in dit onderzoek en wordt aan het eind van het onderzoek vernietigd.*
- Heb je hier nog vragen over?
- *Dan start ik de opname, en mag je nog even officieel toestemming geven.*

Toestemming

- Ben je voldoende geïnformeerd over het onderzoek door middel van een separaat informatieblad? Heb je het informatieblad gelezen en de mogelijkheid gehad om vragen te stellen? Zijn die vragen voldoende beantwoord?

- Neem je vrijwillig deel aan dit onderzoek? Is er geen expliciete of impliciete dwang voor jou om aan dit onderzoek deel te nemen? Je hoeft een vraag niet te beantwoorden als je dat niet wil. Is het duidelijk dat je deelname aan het onderzoek op elk moment, zonder opgaaf van reden, kan beëindigen?
- Geef je toestemming om de gegevens die gedurende het onderzoek bij jou worden verzameld te verwerken zoals is opgenomen in het informatieblad?
- Geef je toestemming om tijdens het interview opnames te maken en je antwoorden uit te werken in een transcript?
- Geef je toestemming om je antwoorden te gebruiken voor (anonieme) quotes in de onderzoekspublicaties? (*Niet verplicht.*)

Kern

Algemeen

- Hoeveel leerlingen vallen er ongeveer in jouw regio?
  - Hoeveel klassen?
- Waar zitten de scholen in jouw regio in de planning wat betreft programmeren? Zijn er scholen die het onderdeel nog niet afgerond hebben?
- (2) Wat weet je over de begeleiding op die scholen? Is de co-teacher in staat om met sommige inhoudelijke vragen te helpen?
- (2) Hoe vaak hebben jullie in het algemeen contact met leerlingen? Hoe snel kunnen jullie de leerlingen antwoord geven?

Vragen van leerlingen

- Hoeveel vragen schat je in dat jullie per les hebben gekregen?
  - [Eventueel]: Meerdere per les, ongeveer één per les, minder?
  - Krijg je van alle leerlingen vragen, of is het een kleine groep die vragen stelt? Hoe schat je de verdeling in?
    - Hoeveel leerlingen hebben minstens één keer een vraag gesteld?
  - Zit er verschil in de onderwerpen qua hoeveelheid vragen?
    - *Onderwerpen zijn variabelen, selectie, iteratie, functies.*

Ik ben dus benieuwd naar de fouten die leerlingen maken. Dat kunnen fouten zijn waardoor er een foutmelding verschijnt, maar ook fouten waardoor het programma iets anders doet dan de leerling verwacht had. We zijn vooral geïnteresseerd in de fouten waardoor leerlingen vastlopen, en waarvan ze zelf dus ook al zien dat de fout bestaat. Het kan ook zijn dat een leerling de opdracht niet begrepen heeft, en daardoor een programma schrijft dat niet doet wat de opdracht bedoeld had, maar dat valt niet onder de fouten waar we in dit onderzoek naar kijken.

- Welk deel van de vragen gaat over een fout waarop de leerling is vastgelopen, dus een foutmelding of een 'hij-doet-het-niet' fout?
  - [Eventueel]: *Ten opzichte van bijvoorbeeld inhoudelijke vragen over de stof, niet over de code die de leerling zelf heeft geschreven.*
  - *Deze vragen zijn voor mijn onderzoek het meest interessant, dus laten we daar nog even verder naar kijken.*

Foutmeldingen

- *[Terugkomen op aandeel vragen over fouten.]* Hoe vaak gaat het bij deze vragen om een foutmelding?
- Welke foutmeldingen zie je het meest terugkomen?

- Hoe help je leerlingen die fouten op te lossen?
  - Kun je een voorbeeld geven van een foutmelding en hoe je de leerling daarmee geholpen hebt?
  - Welke kennis denk je dat er ontbreekt bij de leerlingen, waardoor ze die fout niet zelf op kunnen lossen?

Hij-doet-het-niet fouten

De andere categorie fouten is dat de leerling een programma heeft dat niet doet wat ze verwacht hadden.

- Hoe vaak krijg je dit soort vragen?
- Kun je een voorbeeld geven van een leerling die zo'n soort vraag had?
  - Hoe heb je die leerling geholpen?
- Zie je dat leerlingen vaak dezelfde soort fouten maken? Zijn er een paar categorieën die vaak terugkomen?
  - Hoe help je leerlingen met deze soorten fouten?
  - Waarom lukt het leerlingen niet om deze fouten zelf op te lossen? Welke kennis of vaardigheden missen ze?
    - Is dat verschillend voor de verschillende categorieën?

Dat waren mijn vragen over de vragen die leerlingen stellen. Wil je daar nog iets aan toevoegen?

Fouten in de opdrachten

Jullie kijken ook de opdrachten na die leerlingen ingeleverd hebben, daar heb ik ook nog een aantal vragen over.

Ik neem aan dat niet alle ingeleverde opdrachten helemaal foutloos zijn, maar er is natuurlijk een verschil tussen fouten die de leerling opgemerkt heeft, maar waarvan ze niet weten hoe die opgelost kan worden, en fouten die ze zelf niet tegen waren gekomen. Die laatste vind ik niet zo interessant, dus als ze bijvoorbeeld de opdracht niet begrepen of niet helemaal uitgevoerd hebben.

- Zie je fouten in de ingeleverde programma's waarvan je verwacht dat leerlingen die zelf ook opgemerkt hebben?
  - Kun je een voorbeeld geven?
- Waar ligt het aan dat de leerlingen deze fouten niet opgelost hebben?

Slot

- Wil je nog iets kwijt dat we niet besproken hebben?
- *Om je een idee te geven wat we nu verder gaan doen in dit onderzoek: ik ga zometeen eerst dit interview uitschrijven en met de andere interviews samen kijken of we tot een overzicht van fouten en oplossingen komen. Dan ga ik ook kijken of ik een paar leerlingen een logboek van fouten en uiteindelijk een ontwerp maken van lesmateriaal om leerlingen te ondersteunen bij het oplossen van fouten. Dat zou dan uiteindelijk in het materiaal van Co-Teach verwerkt kunnen worden.*
- *Bedankt!*

A.3 Logboek voor leerlingen

Het logboek wordt door leerlingen online ingevuld, op een speciaal voor dit doel ontwikkelde website.[1] Zodra een leerling een vraag heeft beantwoord, wordt het antwoord opgeslagen op de server. De vragen zijn gekoppeld aan een willekeurig gegenereerde code, die is opgeslagen op de computer van de leerling. Drie uur na het starten van het logboek wordt deze code weer verwijderd, en zijn de gegevens dus niet meer aan de leerling te linken. Deelnemers kunnen op elk moment terug naar een vorige pagina van de website om hun antwoorden aan te passen of alle gegevens te verwijderen. Zodra de gegeven toestemming voor het verwerken van de gegevens wordt ingetrokken, worden alle gegevens van de server en van de computer van de leerling verwijderd.

Hieronder staat een tekstuele weergave van de vragen die aan leerlingen gesteld worden. Verduidelijkingen staan *schuingedrukt* weergegeven.

Welkom

Iedereen maakt fouten. Dat is goed, want van fouten leer je. Maar zoals je misschien wel gemerkt hebt, kan een klein foutje bij het programmeren ervoor zorgen dat je helemaal vastloopt. Normaal gesproken zou je tijdens een les informatica dan je docent om hulp vragen, die je weer op weg helpt, maar er zijn ook scholen die informatica geven zonder dat ze een informaticadocent hebben.

Daarom onderzoeken wij hoe we die leerlingen beter kunnen helpen met het oplossen van de fouten die ze maken bij het programmeren, ook zonder dat daar een docent bij nodig is. We willen weten op welke fouten jullie vastlopen en dus vragen we jullie om gedurende één les in dit logboek bij te houden welke programmeerfouten jullie tegenkomen. Je docent zal tijdens deze les geen vragen over programmeren beantwoorden, omdat we willen onderzoeken hoe jullie zelfstandig met fouten omgaan.

Het onderzoek wordt uitgevoerd door Arthur Rump van de faculteit Behavioural, Management and Social Sciences op de Universiteit Twente. Als je vragen hebt, kun je die stellen via a.h.j.rump@student.utwente.nl.

Dit moet je weten over dit logboek:

- Alle gegevens worden anoniem verwerkt en alleen gebruikt voor dit onderzoek.
- Vanaf het starten van het logboek blijven de gegevens maximaal drie uur lang aan jouw computer gekoppeld, of totdat je het logboek afsluit. Daarna worden je antwoorden volledig geanonimiseerd en kunnen ze dus niet meer aan jou gelinkt worden.
  - Tot die tijd kun je terugkomen naar deze website om aan te passen wat je hebt ingevuld, of om je toestemming in te trekken en alle gegevens te verwijderen. Dat kun je doen door terug te komen naar deze pagina, het vinkje hieronder weer uit te schakelen en op de knop ‘Opslaan’ te klikken.
- Meedoen is volledig vrijwillig, je bent tot niets verplicht.
- Je kunt op elk moment stoppen met het invullen van het logboek.

Als je mee wilt doen aan het onderzoek, en hiermee akkoord gaat, zet dan het vinkje hieronder aan en ga verder naar het logboek.

Onderaan staan een checkbox met de tekst Ik ga akkoord *en daaronder een knop* Verder*. Als de leerling de checkbox aanvinkt en op die knop klikt, wordt de volgende pagina getoond.*

[1] De code van deze website is beschikbaar via https://github.com/arthurrump/HelpEenError/tree/main/logboek.

Over jou

Fijn dat je mee wilt doen aan dit onderzoek! Voor we verdergaan, willen we nog een paar dingen van je weten.

Wat is je schoolniveau? *Hierbij zijn er twee keuzemogelijkheden:*

- Havo
- Vwo

Hoe goed ben je in programmeren?

De antwoordmogelijkheden zijn een 5-punts likertschaal van Niet zo goed *tot* Heel goed.

Heb je voor deze lessen al eerder geprogrammeerd?

De antwoordmogelijkheden zijn een 5-punts likertschaal van Nooit *tot* Heel vaak.

Onderaan staat weer een knop Verder.

Logboek

Op deze pagina kun je bijhouden welke fouten je tegenkomt tijdens deze les. Je hoeft alleen fouten toe te voegen waarvan je niet direct weet wat je ermee moet doen. We kijken naar twee soorten fouten: je kunt een foutmelding krijgen als Python vindt dat er iets niet klopt aan de code (te herkennen aan de rode tekst, bijvoorbeeld in de Console van Replit), of het kan zijn dat je programma wel uitgevoerd wordt, maar niet doet wat de bedoeling was.

Klik aan het eind van de les op de knop *Afsluiten* om de laatste vragen te beantwoorden en het logboek af te sluiten.

Wat gaat er mis? *Hierbij zijn er twee keuzemogelijkheden:*

- Ik krijg een foutmelding
- Het programma werkt niet zoals ik verwacht

Indien de eerste optie wordt gekozen, worden de volgende vragen gesteld, beide te beantwoorden in een groot tekstvak:

Wat is de foutmelding? Kopieer de foutmelding en plak die hier. Gebruik je Replit? Let dan op: selecteer de foutmelding in de Console en gebruik de rechtermuisknop om te kopiëren. De sneltoets Ctrl+C werkt niet in de Console.

In het tekstvak staat een voorbeeld van een foutmelding in Python die de leerling zou kunnen krijgen. De getoonde error wordt willekeurig gekozen uit een set van zes syntax en runtime errors.

Wat ga je doen om de fout op te lossen?

In het tekstvak worden enkele voorbeelden gegeven: Bijvoorbeeld een klasgenoot om hulp vragen, de code nog eens rustig doorlezen, in Fundament opzoeken hoe de code eruit moet zien, ... *Dit is placeholder tekst in het tekstvak. Deze wordt in een grijze kleur weergegeven en verdwijnt zodra er iets in het tekstvak getypt wordt.*

Indien de tweede optie wordt gekozen, verschijnen deze drie vragen, opnieuw te beantwoorden in een groot tekstvak:

Wat verwachtte je dat het programma zou doen?

Wat doet het programma?

Wat ga je doen om de fout op te lossen?

Bij de eerste twee van deze vragen staat geen placeholder tekst, bij de derde staat dezelfde tekst als wanneer er voor de optie Ik krijg een foutmelding *zou zijn gekozen.*

Klik op de knop *Toevoegen* om deze fout toe te voegen aan het logboek. Daarna kun je deze vragen opnieuw beantwoorden bij de volgende fout die je tegenkomt.

Onderaan staat een knop Toevoegen *waarmee de antwoorden worden opgeslagen en een knop* Afsluiten *waarmee het logboek wordt afgesloten en de leerling doorgaat naar het scherm met de afsluitende vragen. Met die eerste verschijnen de antwoorden ook in de lijst van fouten die aan het logboek zijn toegevoegd, zie figuur A.1. Elke fout heeft hier ook een knop om die weer uit het logboek te verwijderen.*

Afsluiting

Tijdens deze les heeft je docent jou niet geholpen bij het oplossen van programmeerfouten en geen vragen beantwoord over programmeren. Je moest dus zelfstandiger aan het werk. Wij zijn benieuwd hoe die zelfstandigheid invloed heeft gehad op de manier waarop jij met fouten bent omgegaan.

Hoeveel van de fouten in het logboek heb je kunnen oplossen?

De antwoordmogelijkheden zijn een 5-punts likertschaal van Geen *tot* Allemaal.

Hoeveel fouten heb je in deze les opgelost in vergelijking met andere lessen?

De antwoordmogelijkheden zijn een 5-punts likertschaal van Veel minder *tot* Veel meer.

Hoeveel tijd heb je in deze les besteed aan het oplossen van fouten in vergelijking met andere lessen?

De antwoordmogelijkheden zijn een 5-punts likertschaal van Veel minder *tot* Veel meer.

Dat waren alle vragen! Klik hieronder op *Opslaan en afsluiten* om je antwoorden op te slaan en het logboek te beëindigen.

Bedankt

Als de leerling op Opslaan en afsluiten *klikt of de website na drie uur nog open staat, verschijnt het bedankscherm. Alle gegevens worden dan uit de browser verwijderd, waardoor de antwoorden niet meer aan die computer te koppelen zijn.*

Bedankt voor je deelname aan dit onderzoek! Je antwoorden zijn nu anoniem opgeslagen en dus niet meer aan jou te koppelen. Als je nog vragen hebt over het onderzoek, kun je contact opnemen met Arthur Rump via a.h.j.rump@student.utwente.nl.

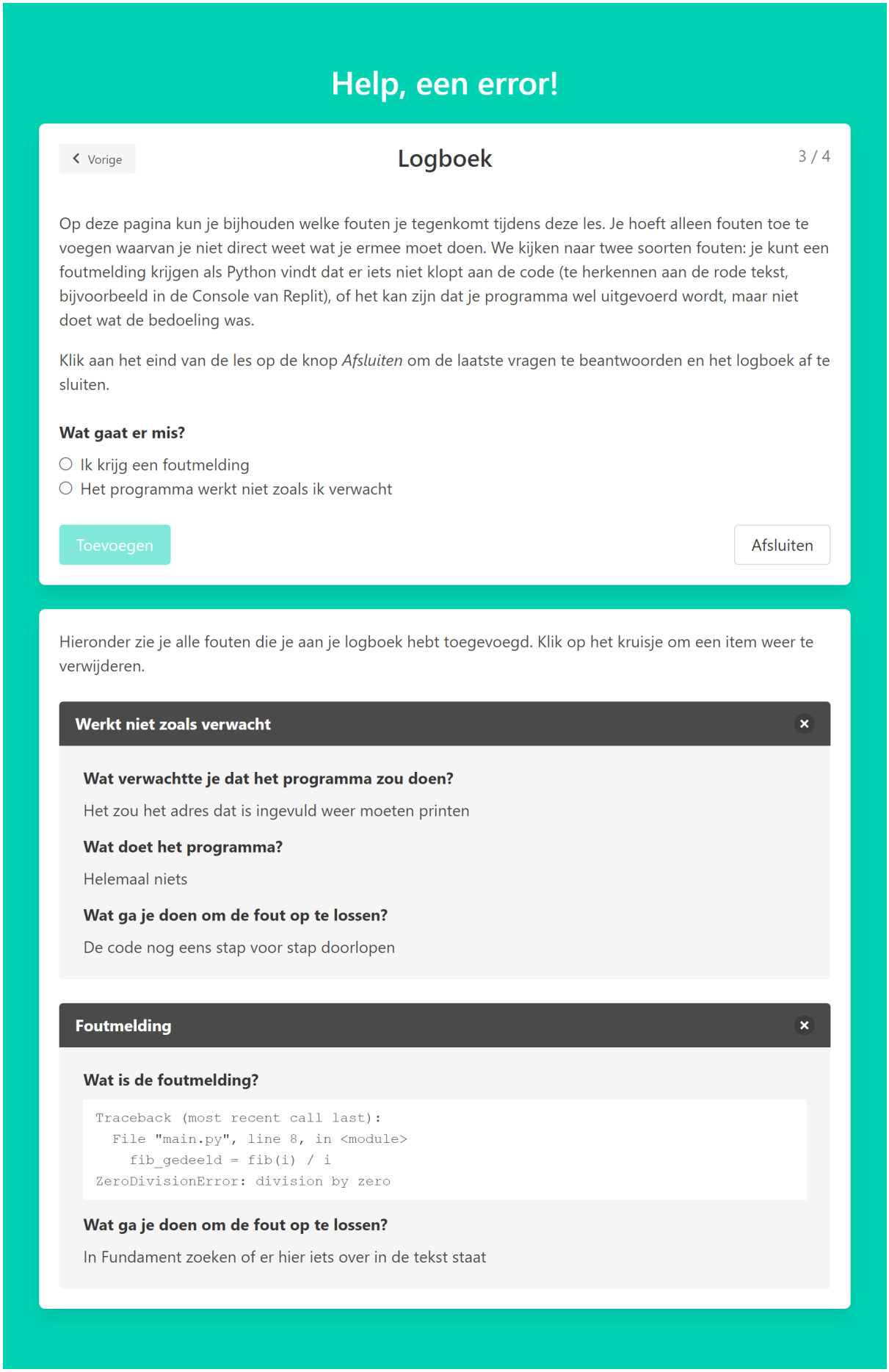
A.4 Interviewleidraad expertreview

De structuur van dit interview is vergelijkbaar met de andere interviews.

Inleiding

Intro

- *Bedankt dat je mee wilt doen aan dit interview.*
- *We gaan het hebben over het zelfstandig oplossen van programmeerfouten door leerlingen.*
- *Dit is voor een onderzoek voor Co-Teach, waarbij informatica op scholen wordt aangeboden die geen vakdocent voor informatica hebben. Voor programmeren gebruiken ze online materiaal en filmpjes.*



Figuur A.1: De fouten die de leerling heeft toegevoegd aan het logboek worden op de pagina weergegeven.

- *We weten dat het oplossen van fouten over het algemeen in één-op-één hulp door een vakdocent gebeurt, wat in dit geval dus niet mogelijk is.*
- *Ik heb een ontwerpvoorstel gemaakt om in het lesmateriaal meer aandacht voor het oplossen van programmeerfouten, daar gaan we het over hebben en ik ben benieuwd hoe jij daar als ervaren vakdocent naar kijkt.*
- Heb je nog vragen over het onderzoek?

Organisatorisch

- *Je hebt een informatieblad gekregen met informatie over het onderzoek, ik zal nog even kort de belangrijke punten aanstippen:*
- *Ik verwacht dat we ongeveer 15 minuten nodig hebben voor het hele interview.*
- *Je mag op elk moment stoppen met het interview, of ervoor kiezen om een vraag niet te beantwoorden. Als je tijdens het onderzoek stopt met je deelname, worden alle gegevens die we hebben verzameld verwijderd. Je bent tot niets verplicht.*
- *Ik wil dit interview graag opnemen. De opname wordt uitgewerkt tot een transcript voor gebruik in dit onderzoek en wordt aan het eind van het onderzoek vernietigd.*
- Heb je hier nog vragen over?
- *Dan start ik de opname, en mag je nog even officieel toestemming geven.*

Toestemming

- Ben je voldoende geïnformeerd over het onderzoek door middel van een separaat informatieblad? Heb je het informatieblad gelezen en de mogelijkheid gehad om vragen te stellen? Zijn die vragen voldoende beantwoord?
- Neem je vrijwillig deel aan dit onderzoek? Is er geen expliciete of impliciete dwang voor jou om aan dit onderzoek deel te nemen? Je hoeft een vraag niet te beantwoorden als je dat niet wil. Is het duidelijk dat je deelname aan het onderzoek op elk moment, zonder opgaaf van reden, kan beëindigen?
- Geef je toestemming om de gegevens die gedurende het onderzoek bij jou worden verzameld te verwerken zoals is opgenomen in het informatieblad?
- Geef je toestemming om tijdens het interview opnames te maken en je antwoorden uit te werken in een transcript?
- Geef je toestemming om je antwoorden te gebruiken voor (anonieme) quotes in de onderzoekspublicaties? (*Niet verplicht.*)

Kern

Ik heb je vooraf een ontwerp voor een hoofdstuk over if-statements gestuurd dat meer aandacht geeft aan het oplossen van programmeerfouten.

- Wat was je eerste indruk van dit materiaal?
- Zou je materiaal met dit soort oefeningen willen gebruiken?
  - Wat zou jij aanpassen als je dit materiaal zou overnemen?

In vorige interviews gaven meerdere docenten aan dat zij denken dat het niet kunnen oplossen van een fout vooral komt door een gebrek aan zelfvertrouwen.

- Helpt dit materiaal bij het bouwen van zelfvertrouwen?
  - *In de literatuur over zelfvertrouwen blijken succeservaringen de meest effectieve manier om zelfvertrouwen te verhogen.*
  - Denk je dat de oefeningen hier daarbij helpen?
  - Hoe is het niveau van de oefeningen?
    - Leiden ze tot een succeservaring voor de leerlingen die dat nodig hebben?

- Welke kennis missen leerlingen nog als ze alleen dit materiaal volgen?

Slot

- Wil je nog iets kwijt dat we niet besproken hebben?
- *Deze evaluatie is de afronding van het onderzoek, dus ik ga dit uitwerken met de andere evaluaties en dan mijn conclusies schrijven.*
- *Bedankt!*

Bijlage B

Thematische codering van interviews

In deze bijlage staat voor elk afgenomen interview een thematische codering. Hierbij gebruiken we de vragen beschreven in sectie 3.3 als leidraad en geven per vraag één of meerdere kenmerkende citaten met een samenvatting van wat de respondent daar verder over te zeggen had.

B.1 Interviews met vakdocenten

B.1.1 Interview 1

B.1.1.1 Wat zijn de fouten waarbij je leerlingen vaak moet helpen?

- "In de vierde klas zie je meestal **syntaxfouten**."

  De docent schat in dat syntaxfouten ongeveer 80% van de fouten zijn waarop leerlingen in klas 4 vastlopen. Die schaart daar ook inconsistent hoofdlettergebruik in variabelen onder, en natuurlijk het vergeten van een dubbele punt of verkeerde indentatie. Die laatste worden wel snel minder als leerlingen doorkrijgen wat het betekent. Vwo-leerlingen wisten zich sneller over syntaxfouten heen te zetten dan havo-leerlingen, en ook in hogere klassen was het minder een probleem.

- "In vwo heb ik dan natuurlijk ook nog lijsten gedaan, daar krijg je ook wel een hoop **indexfouten**."

  De leerlingen waren wel goed in staat om deze fouten op te lossen, nadat ze uitleg hadden gekregen over wat de foutmelding betekende. Op basis van die foutmelding hadden ze niet door wat er fout ging.

- "Meneer, de test wordt niet groen."

  Soms zijn dit een soort syntaxfouten, dat ze hun output niet correct geformatteerd hebben. Het komt ook voor dat het meer algoritmische fouten zijn, bijvoorbeeld de verkeerde variabelen bij elkaar optellen of een variabele waarvan je de waarde later nodig hebt, hergebruiken. Door tests te gebruiken, creëert de docent foutmeldingen voor programma's die niet werken zoals de bedoeling was.

B.1.1.2 Hoe help je leerlingen bij het oplossen van een fout?

- "Op het begin probeerde ik ze allemaal **individueel** te helpen, maar dat werd al gauw **onhandelbaar**."
- "Daar heb ik dan een linkje naar die uitlegomgeving ingezet, zodat ze daaruit kunnen **kopiëren/plakken**."

Om de eerste lading syntaxfouten te voorkomen, geeft de docent de leerlingen toegang tot de voorbeelden uit de uitleg. Die code kunnen ze dan overnemen en aanpassen. Zo hebben ze een goed startpunt bij het maken van een opgave, en hebben ze minder last van syntaxfouten. De leerlingen kunnen bij de introductie van een nieuw onderdeel dan ook terugvallen op wat ze al wel weten: ze beginnen met een stuk code en passen dan de variabelenamen, de condities etc. aan.

- "Ik ben wel een beetje m'n **uitleg** erop gaan aanpassen, dat ik veel **explicieter** ben **over syntax**."

  Bij de introductie van een nieuwe constructie leest de docent nu duidelijk alle onderdelen voor, inclusief de dubbele punt en de indentatie. Die heeft ook het gevoel dat het werkt, maar het hangt er natuurlijk ook vanaf hoe goed de leerlingen opletten.

- "Dan ga ik in **stapjes af van fout richting oplossing** en dan op een gegeven moment nemen zij het over en dan gaan ze gauw typen."

  De docent begint bij de waarom-vraag: "waarom werkt dit niet?" Zo loopt hij samen met leerlingen het oplossingsproces door, in kleine stapjes, zodat de leerling het over kan nemen zodra die begrijpt wat er fout gaat en hoe dat opgelost kan worden. Soms gebruikt de docent ook een debugger om de leerling stap voor stap hun eigen code te laten zien, maar die verwacht niet van de leerlingen dat ze dat zelf zouden kunnen of doen.

B.1.1.3 Waar ligt het aan dat leerlingen een fout niet kunnen oplossen?

- "Maar puur op basis van de **foutmelding hadden zij niet door** dat het een indexfout was."

  In het geval van indexfouten (bij vwo) was de foutmelding niet duidelijk over wat er nu fout was. Zodra de docent vertelde wat een indexfout betekende, waren leerlingen goed in staat die op te lossen.

- "Het lastige, voornamelijk in de vierde klas, (...) is dat ze die **foutmelding** zelf **niet lezen**. Er staat iets roods, en dan is het paniekstand."

  Soms is het hardop voorlezen van de foutmelding al genoeg om de leerling te laten begrijpen wat er moet gebeuren, maar soms is er ook een beetje vertaling nodig, omdat de foutmelding in het Engels is. In de meeste gevallen proberen leerlingen het niet eens te lezen. Dat heeft er ook mee te maken dat de docent in de buurt is om te vragen, dus er is weinig druk om het zelf te moeten doen. Bij syntaxfouten, de fouten die leerlingen aan het begin veel hebben, zijn de foutmeldingen ook weinig behulpzaam, waardoor leerlingen denken dat ze in het algemeen weinig aan een foutmelding hebben.

- "Ik **beloon ze te weinig** misschien om het zelf op te lossen."

  De docent heeft kleine klassen en kan leerlingen dus veel helpen, waardoor de druk voor leerlingen om het zelf te doen kleiner is.

- "Zij **weten niet precies waar ze op moeten letten**."

  De leerlingen hebben nog geen ervaring en hebben dus minder snel door wat de oorzaak van een fout kan zijn. De docent ziet een rood streepje aan het eind van een while en weet meteen dat daar waarschijnlijk een dubbele punt mist, maar de leerlingen hebben die ervaring nog niet. Na verloop van tijd merk je wel dat de leerlingen het leren, maar bij ieder stukje syntax begint het opnieuw.

B.1.1.4 Hoe vaak lopen leerlingen vast?

- "Het kan echt wel zijn op de orde van **eens per tien minuten**, (...) en dan komen ze ook echt bijna geen enkele stap verder."

  Dat is, grof geschat, **bij 20% van de leerlingen** het geval. Voor de gemiddelde leerling eerder eens per kwartier, 20 minuten.

B.1.2 Interview 2

B.1.2.1 Wat zijn de fouten waarbij je leerlingen vaak moet helpen?

- "Dat is natuurlijk **super verschillend**."

  Een deel van de leerlingen komt moeilijk op gang en loopt al vast op de basis, die weten niet hoe ze de structuur neer moeten zetten. Leerlingen die verder zijn maken minder syntactische fouten, maar meer semantische, bijvoorbeeld in een chaotische opbouw van hun programma. Dat uit zich dan doordat het programma niet doet wat ze verwacht hadden.

- "Wat is nou precies de **syntax** en hoe gebruik ik die nou?"

  Dat lijkt in havo 4 een groter probleem te zijn dan bijvoorbeeld in vwo 5, die pakken dat soort dingen sneller op.

- "De constructies, de **juiste opzet**, het goed nadenken over wat voor variabelen gebruikt, wat voor lusjes je nou precies gaat gebruiken, wat voor voorwaarden je nodig hebt."

  Deze problemen komen in vwo 5 meer voor, en uit zich dan in een programma dat niet doet wat er verwacht werd.

B.1.2.2 Hoe help je leerlingen bij het oplossen van een fout?

- "Ik ben wel fan van **print-statements** of **stapje voor stapje doorlopen** als dat kan."

  Dat past de docent toe bij beide groepen leerlingen, zowel de leerlingen die moeilijk op gang komen als diegenen die al wat verder zijn. "Doe nou even één stapje, doe een print statement erbij, kijk eens wat er gebeurt, gaat dat goed, nou dan ga je een stapje verder." De docent gebruikt hierbij ook wel eens een debugger, om de leerlingen precies te laten zien wat er gebeurd, en dan de vraag te stellen of dat is wat de leerling wilde dat er zou gebeuren.

- "Dat groepje te pakken en ze met elkaar een **basis** te laten **leggen**."

  Bij een grotere opdracht helpt de docent de zwakkere leerlingen op weg, zodat ze met een goede basis kunnen beginnen, en minder snel vastlopen op een fout in de basis van hun programma.

B.1.2.3 Waar ligt het aan dat leerlingen een fout niet kunnen oplossen?

- "Zeker in klas 4 is het toch vaak wel dat **analytisch denkvermogen**, dat **abstracte**, dat **opdelen van het grote probleem in kleinere problemen**."

  Het stap voor stap gestructureerd werken en opdelen van het probleem is waar de meeste leerlingen door vastlopen. Dat is de reden dat ze fouten maken, maar ook dat het niet lukt om een fout op te lossen.

- "In eerste instantie ook wel een beetje de syntax, alleen vooral aan het begin omdat ze **eraan** moeten **wennen**."

- "Ze zijn gewoon bang om die fout te maken en dan durven ze niet te beginnen of iets te proberen."

  Leerlingen die **zelfstandiger** zijn en meer **doorzettingsvermogen** hebben, kunnen fouten vaak zelf oplossen, zeker als het om simpelere syntaxfouten gaat. Die leerlingen maken die fouten ook, maar gaan zelf op zoek naar de oplossing, zonder dat er hulp van de docent nodig is. De docent merkt dat vwo-leerlingen minder snel opgeven, terwijl havo-leerlingen eerder met "hij doet het niet" komen en opgeven. Voor een deel is het ook een kwestie van **zelfvertrouwen**. Zelfverzekerde leerlingen die niet schrikken van een fout zijn beter in staat om een fout op te lossen.

  Bij Co-Teach zou dit wel een andere rol kunnen spelen, omdat de leerlingen "moeten zwemmen", ze hebben geen makkelijke uitweg. De docent geeft aan dat die altijd wil helpen, en de leerlingen misschien ook wel eens wat meer moet laten zwemmen.

- "Het is mijn eigen valkuil dat ik **te snel** leerlingen wil **helpen**."

  De docent wil leerlingen graag op weg helpen, waardoor leerlingen het minder zelf hoeven te proberen.

B.1.2.4 Hoe vaak lopen leerlingen vast?

- "Het zijn eigenlijk altijd dezelfde leerlingen die **continue** vastlopen."

  Een deel van de leerlingen (die met weinig zelfvertrouwen) lopen continue vast, elke keer als de docent langskomt hebben ze eigenlijk wel hulp nodig. Deze leerlingen geeft de docent steeds kleine opdrachtjes, om ook het zelfvertrouwen op te krikken en de leerling te laten zien dat ze het best kunnen.

B.1.3 Interview 3

B.1.3.1 Wat zijn de fouten waarbij je leerlingen vaak moet helpen?

- "Gewoon de kleine foutjes, die komen echt het meeste voor. De puntkomma's, de komma's."

  De **syntaxerrors** komen het meeste voor. Extra moeilijkheid is dat het regelnummer bij de foutmelding vaak niet helemaal klopt. Deze fouten zijn ca. 80-90%.

- "Dan hebben ze de **logica** even **niet** helemaal **helder**."

  Dan doet het programma niet meer wat ze verwacht hadden, bijvoorbeeld omdat er variabelen door elkaar gehaald worden.

B.1.3.2 Hoe help je leerlingen bij het oplossen van een fout?

- " 'Kijk eerst even een **regel ervoor**, wat daar fout is.' "

  Bij syntaxerrors, omdat het regelnummer dat wordt aangegeven meestal niet correct is. Meestal komen de leerlingen daar echter niet uit.

- "Een kwestie van **aanwijzen** (...) en dan komen ze ook wel weer verder."

  Als een leerling niet verder komt met een globale aanwijzing waar ze moeten kijken (vorige punt), wijst de docent de precieze fout aan. Leerlingen hebben dan snel door hoe het opgelost moet worden.

- "Als het ergens fout gaat, moet je weer **terug naar de basis** en dan weer proberen op te bouwen tot het punt waar je bent."

Als het programma niet werkt zoals bedoeld, gaat de docent met leerlingen een stapje terug: wat is nou eigenlijk het doel, en hoe pakken we dat aan. De docent ziet hier nog een parallel met wiskunde: het opbouwende karakter. Elk stapje is nodig, anders raakt de leerling het kwijt. De docent helpt leerlingen met secuur kijken, wat er nou precies gebeurt en of de leerling dat snapt.

- "Het is heel specifiek **leerlinggevoelig**. (...) Het is vaak het inschatten, (...) waar staan ze dan?"

  "Waar zit die leerling? Waar gaat die fout? Waar hapert het? En dat is het schakelmoment waar een docent een hele belangrijke rol heeft."

  Als docent probeer je eerst te peilen waar de leerling staat, zodat je ze vanaf dat punt kunt helpen en niet steeds bij nul begint. Het is heel lastig voor een leerling om dat zelfstandig te doen. (Dus dat wordt bij Co-Teach pittig.)

- "Ga effe zelf **zoeken online**, wat is er allemaal, wat kun je."

  Vooral bij functies verwijst de docent leerlingen ook naar online bronnen, omdat er veel verschillende manieren zijn om het te bekijken. Er is online veel te vinden. Leerlingen hebben dan wel hulp nodig bij het interpreteren van dingen die ze online vinden, vaak ook omdat ze te snel gaan.

B.1.3.3 Waar ligt het aan dat leerlingen een fout niet kunnen oplossen?

- "De **snelheid**, dat ze denken 'oh, ik weet het wel, dat is makkelijk'."

  De leerlingen overschatten zichzelf en blijven daardoor hangen in fouten. De leerling denkt dat die het kunstje doorheeft en dat gewoon kan doen, maar onderschat de precisie die vereist is bij het programmeren. "Dit is toch goed? Dit hebben we net ook gedaan. Dat is toch hetzelfde? Wat is er nu anders dan?" De docent maakt een vergelijking met wiskunde: als de docent het voordoet, is het heel simpel, maar als een leerling het zelf moet doen, moeten ze toch meer nadenken dan verwacht.

  Het gevolg is dat ze dan ook snel denken "ik snap het niet." Dat is bij havo-leerlingen iets meer het geval, vwo gaat wat makkelijker, al is dat wel erg generaliserend.

- Bij functies: "**Niet weten wat ze eigenlijk aan het doen zijn**, dat is even het..."

  Bij functies is het vooral lastig als je het doel niet duidelijk voor ogen hebt, waar is dan een functie voor nodig? Als de leerling dan ook niet helder heeft wat het idee van een functie is, kunnen ze er weinig mee beginnen. Bij gestructureerde opdrachten kunnen ze dan nog wel de stappen volgen, maar bij een vrijere opdracht weten ze niet meer hoe ze het aan moeten pakken.

B.1.3.4 Hoe vaak lopen leerlingen vast?

- "Gemiddeld zal een leerling **één, twee keer in de les** daar iets over vragen."

  Dat zijn dan korte aanwijzingen en dan kan de leerling weer verder, maar die "tikjes" hebben leerlingen wel nodig.

B.2 Interviews met vaksteunpunten

B.2.1 Eerste serie (2022)

B.2.1.1 Interview 1

B.2.1.1.1 Hoeveel vragen stellen leerlingen?

- "Nee, eigenlijk niks."

  Van ongeveer 20 leerlingen is er geen enkele vraag gekomen. Deze vaksteunpuntassistent kijkt alleen ingeleverde opdrachten na. Op het moment van het interview had die ook alleen nog de eerste opdrachten nagekeken en hadden veel leerlingen de latere opdrachten ook nog niet helemaal afgerond, ook al waren de programmeerlessen in principe al wel afgelopen.

B.2.1.1.2 Over welke fouten stellen leerlingen vragen? Niet van toepassing.

B.2.1.1.3 Hoe help je leerlingen die een bepaalde vraag hebben? Niet van toepassing.

B.2.1.1.4 Welke soorten fouten maken leerlingen in de ingeleverde opdrachten?

- Verwarring over het combineren van if-statements

  De assistent geeft het voorbeeld van een programma waar je een kleur (blauw, groen of geel) moet invullen en het programma gebruikt een if-statement om te bepalen wat er dan moet gebeuren. Twee (van de 15) leerlingen lieten het programma een eigen foutmelding printen als de kleur niet blauw was. Bij het invullen van groen printte het programma dan eerst nog "Niet blauw" of iets dergelijks. Deze regel had helemaal aan het eind moeten staan als de input geen van de drie kleuren was.

  Het is niet duidelijk of dit het geval is dat de leerlingen niet doorhadden wat de bedoeling van de opdracht was, of dat niet voor elkaar kregen en het maar zo gelaten hebben. In dit geval is het waarschijnlijk dat leerlingen wel begrepen dat het programma niet helemaal werkt zoals bedoeld.

B.2.1.1.5 Overige observaties

- De co-teacher heeft zelf echter ook een beetje programmeerkennis, waardoor de leerlingen voor de meeste vragen daar wel terecht kunnen.

B.2.1.2 Interview 2

B.2.1.2.1 Hoeveel vragen stellen leerlingen?

- "Ik denk in totaal, over code zelf, twee vragen."

  Dat waren twee vragen van dezelfde leerling, op een populatie van ongeveer 40 à 45 leerlingen, de helft uit die regio met drie scholen. Het speelt ook mee dat de leerlingen nog niet allemaal klaar waren het onderdeel programmeren.

B.2.1.2.2 Over welke fouten stellen leerlingen vragen?

- "De uitkomst klopte niet met hoe het hoorde te zijn, waarom is dit zo?"

  De assistent merkte op dat leerlingen het einddoel vaak nog wel in zicht hebben, maar niet wat er moet gebeuren om dat doel te bereiken, ook als de opdracht aangeeft dat een

bepaalde functie of constructie gebruikt moet worden. In dit geval had de leerling dus het doel van de opdracht wel begrepen en opgemerkt dat diens eigen programma daar niet mee overeen kwam, maar tegelijkertijd had de leerling ook een deel van de opdracht niet begrepen.

- "Het was meer een soort van vragen om advies."

  De tweede vraag van deze leerlingen ging meer over de aanpak van een opdracht, en een vraag om uitleg over hoe een bepaalde constructie daar gebruikt kon worden.

B.2.1.2.3 Hoe help je leerlingen die een bepaalde vraag hebben?

- "Ik had wat hints gegeven. (...) Je moet deze instructie gebruiken, en dit en dit."

  Voor een deel waren deze hints ook herformuleringen van punten uit de opdracht. Bij de andere vraag wees de assistent de leerling op een stukje voorbeeldcode bij de opdracht om een beginnetje mee te maken en het algemene idee van hoe een oplossing eruit zou kunnen zien.

B.2.1.2.4 Welke soorten fouten maken leerlingen in de ingeleverde opdrachten?

- "Ik had één opdracht waar een error in zat. (...) Er kwam een error van date is geen bestaande variabele."

  Vermoedelijk was dit een name error, waarbij de leerling was vergeten om een variabele te declareren voor gebruik (of een andere naam gebruikte). Het kan ook zijn dat de leerling hier een import regel was vergeten om de datetime functies te gebruiken. Als de leerling het programma had uitgevoerd was deze fout snel duidelijk geworden, maar de assistent vermoedde dat deze leerling dacht dat die wel wist hoe het werkt en het gewoon heeft ingeleverd zonder te testen.

B.2.1.2.5 Overige observaties

- "Die dacht gewoon"ik weet dat wel, ik doe dit riedeltje, het is goed.""

  De leerling die een programma met een duidelijke fout inleverde was waarschijnlijk overmoedig geraakt omdat die al vaker dat soort programmaatjes had geschreven bij andere opdrachten en daardoor een beetje laks was geworden.

- "Maar ik denk dat zij nooit echt vragen stellen, ook al moeten ze dat misschien wel doen. Dat is echt zo'n stukje leeftijd en ook de hele opzet van Co-Teach dat ze niet weten wie de CTI support is."

  De assistent denkt dat de drempel om vragen te stellen hoog ligt voor de leerlingen en dat er daardoor ook weinig vragen komen.

B.2.1.3 Interview 3

B.2.1.3.1 Hoeveel vragen stellen leerlingen?

- "Wat ik in ieder geval heb geconstateerd is dat leerlingen redelijk weinig vragen stellen."

  Die vragen komen ook niet per se tijdens de les, maar vooral ook als leerlingen met huiswerk bezig zijn. De studentassistenten zijn natuurlijk niet altijd bereikbaar dus het kan even duren voor een antwoord komt. Dat zou een reden kunnen zijn dat leerlingen weinig vragen stellen.

B.2.1.3.2 Over welke fouten stellen leerlingen vragen?

- "Het grootste deel van de vragen was dat ergens een error kregen, (. . .) als gewoon een syntax error."

  Bijvoorbeeld ook variabelenamen die niet goed geschreven waren, vooral omdat de leerlingen ingewikkelde namen gebruiken die ze dan later vergeten of niet meer weten welke variabele waarvoor bedoeld was.

- "Functiegebruik en hoe dat precies werkte."

  In dit geval ging het niet om een fout, maar meer om uitleg over het principe.

B.2.1.3.3 Hoe help je leerlingen die een bepaalde vraag hebben?

- "Ik hoopte zo min mogelijk het antwoord te geven. (. . .)"Kijk eens goed naar de variabelenamen.""

  De assistent probeerde leerlingen een hint in de goede richting te geven, bijvoorbeeld ook aangeven op welke regels ze moeten letten

B.2.1.3.4 Welke soorten fouten maken leerlingen in de ingeleverde opdrachten?

- "Meeste zit vooral in gewoon de functies."

  De assistent vermoed dat leerlingen denken dat het programma van boven naar beneden uitgevoerd wordt en dat de functies dan ook uitgevoerd worden. Dat uit zich dan in een programma dat niet werkt zoals verwacht.

- "Je kreeg niet de goede uitkomst."

  Dat is dus het geval bij bovenste fouten met functies, maar het gebeurde ook doordat leerlingen een verkeerde loopconstructie gebruikten.

B.2.1.3.5 Overige observaties

- "Zij denken"Oh, misschien krijg ik pas over drie dagen antwoord. Weet je, laat maar, ik zoek het wel uit.""

  De assistent vermoed dat leerlingen ook weinig vragen stellen omdat het lang kan duren voor er een antwoord is. De hulp komt dus niet op het moment dat het nodig is.

- "Naja,"Ik weet niet waarom en de studentassistent reageert toch pas laat, ik lever het wel gewoon in, ik krijg het later wel terug.""

  Waarom leveren leerlingen iets in waarvan ze zelf (hoogstwaarschijnlijk) ook hebben gezien dat het niet het goede antwoord is: waarschijnlijk omdat het lang kan duren voor ze hulp kunnen krijgen, dus dan geven leerlingen het gewoon op. Iets inleveren is nog altijd beter dan niets.

B.2.2 Tweede serie (2024)

B.2.2.1 Interview 1

B.2.2.1.1 Hoeveel vragen stellen leerlingen?

- "Ik heb wel eens weken gehad dat ik gewoon van een klas 10, 20 mails kreeg."

  Deze assistent geeft aan dat als het op school door de co-teacher wordt aangemoedigd, ze in het algemeen best veel mails binnen krijgen.

- “Maar ik heb ook klassen gehad waarvan ik gewoon überhaupt geen mail krijg.”

  Dat is het andere uiterste. Er zit dus veel verschil tussen verschillende scholen. De assistent vermoedt dat de docenten op deze school leerlingen niet herinneren aan de mogelijkheid om vragen te stellen.

- “Over programmeren specifiek heb ik, denk ik, twee vragen in totaal gekregen. Echt heel weinig.”

  Over programmeren stelden de leerlingen veel minder vragen dan over andere onderwerpen.

B.2.2.1.2 Over welke fouten stellen leerlingen vragen?

- “Een iemand had heel zijn code in een oneindige loop gezet.”

  Tot op zekere hoogte klopte dat ook met de opdracht: het programma moest ook blijven doorlopen, maar deze leerling had ook de imports en dergelijke binnen die loop geplaatst.

- “Die ander had, geloof ik, een syntaxfout in een while loop.”

  Dit was een typische fout met een dubbele punt die ergens wel of niet stond waar die wel of niet hoorde te staan.

B.2.2.1.3 Hoe help je leerlingen die een bepaalde vraag hebben?

- “Dus toen heb ik daar uitleg over gegeven.”

  De assistent heeft de leerlingen bij deze fouten geholpen door ze uit te leggen wat er fout gaat en hoe het wel zou werken.

B.2.2.1.4 Welke soorten fouten maken leerlingen in de ingeleverde opdrachten?

- “Er zijn heel veel syntax fouten.”

  De leerlingen laten wel zien dat ze een beetje begrip hebben van hoe constructies als if-statements en while-loops werken, maar maken in de opdrachten toch nog veel syntaxfouten. De assistent denkt dat de module te snel gaat voor deze leerlingen.

- “Alles bij elkaar zetten wat ze hebben geleerd is nog best wel pittig.”

  Leerlingen lijken het moeilijk te vinden om alle losse onderdelen te combineren in een werkend programma. Dan lijken ze wel een vaag idee te hebben van hoe het eruit zou moeten zien, maar snappen ze toch niet helemaal wat ze aan het doen zijn. Ook hier ziet de assistent wel ChatGPT werk langskomen, maar dat werkt meestal niet volledig. Sommige leerlingen hebben dan een goed startpunt en kunnen er wat van maken, maar anderen snappen het duidelijk niet en kunnen op die basis ook niet voortbouwen.

- “Heel veel hebben onnodige variabelen, die ze dan toevallig niet gebruiken of hebben meerdere variabelen met dezelfde naam.”

  Deze leerlingen laten daarmee zien dat ze toch niet helemaal snappen wat ze aan het doen zijn, zelfs als ze er wel een werkende oplossing uit weten te krijgen.

B.2.2.1.5 Overige observaties

- “Het meeste is heel veel theorie, en dat zou een docent best wel prima moeten kunnen.”

  De assistent denkt dat de co-teachers leerlingen ook best veel kunnen helpen, als ze zich ook een beetje in de stof verdiepen. Of ze dat ook doen verschilt natuurlijk per docent. Als

de leerlingen echt gaan programmeren wordt dat wel lastiger, omdat daar meer ervaring bij komt kijken.

- "Je merkt dan toch dat leerlingen bij makkelijkere modules sneller vragen stellen."

  Deze assistent denkt dat programmeren een van de moeilijkere modules is en dat leerlingen dan juist minder vragen stellen omdat ze er langzaam doorheen komen, misschien een week of twee achterlopen en dat niet willen toegeven.

- "Ik vermoed dat er ook wel een klein beetje ChatGPT bij is komen kijken."

  Een andere reden dat er weinig vragen komen zou kunnen zijn dat leerlingen andere bronnen vinden, bijvoorbeeld door ChatGPT te gebruiken, maar er is online sowieso een heleboel te vinden over programmeren.

- "Ik denk dat de drempel een beetje hoog is."

  De assistent denkt dat leerlingen niet snel een mail willen sturen. Er is wel verschil tussen leerlingen: sommigen stellen een duidelijke vraag als ze hulp nodig hebben in een nette mail, anderen geven het op en laten het er maar bij zitten.

- "We vergeten dat de syntax echt heel erg belangrijk is."

  De assistent denkt dat er in het programma teveel focus wordt gelegd op de concepten en daardoor de syntax wordt vergeten, terwijl die wel heel belangrijk is als je echt praktisch aan de slag wil met programmeren. Meer kleine opdrachtjes zouden daarbij kunnen helpen, want nu is de sprong naar een grote opdracht best groot.

B.2.2.2 Interview 2

B.2.2.2.1 Hoeveel vragen stellen leerlingen?

- "Vergeleken met andere onderdelen binnen Co-Teach is het niet duidelijk veel over programmeren."

  Bij dit vaksteunpunt komen er over programmeren ongeveer even veel vragen binnen als over andere onderwerpen. Dat is er dan midden in de module eentje per week en tegen het einde, als de eindopdracht eraan komt, iets meer.

B.2.2.2.2 Over welke fouten stellen leerlingen vragen?

- "Ik heb eigenlijk maar een echt concreet voorbeeld, dat is dus met import statements."

  Een leerling had een import verkeerd en kreeg daardoor een foutmelding bij het aanroepen van een functie die geïmporteerd had moeten worden, maar dat dus niet was.

- "Vragen over het platform zelf."

  Bijvoorbeeld als iets niet werd goedgekeurd, terwijl het wel correct was. Dus geen programmeerfouten, maar meer organisatorisch.

B.2.2.2.3 Hoe help je leerlingen die een bepaalde vraag hebben? Niet besproken.

B.2.2.2.4 Welke soorten fouten maken leerlingen in de ingeleverde opdrachten?

- "Soms komen ze op de toets en dan volgens mij hebben ze geen idee wat een if-statement is."

De assistent merkt op dat sommige leerlingen misschien wel vragen hadden moeten stellen...

- "Dat combineren. Ik had het gevoel dat ze daar ook wel wat moeite mee hadden."

  Het combineren van verschillende functionaliteiten in een groter programma gaat regelmatig mis. In het meest extreme voorbeeld had een leerling volledig losse stukjes code geschreven voor deelopdrachten, zonder enige verbinding daartussen.

B.2.2.2.5 Overige observaties

- "Als ze vragen hebben, dan kunnen ze die wel aan hem stellen, maar eigenlijk paast hij het gelijk door." en "Die wil het ook begrijpen. Die probeert ze echt wel te helpen."

  De ene co-teacher verdiept zich in de stof en probeert leerlingen veel zelf te helpen, terwijl een ander de vragen allemaal direct doorstuurt naar het vaksteunpunt. Het maakt ook verschil of die docent leerlingen aanraadt om een vraag te stellen bij het vaksteunpunt, want dan komen er ook meer vragen.

- "Ze redden zichzelf wel redelijk."

  De assistent denkt dat leerlingen er redelijk zelf uitkomen, bijvoorbeeld omdat de co-teacher ze kan helpen, of omdat ze gebruik maken van ChatGPT.

- "Ik denk wel dat als eentje eenmaal over die drempel heen is, dat misschien dan ook wel makkelijker is."

  Als een leerling eenmaal een vraag heeft gesteld (en antwoord heeft gekregen) lijkt die leerling ook sneller terug te komen met een nieuwe vraag.

- "Misschien ook een paar keer in het platform zelf gewoon neerzetten:"Kom je er niet uit? Vraag het!""

  De assistent lijkt het een goed idee om leerlingen meer aan te moedigen de interactie op te zoeken. Een andere manier zou kunnen zijn door meer dingen tussendoor al door de vaksteunpunten laten bekijken.

B.2.2.3 Interview 3

B.2.2.3.1 Hoeveel vragen stellen leerlingen?

- "Van mij mochten de leerlingen echt aanzienlijk meer vragen."

  Deze assistent krijgt niet veel vragen over programmeren, en vind dat het er best wel wat meer mogen zijn.

- "Van de docenten hoor ik heel vaak:"Ja, de leerlingen snappen het niet en het is heel moeilijk voor ze." Maar vervolgens stelt niemand een vraag."

  Van de co-teachers zonder programmeerkennis hoort deze assistent wel dat de leerlingen het in de les echt moeilijk vinden, maar nog steeds krijgt die weinig vragen.

B.2.2.3.2 Over welke fouten stellen leerlingen vragen?

- "Veel leerlingen lezen de opdracht niet goed en dan sturen ze mij een vraag van"Oh, ik snap niet wat ik hier moet doen.""

  De grootste moeilijkheid voor leerlingen lijkt het lezen van de opdracht, daar methodisch de informatie uithalen en dan rustig nadenken. De assistent ziet dat veel leerlingen gewoon

beginnen en dan vastlopen op iets wat ze niet begrijpen omdat ze de opdracht niet goed hebben gelezen.

- "Echt Python errors valt relatief mee, daar krijg ik niet zo snel vragen over."

  Leerlingen komen zelden met een vraag naar aanleiding van een foutmelding. De assistent merkt op dat die over het algemeen ook makkelijk te Googelen zijn.

B.2.2.3.3 Hoe help je leerlingen die een bepaalde vraag hebben?

- "We hebben ook de uitwerkingen natuurlijk, dus het is vrij makkelijk om dan even te denken: dus dan moet ik ze een beetje die kant op duwen."

  De assistent kijkt vaak naar de uitwerkingen om te zien welke richting de leerlingen uit moeten en probeert ze dan die kant uit te sturen.

B.2.2.3.4 Welke soorten fouten maken leerlingen in de ingeleverde opdrachten?

- "Zelfs de syntax van sommige [onderdelen] is gewoon een beetje lastig."

  De assistent ziet op toetsvragen dat sommige leerlingen de syntax niet goed kennen, maar merkt ook dat dat waarschijnlijk gewoon de leerlingen zijn die niet geleerd of geoefend hebben.

B.2.2.3.5 Overige observaties

- "Dat verschilt natuurlijk per school: ik heb op één school een docent die echt zelf ook programmeert en dat ook leuk vindt. (...) Bij een andere school twee docenten die echt niks van programmeren afweten."

  Er zit wel verschil in hoeverre de co-teachers zich in de stof verdiepen en leerlingen kunnen helpen. Deze assistent heeft dus contact met een aantal docenten die er helemaal niets van weten, maar ook met een docent die zich er wel in verdiept en het ook leuk vindt. Dat merkt de assistent ook in de hoeveelheid vragen: bij die eerste zijn dat er meer dan bij de tweede.

- "Ik denk dat leerlingen eerder aan ChatGPT iets vragen dan aan mij."

  De assistent denkt dat dat ook positief kan zijn, omdat het de druk op Co-Teach iets verlaagt, maar is wel bezorgd over hoeveel leerlingen ervan leren. Ook merkt die op dat veel opdrachten in Co-Teach wel een beetje aangepast moeten worden er goed om te kunnen gaan.

- "Zodra leerlingen ook vragen stellen, gaan ze wel meer vragen stellen. Je merkt wel dat dat vertrouwen dan begint te lopen."

  Het zijn vaak dezelfde leerlingen die wel vragen stellen. De assistent merkt ook op dat het vooral meiden zijn die vragen stellen, de jongens veel minder.

- "Het is heel overwhelming voor hen: hier heb je een programma en succes, zeg maar."

  De assistent denkt dat programmeren ook ingewikkeld is voor de leerlingen: het is veel in één keer. Die merkt ook dat het voor irritatie zorgt bij leerlingen en co-teachers: de leerlingen klagen erover dat ze het niet snappen en dat de docent het ook niet begrijpt. En de co-teachers raken dan geïrriteerd omdat ze een klas vol leerlingen hebben die geïrriteerd zijn en ze niet direct hulp kunnen krijgen.

B.3 Expertreviews

B.3.1 Interview 1

B.3.1.1 Helpt dit materiaal om het zelfvertrouwen van leerlingen in het oplossen van programmeerfouten te vergroten?

Over de oefeningen met syntaxfouten:

- "Ik denk dat deze oefening, dat die werkt, leerlingen gaan hier iets van leren."

  De docent merkt wel op dat het een beetje afhangt van een gevoel voor detail, dus leerlingen moeten wel duidelijk beseffen dat een dubbele punt ergens een relevant detail is.

- "Het voelt zo overzichtelijk aan, omdat het heel contained en beperkt is."

  Binnen deze oefening zal het wel goed gaan, en dus een succeservaring opleveren. Het is de vraag hoe zich dat vertaalt, in het bijzonder naar andere foutmeldingen (want dat zijn er vrij veel), maar ook als leerlingen zelf bezig zijn, code schrijven en daar een foutmelding bij krijgen.

Over de oefeningen met een logische fout:

- "Het is een begrijpend lezen oefening."
- "Ik denk dat je hier extreme resultaten krijgt."

  Diegenen die het zien hebben het antwoord meteen, diegenen die het niet zijn zullen er lang naar zitten staren. En de leerlingen die het niet zien zijn ook de leerlingen die toch al een beetje "shaky" waren.

- "Als je dit in klassenverband doet? Dan roept er eentje door de klas. (...) Bam, hele oefening weg."

  Deze oefening mist iets constructiefs: de oplossing is te simpel voor hoe ingewikkeld de oefening eigenlijk is. Leerlingen kunnen dan weg komen met een heel oppervlakkige benadering.

B.3.1.2 Welke kennis missen leerlingen nog om een fout op te lossen op basis van dit materiaal?

- "Je bent nu eigenlijk nog een beetje bezig op basisschoolniveau, rekenoefeningen waarbij je eigenlijk op de puntjes het antwoord in moet vullen."

  En dat kan ook zeker nuttig zijn, maar we moeten wel de lijn naar de echte wereld, waar leerlingen zelf code schrijven in een vrije opdracht, in de gaten houden. De docent haalt een voorbeeld aan van YouTube tutorials: leerlingen vinden het vaak heel prettig om mee te typen met zo'n filmpje. Daar kan ook een tussenvorm in zitten, waar daar af en toe iets open gelaten wordt om zelf in te vullen.

- "Je zou je af kunnen vragen of je dan bijvoorbeeld een taal zou kunnen maken die minder syntaxgevoelig is."

  De docent refereert hier aan Hedy (Hermans, 2020). Niet alleen is de syntax daar in het begin minder strikt, de leerling gaat ook "bijna een socratische dialoog aan met de computer en met de leerstof" naarmate meer van de echte Python-syntax geïntroduceerd wordt.

- "Begrijpend code lezen."

De oefening met een logische fout is niet alleen een oefening in begrijpend lezen van de opdracht, maar ook van de code. De leerlingen moeten ook een cognitief schema vormen van dat programma, dus dat moeten ze dan wel kunnen. Daar zit nog een stap voor dat leerlingen überhaupt internaliseren dat er meer is dan de syntax, maar dat het ook echt een semantische betekenis heeft.

- "Hoe kun je dit soort problemen nu aanvliegen? Hoe kun je ze kleiner of behapbaar maken? (...) Hoe tem je deze draak?"

  Leerlingen hebben waarschijnlijk wel behoefte aan algemene probleemoplossingsstrategieën. Dat moet wel meteen concreet zijn, want anders wordt het abstract gewauwel.

- "Je moet een bepaald soort bestendigheid hebben tegen in een probleem vastzitten."

  De docent maakt een vergelijking met wiskunde A t/m D: als je wiskunde B kiest moet je er een beetje tegen kunnen om 30 seconden naar een oefening te kijken en nog niet te weten wat je moet doen. Bij wiskunde A is die tolerantie lager en bij wiskunde D juist nog veel hoger. Dat zouden verschillende fases kunnen zijn die we bij programmeren ook kunnen hanteren.

- "Relatie zullen we het maar even niet over hebben."

  De docent haalt de zelfdeterminatie theorie aan als we het over motivatie hebben: autonomie, relatie en competentie. Op het punt van relatie maakt die zich vooral zorgen in de constructie van Co-Teach.

B.3.2 Interview 2

Bij dit interview is de opname mislukt en de resultaten zijn gebaseerd op aantekeningen die direct na afloop zijn gemaakt. Hierdoor kunnen we geen typerende citaten geven.

B.3.2.1 Helpt dit materiaal om het zelfvertrouwen van leerlingen in het oplossen van programmeerfouten te vergroten?

- Het helpt sowieso met herkenning: leerlingen hebben veelvoorkomende fouten in ieder geval eerder gezien.

  Deze docent weet niet in dat dat per se helpt bij het zelfvertrouwen van leerlingen, maar het geeft ze in ieder geval wel een vorige ervaring om op terug te vallen.

- De oefeningen met syntaxfouten gaan waarschijnlijk wel goed.

  De oefeningen op syntax zullen waarschijnlijk wel een succeservaring opleveren.

- De stap naar de oefeningen met een logische fout is groot.

  Deze docent zou daar een apart hoofdstuk aan willen besteden: om leerlingen eerst te introduceren met het idee dat er iets is als een logische fout en ze meer handvatten te bieden bij het oplossen van dergelijke fouten. De docent merkte ook op dat deze oefening ook veel op een begrijpend lezen oefening leek, wat wat hen betreft iets anders is dan programmeren.

B.3.2.2 Welke kennis missen leerlingen nog om een fout op te lossen op basis van dit materiaal?

- Opmaak van code: logische witruimte helpt voor de leesbaarheid.

De docent maakt hier een vergelijking met leerlingen die net leren schrijven: die schrijven aan een stuk door, maar op een gegeven moment leren we ze ook om het verhaal op te delen in alinea's. Met witruimte kun je in een programma ook "alinea's" aanbrengen die de structuur van de code beter laten zien, waardoor het makkelijker te lezen en te debuggen is.

Het kan ook helpen bij een fout die de docent leerlingen vaak ziet maken: als er geen witregel na het blok bij een if-statement staat, denken leerlingen soms dat de volgende, niet ingesprongen regel ook nog bij dat blok hoort. Met een nettere opmaak wordt dat ook duidelijker.

- Logische fouten zitten vaak in de keuzepunten.

  Vaak zitten logische fouten op een punt waar keuzes gemaakt worden in een programma, dus bij if-statements en loops. Als je dus op zoek bent naar een fout in een programma, zijn dat de punten die je goed moet bekijken.

Bijlage C

Ontwerp

Hier volgt een ontwerpvoorstel op basis van een enkel hoofdstuk over if-statements. Dit ontwerp is ook, met een betere opmaak, online te vinden via https://arthurrump.github.io/HelpEenError. De schuingedrukte *hints* zijn standaard ingeklapt, dus de tekst wordt pas weergegeven als de lezer dat blok uitklapt.

Dit materiaal behandelt grofweg dezelfde stof als hoofdstuk 3.4 uit Fundament Informatica, maar is uitgebreid met oefeningen waarin leerlingen bestaande programma's verbeteren, bijvoorbeeld om een foutmelding of een logische fout op te lossen. In de voorgaande hoofdstukken zijn booleaanse waarden, vergelijkingsoperatoren en booleaanse operatoren besproken.

If-statements

Tot nu toe deden alle programma's die we hebben geschreven steeds hetzelfde: alle regels code werden altijd één voor één uitgevoerd. Soms willen we echter dat er dingen anders lopen in een bepaalde situatie. Zie dit voorbeeld:

```
leeftijd = 16
if leeftijd < 18:
    print("Sorry, je bent niet stemgerechtigd")
```

```
> Sorry, je bent niet stemgerechtigd
```

Hier wordt de melding "Sorry, je bent niet stemgerechtigd" weergegeven, omdat de ingevulde leeftijd onder de 18 is. Als we echter de leeftijd veranderen, dan wordt de melding niet geprint:

```
leeftijd = 32
if leeftijd < 18:
    print("Sorry, je bent niet stemgerechtigd")
```

Het keyword `if` geeft aan dat we code voorwaardelijk uitvoeren. Na `if` staat een voorwaarde: deze is *waar* (`True`) of *niet waar* (`False`). De regel wordt afgesloten met een dubbele punt. De code die onder het if-statement staat wordt alleen uitgevoerd als de voorwaarde *waar* is. Alle regels die bij het if-statement horen moeten met dezelfde hoeveelheid spaties worden ingesprongen:

```
leeftijd = 16
if leeftijd < 18:
    print("Sorry, je bent niet stemgerechtigd")
    print("Kom over", 18 - leeftijd, "jaar terug")
```

```
> Sorry, je bent niet stemgerechtigd
> Kom over 2 jaar terug
```

Oefening

Alice heeft de volgende stukjes code met een if-statement geschreven, maar ze krijgt steeds een foutmelding. Kijk goed naar de voorbeelden hierboven en verbeter haar code:

```
getal = 9
if getal >= 10
    print("Dat getal heeft twee of meer cijfers.")
```

```
>   Cell In[4], line 2
>     if getal >= 10
>                    ^
> SyntaxError: expected ':'
```

```
temperatuur = -2
if temperatuur <= 0:
print("Het vriest!")
```

```
>   Cell In[5], line 3
>     print("Het vriest!")
>     ^
> IndentationError: expected an indented block after 'if' statement on line 2
```

```
antwoord = input("Wil je doorgaan? (ja/nee)")
if antwoord = "ja":
    print("Oké")
```

```
>   Cell In[6], line 2
>     if antwoord = "ja":
>        ^
> SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
```

Hint: Kijk nog eens goed naar de vergelijkingsoperatoren. = betekent niet hetzelfde als ==!

```
wachtwoord = input("Wat is het wachtwoord?")
if wachtwoord == "Super$ecret":
    print("Correct!")
      print("Je bent nu ingelogd.")
```

```
>   Cell In[7], line 4
>     print("Je bent nu ingelogd.")
>     ^
> IndentationError: unexpected indent
```

```
geboortejaar = 1953
fi geboortejaar > 1945 and geboortejaar < 1965:
    print("Ok, boomer")
```

```
>   Cell In[8], line 2
>     fi geboortejaar > 1945 and geboortejaar < 1965:
>        ^
> SyntaxError: invalid syntax
```

Else

Tot nu toe hebben we alleen code uitgevoerd als de voorwaarde waar is, of die code overgeslagen. Het komt vaak voor dat je iets wilt doen als de voorwaarde waar is, of iets anders als de voorwaarde niet waar is. Daarvoor kunnen we gebruik maken van `else`:

```
btwTarief = "laag" # "laag" of "hoog"
prijs = 12

if btwTarief == "hoog":
    print("Je betaalt 21% btw.")
    prijs = prijs * 1.21
else:
    print("Je betaalt 9% btw.")
    prijs = prijs * 1.09

print("Prijs (incl. btw):", prijs)
```

```
> Je betaalt 9% btw.
> Prijs (incl. btw): 13.080000000000002
```

De code onder `else:` wordt alleen uitgevoerd als de voorwaarde niet waar is. Net als bij het if-statement moeten de regels die bij elkaar horen onder else dezelfde indentatie hebben. Het is ook belangrijk dat de `else:` regel dezelfde indentatie heeft als de `if ...:` regel waar die bij hoort.

Oefening

Bob heeft een aantal programma's met een if/else-statement geschreven, maar hij krijgt steeds een foutmelding. Kijk nog eens goed naar de voorbeelden en verbeter Bobs code:

```
totaalBedrag = 22
if totaalBedrag < 20:
    print("De verzendkosten zijn €4,95")
else
    print("Je pakket wordt gratis verzonden!")
```

```
>   Cell In[9], line 4
>     else
>         ^
> SyntaxError: expected ':'
```

```
stoplicht = "groen"
if stoplicht == "rood":
    print("Je moet stoppen.")
    else:
        print("Je mag doorlopen.")
```

```
>   Cell In[11], line 4
>     else:
>     ^
> SyntaxError: invalid syntax
```

```
metSuiker = True
print("Tijd voor koffie!")
if metSuiker:
```

```
    print("Met suiker, natuurlijk.")
else:
      print("Zonder suiker.")
    print("Ik drink mijn koffie zwart.")
```

```
>   File <tokenize>:8
>     print("Ik drink mijn koffie zwart.")
>     ^
> IndentationError: unindent does not match any outer indentation level
```

Oefening

Carol heeft een programma geschreven dat bepaalt of een speler dubbele ogen heeft gegooid bij een spelletje Monopoly. Ze krijgt geen foutmeldingen, maar het programma werkt toch niet helemaal zoals Carol bedoeld had. Als de speler dubbel gooit, dan print het programma dat de volgende speler aan de beurt is, terwijl je juist dan nog een keer mag gooien. Andersom gaat het ook fout. Verbeter het programma zodat het wel voldoet aan de eisen.

```
import random

# Uit de range 1 tot 7, dus dat is 1 t/m 6
eersteWorp = random.randrange(1, 7)
tweedeWorp = random.randrange(1, 7)

print("Eerste worp:", eersteWorp, "Tweede worp:", tweedeWorp)
print("Je mag", eersteWorp + tweedeWorp, "stappen zetten.")

if eersteWorp != tweedeWorp:
    print("Je hebt dubbel gegooid, je mag nog een keer gooien!")
else:
    print("De volgende speler is aan de beurt.")
```

```
> Eerste worp: 2 Tweede worp: 2
> Je mag 4 stappen zetten.
> De volgende speler is aan de beurt.
```

Oefening

Hier een standaard schrijf-een-programma-met-if/else-oefening, zoals in de originele methode.

Elif

Soms zijn er meer dan twee opties, zoals bij de BTW: voor enkele producten geldt een tarief van 0%. Je kunt natuurlijk een tweede if-statement in het else-blok zetten om de juiste optie te kiezen:

```
btwTarief = "geen" # "hoog", "laag" of "geen"
prijs = 24

if btwTarief == "hoog":
    print("Je betaalt 21% btw.")
    prijs = prijs * 1.21
else:
    if btwTarief == "laag":
```

```
        print("Je betaalt 9% btw.")
        prijs = prijs * 1.09
    else:
        print("Je betaalt geen btw.")
        # prijs blijft dus gelijk

print("Prijs (incl. btw):", prijs)
```

```
> Je betaalt geen btw.
> Prijs (incl. btw): 24
```

Als je nog veel meer dan drie opties hebt, kan je code daardoor heel onoverzichtelijk worden. Gelukkig is er ook een manier om het korter op te schrijven met `elif`:

```
btwTarief = "geen" # "hoog", "laag" of "geen"
prijs = 24

if btwTarief == "hoog":
    print("Je betaalt 21% btw.")
    prijs = prijs * 1.21
elif btwTarief == "laag":
    print("Je betaalt 9% btw.")
    prijs = prijs * 1.09
else:
    print("Je betaalt geen btw.")
    # prijs blijft dus gelijk

print("Prijs (incl. btw):", prijs)
```

```
> Je betaalt geen btw.
> Prijs (incl. btw): 24
```

Hier wordt eerst gecontroleerd of het `btwTarief` gelijk is aan "hoog". Als dat zo is, wordt het eerste blokje code uitgevoerd en de rest overgeslagen. Zo niet, dan wordt dat blok overgeslagen en de vergelijking `btwTarief == "laag"` gecontroleerd. Als die waar is, dan wordt het tweede blokje code uitgevoerd (en de derde overgeslagen). Als die ook niet waar is, dan wordt de code in het derde blokje uitgevoerd.

Een `elif` komt altijd na een `if`-blok en in zo'n rijtje komt de `else` altijd als laatste. Je kunt zoveel `elif`s toevoegen tussen de `if` en `else` blokken als nodig is in je programma.

Oefening

Dave heeft een aantal programma's geschreven met `elif`, maar hij krijgt steeds een foutmelding. Kijk nog eens goed naar de voorbeelden en verbeter zijn code:

```
antwoord = input("Wil je doorgaan? (ja/nee)")
if antwoord == "ja":
    print("Oké, we gaan door!")
else if antwoord == "nee":
    print("Goed, dan stoppen we hier.")
else:
    print("Dat is geen goed antwoord.")
```

```
>   Cell In[14], line 4
>     else if antwoord == "nee":
```

```
>          ^
> SyntaxError: expected ':'
```

```
fruit = "peer"
if fruit == "appel":
    print("Een appeltje voor de dorst!")
else:
    print("Dat fruit ken ik niet.")
elif fruit == "banaan":
    print("Zo krom als een banaan.")
elif fruit == "peer":
    print("Hij is een toffe peer!")
elif fruit == "pruim":
    print("Deze uitdrukkingen zijn niet te pruimen.")
```

```
>   Cell In[13], line 6
>     elif fruit == "banaan":
>     ^
> SyntaxError: invalid syntax
```

Oefening

Eve heeft een programma geschreven dat de gebruiker groet afhankelijk van de tijd: goedemorgen tussen 6:00 en 12:00, goedemiddag tussen 12:00 en 18:00, goedenavond tussen 18:00 en 0:00 en goedenacht tussen 0:00 en 6:00. Ze krijgt geen foutmeldingen, maar het programma werkt toch niet helemaal zoals Eve bedoeld had. Om 6:15 print het programma bijvoorbeeld "Goedenavond!", terwijl het dan natuurlijk ochtend is. Dat gebeurt ook om bijvoorbeeld 12:43 (in de middag) en 0:26 ('s nachts). Verbeter het programma zodat het wel voldoet aan de eisen.

Tip: vervang de regel `uur = datetime.now().hour` door bijvoorbeeld `uur = 6`, om te kijken welke groet geprint wordt om 6 uur 's ochtends.

```
from datetime import datetime

uur = datetime.now().hour

if uur > 0 and uur < 6:
    print("Goedenacht!")
elif uur > 6 and uur < 12:
    print("Goedemorgen!")
elif uur > 12 and uur < 18:
    print("Goedemiddag!")
else:
    print("Goedenavond!")
```

```
> Goedemorgen!
```

Hint: Wat is het verschil tussen > en >= en tussen < en <=?

Oefening

Fred heeft een kluis gemaakt waarbij de code afhangt van de datum: de code bestaat uit vier cijfers, waarbij de middelste twee de dag van de maand aanduiden. Het eerste cijfer van de code is een 2 en de laatste een 8. Hij krijgt geen foutmeldingen, maar het programma werkt toch niet helemaal zoals Fred bedoeld had. Op de 12de van de maand zou de code `2128` moeten zijn,

maar de kluis gaat niet open. Verbeter het programma zodat de kluis wel met de juiste code geopend kan worden.

```
from datetime import datetime

dag = datetime.now().day

if dag < 10:
    dagCode = "0" + str(dag)
else:
    dagCode = str(dag)

kluisCode = "2" + dagCode + "8"

invoerCode = input("Wat is de code van de kluis?")

if len(invoerCode) != 4:
    print("Dat is geen geldige code.")
elif invoerCode == dagCode:
    print("De kluis is geopend.")
else:
    print("Dat is niet de juiste code.")
```

```
> Dat is niet de juiste code.
```

Oefeningen

Hier volgen nog 7 schrijf-een-programma-met-if/elif/else-oefeningen, zoals in de originele methode.