Bachelorscriptie
Informatica

Radboud Universiteit

Voor de draad ermee!

Een vakdidactische verkenning van programmeren met threads

Auteur:
Mathijs Vos
mathijsvos@student.ru.nl

Inhoudelijk begeleider:
Erik Barendsen

Tweede lezer:
Sjaak Smetsers

5 juli 2013

mutex.lock();

Samenvatting

In deze scriptie onderzoeken we enkele vakdidactische aspecten van programmeeronderwijs met betrekking tot threads. Die aspecten zijn: leerdoelen, moeilijkheden bij studenten, onderwijsaanpak en toetsing. De leerdoelen op de OU blijken in termen van Bloom-niveau lager geformuleerd dan die op de RU, hoewel de OU-docenten qua mening dichter bij de RU-leerdoelen zitten. Studenten blijken vooral moeilijkheden te hebben met elementaire programmeeraspecten.

Dankwoord

Allereerst bedank ik mijn begeleider Erik Barendsen, voor het wekelijks vrijmaken van tijd voor een voortgangsbespreking over de scriptie en voor alle hulp en feedback. Verder bedank ik Harrie Passier en Harold Pootjes van de Open Universiteit voor het vrijmaken van tijd en het laten aftappen van hun PCK; Annemiek Herrewijn van de Open Universiteit voor het voorzien in tentamenuitwerkingen en lesmateriaal; Sjaak Smetsers van de Radboud Universiteit voor het voorzien in tentamenuitwerkingen en voor het vrijmaken van tijd voor een PCK-interview.

Figuur 1: Threads.

Inhoudsopgave

1 Inleiding

Bij het geven van programmeeronderwijs komt het onderwerp "threading" al snel aan bod. Mede doordat we in hedendaagse pc's steeds vaker multicore-processoren aantreffen, wordt het onderwerp steeds belangrijker voor studenten die leren programmeren. Multithreaded programma's kunnen van meerdere processorkernen gebruik maken of het principe van interleaving gebruiken. Hierdoor kan een programma sneller uitgevoerd worden of kan de gebruikservaring worden verbeterd, door bijvoorbeeld het ene deel van een computerprogramma verder te laten gaan terwijl een ander deel een bestand van internet download.

De Open Universiteit leert studenten programmeren met threads of *draden* in het vak Objectgeörienteerd Programmeren in Java 2. Men vraagt zich er af of de manier van onderwijs geven over het onderwerp draden nog wel voldoet. Er lijkt geen structurele methode te bestaan om een programma dat met draden werkt te ontwerpen. Dit is te zien in de manier van onderwijzen, waar op dit moment theorie en voorbeelden gegeven worden, maar geen structurele manier om te werken met threads wordt aangeleerd.

Om het onderwijs te kunnen verbeteren, is het belangrijk om eerst te kijken wat er dan precies verbeterd moet worden en hoe dat dan kan. Om te weten wat verbeterd moet worden, is het nodig om een beeld te hebben van de bestaande situatie. Met deze scriptie onderzoeken we wat de huidige stand van zaken is in het programmeeronderwijs.

2 Probleemstelling

We willen de huidige toestand van het programmeeronderwijs met betrekking tot threading onderzoeken: zowel de kennis en lesmethoden van de docenten, als de moeilijkheden die studenten ondervinden bij het leren van het onderwerp. De 'toestand' bestaat uit de vier PCK-aspecten van Magnusson et al. (1999):

- Kennis over leerdoelen
- Kennis over het (on)begrip van studenten
- Kennis over manier van onderwijzen
- Kennis over manier van toetsen

De PCK-aspecten van Magnusson et al. (1999) zijn oorspronkelijk bedoeld om de vakdidactische kennis van docenten in te delen in een aantal categoriën. Dat omvat de persoonlijke mening, ervaringen en kennis over de stof van de docent. Hier gebruiken we dezelfde PCK-aspecten in een andere context. We classificeren geen mening van een docent, maar een toestand van het onderwijs in het algemeen. Het blijkt dat PCK-analyse niet alleen geschikt is om de kennis van een docent in beeld te brengen, maar dat dezelfde analysetechnieken ook geschikt zijn om leerboeken te analyseren (Saeli, 2012). Hoewel Magnusson's indeling daar niet direct voor bedoeld is, is het ook in ons geval een nuttige indeling.

We formuleren dus onze hoofdvraag als volgt en baseren de deelvragen op de PCK-aspecten van Magnusson:

- Wat is de huidige vakdidactische toestand in het threadingonderwijs?
  - Welke leerdoelen worden er gesteld?
  - Waar hebben studenten moeite mee?
  - Welke onderwijsaanpak wordt er gebruikt?
  - Hoe wordt het onderwerp getoetst?

3 Achtergrond

3.1 Draden

Hoewel het concept van draden of *threads* al sinds jaar en dag bestaat, worden draden meer en meer gebruikt in programma's en apps. Threads worden bijvoorbeeld gebruikt om te voorkomen dat de interface van een programma 'vastloopt' wanneer een langdurende berekening moet worden uitgevoerd, of wanneer bepaalde taken (tegelijkertijd) op de achtergrond moeten worden uitgevoerd (bijvoorbeeld een webbrowser die verschillende afbeeldingen tegelijkertijd inlaadt).

Waar tot enkele jaren geleden de meeste pc's nog een single-core-processor hadden en de threads afgewisseld *interleaved* werden uitgevoerd, beschikken steeds meer pc's en smartphones over multi-core-processoren, waarmee threads echt simultaan uitgevoerd kunnen worden. Voor bepaalde applicaties zorgt dat niet alleen voor een verhoogd gebruiksgemak (doordat de interface te allen tijde responsief blijft), maar kunnen ook de prestaties van het programma verbeterd worden doordat simpelweg meer dingen tegelijkertijd berekend kunnen worden.

Met draden kunnen verschillende onderdelen van een programma simultaan uitgevoerd worden. In de praktijk worden die delen niet altijd *echt* gelijktijdig uitgevoerd, maar krijgen de draden om en om een klein stukje rekentijd. Dit heet *interleaving*. Het gebruik van draden is vaak handig en in veel gevallen zelfs onvermijdelijk als het de bedoeling is dat een computerprogramma te bedienen blijft, ook al is het met een langdurende berekening bezig.

Wanneer met threads niet zorgvuldig wordt om gegaan, kan dit tot problemen leiden. Stel dat twee threads toegang hebben tot een gemeenschappelijke variabele. De gemeenschappelijke variabele bevat een eenvoudige teller. De ene thread verhoogt de teller, de andere kopieert de teller naar een lokale waarde. De volgorde waarin de threads rekentijd krijgen (en de lengte van die rekentijd), is onvoorspelbaar. Het is dus niet zo, dat wanneer de ene draad de teller heeft mogen verhogen, dat meteen de andere aan de beurt is. We bekijken het volgende codevoorbeeld:

```
int a = 0; // Gemeenschappelijke variabele
// Thread 1
public void run() {
  while(true)
    a++;
}

// Thread2
public void run(){
  while(true)
  {
    int b = a;

    if(a>b)
      throw new RuntimeException("Oops");
  }
}
```

Stel de regel 'int b = a' is zojuist uitgevoerd, waardoor de waarde van a gekopieerd is naar variabele b. Het zou kunnen, dat op dat moment de uitvoering van thread 2 gestopt wordt, en dat thread 1 weer even rekentijd krijgt. Die verhoogt de waarde van a (eventueel meerdere keren). Dan krijgt thread 2 weer de beurt, en dan zal blijken dat a een grotere waarde heeft dan b. Een illegale situatie is ontstaan.

De oplossing is het gebruik van synchronisatie. Synchronisatie is een verzamelterm voor verschillende technieken (locks, mutexen, semaforen, queue's etcetera). Daarmee kan worden afgedwongen dat de ene thread pas verder mag als de andere aan een bepaalde voorwaarde heeft voldaan. In het bovenstaande voorbeeld zou het probleem worden opgelost door af te dwingen dat thread 2 steeds a naar b kopieert én de controle $a > b$ uitvoert, alvorens thread 1 weer de beurt kan krijgen.

In Java (waar de focus in deze scriptie op ligt) gebeurt deze synchronisatie middels het statement *synchronized* en de methoden *wait()* en *notifyAll()*. Op de details van deze technieken gaan we hier niet in. Meer informatie is te vinden in de Java-documentatie, bijvoorbeeld op `http://docs.oracle.com/javase/tutorial/essential/concurrency/interfere.html`.

3.2 Leerdoelen

Het is bij cursussen zoals die op hogescholen en universiteiten gegeven worden gebruikelijk om vooraf een aantal *leerdoelen* vast te stellen. Zulke leerdoelen bestaan uit vaardigheden en kennis die een student zou moeten hebben geleerd tegen het einde van de cursus. Dat betekent dat het gebruikte lesmateriaal, de lessen zelf en alle gerelateerde zaken zoals oefenopgaven, in het teken van die leerdoelen moeten staan, zodat de student ook de mogelijkheid heeft om inderdaad aan de gestelde leerdoelen te voldoen. Bovendien vormen die leerdoelen een manier voor de student om na te gaan of hij inderdaad alle stof beheerst.

Bloom, Engelhart, Furst, Hill en Krathwohl (1956) stelden een indeling voor leerdoelen voor. Die indeling bestaat uit zes niveaus, met oplopende moeilijkheidsgraad.

Weten – de student bezit bepaalde kennis: hij kent begrippen, definities en andere feiten. *Hoe wordt een thread in Java gestart?*

Begrijpen – de student kan zijn kennis interpreteren en logisch reproduceren. *Zijn threads een geschikte oplossing voor dit probleem?*

Toepassen – de student kan zijn kennis en inzicht gebruiken in nieuwe situaties. *Welke synchronisatieprimieteve(n) kan je gebruiken om dit probleem op te lossen?*

Analyseren – de student kan ordenen naar inhoud en relaties aangeven. *Vergelijk de gegeven oplossingen. Welke voldoet het beste aan alle eisen?*

Synthetiseren – de student kan zelf elementen combineren tot een geheel en zo iets nieuws maken. *Implementeer een oplossing voor het gegeven parallelle probleem.*

Evalueren – de student kan evalueren en oordelen over ideeën of kwaliteit van het werk. *Is het verstandig om hier synchronisatie te gebruiken? Waarom (niet)?*

Een uitgebreide beschrijving van de taxonomie is te vinden in figuur 2.

3.3 PCK

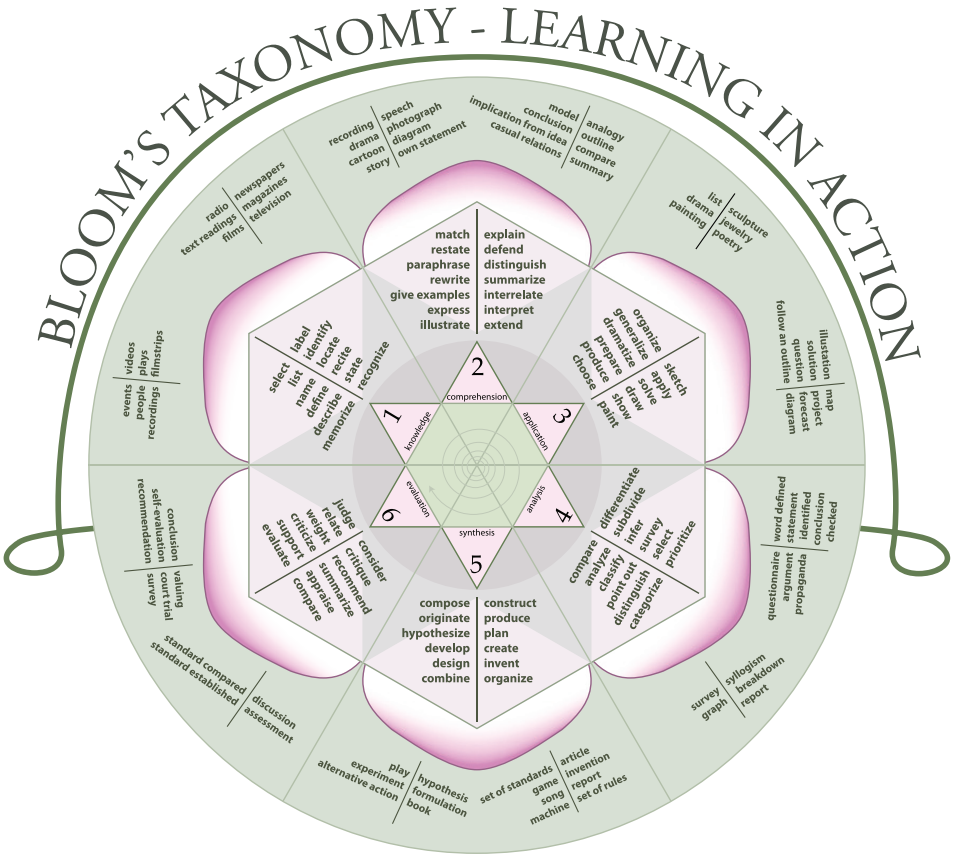
Leraren hebben, bewust of onbewust, *pedagogical content knowledge* of *PCK* op het gebied van de onderwerpen die zij doceren. Die PCK bevat hun kennis over hoe het betreffende onderwerp gedoceerd kan worden en bestaat onder andere uit methoden om stof aan anderen uit te leggen (de beste manier om een onderwerp de representeren, voorbeelden, demonstraties), begrip van wat het onderwerp lastig maakt voor anderen (kennis over begrip, misvattingen, voorkennis op verschillende leeftijden) en methoden om de heersende misvattingen de wereld uit te helpen (Shulman, 1986).

Een methode om er achter te komen wat de leerdoelen volgens docenten zouden moeten zijn, wat hun lesaanpak is, en in hoeverre zij weten waar de studenten moeilijkheden ondervinden, is het analyseren van de PCK. Het afnemen van een semi-gestructureerd interview aan de hand van een Content Representation of *CoRe*-tabel (Loughran, Mulhall & Berry, 2004) blijkt een geschikte aanpak hiervoor (Barendsen & Henze, 2012). Een voorbeeld van een (lege) CoRe-matrix is te vinden in appendix A.

3.4 Programmeerdidactiek

Van eerstejaars studenten die over basis programmeervaardigheden beschikken, mag worden verwacht dat zij een probleem volgens deze stappen aanpakken (McCracken et al., 2001):

1. Het probleem abstraheren uit de beschrijving



Figuur 2: De taxonomie van Bloom, weergegeven in een bloemvormig diagram. (licentie: Creative Commons Attribution-Share Alike 3.0 Unported; auteur: K. Aainsqatsi)

2. Het probleem opdelen in deelproblemen
3. De deelproblemen opdelen in deeloplossingen
4. Het samenvoegen van de deeloplossingen tot een werkend programma
5. Evalueren van de oplossing en indien nodig de stappen nogmaals doorlopen

Wanneer studenten leren werken met threads, nemen we aan dat zij reeds over basis programmeerkennis beschikken en een probleem volgens bovenstaand stappenplan oplossen. Tentamens richten zich meestal op het toetsen van stappen 3 en 4, wegens de beperkte tijd die studenten hebben om het probleem op te lossen (McCracken et al., 2001). McCracken et al. (2001) analyseerden studentuitwerkingen bij een programeeropdracht van eerstejaars informaticastudenten. De gemaakte uitwerkingen kregen een score op vier onderdelen:

Execution – kon het programma worden uitgevoerd?

Verification – ging het programma op de juiste manier om met gebruikersinvoer?

Validation – deed het programma wat het moest doen?

Style – voldoet de programmacode aan de standaarden?

Uit de studie van McCracken et al. (2001) bleek dat studenten in het algemeen laag scoorden, maar het best presteerden op *execution* (wat betekent dat ze programma's schreven die te compilen en uit te voeren waren) en *style* (wat betekent dat ze goed uitziende broncode produceerden). De meeste studenten hadden echter geen idee hoe ze de opgave moesten oplossen.

In een studie naar *invariant based programming* analyseerden Mannila (2010) studentuitwerkingen. Die uitwerkingen vormen zogenaamde *rich data*. Om hier zinnige gegevens uit te krijgen is het van belang om op enigszins gestructureerde wijze de gemaakte uitwerkingen te analyseren. De aanpak die hiervoor is gebruikt in Mannila (2010) is als volgt:

- Eerst wordt een subset van de uitwerkingen bekeken (een voor een).
- De uitwerkingen worden vergeleken met de opgave.
- Iedere uitwerking kan geen, een, of meer dan een soorten fouten bevatten.
- Alle soorten fouten worden genoteerd. Na het bekijken van de subset van uitwerkingen hebben we dus een lijstje met verschillende soorten fouten.
- Soorten fouten die erg gedetailleerd zijn, kunnen worden samengevoegd tot één meer abstracte soort fout.
- Nu worden alle uitwerkingen bekeken, waarbij wordt bijgehouden welke uitwerking welke soorten fouten bevat.

Er zijn twee soorten fouten te onderscheiden: studenten hebben moeilijkheden met de syntaxis en semantiek van de betreffende programmeertaal, en studenten hebben moeite met het bepalen welke taalelementen te gebruiken en ze tot één werkend programma samen te voegen (Soloway et al., 1982). We zullen in ons geval de syntactische fouten buiten beschouwing laten – het draait immers om het begrip dat de studenten van threads hebben, en niet zozeer om of zij de benodigde syntaxis precies kennen.

De methode die in Soloway et al. (1982) wordt gebruikt om studentenuitwerkingen te analyseren is als volgt:

- De oplossing die gegeven had moeten worden (in de vorm van een programma) wordt opgedeeld in stukken.
- Van iedere studentuitwerking krijgt ieder deel een score naar gelang de gelijkenis met de correcte oplossing.
- Voor ieder deel wordt de meest gegeven oplossing genomen, en al deze delen worden samengevoegd tot één uitwerking die aangeeft op welk onderdeel de meeste fouten werden gemaakt (als het resulterende programma correct is, is er geen één onderdeel waarbij de meerderheid fouten maakt).

4 Methode

We stelden ons de volgende vragen:

- Wat is de huidige vakdidactische toestand in het threadingonderwijs?
    1. Welke leerdoelen worden er gesteld?
    2. Waar hebben studenten moeite mee?
    3. Welke onderwijsaanpak wordt er gebruikt?
    4. Hoe wordt het onderwerp getoetst?

We bekijken een aantal informatiebronnen en zullen daarmee proberen bovenstaande vragen te beantwoorden.

4.1 Lesmateriaal

Om te achterhalen wat de leerdoelen op dit moment zijn, zullen we gaan kijken naar lesmateriaal dat gebruikt wordt in programmeercursussen waarin ook threading aan bod komt. We bekijken het dictaat en een tentamen van de cursus *Objectgeörienteerd programmeren in Java 2* van de Open Universiteit, en kijken naar een tentamen van het vak *Objectoriëntatie* van de Radboud Universiteit Nijmegen. Niet alleen kijken we naar de leerdoelen die letterlijk in het genoemde dictaat staan geschreven, maar ook proberen we aan de hand van de tentamenopgaven terug te redeneren wat de bijbehorende leerdoelen zijn. We zullen daarbij gebruik maken van de taxonomie van Bloom (Bloom et al., 1956). Met behulp van deze data zullen we proberen deelvraag 1 te beantwoorden. Bovendien kunnen we de tentamenopgaven ook gebruiken om te bekijken wat de manier van toetsen is, waarmee we deelvraag 4 kunnen beantwoorden.

4.2 Tentamenuitwerkingen

We bekijken uitwerkingen van studenten bij tentamens met opgaven over threading. De tentamens zijn afkomstig van zowel de Open Universiteit als de Radboud Universiteit. De tentamens zullen we analyseren op soortgelijke wijze als in Mannila (2010). Met deze data beantwoorden we deelvraag 2, over moeilijkheden die studenten ondervinden.

4.3 PCK-interviews

We interviewen een aantal docenten die een programmeervak geven waarin threading aan bod komt. Met die interviews proberen we hun PCK te achterhalen. Docenten hebben vaak (onbewust) een grote hoeveelheid kennis over hoe het onderwerp onderwezen wordt of hoe het onderwezen zou moeten worden. We interviewen docenten van programmeervakken van zowel de Open Universiteit als de Radboud universiteit. Zo krijgen we informatie over de manier van onderwijzen, lesstrategieën en de problemen die met het onderwijzen verbonden zijn, en dus kunnen we deelvraag 3 beantwoorden. Bovendien zullen we docenten vragen over de manier van toetsen, of hoe die volgens hen zou moeten zijn. Daarmee beantwoorden we een deel van deelvraag 4.

5 Resultaten

5.1 Lesmateriaal

5.1.1 Open Universiteit

Om te achterhalen of hetgeen dat getoetst wordt overeenkomt met hetgeen dat geleerd had moeten worden, is het van belang om te achterhalen welke *leerdoelen* schuil gaan achter vakken waarin met threads wordt gewerkt. In het lesmateriaal van de Open Universiteit (behorende bij het vak *Objectgeörienteerd programmeren in Java 2*) troffen we de volgende leerdoelen expliciet vermeld aan.

Na het bestuderen van deze leereenheid wordt verwacht dat u

- *weet in welke gevallen toepassing van draden nuttig is*
- *weet wat draden zijn en dat hun voortgang onvoorspelbaar is*
- *met behulp van de interface Runnable en de klasse Thread draden kunt creëren en starten*
- *weet hoe en in welke gevallen u een subklasse van Thread kunt gebruiken in plaats van een implementatie van Runnable*
- *weet in welke vijf toestanden draden kunnen verkeren en welke overgangen daar tussen zijn*
- *de verwerking van een draad kunt onderbreken door middel van de methoden Thread.sleep en Thread.yield*
- *weet hoe de control flow bij het gebruik van draden verloopt*
- *de problemen onderkent die optreden bij synchronisatie van draden, met name als die draden gemeenschappelijke variabelen gebruiken*

Leerdoel	**Bloom-niveau**
weet in welke gevallen toepassing van draden nuttig is	Weten
weet wat draden zijn en dat hun voortgang onvoorspelbaar is	Weten
met behulp van de interface Runnable en de klasse Thread draden kunt creëren en starten	Toepassen
weet hoe en in welke gevallen u een subklasse van Thread kunt gebruiken in plaats van een implementatie van Runnable	Begrijpen
weet in welke vijf toestanden draden kunnen verkeren en welke overgangen daar tussen zijn	Weten
de verwerking van een draad kunt onderbreken door middel van de methoden Thread.sleep en Thread.yield	Weten
weet hoe de control flow bij het gebruik van draden verloopt	Weten
de problemen onderkent die optreden bij synchronisatie van draden, met name als die draden gemeenschappelijke variabelen gebruiken	Begrijpen
kunt aangeven wat het betekent als een klasse of methode synchronized is	Begrijpen
de volgende begrippen kent: concurrency, parallellisme, target van een thread, interleaving, dispatching, starvation, deadlock, scheduler, communicatie en synchronisatie.	Weten

Tabel 1: De leerdoelen van de Open Universiteit, gespecificeerd naar Bloom-niveaus.

- *kunt aangeven wat het betekent als een klasse of methode synchronized is*
- *de volgende begrippen kent: concurrency, parallellisme, target van een thread, interleaving, dispatching, starvation, deadlock, scheduler, communicatie en synchronisatie.*

We deelden de leerdoelen in naar de niveaus van Bloom et al. (1956). Dat leverde het volgende resultaat in tabel 1.

Opvallend is dat de leerdoelen op de laagste drie Bloom-niveaus zijn opgesteld. De meeste leerdoelen zijn zelfs op het laagst mogelijke niveau (*weten*) gesteld. Voor een programmeercursus is het merkwaardig dat het blijkbaar niet nodig is dat studenten ook daadwerkelijk met threads leren werken, maar dat zij enkel een aantal begrippen behoeven te kennen.

In het bij de cursus behorende lesmateriaal worden eenvoudige opgaven gegeven, die in de aard zijn van vragen als “geef de wijziging aan deze klasse zodat je draden kunt gebruiken” en “beschrijf wat er hier gebeurt”. Behalve het starten en stoppen van threads wordt niet veel implementatie verwacht van studenten. Dit komt overeen met de gestelde leerdoelen: het lijkt vooral gericht op *weten* en niet zozeer op *kunnen*. Het lesmateriaal bevat een redelijk groot hoofdstuk over synchronisatie. Dit bestaat echter vooral uit beschrijvende tekst, en het kleine aantal bijbehorende opgaven is simpel (“beschrijf waarom het hier mis gaat”, maar geen opdracht om synchronisatie te implementeren of te repareren).

5.1.2 Radboud universiteit

Voor het vak Object-Oriëntatie van de Radboud Universiteit zijn geen specifieke leerdoelen met betrekking tot threading opgesteld. Threads maken ook een veel kleiner deel uit van deze cursus dan in de cursus van de Open Universiteit. Wel bekeken we de leerdoelen die voor het vak als geheel zijn gesteld (tabel 2).

De leerdoelen van de Radboud Universiteit blijken wat bloomhiërarchie betreft een stuk ‘hoger’ opgesteld te zijn. De meeste leerdoelen zitten in de hoogste drie niveau’s. Dit in contrast met de leerdoel van de Open Universiteit, die juist grotendeels op de onderste lagen van het model rusten. De verklaring voor dit verschil komt waarschijnlijk direct uit de aanleiding voor deze scriptie. De OU wil het onderwijs voor de cursus met draden verbeteren, omdat men het gevoel heeft dat het niet voldoet. Een concreet voorbeeld waaruit blijkt dat het RU-onderwijs inderdaad ‘moeilijker’ is, is dat daar het onderwerp synchronisatie behandeld wordt en ook onderdeel is van de toetsing, terwijl bij de OU dit onderwerp op het moment niet of nauwelijks wordt behandeld.

Leerdoel	Bloom-niveau
voor een gegeven probleem een objectgeoriënteerd ontwerp maken dat een bruikbare basis vormt voor een implementatie	Synthetiseren
het ontwerp realiseren als een java programma	Synthetiseren
toepassingen van klassehiërarchieën herkennen en realiseren	Synthetiseren
abstraheren via modulen en klassen	Toepassen
complexere bibliotheken gebruiken	Toepassen
componenten van bestaande objectgeoriënteerde systemen analyseren, begrijpen, aanpassen en uitbreiden	Synthetiseren
kwaliteitseisen (zoals geschiktheid, correctheid, en complexiteit) aan een algoritme of programma benoemen en verifiëren	Evalueren

Tabel 2: De leerdoelen van het vak Object-Oriëntatie van de Radboud universiteit, gespecificeerd naar Bloom-niveaus.

5.2 Tentamenuitwerkingen

5.2.1 Object-Oriëntatie

We bekeken een willekeurige subset van 40 tentamenuitwerkingen van totaal 87 deelnemers van het vak Object-Oriëntatie van de Radboud Universiteit. Deze analyseerden we op soortgelijke wijze als in Mannila (2010).

Allereerst doorliepen we een deel van de 40 tentamens om de verschillende soorten gemaakte fouten op een rij te zetten. Daaruit verkregen we de volgende lijst:

F1 – Onnodig gebruik van threadbesturing De student heeft onnodig of zelfs foutief de methoden `wait()` en / of `notifyAll()` gebruikt. Hierdoor wordt de performance van het programma negatief beïnvloed of kan zelfs een deadlock optreden. Tussen deze twee situaties maken we geen verder onderscheid.

F2 – Onnodig gebruik van synchronisatie De student heeft onnodig gebruik gemaakt van het `synchronized`-statement. Hierdoor is een methode onnodig gesynchroniseerd, waardoor de performance van het programma negatief beïnvloed wordt of een deadlock kan optreden. Tussen deze twee situaties maken we verder geen onderscheid.

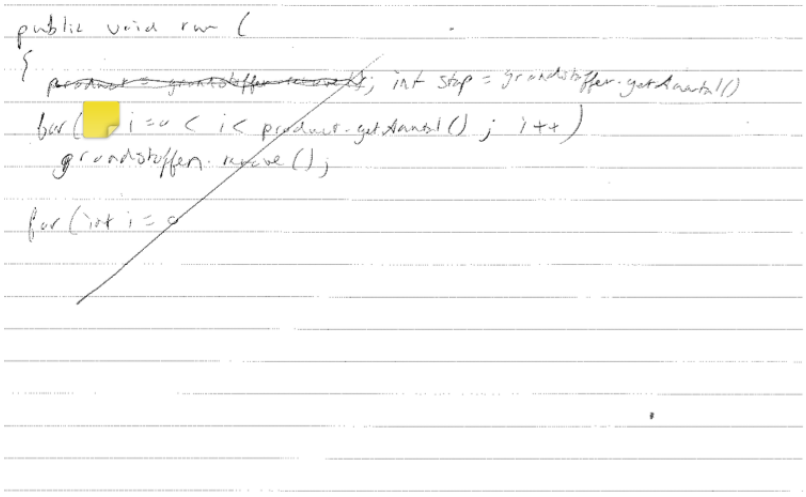
F3 – Weggelaten synchronisatie De student heeft een methode niet `synchronized` gemaakt waar dit wel had gemoeten. Hierdoor kan een inconsistente datastructuur ontstaan, doordat twee draden tegelijkertijd dezelfde variabelewaarden kunnen wijzigen.

F4 – Weggelaten threadbesturing De student heeft geen gebruik gemaakt van de methoden `wait()` en / of `notifyAll()`. Hierdoor zal een methode niet doen wat hij moet doen, doordat niet op het juiste moment wordt gewacht of doordat andere threads niet verder kunnen.

F5 – Gebruik van if in plaats van while, in verband met `notifyAll()` Wanneer `notifyAll()` wordt aangeroepen, worden alle wachtende draden wakker gemaakt, ook al wordt aan de conditie waar de draden op wachten nog steeds niet voldaan. Daarom is het nodig om de conditie in een `while`-lus te plaatsen en de draad opnieuw te laten wachten als de conditie nog niet waar is. Wanneer een student in dit verband een `if`-conditie heeft gebruikt, waardoor de thread niet opnieuw zal wachten en dus in sommige gevallen verder gaat terwijl niet aan de conditie is voldaan, duiden we dit aan met F5.

F6 – Ontbrekende implementatie De student heeft een subopgave niet of slechts deels gemaakt. Een mogelijke oorzaak zou kunnen zijn dat de student niet wist hoe de opgave opgelost moest worden, en daarbij alleen een aantal basiszaken ingevuld heeft. Zie figuur 3 voor een voorbeeld.

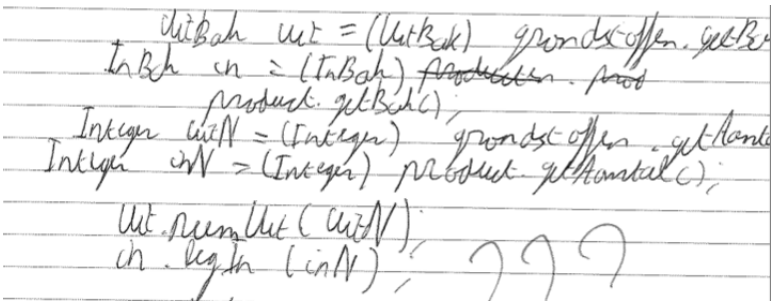
F7 – Opgave niet of maar deels gemaakt De student heeft de gehele opgave niet gemaakt, of is na een aantal subopgaven gestopt. Sommige studenten gaven expliciet aan te weinig



Figuur 3: Een voorbeeld van een deels geïmplementeerde deelopgave.

tijd te hebben gehad, waardoor zij niet toekwamen aan de rest van de subopgaven. Andere studenten vermelden geen reden, hoewel ook daar tijdsgebrek een plausibele reden is.

F8 – Opgavespecifieke fout We kwamen veelvuldig fouten tegen met betrekking tot het gebruik van de objecten `Inbak` en `Uitbak`. Het betreft hier geen fout over draden of programmeren in het algemeen. Dit zou er op kunnen duiden dat de student de opgave niet geheel begrepen heeft. Een voorbeeld van deze fout is bijvoorbeeld het eenmalig aanroepen van de methoden `neemUit` of `legIn`, terwijl dit in een loop had moeten gebeuren. Zie ook figuur 4.

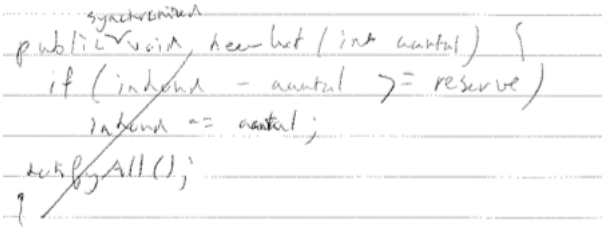


Figuur 4: Een voorbeeld van een verkeerd gebruik van de in-/uitbakken. De aanroepen naar `neemUit` en `legIn` horen plaats te vinden in een loop.

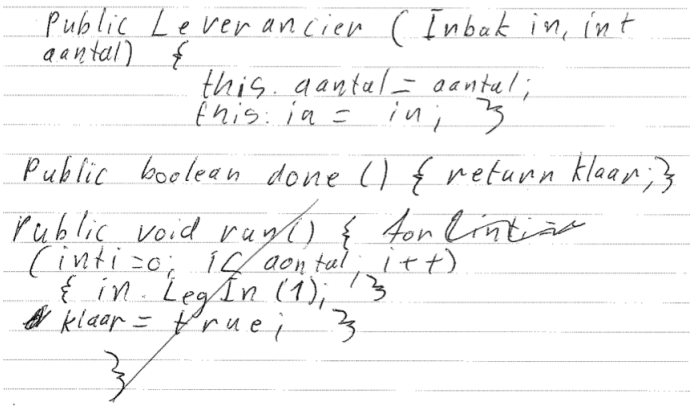
F9 – Verkeerde / missende condities De student heeft een verkeerde conditie (in de vorm van `if`, `while` of `for`) gebruikt of een nodige conditie in het geheel weggelaten. Deze fout heeft niet direct iets te maken met draden, maar geeft wel een indicatie van algemene programmeervaardigheden en begrip van de opgave. Zie ook figuur 5.

F10 – Verkeerde functionaliteit De student heeft een 'overige' fout (dat wil zeggen: een fout die niet in de bovenstaande categorieën past) gemaakt, en daarbij de plank zodanig misgeslagen, dat het programma niet zal doen wat het zou moeten doen. Voorbeelden van dit soort fouten zijn ontbrekende argumenten (zodat niet gecommuniceerd wordt met een ander object, terwijl dat wel nodig is), totaal missende functionaliteit, en implementaties die duidelijk iets heel anders doen dan in de opgave beschreven is. Zie ook figuur 6, 7 en 8.

Bij het noteren van de fouten hebben we syntactische fouten buiten beschouwing gelaten. Daaronder verstaan we niet enkel missende puntkomma's, maar ook afwijkende methodenamen (voor-



Figuur 5: Een voorbeeld van een missende conditie. In dit voorbeeld ontbreekt een `while`-loop die de thread laat wachten zolang `inhoud<aantal`.



Figuur 6: Een fout zoals gemaakt bij deelopgave a. Het object krijgt geen `Uitvoerder`-object, terwijl dit voor de `run`-methode nodig is.

beeld: 'werkIsAf' versus 'werkIsGedaan', of 'size' in plaats van 'length'), verkeerde return-types (bijvoorbeeld void in plaats van int), missende return-statements en meer zaken die een digitale ontwikkelomgeving zou detecteren.

Het kwam een aantal keer voor dat een student bij deelopgave b (over generics) een uitwerking maakte die een klasse bevatte die niet generiek was (in tegenstelling tot wat de bedoeling was). Deze fout hebben we onder F10 geschaard, omdat de fout niet alleen weinig voorkwam, maar ook minder interessant is om los te vermelden omdat de fout niets met threads te maken heeft.

Student	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
1	x	x		x	x	x		x	x	x
2				x	x	x	x	x	x	x
3							x			
4	x	x	x	x		x		x	x	x
5							x	x	x	x
6	x	x		x	x			x		
7				x	x	x				x
8							x		x	x
9							x		x	x
10									x	
11			x	x					x	x
12			x	x			x		x	x
13						x	x			
14	x				x				x	x
15			x	x	x			x		x
16			x	x				x		
17			x	x				x	x	x
18			x	x				x	x	x

19			x	x	x			x	x	x
20			x	x		x	x	x	x	x
21				x				x	x	
22			x	x		x	x		x	
23			x	x		x			x	x
24			x	x		x			x	x
25										x
26					x		x			x
27					x					
28							x			
29										x
30			x	x				x	x	x
Student	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**

5.2.2 Objectgeoriënteerd programmeren in Java 2

Op basis van tentamenuitwerkingen behorende bij het vak Objectgeoriënteerd programmeren in Java 2 van de Open Universiteit maakten we een soortgelijke analyse als van het materiaal van de Radboud Universiteit. In dit vak kwam het creëren, starten en stoppen van threads aan bod. Communicatie tussen threads was geen onderdeel van deze cursus.

Het relevante deel van het tentamen (namelijk de vraag over het onderwerp draden) bestond uit één hoofdvraag met daar onder zeven deelvragen. Daarvan waren zes deelvragen van het soort "geef de noodzakelijke wijzigingen aan de klasse / methode". Één deelvraag betrof het geven van een uitleg waarom twee methoden in een gegeven stuk code *synchronized* waren. Omdat we vooral willen achterhalen welke structurele fouten vaak voorkomen, zijn syntaxisfouten niet erg interessant. De zes eerder genoemde deelvragen gaan juist vooral over syntaxis. Uit de tentamenuitwerkingen bleek ook dat bij deze opgaven nauwelijks fouten werden gemaakt.

De overgebleven redenatievraag over het *synchronized*-statement is de enige vraag waar wat creativiteit voor nodig is. Daarom probeerden we uit de antwoorden bij deze vraag veelgemaakte fouten te extraheren. In totaal bekeken we 30 uitwerkingen. Van de 30 deelnemers hadden 2 studenten het hele onderdeel over draden overgeslagen. Één student had de opgave wel gemaakt, maar daarbij de deelvraag over het *synchronized*-statement overgeslagen. Alle andere studenten hadden de vraag ofwel deels juist, of helemaal juist beantwoord. Omdat er nauwelijks foutieve antwoorden voorkomen (wat waarschijnlijk te wijten is aan de aard van de opgaven), kunnen we uit deze uitwerkingen geen zinvolle gegevens halen.

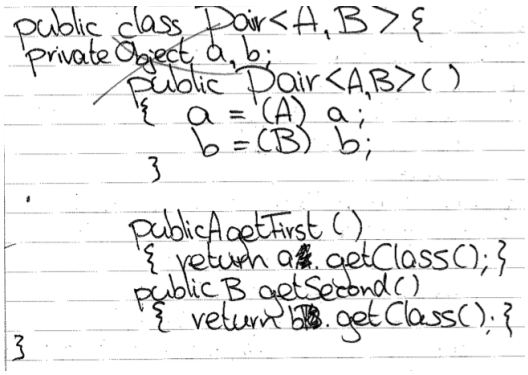
Wel valt op dat de tentamenopgaven, die erg recht-toe-recht-aan en meer een 'invuloefening' zijn, goed passen bij de leerdoelen zoals die momenteel gesteld zijn. Het gaat vooral om het 'weten' (van de syntax) en niet of nauwelijks om het kunnen werken met draden.

5.3 PCK-interviews

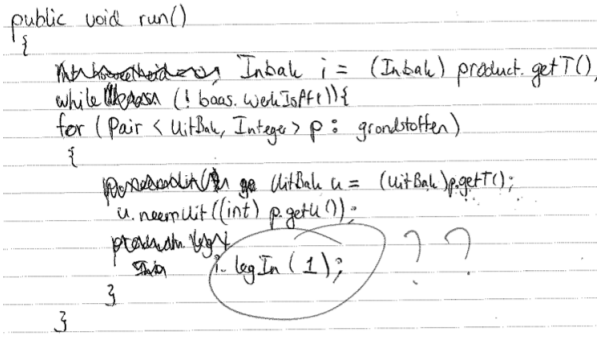
We interviewden drie docenten om hun PCK te achterhalen: twee van de Open Universiteit (docent 1 en 2) en een van de Radboud Universiteit (docent 3).

Opvallend is dat docenten 1 en 2 als reden noemen dat draden steeds meer gebruikt worden, en dat het daarom een belangrijk onderwerp is. Docent 3 daarentegen zegt dat het simpelweg nodig is in een bepaalde klasse van applicaties, om responsiviteit te kunnen garanderen. Docent 1 wil eigenlijk een soort methode of stappenplan hebben, waarmee studenten draadproblemen kunnen oplossen als ze dat stappenplan volgen. Docent 3 geeft aan dat hij niet op de hoogte is van het bestaan van een 'pattern' voor threads of modelleertechnieken daarvoor.

Alle geïnterviewde docenten vinden dat studenten in meer of mindere mate moeten kunnen redeneren over draden. Zo'n redenatie zou dan kunnen bestaan uit het geven van een codevoorbeeld waar een fout in zit, waarbij studenten moeten uitleggen waarom een dergelijke oplossing niet werkt. Docent 3 toetst dit soms in tentamens. Docent 1 en 2 vinden in ieder geval dat het iets is dat studenten moeten kunnen – juist omdat ze niet alleen een oplossing moeten kunnen implementeren, maar ook moeten kunnen nadenken over wat een goede oplossing is.



Figuur 7: Een fout zoals gemaakt bij deelopgave b. In plaats van dat objecten a resp. b worden teruggegeven, wordt de naam van de betreffende klasse genomen.



Figuur 8: Een fout zoals gemaakt bij deelopgave c. In plaats van dat een vooraf gedefinieerd aantal grondstoffen in de bak wordt gelegd, is dit gehardcode op 1.

5.3.1 Docent 1

De docent vindt het onderwerp vooral belangrijk omdat multicore-pc's steeds meer gemeengoed worden, en het belangrijk is om met draden te kunnen programmeren om daar goed gebruik van te maken.

> "Van oudsher is het onderwerp *draden* belangrijk, en volgens mij wordt het ook steeds belangrijker. Het was belangrijk omdat multithreading in computersystemen een belangrijke rol speelt, ook in de pc's van een paar jaar geleden vindt al heel veel multithreading plaats. Maar dat was in veel gevallen niet zo prominent aanwezig voor de gemiddelde programmeur. Vanaf nu richting de toekomst, met multicore-processoren wordt het steeds belangrijker dat je kennis hebt van draden en dat je activiteiten kan splitsen en daardoor sneller kan laten verlopen."

Hoewel het een belangrijk onderwerp is, vindt de docent het lastig om aan te geven waar de grens ligt voor wat te ver gaat om aan studenten te leren.

> "Dit vind ik een hele lastige, omdat ik dat niet precies weet, en dat heeft ook alles te maken met dat we als docenten nog onvoldoende zicht hebben op wat dit onderwerp eigenlijk inhoudt. Enerzijds als je draden wil programmeren met dat er communicatie tussen draden plaatsvindt, en je wil aantonen, formeel of wat minder formeel, dat je programma ook foutloos gaat werken, dan ben ik bang dat dat een paar bruggen te ver is. Dat vergt heel veel kennis van formele technieken."

De docent is eigenlijk op zoek naar een hulpmiddel om studenten in de goede richting te duwen bij het oplossen van threading-problemen.

> "Dat is het grijze gebied waar ik zicht op probeer te krijgen. Zijn we in staat om met een aantal meer eenvoudige hulpmiddelen af te dwingen dat studenten voor een deel wat meer nadenken, wat meer redeneren, over hun programma? Bijvoorbeeld door het inzetten van UML-diagramtechnieken, dat je de opzet van zo'n programma eerst op papier doet, daar over nadenkt, en het dan gaat programmeren. Dat daar een aantal hulpvragen of vuistregels geformuleerd kunnen worden die studenten in ieder geval helpen: als je die nou maar langsloopt dan ga je in ieder geval de goede kant op. Natuurlijk is het dan zo dat je niet kunt bewijzen dat het een en ander dan foutloos werkt, maar toch dat principe van 'eerst denken, dan doen', dat dat veel meer ondersteund en een soort van afgedwongen wordt."

Studenten moeten niet alleen kunnen programmeren, maar ook redeneren. Dat redeneren bestaat vooral uit nadenken over een UML-ontwerp.

> "Dat je aan de hand van je UML-ontwerp kunt uitleggen waarom dit (de oplossing, red.) werkt. Dus niet kunnen bewijzen, maar wel aannemelijk kunnen maken dat dat goed in elkaar zit."

Bovendien is dat belangrijk omdat studenten niet meer gewend zijn om methodisch te werken.

> "Ik heb het vermoeden dat studenten steeds minder opgeleid zijn [...] in het netjes methodisch werken, er daar ook heel veel moeite mee hebben. Zeker bij programmeren is het natuurlijk heel makkelijk dat je gelijk code begint te schrijven en dan net zo lang doorprutst tot het werkt. Die aanpak werkt meestal niet bij een onderwerp als draden, dat moet je veel systematischer aanpakken."

De moeilijkheid ligt er in dat processen dynamisch zijn, en dat dat lastig voor te stellen is.

> "Het is niet een onderwerp zoals klassen en subklassen, dat is allemaal statisch. Dit zijn dynamische processen, en dat is allemaal veel lastiger voor te stellen. Met name als [...] processen op een gegeven moment parallel gaan lopen [...] en dus ook lastiger voor studenten om te begrijpen."

De docent denkt dat het nuttig is om, bijvoorbeeld op een tentamen, eerst de betrokken concepten te laten opschrijven.

> "Een eerste stap zou kunnen zijn: zet eerst eens de concepten op een rij die hier een rol spelen. Als iemand daar niet uit komt, is gelijk duidelijk dat daar al een probleem zit. Ik denk ook dat dat niet alleen voor de student erg handig is [...] maar dat het ook in het onderwijs qua diagnosemiddel een heel handig iets is, omdat je ook veel beter ziet waar het al mis gaat."

5.3.2 Docent 2

De docent vindt dat in het huidige cursusmateriaal te veel aandacht uitgaat naar syntaxis van draden, terwijl synchronisatie niet of nauwelijks aan bod komt.

> "Wat ik zelf ten opzichte van de huidige cursus nog wel zou willen toevoegen, is wat mogelijkheden om synchronisatie tussen twee verschillende draden [...] wat meer uit te diepen en wat meer handvaten te geven dan wat we nu doen, nu behandelen we alleen het woordje 'synchronized' [...] en dat (synchronisatie, red.) lijkt me wel iets dat het interessanter maakt."

De docent wil verder gaan dan alleen de syntax die nu behandeld wordt, het gaat vooral om het kunnen oplossen van het probleem.

> "Draden op zich zijn niet interessant, als je twee onafhankelijke processen hebt dan is het voornamelijk syntax [...] en daar liggen niet de problemen, de problemen liggen in draden en synchronisatie, en dat is waar het echt om gaat. [...] Dat kan erg complex zijn en het is voor onze cursus niet nodig om heel erg de diepte in te gaan, met name

> dingen als deadlock hoeven niet. [...] Wat we nu al doen is een gemeenschappelijke variabele beschermen, zodat twee processen niet tegelijk de waarde kunnen wijzigen, een stapje verder is dat de processen elkaar informeren over dat hun taak er op zit. [...] Nu hebben we veel te veel syntax. [...] Daar gaat het helemaal niet om, je moet denken aan parallelle processen die je gestalte wil geven en het hoe is niet belangrijk om uit te leggen dat het op twee manieren kan, dat vertroebeld het beeld alleen maar."

Studenten moeten met synchronisatie in staat zijn om een simpel producer-consumerprobleem op te lossen.

> "Het producer consumer-probleem [...] in ons practicum bijvoorbeeld obers en koks [...] waarbij de kok bijvoorbeeld een seintje geeft aan de ober dat er weer wat klaar staat, dus wat directere synchronisatie dan we nu doen, namelijk 'kijk maar of er toevallig wat klaar staat en zo ja dan neem je het mee'."

De docent vindt het belangrijk dat studenten op een hoger niveau over draden kunnen nadenken.

> "Dat lijkt me een belangrijk iets, dat ze (studenten, red.) eerst op een hoger niveau, eerst het wat dan het hoe [...], een synchronisatie- of cuncurrencyprobleem kunnen oplossen en het daarna pas implementen. [...] Dat oplossen kan door het eerst te visualiseren, bijvoorbeeld met activity- of sequence diagrams in UML."

De docent vindt dat het onderwerp in zijn algemeen erg complex is.

> "Omdat de mens van nature niet geneigd is parallell te denken. Het is een van de moeilijkste onderwerpen, en dat blijkt ook omdat in Java 7 een nieuwe synchronisatie-API zit, en daar staat dat de oude API van tien jaar geleden, zoals die nu in onze cursus gebruikt wordt, eigenlijk al achterhaald is en dat dingen daardoor behoorlijk fout kunnen lopen. Ik denk dat dat een goed voorbeeld is om de complexiteit te schetsen. Er zijn bijvoorbeeld ook heel complexe methoden om te bewijzen of ergens al dan niet een deadlock zou kunnen zitten"

Doordat er weinig contacturen en geen practica zijn, heeft de docent geen goed beeld van de moeilijkheden die studenten ondervinden.

> "Dat vind ik lastig aan te geven, want als je bij ons lesgeeft dan is dat drie uurtjes voor twintig uur stof, dus als je dan draden behandelt dan vertel je een beetje de grote lijn, maar vaak ontbreekt de tijd om ter plekke een opgave te laten maken. Het enige dat we doen is het geven van een opdracht die studenten thuis moeten uitwerken. Maar de struggle die studenten gehad hebben en welke zijpaden ze genomen hebben, daar heb ik dan helaas geen zicht op."

Het gaat niet alleen om het kunnen maken van een draad, maar ook om het herkennen wanneer deze nodig is.

> "Je moet denk ik ook wel een goed begrip hebben van wat een draad nu precies is, en wanneer je iets met een draad moet doen. Dat soort gedachtengangen moet je eerst hebben. Zou ik überhaupt een draad nodig hebben? Ik denk dat dat soort aspecten mooi geïllustreerd kan worden door iets te vertellen over het besturingssysteem, hoe multitasking werkt; dat is bij ons op dit moment ontbrekende achtergrondkennis."

> "Eigenlijk is het iets dat in de gereedschapskist van iedere programmeur zou moeten zitten: het type probleem herkennen waar draden nodig zijn."

Het onderwerp zou volgens de docent het best te doceren zijn door te beginnen met simpele voorbeelden en dan de complexiteit op te schroeven.

> "Ik denk dat het belangrijk is om wat voorbeelden te geven van problemen die mogelijk via parallelliteit opgelost zouden kunnen worden [...]. Vervolgens zou ik een simpel voorbeeld nemen en daar op hoog niveau een oplossing voor schetsen, bijvoorbeeld een

> bedrijfsproces [...]. Daar kan een mooi UML activity diagram bij, en vervolgens zou je wat meer richting een software-oplossing kunnen gaan. Daarna zou ik een complexer voorbeeld nemen en dat op dezelfde manier uitwerken, en tussendoor wat opgaven geven en er wat over vragen. [...] Als dat gebeurd is zou ik het synchronisatieprobleem introduceren. [...] Behalve dat je dan moet weten hoe draden werken, kan je ook toevoegen hoe je dan dat synchronisatieprobleem kunt oplossen."

Moeten studenten ook kunnen redeneren over draden?

> "Nou, in zoverre, dat ze weten dat bij het gebruik van het woordje "synchronized" er ook een keerzijde aan zit, en op die manier ook zouden kunnen redeneren of het niet op een andere manier kan. [...] Dus dat ze in ieder geval bewust zijn van wat ze aan het doen zijn, en niet alleen wat code inkloppen en dan zien 'oh hij doet het'."

De tentamens zouden minder een invuloefening moeten zijn dan nu het geval is. Maar vragen zijn lastig te construeren.

> "We hebben sowieso in ons materiaal een zelftoets. [...] Die is bedoeld om te testen of studenten de stof beheersen. Zo'n type opgave zou ook in het tentamen horen. De tentamens die we nu hebben [...], zijn vooral een invuloefening. Dat is niet precies het type opgaven dat ik zou willen geven. [...] Je moet altijd oppassen met opgaven: als iemand niet uit het ontwerp komt, dan kan hij verder ook geen punten meer kan scoren. Dus die opgaven zijn lastig te construeren."

Een moeilijkheid is dat bij het nakijken van opgaven, het lastig is om te controleren of een gegeven oplossing werkt.

> "Wat een probleem op zich is, als iemand nou een oplossing met draden heeft gemaakt, om goed te kunnen testen of dat een goede oplossing is. Als een synchronisatieprobleem niet goed is opgelost, is de kans dat het mis gaat vrij klein, je moet echt je best doen om het dan fout te laten gaan. [...] Nu doen we dat vooral door goed naar de code te kijken. [...] Wat wij binnenkrijgen aan oplossingen, daarvan weet je eigenlijk wel dat het goed is. Als een student niet uit een opdracht komt, dan levert hij ook niks in. [...] Als een student iets inlevert, is de kans dat er echt grove fouten in zitten niet zo heel groot. Op het tentamen waar je met papier en pen wat moet doen, is die kans veel en veel groter."

5.3.3 Docent 3

Studenten moeten weten wat ze met threads kunnen, maar ook wat de gevaren zijn.

> "Waar je ze voor kunt gebruiken, en wat typische problemen zijn die je tegen komt waar je threads hebt. [...] Threads gebruik je met name om de responsiviteit te verhogen van applicaties. [...] De 'problemen' hebben te maken met synchronisatie. Waar je voor moet oppassen is dat je een inconsistente datastructuur krijgt als je er niet voor zorgt dat de toegang daartoe gesynchroniseerd is."

Bovendien moeten studenten niet alleen weten welke problemen er zijn, maar ze ook kunnen oplossen.

> "En we gaan nog iets verder, kijkende naar een hele typische vorm: de zogenaamde producer-consumer-threads, waarbij je een of meerdere producenten hebt en een of meerdere consumenten, en het is idee is dat die niet synchroon lopen. [...] Dus: hoe zorg je ervoor dat threads niet nodeloos op elkaar zitten te wachten?"

De studenten moeten dit alles zelf kunnen, en daartoe de bijbehorende primitieven in Java kennen en gebruiken. Op een tentamen zijn opdrachten nooit 'from scratch'. Op het practicum wel: studenten moeten dus wel zo'n opdracht vanaf de grond af aan kunnen opbouwen, maar dat wordt niet getoetst.

> "In een tentamen is dat nooit 'from scratch', omdat het schrijven tijdens het tentamen van een programma zo veel werk kost, dat is niet realistisch. Dan ben je met een boel dingen die niet relevant zijn bezig."

Het onderwerp wordt geïntroduceerd met een simpel voorbeeld.

> "Gewoon een flauw voorbeeldje [...] waarbij ik iets op het scherm teken en laat zien dat de gebruiker op het moment dat er getekend wordt, niets met de applicatie kan. Dus de applicatie reageert niet op wat de gebruiker doet. Typisch iets wat je met een interactief programma niet wil. [...] Het moet zo zijn dat de berekeningen die je doet heel kort zijn, zodat het programma binnen hele korte tijd kan reageren op wat de gebruiker doet. Ik gebruik dat voorbeeld en laat zien hoe je dat kunt oplossen, en dan maak ik een stapje naar processen die gemeenschappelijke data gebruiken. [...] Dan heb je synchronisatie nodig, en dat beschrijf ik dan ook, maar ik laat ook meteen zien wat het gevaar is van synchronisatie; dus dat je de mogelijkheid creëert dat sommige threads heel lang zitten te wachten, of dat je zelfs in een situatie kunt komen waarin threads op elkaar zitten te wachten, dus een deadlock."

Wat 'ingewikkelder' synchronisatieproblemen worden verder alleen genoemd, maar er wordt niet echt iets mee gedaan.

> "Dingen als deadlock en starvation [...] krijgen geen heel prominente aandacht. Dat noemen we, en dan weet je dat het probleem bestaat, maar verder is dat niet zo relevant voor wat wij doen."

Studenten moeten in staat zijn om te redeneren, maar dat wordt niet altijd getoetst.

> "Ik stop wel eens een vraag in een tentamen als 'dit is de situatie, geef aan waarom dit gevaarlijk is'. [...] Dus dat toetsen soms wel, soms niet."

In ieder geval moeten studenten er wel toe in staat zijn om zo'n vraag te beantwoorden:

> "Anders denk ik ook niet dat ze de primitieven op de juiste manier kunnen gebruiken."

Threads zijn een belangrijk onderwerp, omdat ze soms gewoon nodig zijn.

> "Ik vind het wel belangrijk, omdat het je het bij een bepaalde klasse van applicaties gewoon nodig hebt. "

Hoewel het een belangrijk onderwerp is, gaat de stof niet heel diep: ingewikkelde zaken komen niet echt aan bod, tenzij bepaalde dingen er zo dik bovenop liggen dat studenten wel een fout moeten herkennen.

> "De synchronisatieprimitieven die wij behandelen zijn heel elementair. [...] Deadlocks, starvation kan je veel meer over vertellen. Je kunt er ook veel meer mee analyseren, als je het hebt over bewijzen of aannemelijk maken dat er geen deadlock is, dat zijn dingen waar we niks aan doen, daar laten we studenten niet mee oefenen. Het moet er heel dik bovenop liggen, dan vragen we er wat over."

Threads zijn het laatste onderwerp uit de cursus van deze docent. De docent heeft de indruk dat sommige studenten daardoor het onderwerp wat minder serieus nemen en gewoon niet bestudeerd hebben. Maar los daarvan lijken algemene programmeervaardigheden ook te ontbreken:

> "Ik weet niet zo goed wat de oorzaak is waarom studenten niet de vragen over threads kunnen beantwoorden. [...] Bij de cursus is het in het algemeen voor een aantal studenten moeilijk om überhaupt sommige vragen te beantwoorden. Dat duidt vaak op te weinig ervaring, dus dat de basis er niet echt is. Dat heeft niet zozeer met threads te maken. Studenten hebben heel veel moeite met heel elementaire dingen. Daardoor komen die wat geavanceerder onderwerpen, en threads is een geavanceerder onderwerp, in het gedrang. [...] Bij een aantal andere dingen die we behandelen zie je hetzelfde probleem."

Sommige studenten hebben de stof óf helemaal niet bestudeerd, of ze hebben er niets van begrepen, waardoor hun oplossing vaak erg ver van de juiste zit.

> "Als je geoefend hebt met oude tentamens, dan moet het niet al te lastig zijn om de vragen over threads te kunnen beantwoorden. Het hoeft niet precies, maar sommige studenten zitten niet eens in de buurt. Er zijn bepaalde basisregels, bijvoorbeeld heel elementair: als je gaat wachten, dan moet dat binnen een gesynchroniseerde methode zijn, anders heeft dat geen zin. Dat is een basisregel. Maar sommigen schrijven gewoon te pas maar ook vaak te onpas instructies op om een thread te laten wachten."

Aan de hand van ingeleverde practicumopdrachten is moeilijk te achterhalen waar studenten moeite mee hebben.

> "Het gros van de studenten levert een werkend programma in. Maar hoe die werkende oplossing ontstaat, is mij niet zo duidelijk. Een heleboel van de inspiratie die daarvoor nodig was, zou wel eens afkomstig kunnen zijn van medestudenten."

Op het tentamen worden studenten aan de hand van wat vooraf gegeven code ondervraagd.

> "Een probleem met wat voorgegeven code. [...] Ze worden wel een beetje in de juiste richting geduwd. [...] Maar ze moeten wel de essentie van het onderwerp begrijpen om de vraag te kunnen beantwoorden."

Studenten komen ook niet altijd toe aan het implementeren van het programma zoals het had gemoeten.

> "Ze krijgen daar [bij de practicumopdracht, red.] wel eens wat voorgegeven maar het is toch nog een hele hoop programmeerwerk om het goed werkend te krijgen. Zoals altijd met programmeeropdrachten zijn de studenten die het minder goed beheersen heel veel tijd kwijt met de basis te implementeren, zonder daarbij aan de essentie van de opdracht toe te komen. Sommigen komen daar helemaal niet aan toe, die denken 'het werkt toch?', maar dat was het doel niet, maar meer om het programma volgens een bepaalde strategie op te zetten."

6 Conclusie en discussie

6.1 Welke leerdoelen worden er gesteld?

De leerdoelen zoals die op het moment bij de OU geformuleerd zijn, verschillen van wat de leerdoelen zouden moeten zijn naar de mening van de geïnterviewde docenten. De leerdoelen zijn op Bloom-niveau's vrij laag gespecificeerd. Het draait vooral om het kennen van begrippen en syntaxis in Java. Uit de PCK-interviews met docenten 1 en 2 blijkt dat ze syntaxis eigenlijk helemaal niet zo belangrijk vinden: het zou vooral moeten gaan om het (methodisch) kunnen oplossen van problemen. Dat zal ook gevolgen hebben voor de manier van toetsen. Die past op dit moment erg goed bij de gestelde leerdoelen: het OU-tentamen draait ook erg om syntaxis en het kennen van de stof, en veel minder om het kunnen werken met threads.

Docenten 1 en 2 vinden dat het onderwerp draden belangrijk is, omdat het steeds meer voorkomt in allerlei (bedrijfs-)toepassingen. Het is daarom belangrijk dat studenten er ook mee leren werken. Beide docenten benadrukken dat het vooral zou moeten gaan om het kunnen herkennen van een situatie waar draden nodig zijn, en dan het probleem kunnen oplossen – vooral op een hoger niveau (bijvoorbeeld met UML-diagrammen). Dat bij de implementatie meerdere manieren zijn om de oplossing te implementeren is minder belangrijk.

6.2 Waar hebben studenten moeite mee?

Uit de PCK-interviews blijkt dat docenten moeite hebben om aan te geven waar studenten precies moeite mee hebben, of meer specifiek, waarom studenten moeite hebben met threading-gerelateerde opgaven. Zowel docenten 2 en 3 gaven aan dat studenten met practicumopgaven

vaak net zo lang doorprutsen tot het werkt – al dan niet met inspiratie van medestudenten – en als het niet lukt, dan leveren zij niets in. De practicumopgaven geven de docent geen beeld van het begrip van de studenten van het onderwerp. Pas op het tentamen komen allerlei misvattingen naar voren.

Aan de tentamenuitwerkingen is te zien welke soort fouten veel worden gemaakt en in welke combinaties die voorkomen. Fouten F3 en F4 (weglaten van synchronisatie resp. threadbesturing) komen opvallend vaak tegelijk voor. Het zou kunnen dat studenten in die situatie niet herkend hebben dat er synchronisatie nodig was om inconsistente data te voorkomen, of dat zij er simpelweg niet aan gedacht hebben. Fouten F1 en F2 komen ook een aantal keer tegelijk voor (het juist op een verkeerde plaats gebruiken van synchronisatie resp. threadbesturing). Het lijkt er op die studenten lukraak wat statements opschreven, een situatie die docent 3 tijdens het PCK-interview beschreef.

Redelijk opvallend is dat de niet-draad-gerelateerde fouten het vaakst voorkomen. De fouten F8 tot en met F10 zijn zulke fouten. Dat wijst er op dat veel studenten hiaten hebben in hun kennis over programmeren in het algemeen. Het zou kunnen zijn dat zij eigenlijk helemaal geen moeite hebben met threads specifiek, maar dat algemene programmeerkennis en -ervaring ontbreekt, waardoor zij niet weten hoe de opgave aan te pakken. Dat komt ook overeen met het beeld dat docent 3 in het PCK-interview schetst: dat studenten moeite hebben met heel elementaire onderdelen van het programmeren.

6.3 Welke onderwijsaanpak wordt er gebruikt?

De lesaanpak in de lesmodule van de OU lijkt op de manier waarop docent 1 en 2 het onderwerp graag zouden aanpakken, en ook op de manier waarop docent 3 het al aanpakt. Er wordt begonnen met een eenvoudig voorbeeld waarmee wordt gedemonstreerd dat een applicatie niet meer reageert op gebuikersinvoer, wanneer die bezig is met een lange berekening. Vervolgens wordt uitgelegd wat threads zijn en hoe ze gebruikt kunnen worden om het probleem van de niet-reagerende applicatie op te lossen.

Daarna treedt er een verschil op: de lesmodule duikt hierna de wereld van de Java-syntaxis voor draden in. De geïnterviewde docenten willen het anders aanpakken: na het eerste voorbeeld wordt een moeilijker voorbeeld genomen en vervolgens opgelost (docent 1 en 2 willen een dergelijke oplossing eerst op hoger niveau beschrijven). Tenslotte wordt synchronisatie geïntroduceerd, na een voorbeeld dat aangeeft hoe het mis kan gaan als synchronisatie weggelaten wordt.

De lesmodule van de OU gaat kort in op synchronisatie. Daarbij blijft het echter bij een beschrijving van wat mis kan gaan en hoe dat opgelost kan worden. De studenten gaan er niet of nauwelijks zelf mee aan de slag – los van wat syntactische zaken. Bovendien worden zaken als ‘deadlock’ alleen genoemd, maar daar gebeurt verder niets mee. Docent 3 geeft aan begrippen als deadlock te noemen, maar er verder niet al te diep op in te gaan, tenzij het er ‘erg bovenop ligt’.

6.4 Hoe wordt het onderwerp getoetst?

De toetsing bij de OU komt overeen met de leerdoelen: er worden veel syntactische ‘weetjes’ gevraagd. De betrokken docenten gaven in de interviews aan dat ze ook dit graag willen veranderen. Het is echter lastig om goede vragen te construeren, omdat voorkomen moet worden dat als studenten niet uit de ene vraag komen, dat ze dan ook de andere niet kunnen beantwoorden. Bovendien is de beschikbare tijd op een tentamen beperkt, en moet voorkomen worden dat studenten al veel tijd kwijt zijn met randzaken waardoor zij niet meer toekomen aan de essentie van de opdracht.

Het materiaal van de RU (docent 3) is anders opgezet: er wordt een situatie geschetst en er wordt code voorgegeven. De gegeven code is onvolledig en moet worden aangepast en / of aangevuld. De opgave over threads neemt meer ruimte in op het tentamen dan in de cursus: ongeveer $\frac{1}{3}$ deel van het tentamen wordt gevuld met de draadopgave. Daarbij moet worden opgemerkt dat vaardigheid met threads niet het enige is dat wordt getoetst: het gaat ook om het objectgeoriënteerd programmeren in het algemeen. Er is bijvoorbeeld een deelopgave die vraagt om een *generic* klasse te implementeren.

6.5 Discussie en toekomstig onderzoek

Voor deze scriptie bekeken we tentamenuitwerkingen van twee vakken van de Open Universiteit resp. de Radboud Universiteit. De studentuitwerkingen van de OU bleken niet erg geschikt voor onze analyse omdat we veelgemaakte fouten er niet uit konden halen. De lijst met veelgemaakte fouten geeft dus alleen een beeld van de situatie op de RU. Om een algemener beeld van de situatie te krijgen, zou het nodig zijn om de studentuitwerkingen van meer universiteiten te bekijken. Bovendien hebben we alleen gekeken welke fouten voorkomen, maar hebben we geen verbanden tussen die fouten gezocht. Het is goed mogelijk dat er een dieper liggende misvatting heerst bij studenten, waardoor zij meerdere fouten tegelijk begaan. Hier zou nader onderzoek naar gedaan moeten worden.

De tentamenuitwerkingen geven ons bovendien geen beeld van wat de studenten precies denken op het moment dat ze een probleem (al dan niet foutief) oplossen. Waarom studenten threads een lastig onderwerp vinden en veel fouten maken, konden we niet achterhalen aan de hand van tentamenuitwerkingen. Ook docenten hadden hier geen verklaring voor. Een hard-op-denksessie met een student zou hier nuttig zijn, om te achterhalen wat er precies in het hoofd van de student omgaat.

Docenten 1 en 2 zijn directe collega's van elkaar. Ze praten geregeld met elkaar en wisselen hun meningen uit met betrekking tot zaken die we in het PCK-interview gevraagd hebben. Dat hun meningen deels overlappen, zal hier door komen. Een beperkende factor bij het aftappen van de PCK van docenten 1 en 2, is dat zij de studenten nooit aan het werk zien. De tentamens worden door een ander persoon nagekeken, en omdat het materiaal bedoeld is voor thuisstudie, zijn er ook geen practica waar de docenten kunnen rondlopen om te zien waar studenten vastlopen.

Bij het bekijken van de studentuitwerkingen treffen we een zelfde situatie als McCracken et al. (2001): namelijk dat studenten over het algemeen matig scoren, maar het vooral goed doen op syntaxis. Dat wil zeggen: programma's zijn compileerbaar, maar semantisch komen ze niet overeen met de opdracht. Ook hier lijkt het er op dat een behoorlijk aantal studenten niet de juiste oplossing weet te vinden.

Het lesmateriaal van de Open Universiteit bleek consistent met de leerdoelen. Dat was ook het geval voor het tentamen dat we bekeken. Wat de docenten eigenlijk met het vak zouden willen, wijkt hier flink van af. Wel is het zo dat wat de twee OU-docenten zeggen te willen, grotendeels overeenkomt met de opzet en aanpak van het vak dat docent 3 aan de RU geeft.

Momenteel is er een onderzoek gaande waarbij zowel de OU als de RU betrokken zijn. Het doel is uit te zoeken hoe het onderwijs met betrekking tot draden verbeterd kan worden. Uit deze scriptie blijkt dat een dergelijk onderzoek nuttig zou zijn: de docenten van de OU willen hun vak anders gaan aanpakken en hebben daar ideeën over, maar niet alles is even duidelijk. Om er achter te komen waar studenten nu echt moeite mee hebben, zou een hard-op-denksessie met een dradenprobleem gedaan moeten worden.

Literatuur

Barendsen, E. & Henze, I. (2012). *Teacher knowledge versus teacher practice: reflecting on classroom instruction and interaction through pck-related observation.* Paper for the NARST 2012 Conference, March 25-28, Indianapolis, Indiana, USA.

Bloom, B. S., Engelhart, M., Furst, E. J., Hill, W. H. & Krathwohl, D. R. (1956). Taxonomy of educational objectives: Handbook i: Cognitive domain. *New York: David McKay*, *19*, 56.

Loughran, J., Mulhall, P. & Berry, A. (2004). In search of pedagogical content knowledge in science: Developing ways of articulating and documenting professional practice. *Journal of Research in Science Teaching*, *41(4)*, 370âĂŞ-391.

Magnusson, S., Krajcik, J. & Borko, H. (1999). Nature, sources, and development of pedagogical content knowledge for science teaching. *Examining pedagogical content knowledge: The construct and its implications for science education*, *6*, 95–132.

Mannila, L. (2010). Invariant based programming in education–an analysis of student difficulties. *Informatics in Education-An International Journal*(Vol 9_1), 115–132.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., ... Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. *ACM SIGCSE Bulletin*, *33*(4), 125–180.

Saeli, M. (2012). Teaching programming for secondary school: a pedagogical content knowledge based approach. *Technische Universiteit Eindhoven*.

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational researcher*, *15*(2), 4–14.

Soloway, E., Ehrlich, K., Bonar, J. & Greenspan, J. (1982). *What do novices know about programming?* Yale University, Department of Computer Science.

Appendices

A Voorbeeld CoRe-matrix

	Belangrijke ideeën en concepten		
	idee A	idee B	idee C
1. Wat wil je dat de studenten over dit idee leren?			
2. Waarom is het voor de studenten belangrijk om dit te weten?			
3. Wat weet jij nog meer over dit idee (maar wat de studenten nog niet hoeven te weten)?			
4. Problemen en beperkingen verbonden met het onderwijzen van dit idee			
5. Kennis over het denken van de studenten die jouw onderwijzen van dit idee beÃŕnvloedt			
6. Andere factoren die jouw onderwijs van dit idee beïnvloeden			
7. Onderwijsaanpak (en de redenen om die te gebruiken bij dit idee)			
8. Specifieke manieren om achter het begrip of de verwarring van de studenten rond dit idee te komen			

Uit: Loughran, J., Berry, A., & Mulhall, P. (2006). Understanding and Developing Science Teachers' Pedagogical Content Knowledge. Rotterdam: SensePublishers.

B CoRe-matrix docent 1

1. Wat wil je dat de studenten over dit idee leren?	• Het kunnen werken (programmeren) met draden • Niet alleen theoretische kennis over het idee, maar kunnen die kennis ook kunnen toepassen • Het maken, starten en stoppen van een draad	• Het kunnen redeneren over de werking van draden • Herkennen in welke situaties het (niet) nuttig is om draden te gebruiken • Weten dat de voortgang van draden onvoorspelbaar is, en daar ook mee kunnen redeneren (bijvoorbeeld een verschijnsel verklaren)	• Methodisch nadenken over de toepassing van draden (eerst denken, dan doen)
2. Waarom is het voor de studenten belangrijk om dit te weten?	• Draden worden steeds meer toegepast, bijvoorbeeld in systemen als Android. Bovendien wordt met moderne hardware steeds meer mogelijk en krijgt de programmeur steeds meer direct te maken met multithreading-problemen. Als gevolg daarvan is er ook steeds meer vraag vanuit âĂŸde marktâĂŹ naar programmeurs die met threads overweg kunnen (voor in bijvoorbeeld apps)		• Als studenten methodisch kunnen nadenken over een probleem, en daarbij eerst een ontwerp maken voor een oplossing, dan hebben zij in ieder geval alvast een goede basis voor een programma
3. Wat weet jij nog meer over dit idee (maar wat de studenten nog niet hoeven te weten)?	• Synchronisatie tussen draden	• Formeel aantonen dat het draad-programma doet wat het moet doen	

4. Problemen en beperkingen verbonden met het onderwijzen van dit idee	• Er is te weinig tijd beschikbaar om alle onderwerpen uitgebreid te behandelen • Het onderwerp is erg dynamisch en complex • Synchronisatie tussen draden is wellicht te ingewikkeld om aan propedeusestudenten aan te bieden		• Er zijn in UML niet veel mogelijkheden om diagrammen m.b.t. draden te maken • Er is in lesmateriaal (boeken e.d.) niets te vinden over een methodische aanpak om een dradenprobleem op te lossen
5. Kennis over het denken van de studenten die jouw onderwijzen van dit idee beïnvloedt	• Het is een propedeusevak, studenten zijn wellicht nog niet ver genoeg om het onderwerp âĂŸdiepâĂŹ te behandelen • Toenemend aantal studenten heeft moeite met het aanleren van meer abstracte zaken • Studenten zijn minder gemotiveerd / minder bereid er hard voor te werken		

6. Andere factoren die jouw onderwijs van dit idee beïnvloeden			• Het is verleidelijk om meteen aan de code te beginnen, zonder er van tevoren goed over na te denken • Modelleervakken staan los van de programmeervakken waarin de modelleertechnieken bruikbaar zijn, weinig integratie tussen de vakgebieden, terwijl het beter zou zijn als een onderwerp als modelleren gedurende de hele opleiding in meerdere vakken toegepast wordt
7. Onderwijsaanpak (en de redenen om die te gebruiken bij dit idee)		• Het aanbieden van een casus, en studenten laten redeneren over de geschiktheid van een geboden oplossing ten opzichte van gestelde eisen, of de geboden oplossing zodanig aan laten passen zodat aan die eisen voldaan wordt • Aan de hand van een UML-ontwerp aannemelijk kunnen maken dat een programma goed in elkaar zit • Studenten laten redeneren over de voortgang van draden, en daarbij gegeven een stuk programmacode een bepaald verschijnsel laten verklaren	• Studenten dwingen om eerst een ontwerp voor een draadprogramma te maken in bijvoorbeeld UML, en dan te laten redeneren over het gemaakte ontwerp • Modelleren met UML meteen in het begin introduceren, zodat studenten vanaf het begin leren om methodisch te werken

8. Specifieke manieren om achter het begrip of de verwarring van de studenten rond dit idee te komen	• Redeneren studenten in termen van concepten die zij hebben geleerd? • Controleren of studenten daadwerkelijk de methodische aanpak (ontwerpen in UML en daarna implementeren) gebruiken • Studenten de relevante concepten laten opsommen		

C CoRe-matrix docent 2

1. Wat wil je dat de studenten over dit idee leren?	• Synchronisatie tussen verschillende draden kunnen programmeren (meer uitgediept dan nu het geval is; nu wordt het synchronized-keyword gebruikt, maar wordt synchronisatie tussen twee verschillende draden (als de een klaar is, kan de andere starten) niet behandeld).	• Synchronisatie kunnen bedenken in termen van processen • Problemen classificeren (herkennen wanneer draden nodig zijn)
2. Waarom is het voor de studenten belangrijk om dit te weten?	• Draden op zich zijn niet zo interessant (als er twee losse processen zijn), de problemen ontstaan bij synchronisatie. Dat kan heel complex zijn. Te diep (dingen als deadlocks) is niet van belang, maar basale synchronisatie (bijv. beschermen van gemeenschappelijke variabele) of draden die elkaar informeren dat ze klaar zijn is nuttig.	• Studenten zijn geneigd om meteen te gaan programmeren, zonder eerst goed na te denken

3. Wat weet jij nog meer over dit idee (maar wat de studenten nog niet hoeven te weten)?		• Dining philosophers, round robin scheduling en dergelijke principes zijn niet van belang voor deze cursus
4. Problemen en beperkingen verbonden met het onderwijzen van dit idee	• Voorbeelden van oplossingen m.b.t. synchronisatie die niet goed werken, werken niet op alle pcâĂŹs hetzelfde. • Studenten leveren niet-werkende oplossingen meestal niet in, dus op het tentamen zijn veel meer fouten te zien dan in de ingeleverde opdrachten (ook omdat op het tentamen trial-and-error niet mogelijk is).	• Er is beperkte tijd beschikbaar, dus er kan niet op alles diep worden in gegaan • Bij toetsing vormen stapelvragen een probleem.
5. Kennis over het denken van de studenten die jouw onderwijzen van dit idee beïnvloedt	• Het blijft altijd zoeken naar een optimum van dingen wel en niet synchroniseren (een student zou kunnen denken dat het altijd goed gaat als hij alles maar synchronized maakt, maar dat geeft een flinke impact op performance).	• Mensen denken van nature niet in termen van parallelle processen, dat maakt het onderwerp erg complex
6. Andere factoren die jouw onderwijs van dit idee beÃŕnvloeden	• De Java-API die nu gebruikt wordt is eigenlijk al achterhaald. • Het is moeilijk om een oplossing te testen, omdat als het ontwerp fout is, de uitvoering niet altijd fout verloopt (fouten zijn lastig reproduceerbaar).	• Studenten maken de opgaven thuis, waardoor uiteindelijk alleen de gegeven oplossing te zien is, maar hoe een student tot zoâĂŹn oplossing gekomen is, is niet te achterhalen. Dat maakt het lastig er achter te komen waarom studenten draden een lastig onderwerp vinden.

7. Onderwijsaanpak (en de redenen om die te gebruiken bij dit idee)	• Voorbeelden geven van problemen die met parallelliteit opgelost kunnen worden (bijvoorbeeld een langdurige taak waarbij de user interface moet blijven reageren of het producer-consumerprobleem) • In het begin eenvoudige opgaven geven met stukken voorgegeven code waar nog wat aan veranderd moet worden. • De opgaven steeds wat moeilijker maken en uiteindelijk synchronisatie introduceren.	• Studenten een ontwerp laten visualiseren, bijvoorbeeld met activity diagrams uit UML, of sequence diagrams, waarin draden en de synchronisatie daartussen wordt weergegeven • Een voorbeeld van een simpel probleem geven (bijvoorbeeld een bedrijfsproces waarin verschillende dingen tegelijkertijd gebeuren) en op hoog niveau (d.w.z.: met diagrammen) een oplossing bespreken
8. Specifieke manieren om achter het begrip of de verwarring van de studenten rond dit idee te komen	• Vragen of een bepaalde gegeven oplossing optimaal is. • Studenten een zelftoets laten maken.	• Studenten een eerste versie laten inleveren, waar geen cijfer voor wordt gegeven, maar waar aan de hand van de uitwerkingen de veelgemaakte fouten worden besproken.

D CoRe-matrix docent 3

1. Wat wil je dat de studenten over dit idee leren?	• Studenten moeten weten wanneer ze threads kunnen gebruiken • Studenten moeten weten wat typische problemen zijn die optreden bij het gebruiken van threads, en die problemen ook kunnen oplossen, bijvoorbeeld het gebruik van synchronisatie, om te voorkomen dat inconsistente datastructuren ontstaan bij gelijktijdig bewerken van datastructuren; maar ook ervoor zorgen dat threads niet nodeloos op elkaar wachten.
2. Waarom is het voor de studenten belangrijk om dit te weten?	• Om de responsiviteit van applicaties te verbeteren, kan men threads te gebruiken; dus bij een bepaalde klasse applicaties is het gewoon nodig. Het sneller uitvoeren van een bepaalde berekening is niet direct een reden, alleen al omdat het oplossen van het probleem op precies dezelfde manier zou gebeuren.

3. Wat weet jij nog meer over dit idee (maar wat de studenten nog niet hoeven te weten)?	• Er zijn meer synchronisatieprimitieven beschikbaar dan worden behandeld in de cursus. • Je kunt meer vertellen over en analyseren aan deadlock en starvation dan nu gedaan wordt.
4. Problemen en beperkingen verbonden met het onderwijzen van dit idee	• Practica worden begeleid door studentassistenten, waardoor pas op het tentamen een goed beeld van het begrip van studenten verkregen wordt
5. Kennis over het denken van de studenten die jouw onderwijzen van dit idee beïnvloedt	• Op tentamens worden opgaven nooit âĂŸfrom scratchâĂŹ opgebouwd, omdat dat te veel tijd en werk kost en daardoor niet realistisch is. Met practicumopgaven gebeurt dit wel. • Studenten van verschillende studierichtingen moeten het vak kunnen volgen (daardoor komen onderwerpen als deadlock en starvation niet heel prominent aan de orde). • Het is moeilijk te bepalen wat studenten lastig vinden, omdat niet altijd duidelijk is wat de oorzaak is van het niet kunnen beantwoorden van een vraag. Het is voor een aantal studenten Ãijberhaupt moeilijk, omdat ze moeite hebben met basis programmeertaken, bijvoorbeeld door te weinig ervaring waardoor een bepaalde basis ontbreekt. • Het onderwerp is het laatste onderwerp uit de cursus, waardoor sommige studenten het minder serieus nemen en de stof niet (goed) bestuderen. • Studenten leveren voor practicumopdrachten bijna altijd een werkend programma in, waarbij niet altijd duidelijk is hoe dat programma tot stand is gekomen (bijvoorbeeld dankzij âĂŸinspiratieâĂŹ van medestudenten). Bovendien zijn studenten er meer op gebrand een programma te maken dat âĂŸhet doetâĂŹ, maar is het programma niet altijd volgens de juiste eisen opgezet.
6. Andere factoren die jouw onderwijs van dit idee beÃŕnvloeden	• De stof die behandeld wordt, is in principe voldoende. Er is meer te behandelen, maar daar is niet zo zeer een noodzaak voor. Tijd is dus geen beperkende factor.

7. Onderwijsaanpak (en de redenen om die te gebruiken bij dit idee)	• Beginnen met een simpel voorbeeld, waarbij de applicatie niet te gebruiken is tijdens een langdurige operatie • Vervolgens wordt een oplossing voor het probleem getoond. • In een moeilijker voorbeeld gebruiken verschillende processen dezelfde data, waar synchronisatie aan te pas komt • Het gevaar van synchronisatie wordt getoond (deadlocks e.d.) • Vervolgens wordt een practicumopgave gegeven, met voorgegeven code waar een probleem in zit dat opgelost moet worden.
8. Specifieke manieren om achter het begrip of de verwarring van de studenten rond dit idee te komen	• Soms wordt een redenatievraag in het tentamen verwerkt (âĂŸgeef aan waarom de gegeven situatie gevaarlijk isâĂŹ). • Op het tentamen moeten studenten een opgave oplossen, waarbij ze de essentie van het onderwerp begrepen moeten hebben om de vraag te kunnen beantwoorden. Studenten worden in de opgave een beetje gestuurd, maar niet te veel.

mutex.unlock();