Programmeren met PRIMM didactisch bekeken

PRIMM is een didactische aanpak die focust op het aanbrengen van programmeerconcepten. Maar waarom zou je deze methodiek gebruiken, wat is de meerwaarde van samenwerkend leren hierbij en hoe kan je dit allemaal in jouw lespraktijk realiseren?

Deze good practice maakt deel uit van het PWO-project:
“De effecten van CSCL (Computer Supported Collaborative Learning) in de virtuele en fysieke ruimte op sociaal en cognitief gedrag”

Lisa Caenen – Stijn Coussement – Peter Dejonckheere – Hermien Dekoninck – Tony Opsomer
2020-2022 hogeschool VIVES

versie 2022-11-16

Wat is PRIMM en waarom zou je het gebruiken?

PRIMM is een manier om nieuwe programmeerconcepten aan te reiken aan beginners. De grote kracht van deze methodologie zit enerzijds in haar heldere **lesstructuur** met stapsgewijze lesopbouw en anderzijds in de **flexibiliteit** die je als leerkracht behoudt t.a.v. de gebruikte werkvormen.

De lesstructuur bestaat uit **vijf stappen**: Predict, Run, Investigate, Modify en Make. Na elke stap krijgt een leerling meer inzicht in en eigenaarschap over de code. De belangrijkste klemtonen van elke fase vind je hieronder terug:

1 **Predict** (voorspel):
- Leerkracht: moedig discussie aan;
- Leerlingen: achterhaal hoe de code functioneert zonder deze evenwel uit te voeren.

2 **Run** (voer uit):
- Leerkracht: bied leerlingen de code die ze moesten voorspellen integraal aan in een geschikte programmeeromgeving zodat geen code ingetypt hoeft te worden;
- Leerlingen: voer de code uit en controleer de uitvoer van het programma t.o.v. de gemaakte voorspelling.

3 **Investigate** (onderzoek):
- Leerkracht: maak gebruik van diverse verwerkingsopdrachten (bv. traceren van het verloop van waarden doorheen een programma, uitvoeren van kleine opdrachtjes en beantwoorden van vragen, …) om leerlingen de werking van de code beter te doen begrijpen en het ontstaan van misconcepties te voorkomen;
- Leerlingen: ga gericht op onderzoek om de werking van de code volledig te doorgronden.

4 **Modify** (wijzig):
- Leerkracht: voorzie opdrachten waarbij leerlingen geleidelijk de code leren aanpassen om extra functionaliteiten in te bouwen zonder hen daarbij in de valkuilen van zelfontdekkend leren te duwen;
- Leerlingen: pas toe wat je in de vorige stappen geleerd hebt.

5 **Make** (maak):
- Leerkracht: voorzie een of meer compacte, nieuwe programmeeropdrachten waarin leerlingen de geleerde programmeerconcepten in een andere context toepassen;
- Leerlingen: transfereer de opgedane kennis en vaardigheden naar een nieuwe context.

De flexibiliteit waarmee je als leerkracht met PRIMM aan de slag kan, is een groot pluspunt. Zo kan je ervoor opteren om binnen elke PRIMM-fase een meer **leerkrachtgestuurde aanpak of** eerder **leerlinggecentreerde werkwijze** toe te passen.

Wanneer leerlingen aan de slag gaan, is het **sterk aanbevolen** om hen per twee te laten **samenwerken**[1] al kunnen ze indien nodig ook individueel aan de slag mits de bevindingen regelmatig klassikaal teruggekoppeld worden. Eventueel kan een oefening uit de make-fase ook als summatief toetsinstrument gebruikt worden

[1] Zie uitleg omtrent pair programming op de volgende pagina's.

Samenwerkend leren m.b.v. pair programming toepassen in de klas

Samenwerkend leren verwijst naar een onderwijssituatie waarbij leerlingen samen verantwoordelijk zijn voor het bereiken van een gemeenschappelijk doel. Hiervoor voeren ze taken uit in interactie met elkaar. Mits goed toegepast én begeleid, verbetert samenwerkend leren de leerresultaten van leerlingen.

Samenwerkend leren is effectiever wanneer leerlingen bij een opdracht van elkaar afhankelijk zijn en dus ervaren dat zij elkaar nodig hebben om de opdracht te vervullen. Door pair programming toe te passen, speel je in op dat aspect.

Pair programming is een didactische aanpak waarbij leerlingen per twee samenwerken om een programmeerprobleem op te lossen. Leerlingen die weinig zelfvertrouwen hebben of die nog niet zo'n geavanceerde programmeervaardigheden verworven hebben, hebben baat bij deze manier van werken.

In de klassieke benadering van pair programming maken beide leerlingen gebruik van één computer. Eén leerling (= de driver) focust op de implementatie van het opgeloste probleem door de computer te bedienen of de antwoorden op papier te noteren en voert uit wat de andere leerling aangeeft. De andere leerling (= de navigator) houdt dan weer de taakdoelstelling in de gaten, zorgt dat het probleem/de vraag opgelost raakt, merkt eventuele fouten op en geeft hiervoor de nodige instructies aan de driver. Regelmatig wisselen driver en navigator van rol.

De belangrijkste voordelen van pair programming zijn:

- Een vermindering van de cognitieve belasting voor elke leerling omdat de taken gescheiden worden. Zo focust de driver zich op het noteren van antwoorden of het correct intypen van code en hoeft deze zich dus veel minder te concentreren op intensievere taken zoals het lezen van de instructies, het daadwerkelijk oplossen van het probleem en het ontwerpen van een algoritme omdat deze taken door de navigator uitgevoerd worden;
- Een toename van het zelfvertrouwen om een oplossing te vinden voor het gestelde probleem;
- Een betere kwaliteit van het afgeleverde programmeerwerk (minder fouten, verhoogde efficiëntie en elegantere code) omdat driver en navigator elkaar aanvullen

Om pair programming succesvol als hefboom voor inzicht in de aangereikte programmeerconcepten te kunnen toepassen, is het echter belangrijk om rekening te houden met volgende aspecten:

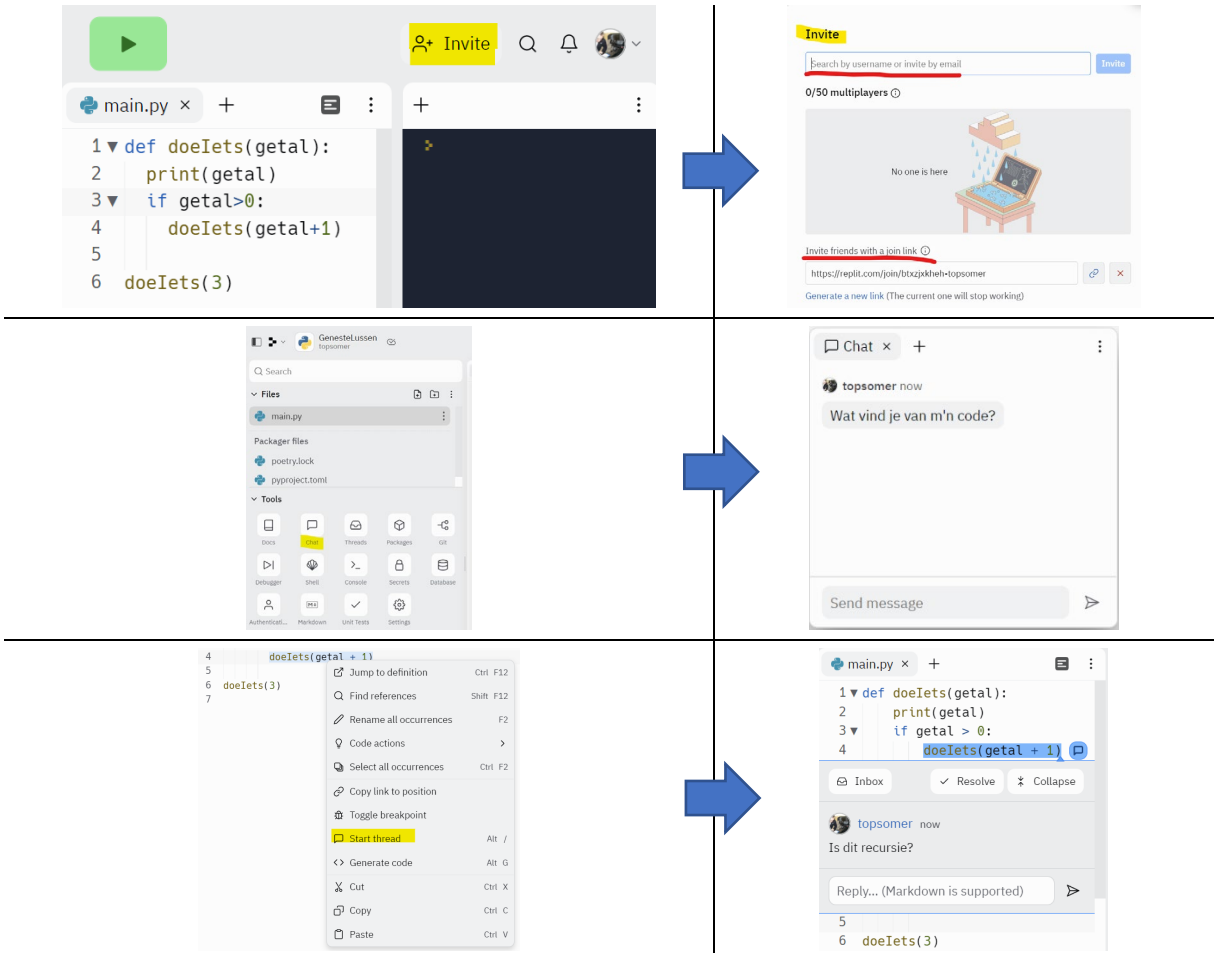
- Laat leerlingen niet zelf kiezen met wie ze samenwerken. Wanneer vrienden samenwerken, is er soms onvoldoende ruimte voor discussie over de invulling van de opdracht en ontstaat vaak snel afleiding. Je bepaalt dus bij voorkeur zelf de groepen en probeert daarbij gemengde en cognitief heterogene niveaugroepjes te bekomen. Zwakkere leerlingen presteren daar immers beter en sterkere studenten worden in dat geval meer uitgedaagd om samenwerkingsvaardigheden te ontwikkelen en leerinhouden te expliciteren. Extreem sterke leerlingen kan je dan weer beter in cognitief homogene groepjes verdelen. Geef sowieso steeds aan wie initieel welke rol moet opnemen;
- Zowel driver als navigator moeten gemotiveerd zijn voor de opdracht en er evenwaardig willen toe bijdragen;
- Zorg dat driver en navigator zich aan hun rol houden zonder dat de ene de andere domineert en bewaak dat beiden op elk moment aan eenzelfde taak werken. Goede communicatie tussen

driver en navigator is extreem belangrijk. Dit vergt niet alleen de nodige oefening door de leerlingen maar ook en vooral voldoende adequate begeleiding door jou als leerkracht. Zo ga je actief rond in de klas en luister je of de groepjes daadwerkelijk gericht bezig zijn met de taak, (bv. leggen ze elkaar uit wat ze willen doen, bespreken ze concepten, opperen ze ideeën). Indien nodig stimuleer je dit gedrag (bv. door het voor te doen);

- Controleer of driver en navigator regelmatig van rol wisselen. Je kan hiervoor op basis van tijd (bv. om de 10') werken of het aantal opgeloste deelvragen als referentie gebruiken.
- Durf de groepssamenstelling regelmatig te wisselen (bv. na elke PRIMM-fiche).

Als alternatieve benadering van pair programming kan gebruik gemaakt worden van een ontwikkelomgeving waarin realtime samenwerkingsmogelijkheden aanwezig zijn. In dat geval werken zowel de driver als de navigator afzonderlijk aan een computer. Code die door de driver ingegeven wordt, wordt ogenblikkelijk zichtbaar voor de navigator waarop deze middels chat of annotaties de code of het denkproces kan vormgeven, bijsturen, enz. Vanzelfsprekend kan je op deze manier ook programmeeronderwijs op afstand faciliteren.

Replit biedt – zelfs al heb je slechts een gratis basisaccount – dergelijke functies aan onder de noemer Multiplayer. Meer info vind je op https://docs.replit.com/tutorials/pair-programming-using-multiplayer-with-repl-it.



Hoe kan je met PRIMM aan de slag in jouw lessen programmeren?

De 9 aangereikte PRIMM-fiches omtrent programmeren in Python helpen je om het gros van de leerplandoelen omtrent computationeel denken in de tweede graad van de doorstroomrichtingen in het secundair onderwijs te realiseren of zelfs te verdiepen.

Zo vind je in het leerplan wiskunde C van de tweede graad van het Katholiek Onderwijs Vlaanderen volgende doelstelling terug:

II-WisS-d60	De leerlingen ontwerpen algoritmes om problemen digitaal op te lossen.
	Concepten van computationeel denken: decompositie, patroonherkenning, abstractie, algoritme. Organisatie, modellering, simulatie en digitale representatie van informatie. Debuggen. Principes van programmeren: opeenvolging, herhalingsstructuur, keuzestructuur. Ingebouwde functies. Elementen van programmeertalen: variabelen, datatypes, operatoren, parameters, condities, procedures of functies.

Gelijkaardige elementen vind je overigens ook in het leerplan basisvorming – doorstroom voor de tweede graad secundair onderwijs van het GO! terug:

ET NR	Computationeel denken en handelen.
4.5	De leerlingen ontwerpen algoritmen om problemen digitaal op te lossen.
Met inbegrip van kennis	
Feitenkennis	
Conceptuele kennis	
- Concepten van computationeel denken: decompositie, patroonherkenning, abstractie, algoritme - Organisatie, modellering, simulatie en digitale representatie van informatie - Debuggen: testen en bijsturen - Principes van programmeren: sequentie, herhalingsstructuur, keuzestructuur - Ingebouwde functies - Elementen van programmeertalen: variabelen, datatypes, operatoren, parameters, condities, procedures of functies	
Procedurele kennis	
- Toepassen van principes van computationeel denken: decompositie, patroonherkenning, abstractie, algoritme - Toepassen van principes van organisatie, modellering, simulatie en digitale representatie van informatie - Toepassen van principes om te debuggen - Toepassen van principes van programmeren: sequentie, herhalingsstructuur, keuzestructuur - Toepassen van controlestructuren en eenvoudige gegevensstructuren bij het formuleren van algoritmen - Toepassen van principes om algoritmen bestaande uit een aantal samenwerkende procedures te ontwerpen en te implementeren in een programmeeromgeving	

De manier waarop je de uitgewerkte PRIMM-fiches in je lespraktijk inzet, kan je vrij kiezen als leerkracht. Zo kan je de fiches als leidraad voor een onderwijsleergesprek gebruiken of als werkbundel aan de leerlingen bezorgen. Niet alle PRIMM-fasen hoeven trouwens in eenzelfde les aan bod te komen. Misschien opteer je ervoor om de predict-fase vooraf en individueel te laten uitvoeren door de leerlingen en geef je de make-fase liever als huistaak mee? Ook de keuze om leerlingen niet in elke fase te laten samenwerken en om daarbij al of niet gebruik te maken van de samenwerkingsmogelijkheden binnen een ontwikkelomgeving, bepaal je vrij.

Om al deze vrijheden mogelijk te maken, is het logisch dat de PRIMM-fiches aangepast en/of uitgebreid kunnen worden. Alleen op die manier kunnen ze ook in jouw lespraktijk succesvol ingezet worden. Om dit te faciliteren, kan je de aanpasbare versies van deze documenten via mail (tony.opsomer@vives.be) aanvragen. De verkregen documenten kan je daarna onder deze Creative Commons licentie verder bewerken:

Hoe evalueer je de programmeeropdrachten in de maak-fase?

Programmeeropdrachten hebben vaak meer dan een mogelijke oplossing. Het evalueren van dergelijke opdrachten omvat – ongeacht of ze individueel of samen opgelost worden - dan ook vele facetten. Hieronder vind je een overzicht van de belangrijkste elementen. Afhankelijk van wat de leerlingen reeds aangeleerd kregen, kan je bepaalde kenmerken weglaten en/of een groter gewicht toekennen:

	Score op 15
Algoritme: • Wordt vertrokken vanuit **deelproblemen** (decompositie)? • Wordt voor elk deelprobleem een **algoritmische voorstelling** gemaakt? • Worden eventuele terugkerende elementen herkend (**patroonherkenning**)? • Wordt een **teststrategie** (bv. bepalen van grenswaarden?) bedacht om na te gaan of het algoritme/programma correct reageert/functioneert?	**Proces:** **-1 0 1 2 3**
Programma: • Lijkt het gestelde probleem opgelost en **functioneert** het programma correct met de – in de opgave – aangereikte testwaarden? • Werkt het algoritme/programma **in alle gevallen betrouwbaar** (dus ook met testwaarden die niet in de opgave stonden, bv. buiten, tussen en op eventuele grenswaarden) correct én zonder kans op het ontstaan van een oneindige lus? • Maakt het programma **efficiënt** gebruik van de beschikbare **programmeerstructuren** (opeenvolging, selectie, herhaling) en worden **wiskundige concepten** zoals booleaanse logica, bewerkingen, … efficiënt ingezet? • Worden een of meer **geavanceerdere technieken** gebruikt zoals bv. nesten, eigen functies of recursie?	**Product:** **-1 0 1** **-1 0 1** **-1 0 1 2 3** **0 1 2**
Leesbaarheid algoritme / programmacode: • Is de **layout/structuur** van het algoritme/de code overzichtelijk en logisch? • Is de programmacode helder **gedocumenteerd**? • Krijgen variabelen een **betekenisvolle naam**?	**Product:** **-1 0 1 2**
Kritische houding: • Wordt luisterbereid en oplossingsgericht omgegaan met problemen en het debuggen van een algoritme/programma?	**Proces:** **-1 0 1**
Creativiteit: • Is de **vormgeving** van de eventuele gebruikersinterface of de manier waarop de gebruiker in interactie treedt met het programma voldoende helder? • Worden **uitbreidingen** voorzien die niet gevraagd werden vanuit de opgave en die de functionaliteit verbeteren? • Wordt een **alternatieve** maar volledig functionele **oplossing** aangereikt?	**Product:** **0 1 2**

Bronnen

Braught, G., Wahls, T., Eby, L.M. (2011). *The case for pair programming in the computer science classroom*. ACM Transactions on Computing Education. 11 (2).

Croghan, M., Mann, M. (2017). *Solved! Better pupil discussion in the classroom: experiments in collaboration in Harris Academy, Battersea.* London: Nesta

Debongnie, C. (2022). *Didactique de l'informatique : la méthodologie PRIMM.* Ecole polytechnique de Louvain, Université catholique de Louvain,. Prom.: Mens, K., Goletti, O.

Kameda, T., S. Sugimori. (1993). *Psychological entrapment in group decision making: An assigned decision rule and a groupthink phenomenon*. Journal of Personality and Social Psychology. 65: 282–92.

Neutens, T., Coolsaet, K., Wyffels, F. (2022). *Assessment of code, which aspects do teachers consider and how are they valued?* ACM Transactions on computing education 22 (4), 1-27.

Preston, D. (2005) *Pair programming as a model of collaborative learning: a review of the research.* Journal of Computing Sciences in Colleges. 20 (4), 39–45.

Sands, P. (2019) *Addressing cognitive load in the computer science classroom.* ACM Inroads. 10 (1), 44–51.

Sentance, S., Csizmadia, A. (2015). *Teachers' perspectives on successful strategies for teaching Computing in school.* IFIP TC3 Working Conference 2015: A New Culture of Learning: Computing and Next Generations.

Sentance, S., Waite, J., Kallia, M. (2019) *Teaching computer programming with PRIMM: a sociocultural perspective.* Computer Science Education. 29 (2–3), 136–176.

Slavin, R.E. (1995). *Cooperative learning: Theory, research and practice (2nd ed.).* Boston: Allyn & Bacon.

Van Leeuwen, A.; Janssen, J. (2019). *A systematic review of teacher guidance during collaborative learning in primary and secondary education.* Educational Research Review. 27: 71-89.

Voyiatzaki, E., Christakoudis, C., Margaritis, M., Avouris, N. (2004). *Teaching algorithms in secondary education: a collaborative approach.* Proceedings ED Media 2004.

Werner, L., Denning, J. (2009). *Pair programming in middle school: What does it look like?* Journal of Research on Technology in Education. 42 (1), 29–49.