Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Rendement van objectgeoriënteerd programmeeronderwijs

Auteurs
Ernst Koldenhof
Universiteit Utrecht
Email: EKoldenhof@lekenlinge.nl

Johan Jeuring
Universiteit Utrecht
Email: johanj@cs.uu.nl

Sandra Ruth
Email: sandraADruth@gmail.com

Samenvatting
Programmeren neemt een steeds belangrijker rol in binnen het Informaticaonderwijs. Voor het programmeeronderwijs zijn er drie methoden op de Nederlandse markt, waarvan er twee, Enigma en BlueJ, een onderdeel voor objectgeoriënteerd programmeren hebben. Om de leeropbrengst van deze methoden te meten, zijn we een onderzoek gestart waarin deze methoden met elkaar worden vergeleken. In dit artikel beschrijven we de resultaten van een pilot van dit onderzoek met de BlueJ methode. We onderzoeken welke suggesties uit de literatuur zijn toegepast, en wat de vakinhoudelijke vorderingen zijn na een serie van 20 lessen. Hiervoor zijn onafhankelijk van de methode twee toetsen, een tussentijdse en een eindtoets, samengesteld. Uit de vakinhoudelijke vorderingen blijkt dat leerlingen fouten goed kunnen herkennen en dat zij in staat zijn om begrippen te omschrijven. Het implementeren van een klasse aan de hand van een klassendiagram gaat hen goed af, maar ze scoren slechter op de implementatie van een klassendiagram met gekoppelde klassen.

Trefwoorden
Objectgeoriënteerd programmeren, informaticaonderwijs, voortgezet onderwijs, leren programmeren, onderwijsrendement

Rendement van objectgeoriënteerd programmeeronderwijs

Een pilot-experiment met de BlueJ methode

1 Inleiding

Leren programmeren is moeilijk. Programmeervakken hebben vaak minder goede resultaten dan leerlingen en docenten verwachten.

Al jaren lang proberen onderzoekers de oorzaken van de slechte leeropbrengst te verklaren en te zoeken naar mogelijkheden om het leren programmeren te verbeteren (Robins, 2003). Merriënboer en Paas (Merriënboer, 1990) kijken naar hoe experts programma's ontwikkelen en gebruiken deze observaties om te bepalen welke kennis en vaardigheden nodig zijn voor het programmeren.

Andere onderzoekers volgen vooral beginnende programmeurs. Perkins et al. (Perkins, 1989) zien dat beginnende programmeurs twee soorten reacties t.o.v. de opdrachten en leerstof vertonen: een *stop*reactie of een *move*reactie. Bij een *stop*reactie doet de leerling niets meer omdat hij/zij niet zeker weet wat er gedaan moet worden. Deze stopper heeft een negatieve attitude t.o.v. het maken van fouten. Door Murphy et al. (Murphy, 2008) wordt dit gekoppeld aan de karaktereigenschappen van deze leerlingen. Volgens Murphy denkt de stopper dat hij/zij niet slim genoeg is. Murphy stelt dat deze leerling een beeld hanteert waarbij kennis en vaardigheden een vaste eigenschap is (*fixed mindset*). Tegenover het *stop*gedrag staat een gedrag waarbij een leerling wel verder gaat. Een leerling met dit gedrag noemt Perkins een *mover*. Volgens Murphy bezit een *mover* een zelfbeeld waarbij hij/zij meent door inspanning de meeste problemen te kunnen overwinnen (*growth mindset*). Hoe kunnen we *stop*gedrag omzetten in *move*gedrag? Murphy stelt dat een docent hier veel invloed op uit kan oefenen, maar dit is nog niet uitgebreid onderzocht.

Het bestuderen van de verschillen tussen beginnende programmeurs is een derde manier om naar het leren programmeren te kijken. Lahtinen (Lahtinen, 2007) kijkt vooral naar de prestaties van programmeurs. Deze prestaties kunnen in verschillende categorieën worden ingedeeld. Dit is een alternatief voor de Bloom taxonomie. Omdat leerlingen die goed kunnen programmeren op alle Bloom niveaus even goed scoren stelt Lahtinen dat de Bloom taxonomie voor programmeren minder geschikt is om het niveau van de programmeervaardigheden van een leerling te bepalen. Lister et al. (Lister, 2006; Whalley, 2006; Lopez, 2008) hebben een aantal toetsen en resultaten zowel met de Bloom taxonomie als met de SOLO taxonomie (Whalley, 2006) geanalyseerd. De SOLO taxonomie lijkt een beter passende indeling in niveaus te geven. Daarnaast laten zij zien dat het kunnen lezen en in eigen woorden beschrijven van programma's een noodzakelijke voorwaarde is voor het schrijven van programma's. Zij komen tot de conclusie dat de volgorde: lees programmeerregels, beschrijf programmeerregels, schrijf programmeerregels een recept is voor het leren programmeren.

In de bovenstaande onderzoeken wordt geen onderscheid gemaakt tussen procedurele talen en objectgeoriënteerde talen. Vooral onderwerpen als kennis van types, het gebruik van arrays, iteraties, conditionele constructies en combinaties daarvan worden onderzocht en getoetst. Het gebruik van objecten in relaties of verwijzingen is echter een belangrijk

onderdeel van de meeste moderne (versies van) programmeertalen. Bovendien vergt het leren programmeren in een objectgeoriënteerde programmeertaal een aantal andere vaardigheden dan in procedureel georiënteerde talen (Robins, 2003). Voor een evenwichtig onderwijsprogramma zijn beide vaardigheden nodig en moeten beide dus worden aangeboden.

Hoe leren leerlingen programmeren in het Nederlands voortgezet onderwijs? De tijd en inspanningen op dit gebied verschillen nogal per school (Geel, 2008). Het programmeren is door de urenuitbreiding van het vak bij de vernieuwing van de tweede fase belangrijker geworden, en de eindtermen in het examenprogramma geven een basis om dit onderwerp meer verdieping te geven (SLO, 2009).

Er zijn drie methoden voor het vak Informatica ontwikkeld: Informatica Actief, Enigma en Fundament. De methoden maken gebruik van Java als programmeertaal, maar met verschillende aanpakken en ontwikkelomgevingen. Informatica Actief geeft de lesstof van JavaLogo uit. Hierbij wordt voornamelijk de procedurele kant van Java gebruikt en wordt weinig aandacht besteed aan het objectgeoriënteerde deel van de taal. Enigma heeft twee modules waarin Java gebruikt wordt. De eerste legt net als Informatica Actief de nadruk op het procedurele aspect van het programmeren, en de tweede gaat over objectgeoriënteerd programmeren. De module van Fundament heet BlueJ en gebruikt ook BlueJ (www.bluej.org) als ontwikkelomgeving. Deze methode begint direct met objectgeoriënteerde begrippen als object, methode, klasse, etc.

Omdat programmeren een steeds belangrijkere component in het Informaticaonderwijs wordt, is het belangrijk te weten wat de leeropbrengst van de verschillende methoden is. Hiervoor is het belangrijk om toetsen op te stellen die onafhankelijk van deze methoden zijn. Dergelijke toetsen zouden vervolgens bij een schriftelijk examen gebruikt kunnen worden.

In dit onderzoek vergelijken we het rendement van twee lesmethoden, de BlueJ module van Fundament en de module objectgeoriënteerd programmeren van Enigma, op het gebied van objectgeoriënteerd programmeren.

Behalve de methode zijn er nog andere factoren die invloed hebben op de leeropbrengst, en deze beschrijven we in het volgende hoofdstuk. Om na te gaan hoe we methoden op een goede manier met elkaar kunnen vergelijken hebben we een pilot uitgevoerd met de BlueJ methode.

2 Factoren

De leeropbrengst van een lessenserie hangt niet alleen af van de gebruikte lesmethode, maar ook van de motivatie van een leerling, de manier waarop een docent les geeft, de leerdoelen, en de exameneisen spelen. We onderscheiden twee categorieën in deze factoren:

1 De attitude van de leerling t.o.v. het vak informatica.
2 Het Informaticaonderwijs:
   a De leerdoelen.
   b De gebruikte lesmethode.
   c Het curriculum dat de docent voor het vak heeft vastgesteld.
   d De perceptie van de leerlingen van de lessen.

In de volgende paragrafen gaan we kort op deze factoren in.

2.1 ATTITUDE VAN DE LEERLINGEN

De houding van leerlingen heeft een belangrijke invloed op de vakinhoudelijke vorderingen (Murphy, 2008). Bovendien kan de docent invloed op deze houding uitoefenen. Het is daarom belangrijk om te bepalen welke houding een leerling aan het begin van de lessen heeft, en of die houding verandert tijdens en/of na de lessenserie.

2.2 ONDERWIJS

De vorm en uitgangspunten van het onderwijs hebben invloed op de resultaten. We onderscheiden vier subcategorieën: de leerdoelen, de lesmethode, de invloed van de docent en de leservaringen van de leerlingen.

2.2.1 *Leerdoelen*

De leerdoelen van het vak informatica staan beschreven in het examenprogramma Informatica voor het VO (SLO, 2009) en de Handreiking schoolexamen informatica havo/vwo 2007 (Schmidt, 2007). Voor ons onderzoek gebruiken we het gedeelte voor het (objectgeoriënteerd) programmeren.

De begrippen en vaardigheden die in dit examenprogramma staan, zijn verwerkt in de twee lesmethoden die we gebruiken. Uiteraard moeten deze ook terug komen in leerdoelen en het programma dat de docent heeft opgesteld.

2.2.2 *De lesmethode*

De methodiek die wordt gebruikt om begrippen en vaardigheden aan te leren heeft veel invloed op de leeropbrengst. Voor programmeren hoort daar ook het instrumentarium bij zoals de ontwikkelomgevingen en de manier waarop deze wordt gebruikt. Omdat beginnende programmeurs bij het oplossen van problemen met behulp van een programma veel gebruik moeten maken van constructies en oplossingsstrategieën die ze nog moeten leren, is de mate waarin een ontwikkelomgeving daarbij ondersteuning biedt belangrijk.

De twee lesmethoden die we in ons onderzoek bestuderen, zijn BlueJ uitgegeven door Instruct en de module Objectgeoriënteerd programmeren met Java van Enigma. De BlueJ methode gebruikt een gelijknamige ontwikkelomgeving die sterk visueel is ingesteld en op die manier de objectgeoriënteerde begrippen ondersteunt. De methode begint direct met de objectgeoriënteerde begrippen zoals objecten, klasse, methode, constructor etc. en visualisatie van de ontwikkelomgeving sluit daar goed bij aan.

2.2.3 *Perceptie docent*

Ames en Archer (Ames, 1988) laten in hun onderzoek zien dat de sturing van de docent invloed heeft op de motivatie en vakinhoudelijke vorderingen van leerlingen. Het is daarom belangrijk te bepalen op welke manier de docent sturing geeft aan het onderwijs en in hoeverre dit door de methode wordt ondersteund.
Verder is het belangrijk te bepalen hoe de docent het curriculum van het vak informatica interpreteert (Geel, 2009) en dit wordt uitgevoerd. Dit geeft de context waarin de lessen en de toetsen plaatsvinden.

2.2.4 *Perceptie leerlingen*

Niet alleen de manier waarop de docent de lessen wil vormgeven en zijn visie daarop heeft invloed op de resultaten ook de perceptie van deze lessen door de leerling heeft invloed op de vakinhoudelijke resultaten. Een groot verschil tussen deze twee leiden in het algemeen tot slechte resultaten.

3 De onderzoeksmethode

We volgen de lessen informatica bij vier scholen. Twee scholen maken gebruik van de BlueJ methode en twee van de Enigma methode. Om na te gaan hoe de aanpak werkt, voeren we op één school een pilot-experiment uit. Deze school gebruikt de BlueJ methode. Deze methode wordt in twee parallel klassen van het vierde leerjaar van het VWO gegeven. Voor deze leerlingen zijn dit de eerste lessen programmeren. Eén klas (Klas 1) bestaat uit 15 leerlingen en de andere klas (Klas 2) uit 25 leerlingen. We gebruiken 18 lessen van 70 min (inclusief 2 toetsen van ongeveer 60 min). In dit artikel beschrijven we alleen de resultaten van het pilot experiment. Tijdens dit experiment zijn nog niet alle metingen of vragenlijsten ingevuld.

De indeling van de beschrijving van de methode volgt de indeling van de factoren die we in het vorige hoofdstuk hebben beschreven.

3.1 ATTITUDE VAN DE LEERLINGEN

Voor het bepalen van de attitude van een leerling hebben we twee vragenlijsten opgesteld, gebaseerd op (Heersink, 2010). Deze worden aan het begin en aan het einde van de serie lessen afgenomen. Omdat tijdens het pilot experiment de bovenstaande vragenlijst nog niet beschikbaar was, hebben we een andere lijst gebruikt. Met behulp van deze vragenlijst bepalen we de verdeling jongen/meisje, de verdeling over de keuzeprofielen, de motivatie om informatica te kiezen, de voorkennis, en de kennis van een aantal begrippen. Ook vragen we hoe leuk een leerling programmeren vindt in vergelijking met andere onderdelen van het vak informatica, en hoe hij/zij de moeilijkheidsgraad van programmeren ten opzichte van de andere onderdelen in dit vak inschat.

3.2 HET ONDERWIJS

3.2.1 *De leerdoelen*

We verifiëren dat de leerdoelen die in het examenprogramma staan, zijn verwerkt in de twee lesmethoden die we gebruiken.

3.2.2 *De lesmethode*

We analyseren of de methode suggesties voor het programmeeronderwijs aan beginners uit de literatuur heeft overgenomen, en inventariseren en categoriseren de vragen en opgaven van de methode.

3.2.2.1 *Gebruikte suggesties uit de literatuur*

We analyseren in hoeverre een methode de volgende aanwijzingen uit de literatuur volgt:
1 *Het oefenen met het lezen van programmaregels en dit in eigen woorden beschrijven*. Volgens Lister, Simon, Thompson, Whalley, Prasad en Lopez (Lister, 2006; Whally, 2006; Lopez, 2008) is dit een voorwaarde om programma's te kunnen schrijven.
2 *Het gebruik van de 4C/ID methode*. In de 4C/ID methode zoals die door Merriënboer en Paas is ontwikkeld (Merriënboer, 1990), wordt aan een groot project gewerkt waarbij uitgewerkte voorbeelden of oplossing voor deelproblemen worden aangeboden.
3 *Het werken in teams van twee*. Door meerdere programmeurs aan één softwareproduct te laten werken, communiceren deze onderling over het oplossen van fouten, programmastructuren en oplossingsstrategieën. Van één bepaalde vorm van samenwerking het "pair programming", is vastgesteld dat het een positief leereffect heeft (Nagappan, 2003; Salleh, 2008; McDowell, 2006; Simon, 2007).
4 *Het bevorderen van een goede leerstijl en het tegengaan van eventueel stopgedrag van een leerling.*

3.2.2.2 *De categorieën van de vragen en opgaven*

We verdelen de vragen/opgaven van een lesmethode in de volgende categorieën en geven hierbij aan welk niveau van de taxonomieën van Bloom (Bloom, 1956), SOLO, en 4C/ID hierbij horen.
1 Vragen/opgaven waarvan de antwoorden direct uit de lestekst te halen zijn. Bloom niveau 1.
2 Vragen/opgaven waarvan de antwoorden niet direct uit de lestekst te halen zijn maar waarvoor begrip van het betreffende onderwerp nodig is om het antwoord te formuleren. Bloom niveau 2.
3 Vragen en opgaven waarbij de kennis van het onderwerp moet worden toegepast in een situatie die sterk lijkt op de situatie die in de methode behandeld is. Bloom niveau 3, SOLO Unistructural, Merriënboer low-road.
4 Vragen en opgaven waarbij de kennis van het onderwerp moet worden toegepast in een situatie die niet lijkt op een situatie die in de methode behandeld is. Bloom niveau 3, SOLO Multistructural, Merriënboer high-road.

3.2.3 *Perceptie docent*

De perceptie van de docent is in het pilot-experiment niet bepaald.

3.2.4 *Perceptie leerlingen*

De perceptie van de leerlingen is in het pilot-experiment niet bepaald.

3.3 VAKINHOUDELIJKE VORDERINGEN

Om inzicht te krijgen in de vorderingen van de leerlingen nemen we twee toetsen af. De inhoud van de eerste toets stemmen we af op de behandelde stof. De toetsen bestaan uit verschillende typen opgaven. Het verschil tussen de tussentijdse toets en de eindtoets is enerzijds het aantal begrippen en constructies die getoetst worden en anderzijds is foutherkennen niet in de eindtoets opgenomen omdat de beschikbare tijd te kort is.

Er zijn vijf verschillende typen vragen:
1 *Herken constructies*. In deze vragen laten we een klasse zien en vragen we de leerling de regels te geven waar bepaalde begrippen zijn geïmplementeerd. Dit is een vraag op het eerste Bloom-niveau en heeft SOLO niveau Unistructural.
2 *Herken fouten.* In deze vragen laten we een klasse zien waarin een aantal fouten zit. Een leerling moet aangeven op welke regel welke fouten voorkomen. De vragen hebben SOLO niveau Multistructural, en zijn niet eenduidig in één Bloom niveau te plaatsen.
3 *Omschrijf begrippen*. In deze vragen moeten de leerlingen een aantal begrippen omschrijven. Deze vragen zijn op het tweede Bloom-niveau. Het is niet duidelijk op welk SOLO niveau deze vraag zit.
4 *Herken en beschrijf fouten*. In deze vragen laten we een klasse zien waar een aantal fouten in zit. Een leerling moet aangeven in welke regels de fouten voorkomen en beschrijven op welke manier de fouten kunnen worden gecorrigeerd. Hiervoor moet een leerling de begrippen en de bijbehorende constructies kennen. Het is moeilijk om aan dit onderdeel één Bloom niveau toe te kennen. Deze vraag koppelt het eerste Bloom niveau (kennen/weten) aan het tweede Bloom niveau (begrijpen). In de SOLO taxonomie is dit het Multistructural niveau.
5 *Implementeer een programma aan de hand van een klassendiagram*. In dit onderdeel geven we een klassendiagram, en moet de leerling het bijbehorende programma implementeren. De toetsen bevatten twee opgaven met verschillende moeilijkheidsgraden. De eerste bevat een klassendiagram met alleen een constructor, getters en setters. De tweede bevat een klassendiagram met gekoppelde klassen. Deze vraag ligt op het derde Bloom niveau en op het relationeel SOLO niveau.

De begrippen die we in deze toets gebruiken, halen we uit de begrippenlijst van BlueJ.

4 Resultaten

In dit hoofdstuk beschrijven we de resultaten van het pilot-experiment met de methode BlueJ.

4.1 ATTITUDE VAN DE LEERLINGEN

4.1.1 *Algemene gegevens*

In de twee atheneum 4 klassen is 60% jongen en 40% meisje. Het percentage meisjes in deze klassen ligt hoger dan in andere leerjaren. In de literatuur is het percentage meisjes voor dit vak vaak veel lager (Schmidt 2007-2). Verder heeft iets meer dan 68% een NT of NG profiel gekozen, 23% heeft een EM profiel, en 9% heeft een CM profiel.

NT	57%
NG	11%
EM	23%
CM	9%

AFBEELDING 1 Gekozen profielen

4.1.2 *Motivatie*

Slechts 2% van de leerlingen denkt informatica te gaan studeren, en 15% heeft informatica gekozen omdat zij denken dat het vak nuttige kennis voor een vervolgopleiding op kan leveren. Van de leerlingen wil 15% wel wat meer van het vak informatica weten en vindt 12% andere vakken minder boeiend. Iets meer dan de helft van de leerlingen, 54%, vindt informatica wel een interessant vak.

Informatica studeren	2%
Kennis voor vervolg studie	15%
Wil meer weten	15%
Andere vakken minder boeiend	12%
Interessant	54%
Anders	2%

AFBEELDING 2 Keuze motivatie

4.1.3 *De inschatting van de moeilijkheidsgraad*

Voorafgaand aan de lessen schat geen van de leerlingen de moeilijkheidsgraad van het onderdeel programmeren hoger in dan dat van andere onderdelen in dit vak. Door 39% van de leerlingen wordt dit lager ingeschat. 29% van de leerlingen lijkt het even makkelijk, en 32% heeft hier geen mening over.

Makkelijker	39%
Even makkelijk	29%
Moeilijker	0%
Weet het niet	32%

AFBEELDING 3 Inschatting moeilijkheid

4.1.4 *Voorkennis*

We delen de leerlingen in de volgende 3 categorieën in: veel kennis, een beetje kennis en weinig of geen kennis van programmeren. Deze indeling is gebaseerd op de voorkennis van programmeertalen die leerlingen opgeven. Van de 35 leerlingen geeft 72% aan dat ze weinig of geen kennis hebben, 14% geeft aan dat ze enige kennis heeft, en 14% geeft aan veel programmeerkennis te hebben.

Weinig of geen	71%
Een beetje	14%
Veel	14%

AFBEELDING 4 Inschatting eigen kennis

4.2 HET ONDERWIJS

4.2.1 *De lesmethode*

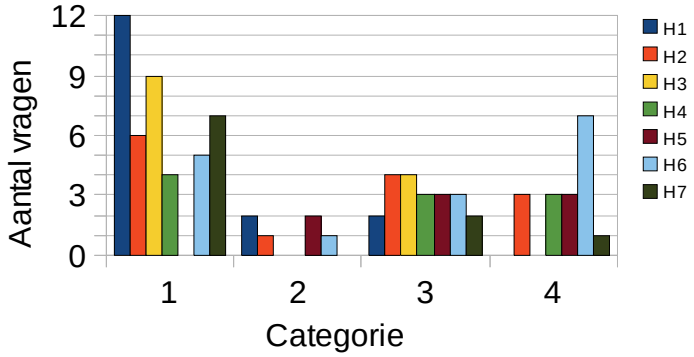
4.2.1.1 Gebruikte suggesties uit de literatuur

Veel van de in hoofdstuk 3 genoemde suggesties uit de literatuur komen niet terug in de BlueJ lesmethode.

1 *Het oefenen met het lezen van programma regels en dit in eigen woorden beschrijven*. De methode heeft geen opdrachten om in eigen woorden programma's of programma-onderdelen te beschrijven.
2 *Het gebruik van de 4C/ID methode*. Het ontwikkelen van software in een context zoals aanbevolen in de 4C/ID methode wordt in de BlueJ methode niet toegepast.
3 *Het werken in teams van twee*. Er komen geen opdrachten voor die het werken in teams van twee eist.
4 *Het bevorderen van een goede leerstijl*. De methode geeft geen voorbeelden of oefeningen met als doel de leerstijl van de leerling te veranderen. Ook zijn er geen aanwijzingen te vinden die aangeven hoe een negatieve houding ten opzichte van het maken van fouten verbeterd kan worden.

4.2.1.2 De categorieën van de vragen en opgaven

Voor de indeling van de vragen en opgaven hebben we de categorieën gebruikt zoals beschreven in 3.2.2.2. Het aantal vragen per categorie staat in afbeelding 5.



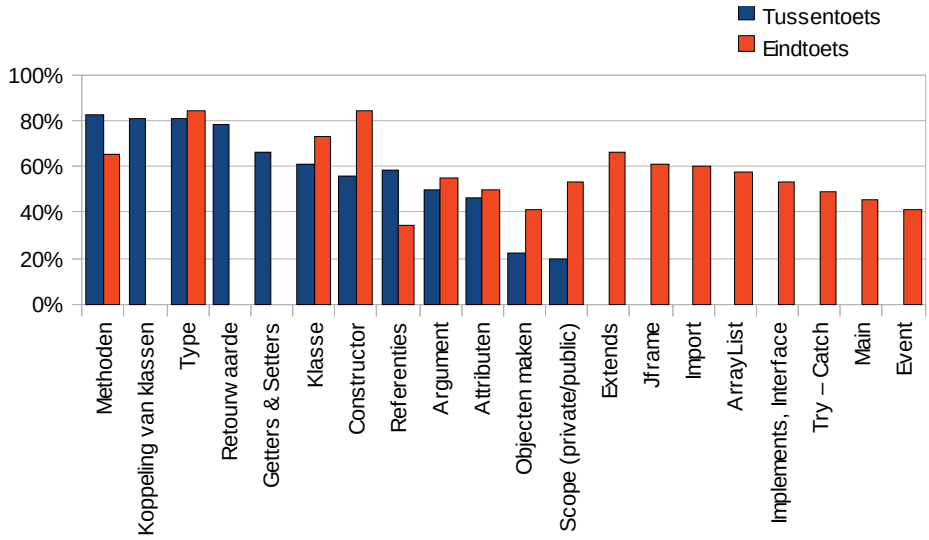
AFBEELDING 5 Aantal vragen en opgaven per categorie en hoofdstuk

We zien dat de meeste vragen in categorie 1 en 3 zitten.

3.4 VAKINHOUDELIJKE VORDERINGEN

3.4.1 *Herken constructies*

De resultaten op de vragen van de categorie herken constructies staan in afbeelding 6. In deze afbeelding vergelijken we de resultaten van de tussentijdse toets en de eindtoets.

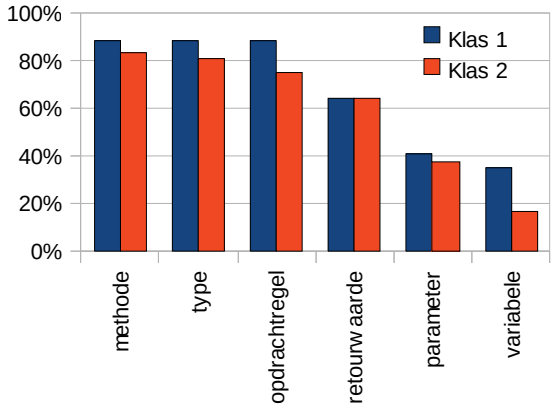


AFBEELDING 6 De resultaten van de tussentijdse- en eindtoets

De meeste begrippen worden in de eindtoets even goed of beter herkend dan in de tussentijdse toets. Alleen de resultaten voor de begrippen methode en referentie wijken af. De nieuwe begrippen Extends t/m Event worden door 60% van de leerlingen goed herkend.

4.3.2 *Herken fouten*

De resultaten van de vragen uit de categorie herken fouten staan in afbeelding 7.
We hebben deze categorie alleen getoetst in de tussentijdse toets.

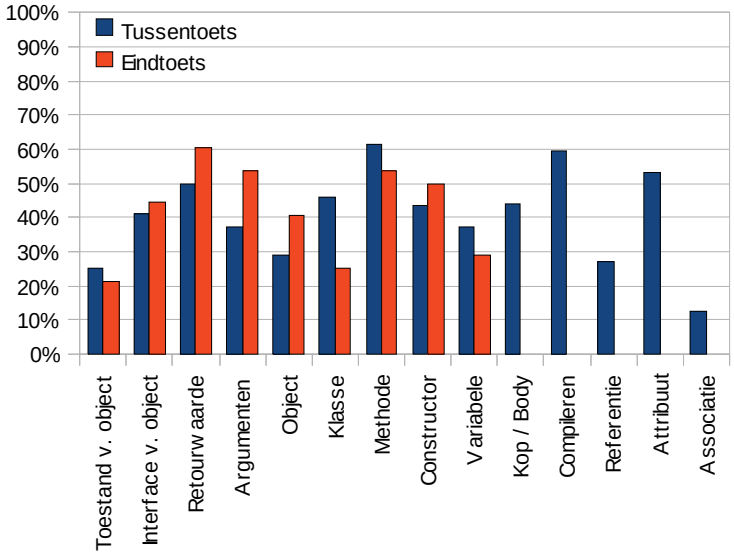


AFBEELDING 7 Het percentage goed herkende fouten

De resultaten van deze categorie vragen geven aan dat een drietal fouten goed wordt herkend. Fouten in de methode, een type en een opdrachtregel worden door ongeveer 80% van de leerlingen herkend, maar fouten in een parameter en variabele worden niet goed herkend.

4.3.3 *Omschrijf begrippen*

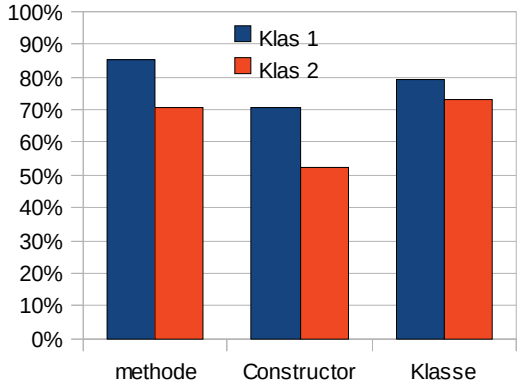
De resultaten van de vragen uit de categorie omschrijf begrippen staan in afbeelding 8. In deze afbeelding zien we dat de resultaten van de eindtoets vergelijkbaar zijn met die van de tussentijdse toets.



AFBEELDING 8 Resultaten begripsomschrijving

4.3.4 *Herken en beschrijf fouten*

De resultaten van de vragen uit de categorie herken en beschrijf fouten staan in afbeelding 9.

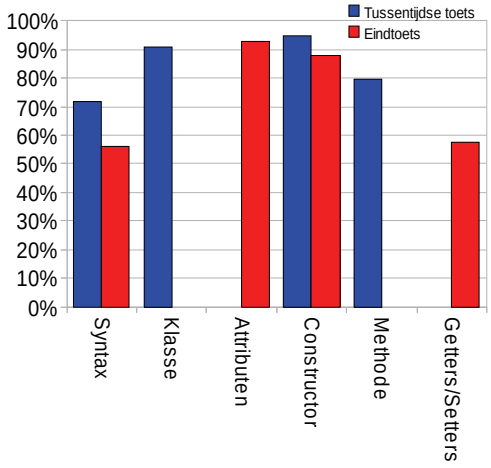


AFBEELDING 9 Het percentage goed beschreven fouten

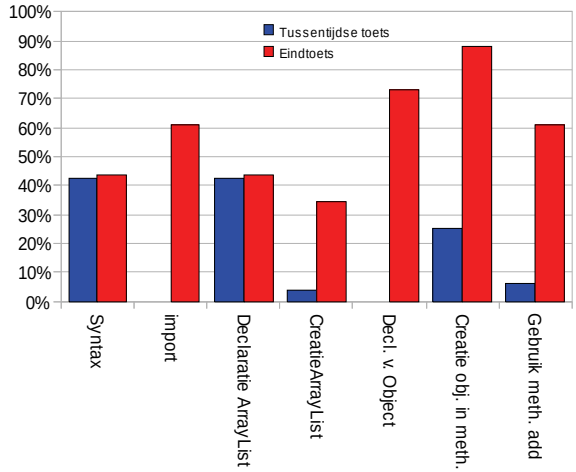
Deze resultaten laten zien dat fouten in de methode en klasse door minstens 70% van de leerlingen goed worden herkend en beschreven. Voor het herkennen en beschrijven van fouten in de constructor gaat het duidelijk minder goed.

4.3.5 *Implementeer klassendiagrammen*

De resultaten van het onderdeel waarin een klassendiagram geïmplementeerd moet worden, staan in afbeelding 10 en 11.



AFBEELDING 10 Implementatie van eenvoudige klassen



AFBEELDING 11 Implementatie van gekoppelde klassen

Deze twee afbeeldingen laten duidelijk zien dat leerlingen goed scoren bij het implementeren van een eenvoudig klassendiagram. Het implementeren van gekoppelde klassen wordt veel minder goed gedaan. Bij de eindtoets gaat het implementeren van gekoppelde klassen beter, maar ook daar blijft het gemiddelde onder de 50%.

5 Discussie

5.1 ATTITUDE VAN DE LEERLINGEN

Het is bekend dat de leerstijl van jongens in het algemeen anders is dan die van meisjes (Milgram, 2007; Murphy, 2006). Deze leerstijlen hebben invloed op de leeropbrengst (Mul, 2006). Verschillende verhoudingen tussen jongens en meisjes in de klas kan zo invloed hebben op de inhoudelijke vorderingen. In het algemeen zou het goed zijn als leerstijlen expliciet worden vastgelegd.
Leren programmeren is moeilijk, maar de leerlingen uit het pilot-experiment denken juist dat het relatief makkelijk is. Wordt dit verwachtingspatroon bijgesteld? In het pilot experiment hebben we daar geen informatie over maar in het experiment waarbij we meerdere scholen betrekken hebben we dat wel.
Meer dan een kwart van de leerlingen in deze klas geeft aan een behoorlijke voorkennis van programmeren te hebben. In vergelijking met andere vakken op het VO is dat nogal veel. De antwoorden op de vragen over de betekenis van een aantal begrippen geven echter aan dat de opgegeven kennis over de gevraagde begrippen juist niet groot is.

5.2 HET ONDERWIJS

5.2.1 *De lesmethode*

5.2.1.1 Gebruikte suggesties uit de literatuur

Het is opvallend dat vrijwel geen van de suggesties uit de literatuur in deze lesmethode zijn toegepast. In de methode is er wel genoeg lesmateriaal om deze suggesties toe te passen. Uiteraard zijn niet alle suggesties uit de literatuur even makkelijk aan de methode toe te voegen. We geven hieronder enkele voorbeelden.

1 In de methode worden verschillende begrippen uitgelegd aan de hand van diverse code fragmenten. Deze code fragmenten kunnen gebruikt worden voor opdrachten waarbij een leerling een beschrijving hiervan moet geven.
2 De opdrachten uit de lesmethode zijn te klein voor de 4C/ID methode. Hiervoor zou een nieuwe opdracht gemaakt moeten worden, die gebruik maakt van de context en voorbeelden uit de methode.
3 De beschikbare opdrachten lenen zich goed voor "pair programming". Hiervoor hoeft alleen de instructie van sommige opdrachten aangepast te worden.

5.2.1.2 De categorieën van de vragen en opgaven

Uit de analyse van de vragen en opgaven valt op dat er vrijwel geen vragen en opgaven in de tweede categorie zitten (zie afbeelding 5). Hierbij moeten we wel opmerken dat de vragen en opgaven waarbij een begrip beschreven moet worden vrijwel allemaal in categorie 1 geplaatst zijn. Deze beschrijvingen zijn letterlijk in de methode terug te vinden. Vragen waarin een uitleg van een aantal programmaregels wordt gevraagd komen niet voor. Deze zouden in de tweede categorie vallen. Vragen uit de verschillende hoofdstukken zijn verschillend over de vier categorieën verdeeld. Het lijkt logisch dat dit te maken heeft met het onderwerp van dat hoofdstuk. Toch zien we deze verdeling niet op die manier in de afbeelding terug.

5.3 VAKINHOUDELIJKE VORDERINGEN

De tussentijdse toets laat zien dat de kennis van een aantal begrippen na de helft van het aantal lessen goed is. De eindtoets laat zien dat de kennis van begrippen uit de eerste toets niet veel is toegenomen. Wel worden er meer begrippen getoetst en deze nieuwe begrippen worden wel goed herkend. De geringe toename van de score van de begrippen uit de eerste toets kan worden veroorzaakt door het grote aantal extra begrippen en de toename van complexe constructies die in deze laatste periode van deze lessen serie worden behandeld. Hierdoor kan de automatisering en het vasthouden van de juiste betekenis van de reeds geleerde begrippen worden gehinderd.

Verder scoren de leerlingen op de herken categorieën beter dan op de beschrijf categorieën. Het in eigen woorden beschrijven van een begrip of situatie vergt naast kennis van het specifieke begrip ook de nodige taalvaardigheid. Toch is het beschrijven van programma fragmenten in eigen woorden een belangrijk onderdeel in het leren programmeren zoals dat in de SOLO taxonomie naar voren komt (Whalley, 2006).

Bij het implementeren van klassendiagrammen zien we grote verschillen tussen de twee soorten implementatie. Vooral in de tussentijdse toets wordt voor de implementatie van het gekoppelde klassendiagram slecht gescoord. Bij deze implementatie worden vooral fouten gemaakt bij het definiëren en creëren van objecten. In de eindtoets gaat dat beter, maar toch blijven de scores op deze implementatie minder goed. Verder valt op dat er een correlatie lijkt te bestaan tussen de score voor de syntax en voor de andere onderdelen van de opgave. Er worden bij het implementeren van gekoppelde klassen veel meer syntax fouten gemaakt. Het lijkt alsof het begrip van de implementatie invloed heeft op het wel of niet maken van syntax fouten.

6 Conclusies

6.1 DE ATTITUDE VAN LEERLINGEN

Uit de attitude test zien we dat, indien de verhoudingen jongens/meisjes tussen de klassen verschillend is, er rekening met de verschillende leerstijlen gehouden moet worden.

Verder wordt in de vervolgonderzoeken voor en na de lessen vastgesteld hoe de perceptie van de leerlingen betreffende de moeilijkheidsgraad van het onderdeel programmeren is, zodat kan worden vastgesteld of dit verandert.

Tenslotte is de ingeschatte voorkennis van de leerlingen veel groter dan hij is. In ieder geval is de kennis van objectgeoriënteerde begrippen erg laag en zijn deze lessen dus voor vrijwel iedereen de eerste kennismaking met objectgeoriënteerd programmeren.

6.2 HET ONDERWIJS

6.2.1 *De lesmethode*

Vrijwel geen van de suggesties uit de literatuur over programmeeronderwijs worden in de BlueJ lesmethode toegepast. Er zijn goede mogelijkheden om een aantal van deze aanbevelingen aan de methode toe te voegen. Het beschrijven van codefragmenten en het stimuleren van pair programming is gemakkelijk toe te voegen. Maar het is ook mogelijk om opdrachten in de 4C/ID stijl toe te voegen.

6.3 VAKINHOUDELIJKE VORDERINGEN

De resultaten van de toetsen laten zien dat objectgeoriënteerde begrippen zoals klasse, methode en constructor niet noodzakelijk meer tijd en oefening kosten dan procedurele begrippen als variabele en argument. De gecompliceerdere begrippen zoals de implementatie van gekoppelde klasse scoren laag.

De score voor het herkennen van begrippen zijn beter dan de scores voor het beschrijven van begrippen. Dit geldt ook voor het herkennen en beschrijven van fouten. De toetsen maken niet duidelijk of het beschrijven van begrippen moeilijk is door onvoldoende taalvaardigheid, of door beperkte kennis van het begrip.

De scores van tussentijdse toets en de eindtoets verschillen voor de verschillende onderdelen niet veel van elkaar en lijken daarom betrouwbaar. Het maken van een opdracht volgens de 4C/ID methode, en vervolgens het observeren van de vorderingen van de leerlingen, kan een beeld geven van vaardigheden in het toepassen van de kennis in een nieuwe situatie.

Referenties

Ames, 1988, Carole Ames and Jennifer Archer, Achievement Goals in the Classroom: Students' Learning Strategies and Motivation Processes, Journal of Educational Psychology 80(3), 260 - 267, 1988.

Bloom, 1956, Bloom's taxonomy. http://www.odu.edu/educ/roverbau/Bloom/blooms_taxonomy.html, 1956.

Geel, 2008, Femke van Geel, Een overzicht van het Programmeeronderwijs op Middelbare scholen in Nederland, Universiteit Twente, http://www.utwente.nl/elan/huidige_studenten/overig/OvO/OvO-inf/Eindverslag%20INF.pdf

Heersink, 2010, Daniel Heersink, Barbara M. Moskal. Measuring high school students' attitudes toward computing. In SIGCSE, 446 - 450, 2010

Lahtinen, 2007, Lahtinen, E., A Categorization of Novice Programmers: A Cluster Analysis Study, 19th Programming Psychology Interest, 32- 41, 2007.

Lister, 2006, Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In SIGCSE, 118-122, 2006.

Lopez, 2008, Lopez, M., Whalley, J. L., Robbins P. and Lister, R. Relationships between reading, tracing and writing skills in introductory programming. In ICER, 2008.

McDowell, 2006, Charlie McDowell, Linda Werner, Heather E. Bullock, and Julian Fernald, Pair programming improves student retention, confidence and program quality, COMMUNICATIONS OF THE ACM 49(8), 2006.

Merriënboer, 1990, Jeroen J.G. van Merriënboer and Fred G.W.C. Paas. Automation and schema acquisition in learning elementary computer programming: Implications for the design of practice. Computers in Human Behaviour 6:273–289, 1990.

Milgram, 2007, Donna Milgram, Gender Differences In Learning Style Specific To Science, Technology, Engineering And Math – Stem, http://ezinearticles.com/?Gender-Differences-In-Learning-Style-Specific-To-Science,-Technology,-Engineering-And-Math---Stem&id=658953

Mul, 2006, Hinke M. Mul, Match or Mismatch, De invloed van leerstijlen op ontdekkend en ervaringsleren.

Murphy, 2006, Laurie Murphy, Renée McCauley, Suzanne Westbrook, Brad Richards, Briana B. Morrison and Timothy Fossum, Women Catch Up: Gender Differences in Learning Programming Concepts. In SIGCSE, 2006.

Murphy, 2008, Laurie Murphy, Lynda Thomas, Dangers of a fixed mindset: implications of self-theories research for computer science education. In ITICSE, 2008.

Nagappan, 2003 Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, Suzanne Balik, Improving the CS1 Experience with Pair Programming. In SIGCSE, 2003.

Perkins, 1989, Perkins, D.N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. Conditions of learning in novice programmers. In E. Soloway & J.C. Spohrer (Eds.), Studying the novice programmer, 261–279, Hillsdale, NJ: Lawrence Erlbaum, 1989.

Robins, 2003, Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. Computer Science Education 13(2):137–172, 2003.

Salleh, 2008, Norsaremah SALLEH, Emilia MENDES, and John GRUNDY, Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, TSE-2008-10-0357

Schmidt, 2007, Victor Schmidt. Handreiking schoolexamen informatica havo/vwo. 2007.

Schmidt, 2007-2 Victor Schmidt. Aantrekkelijk informatica-onderwijs voor meisjes en jongens, Enschede, januari 2007 VO/ICT/3584.001/07-001

Simon, 2007, Beth Simon and Brian Hanks, First Year Students' Impressions of Pair Programming in CS1. In ICER, 2007.

SLO, 2009, Nationaal expertisecentrum leerplanontwikkeling SLO. Eindexamenprogramma informatica HAVO/VWO, 2009.

Whalley, 2006, Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar A.P.K. and Prasad, C. An Australasian Study of Reading and Comprehension Skills in Novice Programmers, using the Bloom and SOLO Taxonomies. In Proc. of the 8th Australasian Computing Education Conference, Conferences in Research in Practice in Information Technology, 52: 243-252, 2006.