Leerbaarheid van Programmeertalen

Karel Pieterson

13 Augustus 2009

Thesis voor de titel
Master of Science (M. Sc.)

Tweejarige Deeltijdopleiding Master of Software Engineering

Afstudeerdocent : Prof. Dr. Paul Klint

Stagebegeleider: Dr. Jurgen Vinju

Opdrachtgever: Prof. Dr. Paul Klint

Publicatiestatus: openbaar

Universiteit van Amsterdam,
Hogeschool van Amsterdam,
Vrije Universiteit

UNIVERSITEIT VAN AMSTERDAM

“De autoriteit van hen die onderwijzen is heel vaak een belemmering voor diegenen die willen leren”

Cicero, Romeins filosoof, 106-43 voor Christus

"Bij het leren luisteren wij alleen naar onze eigen gedachten. Daarom kunnen we geen nieuwe gedachten horen, tenzij wij volgens nieuwe methoden leren luisteren en studeren."

George Ivanovitsj Gurdjieff, Grieks filosoof, 1872 -1949

Inhoudsopgave

Samenvatting

De leerbaarheid van een programmeertaal is geen automatisme; er moet goed over nagedacht worden en zorgvuldig gepland worden, zoals over alle onderdelen van de taal.

In de praktijk blijkt dat voor leerbaarheid een scala van hulpmiddelen wenselijk is: theorie gedoceerd door experts, handleidingen en boeken die deze theorie ondersteunen, workshops en practica, hulpmiddelen zoals intelligente programmeeromgevingen, multimedia presentaties en websites enzovoorts. Dit onderzoek laat zien dat er nog geen samenhangend ontwerp is dat gebruikt kan worden om het leren van een programmeertaal te vereenvoudigen. Het leren programmeren in het algemeen lijkt ingewikkelder te zijn dan het leren van bijvoorbeeld Frans.

Toch wijst onderzoek uit dat pedagogische benaderingen zoals exogeen constructivisme beduidend betere resultaten opleveren dan de traditionele objectivistische aanpak, waarbij een docent kennis overbrengt in een collegemodel.

Er is nog een lange weg te gaan om deze nieuwe benaderingen volledig te implementeren in het onderwijs van programmeertalen. Naast het bewust worden van de noodzaak tot formeel specificeren van de requirements die een programmeertaal meer of minder leerbaar maken staan ook de computerondersteunende hulpmiddelen die nodig zijn (zoals intelligente programmeeromgevingen, onderwijs- en toetssystemen en multimedia benaderingen) nog maar in de kinderschoenen.

De taal Rascal is onderzocht op leerbaarheid, waarbij de theoretische concepten uit de literatuur (zoals die in het onderzoek naar voren kwamen) getoetst zijn. Hieruit is gebleken dat Rascal (ondanks het ontbreken van leerbaarheidseisen in het ontwerp) al beduidend beter scoort dan andere talen , en met een relatief geringe inspanning op het niveau kan worden gebracht van talen die speciaal voor onderwijsdoeleinden zijn ontworpen, zoals Pascal.

Dat neemt niet weg dat er nog veel werk te verzetten valt, vooral op het gebied van generieke Computer Aided Instruction; er is nog een lange weg te gaan voordat de didactische ondersteuning van programmeertalen op hetzelfde niveau ligt als de ontwikkelhulpmiddelen, en het definiëren van kwaliteitseisen op het gebied van leerbaarheid even vanzelfsprekend zal zijn als het opstellen van functionele specificaties.

Abstract

The learnability of a programming language is not something that is a given; it has to be well-designed and planned, like any other aspect of that language.

In practice it turns out that there should be a rich variety in tools and teaching aids: theory lessons provided by experts, manuals and books that support the theory, workshops and assignments, and tools like intelligent development environments, multimedia presentations and websites. This paper shows that there is no coherent methodology as yet that can be used to simplify learning a new programming language and become proficient in its use. It seems that learning to program (from a language point of view) is more complicated than to learn French for instance.

Even so research shows that pedagogical approaches like exogenous constructivism have considerable better results than the traditional objectivistic model, in which a teacher imparts knowledge in a class setting.

There still is a long way to go to completely implement these new approaches in the everyday teaching of programming languages. Aside from the realization that requirements need to be specified that make a programming language more or less learnable, the computer aided tools that are necessary (such as intelligent development environments, teaching- and assessment systems and multimedia approaches) are still in their infancy.

The programming language Rascal has been evaluated for learnability, and theoretical concepts from literature (as found in this research) are checked. This evaluation has shown that Rascal (regardless of the lack of learnability requirements in its design) scores considerably better than the average programming language, and with relatively minor effort can be brought up to the level of languages that have been designed specifically for teaching purposes, like Pascal.

Still, there's a lot of work to be done, especially in the field of generic Computer Aided Instruction; there's a long road ahead of us before didactic support of programming languages is on the same level as support for developing computer programs in that same language, and the definition of requirements for learnability is as automatic as the definition of functional specifications.

Voorwoord

Het is voor een docent een interessante ervaring om weer eens als student in de schoolbanken te zitten; ik kan het iedere leraar aanbevelen, al is het alleen maar om de zaken weer eens vanuit het oogpunt van de student te bekijken.

Deze scriptie is het sluitstuk van 2 jaar werk als deeltijdstudent bij de Master of Software Engineering opleiding van de Universiteit van Amsterdam. Het is een interessante periode geweest, waarin ik een hoop geleerd heb en een deel van mijn reeds bestaande kennis kon formaliseren.
Daarnaast heb ik geleerd om via de wetenschappelijke aanpak te werken.

Ik wil hier graag een aantal mensen bedanken die dit mogelijk hebben gemaakt.

Als eerste Anke Robertus, Reza Esmaili en Kees Rijsenbrij van de Hogeschool van Amsterdam voor de toestemming om deze studie te volgen en voor het faciliteren dat ik tijd kreeg.

Mijn docenten bij de master opleiding voor hun geduld met een toch wel eigenwijze student en voor het bieden van een inspirerende omgeving, en mijn medestudenten met wie ik (soms veel te veel) tijd heb doorgebracht in werkgroepen, labs en andere samenwerkingsverbanden.

De studenten (Technische) Informatica van de Hogeschool van Amsterdam die meegedaan hebben aan mijn onderzoek.

Mijn begeleiders bij het Centrum voor Wiskunde en Informatica voor hun hulp en feedback, in het bijzonder Paul Klint en Jurgen Vinju.

En als laatste en belangrijkste, mijn vrouw en dochters voor hun geduld en begrip als ik niet altijd meteen tijd voor ze had, en voor het blijmoedig opvangen van mijn incidentele knorrigheid als er weer eens onder de nodige tijdsdruk een deadline gehaald moest worden…

Karel Pieterson,
Purmerend, 17 augustus 2009

Hoofdstuk 1: Inleiding

Hoewel ik zowel docent als software engineer ben, was het in eerste instantie niet voor de hand liggend om deze twee vakgebieden te combineren in mijn afstudeerproject. Immers, afgezien van terreinen als Computer Aided Instruction (CAI), Computer Assisted Learning (CAL) en courseware lijken er niet al te veel raakvlakken te zijn.
Door met een andere bril op naar programmeertalen te kijken (namelijk het leren versus het gebruiken) bleek er toch opeens een interessant onderwerp te zijn.

Voor een machinebouwer is het natuurlijk om machines zowel als middel en doel te zien, maar voor de meeste software engineers is een programmeertaal een hulpmiddel, en niet het eindproduct van een software engineering proces.
En zelfs voor diegenen die wel betrokken zijn bij de ontwikkeling van een taal lijken een aantal basisprincipes die gangbaar zijn bij de ontwikkeling van andere software opeens geen rol meer te spelen: zo is bijvoorbeeld “gebruikersvriendelijkheid” ver te zoeken in het gros van de programmeertalen. (Als er sprake is van uitgebreide(re) ondersteuning wordt deze vaak geboden door een (extern) ontwikkelhulpmiddel en later toegevoegd.)

Het toeval wil dat op het moment dat ik moest gaan besluiten wat mijn afstudeeronderwerp zou worden bij het Centrum voor Wiskunde en Informatica van de Universiteit van Amsterdam de ontwikkeling van een nieuwe programmeertaal (Rascal) gaande was.
Hierdoor kreeg ik de kans om een onderzoek te doen naar de wijze waarop het leren van een programmeertaal beïnvloed wordt door taalconstructies, onderwijsmethodes, (gebrek aan) ondersteuning vanuit beschikbare ontwikkeltools en het al dan niet bewust integreren van leerbaarheid in het eindproduct.

Deze scriptie is het eindresultaat van dat onderzoek.

De opbouw is als volgt.
Eerst wordt in hoofdstuk 2 de probleemstelling uiteengezet met de bijbehorende scope in de vorm van een onderzoeksvraag, waarna in hoofdstuk 3 wordt uiteengezet hoe het onderzoek is aangepakt.
Hoofdstuk 4 beschrijft de achtergrond en context, en bevat de resultaten van de literatuurstudie die voor deze scriptie uitgevoerd is; vervolgens wordt in hoofdstuk 5 de opzet en uitvoering van het onderzoek beschreven.
In hoofdstuk 6 worden de resultaten van dit onderzoek gepresenteerd en hoofdstuk 7 besluit deze scriptie: hierin worden de conclusies en interpretatie van de onderzoeksresultaten weergegeven, samen met een reflectie op de uitvoering en een indicatie van eventuele onderwerpen en inzichten voor verder onderzoek.

Hoofdstuk 2: Probleemstelling

2.1 De rationale achter programmeertalen

Op het moment dat de eerste computer gecreëerd werd ontstond ook de eerste programmeertaal om deze computer te besturen. Er was immers een communicatiemiddel nodig om problemen uit de werkelijkheid te vertalen naar een vorm waarin deze opgelost konden worden door de machine. De programmeertaal was dan ook primair bedoeld om een vertaalslag te maken van natuurlijke taal naar een voor de machine verwerkbare vorm; de beschikbare functionaliteit had betrekking op beschrijving van oplossingsalgoritmes.

Deze situatie is vandaag de dag nog steeds zo, hoewel de keuze voor een taal veel groter is geworden. In plaats van de Siamese tweeling die hardware en besturing in het begin vormden zijn programmeertalen nu in toenemende mate platformonafhankelijk en multifunctioneel inzetbaar, en zijn bijvoorbeeld declaratieve en imperatieve talen beschikbaar.

Het is vanzelfsprekend dat programmeertalen vanuit een functioneel perspectief ontstaan; het is immers een stuk gereedschap dat probleemoplossing moet ondersteunen. Het wordt gezien als een middel en niet als doel.

2.2 Leerbaarheid

Toch is er in een ander opzicht niet veel vooruitgang geboekt sinds de opkomst van de computer: programmeertalen worden nog steeds primair bedacht en ontwikkeld gebaseerd op functionaliteit en domein. Een enkele uitzondering daargelaten (BASIC, Pascal) wordt er bij nieuwe talen geen aandacht geschonken aan de *leerbaarheid* van programmeertalen. Hoogstens wordt er een laagje vernis over de taal heen gelegd, zoals bij BlueJ en Java.

Ontwikkelingen in de pedagogiek en didactiek, die volop toegepast worden bij het leren van natuurlijke talen lijken voorbij te gaan aan nieuwe programmeertalen; bij recente talen als Java en Ruby is de leercurve niet in de ontwerpeisen opgenomen, en is aan ondersteuning bij het aanleren (in ieder geval door de ontwerpers en ontwikkelaars) geen aandacht besteed.

Robin, Rountree en Rountree merken hierover het volgende op: "*Learning to program is hard however. Novice programmers suffer from a wide range of difficulties and deficits. Programming courses are generally regarded as difficult, and often have the highest dropout rates. It is generally accepted that it takes about 10 years of experience to turn a novice into an expert programmer*." [Rob03]. Als dit fenomeen zich ook op andere terreinen zou voordoen zou het verwerven van nieuwe kennis een wel zeer moeizame aangelegenheid worden, om nog maar te zwijgen over het krijgen van hoge mate van vaardigheid...

Als hier wat langer bij stil gestaan wordt is bovenstaand probleem eigenlijk vreemd: zou het immers niet zo kunnen zijn dat niet alleen de kracht en functionaliteit van een willekeurige programmeertaal invloed heeft op de acceptatie en het gebruik, maar dat ook de steilte van de leercurve daar mee te maken heeft? Het ligt voor de hand te veronderstellen dat talen die niet alleen aan de functionaliteiteisen voldoen, maar ook snel onder de knie te krijgen zijn, zich in een grotere populariteit in onderwijs en bedrijfsleven zouden mogen verheugen. Hierdoor neemt het aantal gebruikers toe, waardoor verdere investeringen in deze taal eenvoudiger te justificeren zijn.

Er is dus een interessant gebied dat open ligt: hoe kan leerbaarheid expliciet in de programmeertalen ingebouwd worden en hoe kunnen methodes en IT- hulpmiddelen toegepast worden in het onderwijzen van programmeren.
Dat dit niet onbelangrijk is moge blijken uit de observaties die Ruven Brooks al in 1977 deed, bij zijn onderzoek naar de cognitieve processen van het programmeren: "*… While these studies have not been intended to provide information on cognitive aspects of programming, they have produced the important finding that errors or difficulties in programming are not random occurrences, produced by random occurrences in the programmer's environment. Instead, they are clearly linked to specific features or properties of programming languages*." [Bro77]

2.3 De problematiek van het leren programmeren

Het leren van een (nieuwe) programmeertaal is niet slechts het aanleren van nieuwe constructen in die taal; bij het toepassen van het geleerde speelt ook probleemoplossend vermogen binnen een specifiek domein een rol.

Het leren programmeren (leren in het algemeen eigenlijk) bestaat uit een gefaseerd proces, waarbij een aantal stappen worden doorlopen. Elke stap is noodzakelijk als bouwsteen in dit proces, waarop een volgende stap weer kan voortbouwen.
Eerst moet de student de syntax van de nieuwe taal leren, de primaire bouwstenen en de hulpmiddelen die ter beschikking staan (fase 1). In fase 2 moet de samenhang tussen concepten duidelijk worden, waarna in fase 3 de semantiek en de verwerkingswijze wordt geleerd.

Na fase 3 kan de student werken met de programmeertaal als gereedschap; vanaf dit moment kan met behulp van deze specifieke programmeertaal een oplossing gemaakt worden voor een specifiek probleem.
De 4^{e} fase van het programmeren is niet taalgebonden: het duidelijk krijgen van het probleem is voor elk willekeurig hulpmiddel hetzelfde.
Het maken van een specifieke oplossing, evalueren hiervan en zoeken naar alternatieven hebben wel weer duidelijk een verband met de (on)mogelijkheden die de programmeertaal biedt.

Bovenstaande indeling is universeel voor het leren in het algemeen, en is oorspronkelijk bedacht door Benjamin S. Bloom (1913 – 1999). Bloom is in wetenschappelijke kring op grote schaal geaccepteerd als autoriteit op het gebied van educatie, en hij heeft in 1956 een

taxonomie ontwikkeld waarin het bovenstaande model past: een classificatie van de verschillende doelstellingen die aan een leerproces ten grondslag liggen. Bloom onderkent hierbij 6 fasen: kennis, begrip, toepassing, analyse, synthese en evaluatie.

Het is mogelijk om het leren van een programmeertaal te projecteren op de fasen van Bloom's taxonomie (zie 4.2.1). Dit zorgt voor een gestructureerde aanpak van (de evaluatie van) het leerproces en leidt tot de volgende tabel:

Blooms categorie	Leren programmeren
Knowledge	Tools, constructs, syntax
Comprehension	Relating concepts
Application	Flow, semantics
Analysis	Understand the problem
Synthesis	Create the solution
Evaluation	Assess other options

Leren programmeren als fasen in Bloom's taxonomie [Por04]

De effectiviteit van het leerproces is afhankelijk van het "zelf doen": "*Programming is not difficult only because of the abstract concepts. Students have also problems in different issues related to program construction. It is important for the learning that the students do programming by themselves. With carefully designed materials and approaches teachers can guide students knowledge and skill construction*." [Lah05]

2.4 Onderzoeksvraag

De onderzoeksvraag die aan deze scriptie ten grondslag ligt is:
"Welke kwaliteitseisen en hulpmiddelen bepalen de leerbaarheid van een programmeertaal?"

Om hier inzicht in te krijgen zal naar de volgende aspecten worden gekeken.

1. Wat zijn gangbare theorieën en concepten op het gebied van cognitie en leren in de context van het programmeeronderwijs? Hoe succesvol zijn deze? Welke kwaliteitseisen zijn hieruit te destilleren?
2. Welke (computergestuurde) hulpmiddelen kunnen ingezet worden?
3. Wat is hiervan toepasbaar bij het leren en doceren van Rascal; welke strategieën zijn succesvol?
4. Lijken de gevonden resultaten ook toepasbaar op andere programmeertalen? Zo ja, hoe? Zo nee, waarom niet?
5. Wat zijn de investeringen die de leerbaarheid van Rascal op een hoger niveau kunnen brengen? Kan er ook iets als een "wet van verminderende meeropbrengst" worden bepaald?

De scope van het onderzoek beperkt zich tot theorieonderzoek naar toepassing van bestaande leermodellen voor programmeurs; er wordt geen onderzoek gedaan naar geheel nieuwe wijzen van leren.

Daarnaast wordt Rascal gebruikt als “proof of concept”. Gezien de beperkte tijd die beschikbaar is zal primair theoretisch gekeken worden of de uitkomsten op grotere schaal bruikbaar zijn; door middel van een klein experiment met studenten zal op empirische wijze onderbouwing worden gezocht.

2.5 Doelstelling

De doelstelling van dit onderzoek is meervoudig. Op basis van de onderzoeksvraag worden de volgende deelresultaten gedefinieerd.

A. In kaart brengen van bestaande inzichten en theorieën op het gebied van cognitie en onderzoeken naar programmeeronderwijs.
B. Inzicht krijgen in het leerproces zoals dat geldt voor programmeertalen. Is dit leerproces vergelijkbaar met het verwerven van kennis op andere terreinen? Zo ja, kan hiervan dan gebruik gemaakt worden? Zo nee, wat is dan anders, wat zijn de consequenties van deze verschillen, en hoe kan hiervoor een oplossing worden geboden?
C. Het vinden van geschikte leerstrategieën en hulpmiddelen binnen de gestelde context.
D. Onderzoeken van de mogelijkheden tot computerondersteuning van het leerproces.
E. Toetsen van bovenstaande bevindingen in een onderzoek aan de hand van de taal Rascal. Daarbij wordt tijdens de ontwikkeling getoetst hoe aan nieuwe gebruikers het meest efficiënt en effectief kennis over Rascal kan worden overgedragen. De bevindingen worden teruggekoppeld naar het ontwikkelteam voor mogelijke toepassing in het ontwikkelproces.
F. Ontwikkelen van (een aanzet tot) een instructiemodel voor Rascal, geschikt voor doceren van de taal aan toekomstige studenten van de opleiding Master of Software Engineering.

Hoofdstuk 3: Onderzoeksmethode

3.1 Methodologie

Binnen dit onderzoek zal gebruik gemaakt worden van literatuuronderzoek en een beperkt empirisch experiment, gericht op het inrichten van een onderwijsmodel.
Gezien de noodzakelijke beperkingen in tijd en omvang van het empirisch onderzoek is de verwachting dat dit experiment resultaten met beperkte statistische relevantie zal opleveren.
Validatie van de gevonden resultaten zal dan ook beperkte kwantitatieve waarde hebben en deels aan de uit de literatuur verkregen inzichten worden getoetst, deels door het gebruik van vragenlijsten bij HBO-informatica studenten.

3.2 Literatuurstudie

Voor de theoretische onderbouwing van dit onderzoek zal gebruik worden gemaakt van bestaand onderzoek op het gebied van cognitie, pedagogiek, didactiek en leertheorie in relatie tot programmeeronderwijs.
Hierbij zal ingegaan worden op de onderdelen 1 en 2 van de onderzoeksvraag, en onderzoeksdoelstelling A tot en met D.

3.3 Empirisch onderzoek

Voor het empirisch onderzoek zal een klein experiment worden opgezet om leerstijlen te inventariseren.
Hiervoor zal gebruik gemaakt worden van studenten van de Hogeschool van Amsterdam, Instituut voor (Technische) Informatica, die een opleiding op het gebied van software engineering volgen.
Deze studenten hebben allemaal minimaal een basiskennis van programmeren in Java, zodat theoretische kennis en praktische vaardigheden aanwezig zijn.
De reden dat gekozen is voor deze groep studenten is tweeledig:

- ze voldoen aan de doelgroep ;
- het is een groep waar ik betrekkelijk eenvoudig toegang toe heb omdat ik software engineering en programmeren doceer op deze opleiding.

Om duidelijk inzicht te krijgen in ervaring en vaardigheid zal alle deelnemers bij aanvang een korte enquête worden afgenomen om eventuele niveauverschillen in kaart te brengen.
In het experiment zullen een aantal relatief eenvoudige opdrachten worden aangeboden die door de deelnemers moeten worden uitgevoerd.

Op basis van gevonden verschillen tussen de resultaten van groepen studenten zal getracht worden om conclusies te trekken, en een antwoord te geven op deelvragen 3 tot en met 5 en zullen doelstellingen E en F van het onderzoek ingevuld worden.

De details van dit experiment worden beschreven in hoofdstuk 5 (Onderzoek), en de resultaten komen aan de orde in hoofdstuk 6 (Resultaten).

Hoofdstuk 4: Theoretische Achtergrond en Context

4.1 Inleiding

In het algemeen volgt het aanleren van programmeren een klassiek patroon: de theorie wordt door een docent aan de groep gepresenteerd, er worden misschien workshops gehouden en er worden oefeningen aangeboden waarin de gedoceerde stof (praktisch) toegepast wordt.
Bovendien wordt het materiaal doorgaans aangeboden van (zeer) concreet tot abstract: we beginnen bij de meest elementaire structuren, zoals types van variabelen, en breiden op basis hiervan de kennis steeds uit naar meer complexe onderwerpen, totdat de specifieke programmeertaal tot in details beheerst wordt.

Er hebben echter belangrijke ontwikkelingen plaatsgevonden op het gebied van onderwijs en er zijn nieuwe hulpmiddelen en inzichten: de hiervoor al genoemde CAI en CAL, e-learning, hulpmiddelen en methodes die zich aanpassen aan de individuele student, praktijkonderwijs, en constructivisme om er maar een paar te noemen.

Vanuit dit gezichtpunt is het op zijn minst bevreemdend dat het programmeeronderwijs hier tot op heden niet of nauwelijks op inspeelt, en dat bij het ontwerp van moderne programmeertalen (zoals Java en Ruby) er geen aandacht geschonken lijkt te worden aan de ondersteuning van het leren van zo'n nieuwe taal.

In dit hoofdstuk zal eerst gekeken worden naar onderzoek op het gebied van bestaande leertheorieën, waarbij de nadruk zal liggen op constructivisme. Met behulp hiervan zal geprobeerd worden om een leerstrategie te formuleren die het aanleren van een programmeertaal eenvoudiger zou moeten maken.
Vervolgens zal gekeken worden naar mogelijke ondersteuning van deze leerstrategie met behulp van IT- hulpmiddelen; onderzoek naar bestaande hulpmiddelen wordt in kaart gebracht, en eventuele bruikbaarheid wordt geëvalueerd.

Als een samenhangend geheel van strategie en hulpmiddelen is ontstaan, zal getoetst worden in hoeverre Rascal daar in past; bovendien zal gekeken worden of het ook mogelijk is om ondersteuning alsnog in te bouwen, indien deze niet aanwezig is.

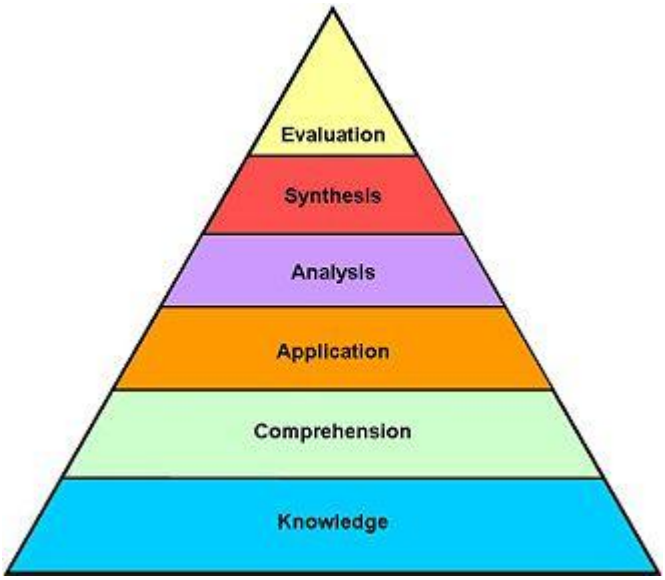
Gebaseerd op de resultaten zal een empirische toetsing worden uitgevoerd, die in hoofdstuk 5 beschreven wordt.

4.2 Programmeeronderwijs: bestaande inzichten, theorieën en pedagogische modellen

Op het gebied van cognitie, en naar het aanleren van vaardigheden is in algemene zin al een grote hoeveelheid onderzoek gedaan. Op basis daarvan zijn meerdere theorieën en modellen ontwikkeld. Hiervan blijken een aantal relevant te zijn vanuit de optiek van het leren programmeren.

4.2.1 De taxonomie van het cognitieve domein van Benjamin Bloom

Het aanleren van programmeertalen wordt ingedeeld in het cognitieve domein, omdat het hier gaat over kennis, begrip en intellectuele vaardigheden. Om een kader te scheppen voor de modellen wordt hier eerst een korte uiteenzetting gegeven van de indeling zoals Bloom deze in de vijftiger jaren heeft ontwikkeld (figuur 1). Hierin wordt de taxonomie van leerdoelstellingen beschreven, die een belangrijk uitgangspunt biedt voor het ontwikkelen van effectief onderwijs.



Figuur 1 Taxonomie van Bloom [Wul05]

De taxonomie beschrijft "*a hierarchy of learning outcomes which range from simple rote memorization to the deeper levels of understanding that allow for expert-level application of acquired knowledge to solve complex domain specific tasks. This deeper level of understanding and the ability to flexibly apply knowledge should be the goal of any instruction*." [Wul05]

Het idee achter deze taxonomie is dat de door de docent gestelde leerdoelen gerangschikt kunnen worden in een hiërarchie, oplopend van eenvoudig tot complex.

Deze hiërarchie kent de volgende niveaus.

- Knowledge (kennis): de student herkent of herinnert zich informatie, ideeën en principes die in ongeveer dezelfde vorm als waarin ze geleerd zijn worden aangeboden.
- Comprehension (begrip): de student vertaalt, begrijpt of interpreteert informatie, gebaseerd op eerder opgedane kennis.
- Application (toepassing): de student selecteert, gebruikt en past gegevens en principes toe om een probleem of een taak op te lossen met een minimum aan sturing.
- Analysis (analyse): de student onderscheidt, classificeert en ziet relaties tussen de aannames, hypothesen, bewijsvoering of structuur van een uitspraak of vraag.
- Synthesis (synthese): de student vormt, integreert en combineert ideeën tot een product, plan of voorstel dat nieuw voor hem is.
- Evaluation (evaluatie): de student beoordeelt op basis van specifieke criteria.

Uitgangspunt van de taxonomie is dat studenten op verschillende niveaus kennis kunnen hebben.

4.2.2 Objectivistische pedagogiek

Dit is de “traditionele” pedagogiek; deze stelt de docent centraal. De docent is de deskundige op een bepaald gebied en de enige bron van kennis. Er is sprake van een actieve docent en een passieve student.

De docent helpt de student bij het verwerven van de onderste laag (kennis); in hoeverre de student daarna doordringt in de hogere lagen van de hiërarchie is afhankelijk van de individuele student; de methode ondersteunt dit niet.

4.2.3 Constructivistische pedagogiek

De objectivistische aanpak lijkt niet geschikt te zijn om de student in staat te stellen zijn volledige potentieel (zoals beschreven in de taxonomie van Bloom) te realiseren, een enkeling daargelaten. Daarom is dit niet de juiste vorm om modern onderwijs in te ontwikkelen.
Om de student te activeren kan gebruik gemaakt worden van de constructivistische pedagogiek, ontwikkeld door Piaget [Dal01].

Deze steunt op drie uitgangspunten:

1. elk persoon vormt zijn eigen kennisrepresentatie, gebaseerd op zijn eigen ervaring. Hieruit volgt dat er geen “enig juiste“ representatie is;
2. mensen leren door actieve verkenning van een kennisgebied, en dit leren vindt plaats wanneer tijdens deze verkenning een inconsistentie ontdekt wordt tussen de huidige representatie van hun (formele) kennis en hun ervaring;
3. leren vindt plaats binnen een sociale context, en interactie tussen studenten is een noodzakelijk deel van het leerproces.

Gebaseerd op bovenstaande uitgangspunten citeert Dalgarno verschillende interpretaties van constructivisme:

"These different interpretations of constructivism have been labelled by Moshman (1982) as endogenous, exogenous and dialectical, as follows:

- *Endogenous constructivism emphasises the individual nature of each learner's knowledge construction process, and suggests that the role of the teacher should be to act as a facilitator in providing experiences which are likely to result in challenges to learners' existing models.* (Ook Jenkins stelt (in [Jen01]): "*... the primary role of a teacher of programming is not as a communicator of information, as in many other subjects or areas of computing. Rather, the teachers main role is that of a motivator."*).
- *Exogenous constructivism is the view that formal instruction, in conjunction with exercises requiring learners to be cognitively active, can help learners to form knowledge representations which they can later apply to realistic tasks.*
- *Dialectical constructivism is the view that learning occurs through realistic experience, but that learners require scaffolding provided by teachers or experts as well as collaboration with peers."*

Het uitgangspunt voor het onderzoek in deze thesis is de tweede vorm, exogeen constructivisme. De reden hiervoor is dat deze pedagogische aanpak aansluit bij moderne onderwijsinzichten en al de dagelijkse praktijk is.
Hierbij is niet alleen een rol weggelegd voor formele instructie, maar daarnaast komen ook in meer of mindere mate andere genoemde wekvormen (practica, workshops, multi-media materiaal en realistische cases) aan de orde. De methode biedt daar niet alleen de ruimte voor, maar is er zelfs expliciet op gebaseerd; de docent speelt daarnaast nog steeds een rol van betekenis in het leerproces van de student, maar de student is eigenaar van zijn eigen proces en kan voortgang bereiken met verschillende hulpmiddelen en leerstijlen.
En, last but not least: vooral in het programmeeronderwijs, dat gebaseerd is op het leren inzetten van computers voor probleemoplossing zou het vreemd zijn als diezelfde computer niet waar mogelijk als hulpmiddel wordt gebruikt!

Dit alles is in tegenstelling tot de objectivistische pedagogiek waarbij de leerbaarheid alleen bepaald wordt door de kwaliteit van het materiaal dat door de docent gepresenteerd wordt en de didactische vaardigheden van de betreffende docent, en de endogene vorm, waarbij de docent geen formele rol van betekenis speelt.

4.2.3 Exogeen constructivisme

Kenmerkend voor exogeen constructivisme is dat er wel plaats is voor directe instructie, maar niet volgens het eenrichtingprincipe van de objectivisten: "*... learners should have some control over the sequence and selection of content, should have the opportunity to actively construct their own knowledge representations and articulate these representations at all stages, and, after instruction should have the opportunity to apply their knowledge to realistic tasks. Constructivist CAL materials that draw on the exogenous view include tutorials that incorporate learner control over sequence...*" [Dal01]

Vanuit deze gedachtegang zou idealiter, naast de directe instructie en het materiaal daarvoor, een CAL omgeving voor het leren van een programmeertaal beschikbaar moeten zijn die de volgende componenten moet bieden:

- toetsen en testen die op ieder willekeurig moment (weer) door een student afgelegd kunnen worden, met een directe feedback over het resultaat;
- hulpmiddelen voor "concept mapping", waarmee grafische mindmaps gemaakt kunnen worden die het kennisgebied beschrijven en waarbinnen vastgelegd kan worden hoe concepten met elkaar samenhangen;
- tutorials die door de student in een individueel te bepalen volgorde kunnen worden doorlopen;
- een intelligente programmeeromgeving (IDE) die de student ondersteunt gedurende zijn groei van beginner tot gevorderde ontwikkelaar;
- realistische taken en voorbeelden die gebruikt kunnen worden om kennis te verwerven en uit te bouwen. Hierbij kan gebruik gemaakt worden van raamwerken, deeloplossingen die verder uitgewerkt moeten worden.

Deze lijst eisen vloeit voort uit de achterliggende gedachten van het exogeen constructivisme, zoals Dalgarno dat beschrijft; als hieraan voldaan wordt kan de student eigenaar worden van zijn individuele leertraject.
Dit is een generieke indeling; in het vervolg van deze scriptie zal getoetst worden of dit ook op Rascal van toepassing is.

4.3 Leerstrategieën

Als gekeken wordt naar de gevolgde leerstrategieën komt bij bestudering van beschikbare onderzoeken aan het licht dat, ongeacht de taal, een student eerst kennis moet verkrijgen van de elementaire structuren. In hun onderzoek naar de problemen van beginnende programmeurs stellen Lahtinen c.s: "*The most common discussion topic in the literature of today seems to be whether imperative or object-oriented approach should be the first. Whatever the approach, at some point the students have to learn the basic structures of the programming languages such as loops, variables, recursion, and parameter passing…Students also have problems with understanding that each instruction is executed in the state that has been created by the previous instructions.*" [Lah05]

Uit hetzelfde onderzoek blijkt dat bij inventarisatie van de leersituatie en –materialen de studenten het volgende vonden:

- alleen studeren is zinvoller dan hoorcolleges volgen;
- alleen werken aan opgaven is zinvoller dan werkcolleges en lab sessies;
- werkcolleges en labsessies zijn zinvoller dan hoorcolleges;
- alleen programmeren is zinvoller dan alleen studeren.

Zowel docenten als studenten vinden voorbeeldprogramma's het meest bruikbare materiaal.
(Het wordt niet duidelijk uit de studie of "alleen" gebruikt wordt in de betekenis van "individueel" of "met meerdere studenten maar zonder begeleiding"; het is wel interessant om te zien dat de inbreng en toegevoegde waarde van een docent in een hoorcollege setting niet hoog aangeslagen wordt…)

Dit betekent dat volgens studenten de volgende werkvormen de voorkeur verdienen (in toenemende mate van effectiviteit):

- hoorcolleges volgen
- alleen studeren
- werkcolleges en labsessies
- alleen werken aan opgaven (practica)

Dit heeft echter wel consequenties voor de wijze van aanbieden van het lesmateriaal. De taal moet geschikt zijn voor "bottom-up" leren, er moet voldoende materiaal van de juiste kwaliteit aanwezig zijn om zelfstudie mogelijk te maken en de mate van interactiviteit tussen de student en de omgeving moet het leerproces ondersteunen.

4.4 Kwaliteitseisen vanuit het oogpunt van leerbaarheid

Manilla en De Raadt hebben een onderzoek gedaan naar de bruikbaarheid van programmeertalen voor het leren programmeren. Daaruit kwam de volgende tabel: [Man06]

	C	C++	Eiffel	Haskell	Java	JavaScript	Logo	Pascal	Python	Scheme	VB
Learning											
Is suitable for teaching (§2.1.1)			✓				✓	✓	✓		
Can be used to apply physical analogies (§2.1.2)			✓		✓	✓	✓		✓		✓
Offers a general framework (§2.1.3)	✓	✓	✓		✓	✓		✓	✓		✓
Promotes a design driven approach for teaching software (§2.1.4)			✓	✓	*1		✓			✓	
Design and environment											
Is interactive and facilitates rapid code development (§2.2.1)				✓			✓		✓	✓	
Promotes writing correct programs (§2.2.2)		*2	✓		*2				*2		
Allows problems to be solved in "bite-sized chunks" (§2.2.3)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Provides a seamless development environment (§2.2.4)			✓		*1						
Support and Availability											
Has a supportive user community (§2.3.1)	✓	✓	✓	✓	✓	✓			✓	✓	✓
Is open source, so anyone can contribute to its development (§2.3.2)									✓		
Is consistently supported across environments (§2.3.3)	✓	✓	✓		✓	✓	✓	✓	✓	✓	
Is freely and easily avalaible (§2.3.4)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Is supported with good teching material (§2.3.5)		✓	✓		✓		✓	✓	✓	✓	✓
Beyond Introductory Programming											
Is not only used in education (§2.4.1)	✓	✓	✓		✓	✓			✓		✓
Is extensible (§2.4.2)	✓	✓	✓		✓				✓		✓
Is reliable and efficient (§2.4.3)	✓	✓	✓		✓	✓		✓	✓		✓
Is not an example of the QWERTY phenomena (§2.4.4)		✓	✓	✓	✓	✓	✓		✓	✓	✓
Author's score	**8**	**11**	**15**	**6**	**14**	**9**	**9**	**7**	**15**	**8**	**9**

*1 Possibly with some IDE's, e.q. BlueJ(http://www.bluej.org)
*2 Possibly with unit testing

Het interessante aan hun onderzoek is niet zozeer bovenstaande tabel (hoewel uit de scores naar voren komt dat er geen voorkeur lijkt te zijn voor gecompileerde versus scripttalen, maar dat OO principes wel belangrijk lijken te zijn); ze hebben echter een aantal kwaliteitseisen vastgesteld die de mate van leerbaarheid bepalen.
Het is dan ook interessant om te zien in hoeverre Rascal al dan niet aan deze kwaliteitseisen voldoet, en welke score Rascal haalt. Deze confrontatie zal in hoofdstuk 6 gebeuren.

4.5 IT hulpmiddelen

In de voorgaande secties van dit hoofdstuk is relatief veel aandacht geschonken aan leermodellen en pedagogische theorie.
Hoewel dit nodig is om een onderbouwing te geven voor de keuzes die gemaakt moeten worden bij het inrichten van een leeromgeving is dit primair een onderzoek naar de wijze waarop een onderdeel van software engineering (het programmeren) kan worden geleerd en welke aspecten daar invloed op uitoefenen (en in welke mate).

Daarom wordt het nu tijd om de delen van de CAL omgeving, die in het empirisch onderzoek gebruikt worden, in te gaan vullen. Voor alle in de voorgaande paragraaf genoemde componenten zal gezocht worden naar een hulpmiddel dat de gewenste functionaliteit biedt (zonder al een concrete implementatie te realiseren; dit zal in de volgende fase plaatsvinden, beschreven in hoofdstuk 5).

4.5.1 Computerondersteuning van het leerproces

Requirement: “een CAL omgeving voor het leren van een programmeertaal [zou] beschikbaar moeten zijn”

In het verleden is al onderzoek verricht naar het ondersteunen van het leerproces van programmeren, vooral vanuit talen als LISP, LOGO en Smalltalk.

Er zijn verschillende manieren mogelijk om naar een indeling van ondersteunende systemen te kijken: Witschial onderkent de volgende vier categorieën [Wit95].

1. Onderwijssystemen, die de gebruiker expliciet kennis laten verwerven over een probleemdomein. Hierbij is het van belang om te onderkennen hoe, en in welke volgorde, kennis aan de gebruiker wordt aangeboden. Dit betekent dat een profiel van de gebruiker moet worden bepaald (de individuele kennis en vaardigheden), waarop de wijze van aanbieden wordt gebaseerd. Het primaire kenmerk van dit type system is dat het een dialoog met de gebruiker aangaat. Een minimale eis is dat het systeem kan onderkennen wanneer hulp nodig is en dan intervenieert.
2. Debuggers, die gebruikt worden als het programmeerwerk al achter de rug is. Ondersteunen het foutzoeken, en zullen normaliter weinig of geen proactiviteit hebben (hoewel nieuwere programmeeromgevingen zoals Eclipse en Netbeans wel potentiële problemen melden). Dit type hulpmiddel zal doorgaans door de programmeur moeten worden geactiveerd.
3. De geïntegreerde ontwikkelomgeving (IDE), waar een verzameling hulpmiddelen (editors, debuggers, profilers en andere tools) voor het ontwikkelproces bij elkaar is gebracht. Hier kan sprake zijn van signalen vanuit de IDE aan de gebruiker die

aangeven dat bepaalde programmeerconstructen potentieel een probleem kunnen gaan opleveren.

4. Microwerelden, leeromgevingen waarin de gebruiker op speelse wijze nieuwe concepten kan ontdekken en verkennen. Hierbij is er geen ondersteuning door een docent, menselijk of computergestuurd; de gebruiker is alleen in de microwereld en bepaalt zelf de studievoortgang. Dit betekent wel dat in het ontwerp van een microwereld het aanleren van goede oplossingsstrategieën een belangrijk aspect is (dit in tegenstelling tot de eerder genoemde onderwijssystemen, die beweren het leerproces te sturen en versnellen). Het microwereld concept is gebaseerd op het werk van Papert en Piaget, waarbij het uitgangspunt is dat de grotere wereld kan worden begrepen vanuit kleinere onderdelen die eenvoudig genoeg zijn om te begrijpen.

Er is natuurlijk overlap te vinden in bovenstaande classificaties. Zo is het tegenwoordig vanzelfsprekend dat programmeertalen een geïntegreerde ontwikkelomgeving bieden als een integraal hulpmiddel, waardoor hulpmiddelen uit categorie 2 en 3 geboden worden. (Zo wordt Rascal binnen het Eclipse framework gebouwd, waardoor allerlei functionaliteiten default beschikbaar komen.)

In een recenter onderzoek is gekeken naar beschikbare tools, en is van uit die invalshoek een indeling gemaakt [Gom05].

1. Hulpmiddelen die een eenvoudige en beperkte ontwikkelomgeving bieden. Hierbij wordt de complexiteit van de ontwikkelomgeving aangemerkt als een bron van problemen bij het leren of onderwijzen van programmeren.
2. Op voorbeelden gebaseerde hulpmiddelen, waarbij de veelheid aan beschikbare voorbeelden als uitgangspunt kan dienen voor het oplossen van nieuwe problemen.
3. Hulpmiddelen die uitgaan van visualisatie en animatie. Hierbij wordt het gedrag van de code als ondersteuning bij het leerproces gebruikt.
4. Simulatie-omgevingen; deze zijn hetzelfde als Witschial's microwerelden.

Gómez-Albarrán stelt in [Gom05]: "*The tools described in the last three categories, while following different approaches, are more related to the understanding of programming concepts and structure, as well as the syntax and semantics of programming languages. These tools are appropriate for students at any level.*"
Hierbij gaat haar voorkeur uit naar de omgevingen die visualisatie, animatie en simulatie ondersteunen: "*When these environments are used to explore ready-to-use programs, students emphasize the importance of context-sensitive documentation windows that relate the visualization to the code. Interactive prediction is generally well received in both kinds of environments. Most students consider interactive prediction to be the feature most conducive to understanding the code.*"

Maar er moet nog veel werk verricht worden naar het effectief genereren en gebruiken van interactiviteit; vragen moeten automatisch gegenereerd worden als de student code invoert. Gómez-Albarrán hierover: "*There are significant computer-assisted instruction and artificial intelligence issues involved in the effective generation of such questions.*"

Helaas zijn er geen raamwerken die gebruikt kunnen worden als uitgangspunt voor een willekeurige programmeertaal X.
In het verleden is al uitgebreid onderzoek gedaan in dit veld. In [And86] wordt op basis van een Lisp tutor een architectuur voor computerhulpmiddelen voorgesteld die bestaat uit een "tutoring engine" en een "problem solver". De engine is verantwoordelijk voor een interactieve stap-voor-stap ondersteuning van een student die een programmeerprobleem oplost, waarna voor bepaling van de juistheid van de oplossing en de relevante feedback gebruikt gemaakt wordt van een "solution trace", die op voorhand door de problem solver is gegenereerd.

Systemen die op basis van deze architectuur werken zijn in staat om een vergelijking te maken tussen de gegenereerde en de ingevoerde oplossing en op basis daarvan intelligente feedback te geven. De complexiteit van dit soort systemen is hoog: er zijn vele wegen die naar Rome leiden. Hoe bepaalt het systeem de juistheid van de oplossing, en hoe wordt berekend wat de kwaliteit is ten opzichte van het gegenereerde antwoord? En hoe wordt dit toepasbaar gemaakt op verschillende programmeertalen, dus inzetbaar voor Java, maar ook voor Rascal en Ruby?

Een recent voorbeeld van een hulpmiddel dat op bovenstaande architectuur is gebaseerd is VIOPE. In een empirisch onderzoek [Car06] geven studenten aan dat ze wel behoefte hebben aan een dergelijk tool, maar dat de restrictieve implementatie (de uitvoer van de oplossing van de student moet overeenkomen met de uitvoer van de standaardoplossing) een beperking in het gebruik is. Het tool probeert het ingevoerde programma te compileren; indien dat faalt wordt geprobeerd relevante feedback te geven om de fout op te lossen. Als uitvoering wel succesvol is wordt gekeken of de uitkomst gelijk is aan de voorgeschreven uitkomst. Het probleem dat daaraan annex is is tweeledig:

- de uitkomst is wel gelijk, maar het programma dat de uitkomst genereert is afwijkend van de standaardoplossing. De student krijgt echter geen feedback over deze verschillen en de voor- en nadelen van de verschillende oplossingen.
- De uitkomst is verschillend. In dit geval krijgt de student alleen een melding van dit feit en is aan zichzelf overgelaten om te proberen een juiste oplossing te vinden.

Ook VIOPE is niet geschikt om een willekeurige taal te ondersteunen; op dit moment is er alleen ondersteuning voor C, C++, Java, SQL en PHP

Er is dus geen generiek raamwerk waar de Rascal leeromgeving in ondergebracht kan worden; waar mogelijk zal de functionaliteit die hier beschreven is door de Eclipse IDE geboden moeten worden, of er zal een dedicated tool moeten worden ontwikkeld.

4.5.2 Toetsen en testen

Requirement: "toetsen en testen die op ieder willekeurig moment (weer) door een student afgelegd kunnen worden, met een directe feedback over het resultaat".

Bovenstaande eis mag gesteld worden aan elke CAL, waarbij een randvoorwaarde is dat de CAL omgeving centraal beschikbaar is via een web-interface. (Dit zorgt ervoor dat er geen platform afhankelijkheden optreden en dat centraal beheer van de toetsen mogelijk is).

Het is niet verwonderlijk dat hieraan tegemoet gekomen wordt door het pakket Blackboard, dat binnen de Universiteit van Amsterdam gevoerd wordt. Ook andere ELO's (Electronische Leer Omgevingen) zoals Dokeos (in gebruik bij de Hogeschool van Amsterdam) bieden dit aan. Het lijkt dus zonder meer mogelijk te zijn deze faciliteit aan te bieden.

4.5.3 Hulpmiddelen voor "concept mapping"

Requirement: "hulpmiddelen voor "concept mapping", waarmee grafische mindmaps gemaakt kunnen worden die het kennisgebied beschrijven en waarbinnen vastgelegd kan warden hoe concepten met elkaar samenhangen".

Hiervoor zijn meerdere hulpmiddelen beschikbaar. Het is niet relevant voor dit hulpmiddel dat het centraal beschikbaar is; studenten moeten in staat zijn om te kiezen voor een implementatie die geschikt is voor hun lokale platform en dat hun werkwijze en conceptuele model ondersteunt.
Een mogelijke constraint is het al dan niet beschikbaar stellen van "standaard" mindmaps, het uitwisselen van gemaakte mindmaps en het maken van mindmaps als opgaven binnen het didactisch model. Als dit gewenst is zal een keuze gemaakt moeten worden voor standaard software, of minimaal een standaard uitwisselingsformaat (indien dit bestaat).

Hoofdstuk 5: Onderzoek

In dit hoofdstuk zal aan de orde komen welke omgeving(en) en hulpmiddelen gebruikt worden voor het uitvoeren van het empirisch onderzoek, en welke keuzes daaraan ten grondslag liggen.

Verder zullen de opzet en de inhoud van het onderzoek beschreven worden.

5.1 Studentbeleving

Om de studentbeleving van het leren programmeren in kaart te brengen zal een eerste enquête afgenomen worden onder de software engineering studenten van het Instituut voor Informatica. De enquêtevragen zijn opgenomen in Appendix C.
Hierbij wordt aan de hele populatie, van eerstejaars tot afstudeerders, gevraagd wat hun ervaringen zijn (geweest) op het gebied van leren programmeren. De doelstelling van deze enquête is om te bepalen welke aspecten als meest ingewikkeld worden ervaren. Hierdoor kan getoetst worden in hoeverre Rascal aan deze problemen tegemoet komt.
Daarna zal een tweede, meer specifieke, enquete gehouden worden waarin verdere verduidelijking van de eerder gevonden resultaten wordt gezocht.

5.2 Toetsomgeving

Het lijkt voor de hand te liggen om de implementatie van een toetsomgeving binnen Blackboard te realiseren. De achterliggende gedachte hier is dat Rascal gebruikt gaat worden in het (toekomstig) onderwijs binnen de Masteropleiding Software Engineering van de Universiteit van Amsterdam en dat de elektronische leeromgeving van de UvA geïmplementeerd is in Blackboard.

Toch heb ik er voor gekozen om de proof-of-concept in de HvA omgeving Dokeos op te zetten. Daar zijn de volgende argumenten voor :

- ik heb geen toegang tot een geschikte test-omgeving binnen Blackboard, maar kan deze wel binnen Dokeos realiseren met behulp van Moodle (het Dokeos hulpmiddel voor het toetsen);
- het geplande empirisch onderzoek wordt uitgevoerd met behulp van informatica studenten van de HvA; deze hebben geen toegang tot Blackboard;
- er zijn standaard formaten voor het maken van vragen, die door meerdere pakketten ondersteund worden. Als de toetsvragen in deze vorm gemaakt worden is de implementatie van een toetssysteem in Blackboard relatief eenvoudig.

5.3 Hulpmiddelen voor “concept mapping”

Het verdient aanbeveling om op voorhand te kiezen voor een standaard hulpmiddel. Hierdoor is het mogelijk om in de toekomst maps beschikbaar te stellen in de vorm van opgaven en (standaard-)uitwerkingen. Met behulp van de beschikbare Rascal documentatie zal aan de studenten gevraagd worden om een mindmap te maken van de concepten van Rascal. Op basis van de resultaten hiervan kan getoetst worden of de concepten voldoend duidelijk naar voren komen.

Een mogelijke kandidaat voor de mindmapping software is Freemind, dat zowel voor Windows, Apple als Linux beschikbaar is. Daarnaast is het gratis.

5.4 Tutorials

Voor de voorloper van Rascal, ASF+SDF, is in het verleden een “guided tour”gemaakt door Paul Klint. In deze guided tour wordt aan de student via de website van de meta-omgeving een overzicht aangeboden waarin (een deel van) de functionaliteit van ASF+SDF wordt getoond. (Zie appendix A voor de link.) Dit was echter een behoorlijke inspanning.
Er zal in het kader van dit onderzoek gekeken worden of er ook moderne tools zijn, waarmee voor Rascal op eenvoudiger wijze toch in een soortgelijke demo (een “screencast”) kan worden voorzien.
Daarnaast zal de nieuwe gebruikersdocumentatie voor Rascal [Kli09-2] aan een onderzoek onderworpen worden.

5.5 Intelligente programmeeromgeving (IDE)

Er zijn op het moment van schrijven van deze thesis twee mogelijke kandidaten voor de keuze van een IDE. Dit zijn Eclipse en Netbeans, die samen een leidende positie hebben op het gebied van onafhankelijke ontwikkelhulpmiddelen.

De IDE die als meest geschikt naar voren komt is Eclipse. De argumenten voor deze keuze zijn:

- Eclipse is een volwassen, professionele omgeving die geschikt is om het gehele ontwikkeltraject te ondersteunen;
- Eclipse is uitbreidbaar, dat wil zeggen dat individuele ontwikkelaars modules kunnen toevoegen die specifieke (taalgebonden) functionaliteit ondersteunen;
- Eclipse is open source en wordt door een actieve community ondersteund;
- Last but not least, Rascal is geïntegreerd in Eclipse.

5.6 Realistische taken en voorbeelden

Hiervoor (in 5.4) is de nieuwe versie van de gebruikersdocumentatie al genoemd. In dit document wordt het EASY paradigma uitgelegd en toegelicht. Als onderdeel van dit document passeert een aantal toepassingsvoorbeelden van Rascal de revue, met de wijze waarop deze voorbeelden uitgewerkt kunnen worden. Dit document zal (hoewel misschien nog niet volledig en / of compleet) aan de studentpopulatie die aan het onderzoek meewerkt worden voorgelegd, met een enquête om te toetsen hoe de kwaliteit ervaren wordt.

Hoofdstuk 6: Resultaten

6.1 Enquêteresultaten enquête 1: aandachtsgebieden met betrekking tot leerbaarheid

Met behulp van de eerste enquête is getracht om in kaart te brengen waar de problemen liggen op het gebied van het leren van een programmeertaal.
De uitkomsten zijn alleen representatief voor de onderzochte groep, namelijk HBO studenten die een informatica opleiding volgen. Toch is deze groep relevant voor het onderzoek, want dit zijn potentieel studenten die eventueel een masteropleiding gaan volgen en daar met Rascal te maken kunnen krijgen.
Daarnaast is de groep respondenten ervaringsdeskundig op het gebied van leren programmeren; aangenomen mag worden dat de uitkomsten ook van toepassing zijn op anderen die programmeren willen leren.

In de eerste enquête is de studenten een aantal vragen gesteld op het gebied van voorkennis, effectiviteit van leermethoden en hulpmiddelen, en problemen die ze tegenkwamen in het leerproces. Hierbij werd veelal gevraagd naar antwoorden op een gedifferentieerde schaal (zie ook appendix C voor de vragen). Er is van 23 studenten een response ontvangen.
De resultaten van deze enquête zien er als volgt uit.

Gevraagd naar ervaring met en motivatie voor het programmeren kwamen de volgende antwoorden.

- Aan het begin van de opleiding had 62,5% van de studenten al enige programmeerervaring.
- Van deze studenten had 33% minder dan een jaar ervaring, 25% tussen de 1 en 2 jaar, 17% tussen de 2 en 5 jaar en 25% meer dan 5 jaar.
- De genoemde programmeertalen zijn C, C++, Visual Basic, C#, Java. Hierbij was de hoogste score voor Java; dit is niet verwonderlijk, gezien het feit dat Java als standaardprogrammeertaal aan de studenten gedoceerd wordt.
- Programmeren wordt hoofdzakelijk gedaan om zelf te leren programmeren en om software te ontwikkelen (82%); slechts 18% gaf aan (ook) anderen te helpen met het leren programmeren.

De antwoorden op effectiviteit van leermethodes geven het volgende beeld.

- Hoorcolleges zijn voor 82% effectief of een beetje effectief; 18% vond het niet effectief. Niemand beschouwt hoorcolleges als zeer effectief.
- Hetzelfde beeld (82% versus 18%) kwam naar voren voor zelf programmeren tijdens hoorcolleges; verschil is wel dat hier 18% dit als een zeer effectieve leermethode beschouwt.
- Zelfwerkzaamheid scoort hoog op het gebied van leren programmeren: totaal 88% ziet dit als een zeer effectieve (59%) of effectieve (29%) leermethode.

- Ook voorbeeldcode van een docent wordt op prijs gesteld: 94% vindt dit een hulpmiddel bij het leerproces. 36% geeft aan het zeer effectief te vinden, en 41% vindt het effectief.
- Ook wordt het gebruik van tutorials (die individueel en zelfstandig doorgewerkt moeten worden) als een goed hulpmiddel ervaren. 82% vindt dit minimaal effectief, waarvan de helft zelfs zegt het een zeer effectieve leerstrategie te vinden. Dit geldt dan weer niet als dit in groepsverband gebeurt: dan vindt slechts 30% het effectief als hoogste waardering.
- Hulp tijdens een practicum wordt als gemiddeld effectief ervaren. Daarbij wordt hulp van een docent hoger aangeslagen dan hulp van een practicumassistent (dit is een ouderejaars student die naast de docent als mede-practicumbegeleider optreedt).
- Zoals al eerder is geconstateerd wordt alleen studeren buiten de collegesetting als (zeer) effectief ervaren. Gevraagd naar het rendement van alleen studeren (thuis of in een mediatheek) zegt 65% dat effectief (35%) tot zeer effectief (30%) te vinden. Niemand zegt dat het niet effectief is. Dezelfde uitkomst geldt voor samen met een ander studeren in deze omgeving.

Ook hulpmiddelen kunnen een rol spelen in het leerproces.

- Een ontwikkelomgeving als Eclipse of Netbeans wordt door 82% als een (goed) hulpmiddel ervaren. Maar liefst 86% vindt dit een (zeer) effectief hulpmiddel.
- Technische boeken, zoals het voorgeschreven lesboek, scoren lager als hulpmiddel. Hierbij wordt zelf gekozen tekst als positiever ervaren dan voorgeschreven boeken.
- Internetondersteuning in de vorm van websites voor het vak/onderwerp worden op prijs gesteld. Net als bij het voorgaande punt worden zelf gekozen sites (vooral forums en tutorials, indien beschikbaar) als positiever ervaren.
- Voorbeeldprogramma's en –uitwerkingen worden door 77% als (zeer) effectief ervaren om het leerproces te ondersteunen; college-ondersteunend materiaal is dat echter niet. Maar 29% vindt dit een (zeer) effectief hulpmiddel.

Als laatste is geïnventariseerd wat als moeilijk ervaren wordt bij het leren programmeren. Hierbij ligt de nadruk op het vinden van compilatiefouten en logische fouten van een programma. Er lijken geen programmeerconcepten als echt moeilijk gezien te worden; de hoogste score hier is voor pointers in talen die daar gebruik van maken.

Een opmerking die gemaakt werd bevestigt het beeld dat zelfstandig en met meerdere bronnen leren goed mogelijk is: "*Twee jaar geleden heb ik Ruby, buiten school om, geleerd.* ***Hier waren vooral online interactieve tutorials en screencasts erg nuttig.*** *Daarnaast hielp het om verschillende boeken door te lezen.*" (Noot: nadruk door vet gedrukte tekst is van mij.)

6.2 De gebruikersdocumentatie van Rascal

Naast de ontwerpdocumentatie [Kli09-1] is de nieuwe gebruikersdocumentatie voor Rascal ontstaan.
Deze documentatie is voorgelegd aan de studenten die meegedaan hebben aan het onderzoek, en aan de hand van een vragenlijst (zie appendix D) is hen gevraagd om een oordeel te geven over deze documentatie.

Hierbij is expliciet gevraagd naar de begrijpelijkheid en de relevantie van de voorbeelden: uit onderzoek is gebleken dat er potentiële valkuilen zijn bij het ontwerpen van voorbeelden [Mal04].
Deze valkuilen zijn:

- het voorbeeld is te abstract;
- het voorbeeld is te complex;
- het voorbeeld past concepten niet consistent toe;
- het concept dat uitgelegd wordt is dusdanig vereenvoudigd dat het doel niet meer bereikt wordt.

De uitwerking van de antwoorden is terug te vinden in paragraaf 6.5.

6.3 Rascal en leerbaarheid

In 4.4 is vanuit bestaand onderzoek gekeken welke kwaliteitseisen van invloed zijn op de leerbaarheid van een programmeertaal. De winnaars waren Eiffel, Python (beide talen die ontworpen zijn met onderwijs als uitgangspunt) en Java.

Hoewel in het geciteerde onderzoek uitgegaan werd van het leren van een eerste programmeertaal, en de gebruikers van Rascal al een of meerdere talen beheersen, is het toch zinvol om een toetsing aan deze kwaliteitseisen uit te voeren.
In de praktijk blijkt namelijk dat:

- bestaande ervaring verschilt in de (soort) programmeertaal: veel van de gebruikers van Rascal beginnen hun programmeerloopbaan met imperatieve talen, en hebben geen ervaring met het functionele paradigma;
- het terrein waarop programmeerervaring is opgedaan loopt van embedded software, geschreven in C of proprietary talen tot webapplicaties in de .NET omgeving;
- het typische toepassingsgebied van Rascal (onderzoek naar software evolutie) is vaak een nieuw kennisgebied voor de gebruikers.

Het spreekt dan ook vanzelf dat er meer aandacht aan het feitelijke probleem kan worden geschonken naarmate het hulpmiddel voor de oplossing van dat probleem eenvoudiger onder de knie kan worden gekregen.

In deze sectie wordt op basis van de criteria van Manilla en de Raadt getoetst wat de huidige score is van Rascal, waar er (dus) nog verbeterpunten zijn, en wat de huidige score is in vergelijking met de beste keuzes uit het onderzoek.

De indeling zoals die door Manilla en de Raadt wordt aangegeven wordt hier gevolgd; er is een onderverdeling naar:

- geschikt voor het leren programmeren;
- ontwerp en omgeving;
- ondersteuning en beschikbaarheid;
- geschiktheid voor andere types c.q. complexere problemen en verdieping.

Voor de confrontatie van Rascal met het model is gebruik gemaakt van het “Rascal Requirements and Design Document” [Kli09-1]. Tenzij anders aangegeven zijn de Rascal feiten ontleend aan dit document. Requirements van Rascal die uit dit document komen worden aangeduid met “Rn”, waarbij de n het nummer van de de gedefinieerde requirement is.

6.3.1 Geschiktheid van Rascal om te leren programmeren

- Geschiktheid om te doceren. Hierbij gaat het er om of een taal ontworpen is om te leren programmeren. De syntax is simpel, de semantiek is eenduidig en hulpmiddelen zijn eenvoudig in het gebruik.

  Het Rascal ontwerp document [Kli09-1] zegt niets over leerbaarheid als requirement. Wel is nadrukkelijk rekening gehouden met eenduidige semantiek (vermijden van ambiguïteit) en eenvoud van syntax, en er is gekozen voor hulpmiddelen die zich in de praktijk hebben bewezen (zoals de Eclipde ontwikkelomgeving).

  Het ontbreken van expliciete leerbaarheidseisen in het ontwerp wordt meer dan goedgemaakt in het usermanual [Kli09-2], waarbij nadrukkelijk rekening is gehouden met het gebruik van Rascal in het onderwijs van de Masteropleiding Software Engineering van de UvA.

  Voldaan aan eis ja/nee: **ja**

- Een taal kan met zogenaamde “micro-werelden” overweg, een kleine, eenvoudige en begrensde omgeving. Dit bevordert verkenningen in beperkte domeinen.
  Rascal is een programmeertaal die ontwikkeld is om gebruikt te worden in specifieke domeinen (die van software- evolutie), een DSL (Domain Specific Language). Het is mogelijk om een domein dat met behulp van Rascal bestudeerd wordt terug te brengen tot een beperkte subset van begrippen, waardoor het domein gezien kan worden als een micro-wereld die met Rascal verkend wordt.

  Voldaan aan eis ja/nee: **ja**

- Een taal verschaft een algemeen begrippenkader. De taal faciliteert het leren van algemene programmeerprincipes, die vervolgens gebruikt kunnen worden om een volgende taal te leren.
  Hierbij kan gedacht worden aan de volgende onderdelen:
  - constructies voor besturing, waarmee de uitvoering van het programma geregeld wordt;
  - constructie voor definitie en manipulatie van data-elementen, waarmee waarden uit het toepassingsdomein gemanipuleerd kunnen worden, en specifieke (complexe) datatypes die een afbeelding zijn uit dat domein;
  - modulariteit, waarmee procedures en functies gemaakt kunnen worden waarbinnen specifieke taken uitgevoerd kunnen worden.

Aan de eerste twee aspecten is voldaan: volgens het ontwerpdocument is binnen Rascal voorzien in “simple structured statements for control flow” en “variable assignments for data flow” die uit de imperatieve programmeerwereld afkomstig zijn. (Dit komt echter niet terug als een expliciete requirement…). Ook datatypering is (beperkt) mogelijk met de in Rascal beschikbare types.
De modulariteit wordt geïmplementeerd door middel van functies, modules en bibliotheken.

De hier verworven kennis kan worden geprojecteerd op andere programmeertalen, evenals de modularisering (zie ook onder 6.1.2).

Voldaan aan eis ja/nee: **ja**

- Een taal is niet alleen implementatiegericht, maar verschaft een kader om probleemoplossend denken te ontwikkelen en omvat meerdere aspecten van het softwareontwikkeling proces. Hierbij wordt de betreffende taal ontwikkeld als een samenhangend geheel van taal, principes, hulpmiddelen en bibliotheken.

  In het usermanual van Rascal [Kli09-2] wordt duidelijk gemaakt dat er een samenhang is tussen “taal, principes, hulpmiddelen en bibliotheken”.
  Het proces van ontwikkelen van software met behulp van Rascal is hierin op basis van verschillende scenario’s beschreven, uitgaande van een algemeen paradigma (Extractie/Analyse/Synthese, EASY) .

  Voldaan aan eis ja/nee: **ja**

6.3.2 Ontwerp van Rascal en omgeving

- Een taal is interactief en ondersteunt “*rapid code development*”. Nieuw verkregen inzichten moeten snel toegepast kunnen worden zonder volledige applicatiecontext; terugkoppeling op voortgang is interactief en ogenblikkelijk.

  Uit de specificatie blijkt dat Rascal zowel een scripting modus als een compilatiemodus kent. Inherent aan scriptingtalen is het snel kunnen toepassen van de taal in een “informele” modus (dus zonder het expliciet opzetten van een applicatiecontext), waarbij de scripting engine voor onmiddellijke terugkoppeling zorgt, zowel in negatieve (fouten) als positieve (resultaten) zin.
  Daarnaast is er de compiler, die programmeerfouten terugkoppelt. (Hierbij moet wel de kanttekening gemaakt worden dat de effectiviteit van deze hulpmiddelen staat of valt met de begrijpelijkheid van de meldingen; de voorouders van Rascal, ASF+SDF en RScript, blinken niet direct uit in dit opzicht…). Daarnaast geeft de runtime component de resultaten weer.
  Als derde component is er de (potentiële) ondersteuning vanuit Eclipse, de ontwikkelomgeving van/voor Rascal; Eclipse maakt het mogelijk real-time terugkoppeling te geven op fouten en problemen.

  Expliciet requirement hiervoor is R1

Voldaan aan eis ja/nee: **ja**

- Een taal moedigt het schrijven van correcte programma's aan. Hierbij kan gedacht worden aan technieken als "*Design by contract*", "*Test driven design*", pre- en postcondities en invarianten, maar ook aan in de taal ingebouwde eisen die hierbij helpen.

  Rascal heeft geen ondersteuning voor genoemde technieken en concepten, omdat deze:
  a. taalonafhankelijk zijn (onderdeel van een ontwikkelmethodiek) en
  b. in het ontwerp van het programma moeten worden gerealiseerd.

  Hierdoor kan aan deze eis alleen op indirecte wijze worden voldaan.

  Door toetsing van de requirements kan een beeld gevormd worden in hoeverre het eenvoudig is om met Rascal correcte programma's te schrijven.
  De volgende requirements spelen hierbij een rol.
  - Requirement R4 stelt dat er een concrete syntax is; hierdoor wordt het schrijven van eenduidige programma's bevorderd.
  - Volgens requirement R10 wordt ernaar gestreefd om features orthogonaal te houden; hierdoor is het aantal manieren om een specifiek programma te schrijven minimaal.
  - R11 vereist dat er een minimum aan ambiguïteit in de syntax moet zijn.
  - R13 verbiedt het gebruik van "null" waarden, waardoor veel voorkomende programmeerfouten uitgesloten worden.
  - R14 vereist het onveranderbaar zijn van alle waarden.
  - R17 definieert een type safe omgeving, waarbinnen veel voorkomende programmeerfouten uitgesloten worden, maar die toch hergebruik van (bewezen) code toestaat.
  - R18 definieert een syntax safe omgeving waardoor programmeurs onmogelijk werkende syntactisch incorrecte programma's kunnen ontwikkelen.
  - R20 vereist dat Rascal traceable en/of debuggable moet zijn, waardoor de uitvoering en status van een Rascal programma gevolgd kan worden en waarbij de status en waarden weergegeven kunnen worden.
  - R21 voorkomt potentiële problemen door voor te schrijven dat functies, datatypes en variabelen verplicht van een type moeten worden voorzien.
  - Als laatste faciliteert R22 hergebruik van modules, waardoor bewezen correcte code opnieuw ingezet kan worden.

  Voldaan aan eis ja/nee: **ja**

- Problemen kunnen in deelstappen opgelost worden. Een taal beschikt hiertoe over modularisatie in de vorm van bijvoorbeeld klassen, functies en procedures.

  Modules worden in Rascal ondersteund. Er is sprake van opdelen van Rascal code in modules (met een .ras suffix), uitbreiden van bestaande Rascal modules (door het

opnemen van de betreffende module in het huidige programma wordt overerving gesimuleerd), import van bestaande modules uit SDF en Java, en het maken van functies die met formele parameters gedefinieerd worden, waarbij op runtime de actuele parameters gesubstitueerd worden.

Expliciete requirements hiervoor zijn R2, R3, R22

Voldaan aan eis ja/nee: **ja**

- Een taal beschikt over een geïntegreerde ontwikkelomgeving. Deze omgeving is zowel geschikt voor het oplossen van de problemen van beginners als voor experts. Rascal is geïmplementeerd binnen Eclipse, die aan beide voorwaarden voldoet. (Een kanttekening die gemaakt moet worden is dat de ondersteuning van Rascal binnen Eclipse nog in een beginfase is.)

  Voldaan aan eis ja/nee: **ja**

6.3.3 Ondersteuning en beschikbaarheid van Rascal

- Er is een actieve gebruikersgroep, die mede voor ondersteuning zorgt. (De levensvatbaarheid en beschikbare middelen voor onderhoud en uitbreiding zijn immers direct afhankelijk van de acceptatie van de taal door programmeurs in het algemeen; hoe groter de groep is die belang heeft bij een taal, hoe beter de toekomst van deze taal verzekerd is!)
  Bij ondersteuning kan gedacht worden aan allerlei vormen, zoals websites, boeken, fora, tutorials, documentatie en cases/oefeningen.

  Niet aanwezig, Rascal is ten tijde van het schrijven van deze thesis nog niet in produktie.

  Voldaan aan eis ja/nee: **nee**

- Een taal is "*open source*", waardoor iedere belanghebbende kan bijdragen aan de ontwikkeling. Het uitgangspunt van de ontwikkelaars van de taal is niet om een commercieel product te maken.

  Hoewel het ontwerp hier niets over zegt, is het nadrukkelijk wel de bedoeling om Rascal als open source taal te realiseren (bron: Paul Klint), en er zijn geen commerciële overwegingen.

  Voldaan aan eis ja/nee: **ja**

- Een taal wordt op meerdere platforms ondersteund en werkt daar op identieke wijze.

Hoewel het ontwerp hier niets over zegt, is door de architectuur (realisatie binnen Eclipse en gebruik van een multi-platform omgeving) multi-platform ondersteuning inherent aan Rascal.

Voldaan aan eis ja/nee: **ja**

- Een taal is gratis, eenvoudig en zonder beperkingen beschikbaar.

  Evenals zijn voorlopers, ASF+SDF en RScript zal Rascal via de websites van het CWI zonder voorwaarden aan alle belangstellenden beschikbaar gesteld worden.

  Voldaan aan eis ja/nee: **ja**

- Een taal wordt ondersteund door kwalitatief goed onderwijsmateriaal.

  Het usermanual [Kli09-2]is al enkele malen genoemd. Dit is zonder meer geschikt als onderwijsmateriaal (hoewel er in door de studenten die dit manual hebben bestudeerd en een er vragenlijst over hebben ingevuld nog wel enkele suggesties voor verbetering zijn gedaan; meer hierover later in deze scriptie).

  Voldaan aan eis ja/nee: **ja**

6.3.4 Geschiktheid van Rascal voor andere types c.q. complexere problemen en verdieping

- Een taal wordt niet alleen voor onderwijsdoeleinden gebruikt. Het is mogelijk om "*echte*" oplossingen te realiseren en de taal wordt in de "*real world*" gebruikt.

  De voorgangers van Rascal, ASF+SDF en RScript, worden gebruikt om echte problemen op te lossen in de software industrie. Er is geen reden om aan te nemen dat Rascal hiervoor ongeschikt zal zijn. Requirements R3, R4, R12 en R20 hebben betrekking op deze eis.

  Voldaan aan eis ja/nee: **ja**

- Een taal is uitbreidbaar. Het is mogelijk om met een kleine subset van de taal te beginnen, maar later kan de toepasbaarheid uitgebreid worden naar een groter scala van problemen.

  Op dit moment is Rascal een DSL die toegespitst is op onderzoek naar software evolutie. Hoewel binnen dit toepassingsgebied geen beperkingen zijn blijft het een domeinspecifieke taal, en kan het geen problemen buiten het domein (bijvoorbeeld ontwikkeling van webapplicaties) aan. Vanuit deze optiek lijkt Rascal niet uitbreidbaar te zijn.

  Het is echter mogelijk om Rascal in de toekomst uit te breiden en wel op twee manieren:

1. door functionaliteit aan Rascal zelf toe te voegen, dat wil zeggen door Rascal uit te breiden met code ;
2. door (gebruikers-)bibliotheken toe te voegen die specifieke doelen realiseren. Te denken valt aan de reeds voor ASF+SDF beschikbare uitbreidingen waardoor specifieke systemen in programmeertalen zoals COBOL en Java geanalyseerd en getransformeerd kunne worden.

Voldaan aan eis ja/nee: **ja**

- Een taal is betrouwbaar en efficiënt. De producten die in een taal gemaakt worden moeten reële compilatie- en verwerkingstijden hebben .

  Requirement R1 zegt dat versnelling van compilatie- en verwerkingstijden het verschil kan maken tussen het wel of niet haalbaar zijn van een bepaalde analyse. Dit zegt echter niets concreets over de performance van Rascal, maar R3 spreekt over " a few minutes max".

  In de ontwikkeling van Rascal is de code geoptimaliseerd. Er is gezocht naar de meest efficiënte datastructuren en algoritmes, waardoor zowel de beheeromgeving van Rascal (de IDE) als de objectcode die door Rascal opgeleverd worden de best haalbare performance hebben. Hierdoor is zowel het werken aan de ontwikkeling van Rascal programma's als het uitvoeren van analyses en syntheses met de gemaakte programma's zo snel als mogelijk is. Rascal compileert naar Java klassen en heeft de performance van Java.

  Voldaan aan eis ja/nee: **ja**

- Een taal moet toekomstvast zijn; het gebruik in het heden en de toekomst mag niet afhankelijk zijn van een eventuele toepassing in het verleden.

  Omdat Rascal (hoewel gebaseerd op ASF+SDF en RScript) nieuw ontwikkeld is, kan er geen sprake zijn van eventuele toepassing in het verleden. Er is weliswaar sprake van compatibiliteit met de voorlopers, maar dat kan niet opgevat worden als een afhankelijkheid. Omdat nieuwe toepassingen en domeinen op zichzelf staan is geen sprake van afhankelijkheid (behalve waar de Rascal gebruiker deze zelf veroorzaakt) en kan Rascal mee evolueren met de toepassingen van de gebruikers.

  Voldaan aan eis ja/nee: **ja**

6.3.5 Score van Rascal op het gebied van leerbaarheid

Als we de balans opmaken van bovenstaande inventarisatie blijkt dat Rascal goed scoort op het gebied van leerbaarheid: met een totaal van 12 punten is het beter dan C++, en maar iets slechter dan de topscorers.

Als Rascal samen met de 3 topscorers wordt weergegeven in tabelvorm ziet het beeld er als volgt uit.

Kwaliteitskenmerk	Rascal	Eiffel	Python	Java
Geschiktheid om te leren programmeren				
Geschiktheid om te doceren	√	√	√	-
Microwereld geschikt	√	√	√	√
Algemeen begrippenkader	√	√	√	√
Niet alleen implementatiegericht	√	√	-	*1
Ontwerp en omgeving				
Interactief en rapid code development	√	-	√	√
Correct programmeren	√	√	*2	*2
Decomponeren problemen	√	√	√	√
Geintegreerde ontwikkelomgeving	√	√	-	-
Ondersteuning en beschikbaarheid				
Actieve gebruikersgroep	-	√	√	√
Open source	√	-	√	-
Multi platform	√	√	√	√
Gratis beschikbaar	√	√	√	√
Kwalitatief goed onderwijsmateriaal	√	√	√	√
Complexere problemen en verdieping				
Niet alleen onderwijs	√	√	√	√
Uitbreidbaar	√		√	√
Betrouwbaar en efficiënt	√	√	√	√
Toekomstvast	√	√	√	√
Totaal	16	15	15	14

* 1 Misschien met aangepaste IDE zoals BlueJ
* 2 Misschien met unit testing uitbreiding

Op dit moment haalt Rascal al een hogere score (16) dan de eerder koplopers (met maximaal 15) als het gaat om kwaliteitseisen.

Het enige aspect waarop Rascal (nog) niet scoort is een actieve gebruikersgroep, maar als in ogenschouw genomen wordt dat Rascal in de nabije toekomst in gebruik genomen gaat mag verwacht worden dat deze gaat ontstaan.

6.4 Concept mapping

Eerder is gesproken over “concept mapping”: het afbeelden van concepten binnen een domein op een mentaal beeld. Hierbij worden abstracte begrippen binnen de te leren stof in onderlinge samenhang gebracht. De persoon die dit proces uitvoert wordt hierdoor aangezet tot het identificeren van sleutelbegrippen en de onderlinge samenhang. Indien dit in een vrije vorm kan worden weergegeven, met behulp van een computerondersteunde techniek, leidt dit tot toepassing van de fasen begrip, analyse en synthese uit de taxonomie van Bloom.

Een mogelijke manier om dit toe te passen bij het leren van Rascal is om de student een deel van het design document van Rascal (het hoofdstuk “Rascal at a glance”) te geven en hem een mindmap te laten maken van de begrippen die daarin ter sprake komen en de samenhang tussen deze begrippen. De uitgewerkte mindmap kan als discussie- en/of beoordelingsmateriaal dienen.

Het voordeel van deze werkwijze is dat de student geactiveerd wordt om zelf een eigen beeld van de onderliggende concepten te vormen, en dat onduidelijkheden, begripsverwarringen en andere problemen die op het abstracte vlak spelen snel en individueel onderkend kunnen worden. Dit resultaat wordt niet of minder bereikt als de begrippen alleen passief (ter kennisname) worden aangeboden.
Het betekent echter ook dat de te beschrijven concepten op de één of ander wijze voor de student ter beschikken moeten zijn; meestal is er wel een handleiding waarin de beschrijving vastgelegd is.

6.5 Enquêteresultaten enquête 2: Rascal usermanual

Het concept van het Rascal usermanual is aangeboden aan de groep studenten die mee hebben gedaan aan het onderzoek. Door 17 studenten is een ingevulde vragenlijst ingeleverd, waarmee deze enquête niet statistisch significant is maar wel een indicatie kan geven over hoe een aantal aspecten ervaren wordt.

Naast het cijfermatig materiaal is er ook de gelegenheid gegeven om opmerkingen als vrije tekst te geven; deze opmerkingen zijn door mij per categorie samengevat.

De antwoorden kunnen worden gerangschikt onder een aantal categorieën.

Easy concepten (vraag 1 – 8).
Hierbij is voornamelijk gevraagd om “rapportcijfers” over de voorbeelden, en is gevraagd of het EASY concept, doel en doelgroep van het manual duidelijk zijn.

	Gemiddeld cijfer	Max	Min
Algemene beschrijving	7	9	6
Voorbeeld security breaches	7	9	4
Voorbeeld DSL Compiler	7	9	4
Voorbeeld renovation	8	9	7
Voorbeeld concurrency	7	9	6

	Ja	Nee
Voordelen Rascal in EASY duidelijk?	75%	25%
Doel en doelgroep manual duidelijk?	92%	8%

<u>Opmerkingen.</u>
Het wordt op prijs gesteld dat er begonnen wordt met een probleemstelling en daarna pas de taal zelf wordt uitgelegd, en de uitleg van het EASY concept wordt als helder ervaren. Er wordt echter niet uitgelegd wat het voordeel is van een nieuwe taal ten opzichte van een

traditionele benadering (waarbij niet aangegeven wordt wat een traditionele benadering dan wel is; ik neem aan dat er gerefereerd wordt aan bestaande programmeertalen en hulpmiddelen).
De voorbeelden / scenario's / use cases zijn verhelderend en helpen context te geven (vooral voor beginners) ; omdat er voorbeelden zijn vanuit verschillende domeinen is er altijd wel een bij die een lezer aanspreekt).

Rascal concepten (vraag 10 – 35).

	Gemiddeld cijfer	Max	Min
Voorbeeld datastructuren	5	6	2
Voorbeeld pattern matching	6	7	3
Voorbeeld enumerators	6	8	4
Voorbeeld comprehensions	7	8	6
Voorbeeld control structures	6	8	3
Voorbeeld functions	7	10	4
Voorbeeld rewrite rules	6	8	3

	Ja	Nee
Values concept en gebruik duidelijk	75%	25%
Datastructuren concept en gebruik duidelijk	33%	67%
Pattern matching concept en gebruik duidelijk	56%	44%
Parsing concept en gebruik duidelijk	78%	22%
Enumerators concept en gebruik duidelijk	89%	11%
Comprehensions concept en gebruik duidelijk	100%	0%
Control structures concept en gebruik duidelijk	78%	22%
Switching concept en gebruik duidelijk	78%	22%
Functions concept en gebruik duidelijk	89%	11%
Rewrite concept en gebruik duidelijk	75%	25%
Constraint solving concept en gebruik duidelijk	33%	67%
Type checking concept en gebruik duidelijk	89%	11%
Execution / flow of control concept en gebruik duidelijk	33%	67%

<u>Opmerkingen.</u>

De voorbeelden worden over het algemeen als te ingewikkeld, abstract of nietszeggend ervaren; er is behoefte aan concretere voorbeelden, met daarna eventueel daarna wat abstractere. Misschien een samenhangend voorbeeld waarin de diverse aspecten aan de orde kunne komen.
Opvallend is dat iedereen aangeeft te begrijpen hoe het zit met comprehensions; dat is een belangrijke constatering, aangezien comprehensions een belangrijke rol spelen in Rascal.

Klassieke voorbeelden (vraag 37 – 46).

	Gemiddeld cijfer	Max	Min
Waardering Problem solving strategies	7	9	6

	Ja	Nee
Begrijpt Hello voorbeeld	100%	0%
Hello voorbeeld heeft toegevoegde waarde	78%	22%
Begrijpt Factorial voorbeeld	89%	11%
Factorial voorbeeld heeft toegevoegde waarde	88%	22%
Begrijpt Colored Trees voorbeeld	78%	22%
Colored Trees voorbeeld heeft toegevoegde waarde	78%	22%
Begrijpt Word Replacement voorbeeld	89%	11%
Word Replacement voorbeeld heeft toegevoegde waarde	78%	22%
Problem Solving voorbeeld heeft toegevoegde waarde	75%	25%

Opmerkingen:
Hier worden de voorbeelden beter beoordeeld. Een student zegt: “De voorbeelden, zoals de coloured trees, zouden terug kunnen komen of naar verwezen worden in het onderdeel Datastructures. Op dat moment had ik meer aan het voorbeeld.”. Een ander constateert hetzelfde: “Ik vind deze voorbeelden vrij goed. Wat misschien handiger was geweest als ze juist bij het vorige hoofdstuk hadden gestaan in plaats van hier bij elkaar.”
Problem Solving wordt ervaren als duidelijk, rustig en uitgebreid wordt er uitgelegd hoe het werkt en hoe de stappen doorlopen worden.
Het hoofdstuk is duidelijk, goed te volgen, logisch en enigszins bekend. Het maakt een stuk duidelijker waarvoor Rascal bedoeld is en is daarmee vergelijkbaar met de voorbeeld situaties die aan het begin werden gegeven.

Mystery box voorbeeld (vraag 51 – 64).

	Gemiddeld cijfer	Max	Min
Waardering voorbeeld Call Graph Analysis	7	8	4
Waardering voorbeeld Component Structure	6	8	3
Waardering voorbeeld Analyzing Structure	6	8	4
Waardering voorbeeld Variabelen	7	9	3
Waardering voorbeeld McCabe	7	8	5
Waardering voorbeeld Data Flow Analysis	6	8	3

	Ja	Nee
Begrijpt totale Mystery Box voorbeeld	56%	44%
Begrijpt Component Structure voorbeeld	33%	67%
Begrijpt Analyzing Structure voorbeeld	33%	67%
Begrijpt Ongeinitialiseerde Variabelen voorbeeld	56%	44%
Begrijpt McCabe voorbeeld	78%	22%
Begrijpt Dataflow Analysis voorbeeld	44%	55%

Opmerkingen:

Het mystery box voorbeeld is vooral voor de beginnende studenten moeilijk te volgen (zie ook de scores / waardering). De lage minimum scores komen dan ook van hen; de ouderejaars studenten zijn positiever en geven dan ook beduidend hogere cijfers.

Uitwerkingen van vragen (vraag 11, 19, 22, 29).

Om te toetsen hoe begrijpelijk de kleine syntax voorbeelden in het usermanual zijn is de studenten gevraagd om een paar kleine opgaven te maken (zie ook appendix D). Hiervoor hadden ze alleen de beschikking over de tekst; de Rascal implementatie was nog niet geschikt voor testgebruik.
De intentie was vragen zodanig te stellen dat met een kleine aanpassing de voorbeelden uit de documentatie gebruikt konden worden als antwoord, en dat de vragen zonder Rascal implementatie beantwoord konden worden.

Opvallend is dat gemiddeld maar 50% van de deelnemers geprobeerd hebben om een antwoord te geven op deze opgaven. Het is niet duidelijk waarom dit zo is.
Daarnaast zijn er weinig correcte antwoorden; de vraag om een eenvoudige enumerator te definieren leverde de meeste juiste uitwerkingen op.

Algemeen Rascal (vraag 66 – 72).

	Ja	Nee
Beter begrip als eerst syntax besproken	100%	0%
Begrijpt voordelen Rascal	88%	22%
Kan Rascal leren op basis van alleen document	45%	55%
Heeft hoorcolleges nodig	67%	33%
Heeft werkcolleges nodig	56%	44%
Heeft practica nodig	89%	11%

<u>Opmerkingen (andere toepassingen van Rascal):</u>
Het meten van metrics (zoals het voorbeeld van McCabe's Cyclomatic Complexity) wordt onderkend als iets dat heel goed blijkt te kunnen met Rascal.
Ook IDEs en dynamische spellcheckers worden gezien als toepassingsgebied.

<u>Algemene opmerkingen met betrekking tot het manual.</u>
Weer komt terug dat de voorbeelden zonder verdere toelichting niet altijd even eenvoudig te begrijpen zijn. De studenten geven aan wel hoorcolleges en practica te willen, hetgeen ook niet verwonderlijk is omdat het manual niet geschreven is om zonder ondersteuning gebruikt te worden.

Een kanttekening die gemaakt moet worden is dat het naar verwachting eenvoudiger wordt als ook Rascal zelf beschikbaar is om de voorbeelden "na te spelen", waardoor de kritiek op de voorbeelden waarschijnlijk kleiner wordt.

Hoofdstuk 7: Evaluatie en Conclusies

7.1 Inleiding

De doelstelling van dit onderzoek is geweest om te bepalen wat de kwaliteitseisen en hulpmiddelen zijn die een programmeertaal “leerbaar” maken, of de leerbaarheid verhogen.

Op basis van bestaande literatuur over leren is literatuuronderzoek gedaan naar mogelijke leerstrategieën en hulpmiddelen voor het leren van een programmeertaal. In dit laatste hoofdstuk worden de uitkomsten van de literatuurstudie geconfronteerd met de resultaten van het beperkte empirische onderzoek.

Verder is de beschikbare Rascal gebruikersdocumentatie getoetst bij een groep HBO-informatica studenten, en is geprobeerd een oordeel te vormen over de geschiktheid van dit materiaal voor deze groep op basis van een vragenlijst over deze documentatie.

Als laatste wordt gepoogd aan te geven wat in een eventueel toekomstig onderzoek aan de orde zou kunnen komen om Rascal verder uit te bouwen om aan de kwaliteitseisen voor leerbaarheid te voldoen.

7.2 Beantwoording onderzoeksvraag

Het uitgangspunt van mijn onderzoek is geweest om in kaart te brengen welke kwaliteitseisen en hulpmiddelen de leerbaarheid van een programmeertaal beïnvloeden. Om deze vraag te beantwoorden zijn een aantal deelaspecten onderkend (in 2.4 en 2.5); de bevindingen die op deze deelaspecten betrekking hebben komen hier aan de orde.

De geraadpleegde literatuur geeft aan dat algemeen het exogeen constructivisme en verwante stromingen als de beste manier voor het geven van programmeeronderwijs beschouwd worden.

Dit betekent dat de beschikbaarheid van materiaal dat zich leent om (deels) zelfstandig door te werken de leerbaarheid van een programmeertaal ten goede komt. De docent zou hoofdzakelijk op moeten treden als een gids, die de student helpt om een startpunt te vinden, initieel benodigde kennis bijbrengt, en gedurende het leertraject een rol heeft als coach en vraagbaak. Dit onderbouwt de keuze voor het exogeen constructivisme als een geschiktere onderwijsvorm voor het leren programmeren dan het objectivisme en andere varianten van het constructivisme, die immers minder verschillende onderwijsvormen aanbieden.

Bovenstaande conclusies lijken onafhankelijk te zijn van de specifieke programmeertaal. De kwaliteitseisen lijken immers niet alleen op de specifieke syntaxis en semantiek van de taal te zijn gebaseerd, maar ook op de concepten die onderliggend aan de constructie van de taal zijn. De leerweg kan zodanig ontworpen en ontwikkeld worden dat de gevonden leerstrategieën en –hulpmiddelen daarin geïmplementeerd worden.

Het exogeen constructivisme lijkt een geschikte methode te zijn om programmeertalen in het algemeen te onderwijzen, onafhankelijk van aspecten als bijvoorbeeld objectoriëntatie. De onderzoeksresultaten en hulpmiddelen die in kaart gebracht zijn getoetst lijken dan ook universeel inzetbaar te zijn, hetgeen hopelijk leidt tot ontwikkeling van generieke hulpmiddelen.

Indien andere programmeertalen (vergeleken met Rascal) in dezelfde mate aan de beschreven kwaliteitseisen voldoen, of betere ondersteuning hebben in de vorm van (CAI-) hulpmiddelen, mag er van uitgegaan worden dat de gevonden resultaten ook toepasbaar zijn op deze andere programmeertalen.

Verder lijkt het zo te zijn dat leerbaarheid een functie is van kwaliteitseisen (requirements) en hulpmiddelen. Het resultaat kan dus door beide componenten worden beïnvloed: hoe beter de leerbaarheidseisen vertaald zijn in de taal hoe minder groot de ondersteuning kan zijn van de hulpmiddelen, en omgekeerd.

De hoge score die Rascal nu al behaalt op kwaliteit kan alleen nog via toevoeging van betere (CAI-)hulpmiddelen significant worden verhoogd, en daarbij zou de focus moeten liggen op een “simulator”. Gezien de inspanning (en dus kosten) die de ontwikkeling hiervan met zich meebrengt is dit echter geen rendabele propositie; het verdient mijns inziens meer aanbeveling de IDE omgeving te voorzien van “intelligentie”, waardoor bij het ontwikkelen (door gebruikers van Rascal) feedback gegeven kan worden en de ontwikkelaar hierdoor sneller met de taal leert werken.

7.3 Deelresultaten

Algemeen.
De toepassing van het exogeen constructivisme sluit aan bij de belevingswereld van studenten, gegeven de uitkomsten van de enquêtes van zowel Lahtinen c.s. als mijzelf, en leidt tot betere resultaten dan andere didactische benaderingen.

Hoorcolleges en de daarbij behorende didactische materialen en technieken zijn in de beleving van studenten minder belangrijk dan websites, forums, voorbeeldcode, zelfstandig studeren en programmeren en een state-of-the-art ontwikkelomgeving met ondersteuning van de programmeertaal in kwestie.

De kwaliteitseisen en hulpmiddelen moeten al aan het begin van de ontwerpfase meegenomen worden in het ontwerp, en (deels) parallel aan de ontwikkeling van de programmeertaal zelf plaatsvinden. Het team dat zich bezighoudt met de ontwikkeling van een programmeertaall moet naast technische expertise ook over didactische en pedagogische kennis beschikken.

Opvallend is dat als de deelnemers aan het onderzoek inderdaad in staat gesteld worden om zelfstandig het usermanual door te werken de resultaten (vooral bij de kleine opgaven) tegenvallen. Dit lijkt in tegenspraak te zijn met de bewering dat zelfstandig studeren in de eigen omgeving als meest effectief ervaren wordt.

Het exogeen constructivisme als didactisch model vereist dat er zowel een “gestuurd” deel als een zelfstandig deel aan het leertraject zit. Beide onderdelen hebben hulpmiddelen nodig om hun doelstellingen te realiseren.
De hulpmiddelen die bruikbaar (kunnen) zijn in het Rascal onderwijs passeren hier nogmaals de revue, met waar van toepassing enige kanttekeningen.

Screencasts.
De in hoofdstuk 5 genoemde screencast van ASF+SDF is gemaakt met het hulpmiddel *Wink*. Een (beperkt) onderzoek naar beschikbare hulpmiddelen heeft uitgewezen dat dit het meest geëigende hulpmiddel lijkt te zijn als er ook van Rascal screencasts gemaakt moeten worden. Hiervoor zijn de volgende redenen aan te voeren:

- Wink is “*cross-platform*”, dus de creatie software is beschikbaar voor meerdere operating systemen (Windows en Linux);
- Wink biedt ondersteuning van opname van audio, hetgeen de mogelijkheid biedt om voice-over toelichting bij te voegen;
- Er is keuze uit meerdere uitvoer formaten: o.a. Flash (naar schatting is op 90% van de PCs de Flash Player geïnstalleerd), hetgeen betekent dat de screencasts zowel op een stand-alone machine afgespeeld kunnen worden als vanaf een website in een browser.
- Wink is freeware (gratis).

In de vergelijking zijn onder andere ook AviScreen, CamStudio en Copernicus betrokken.

Het maken van een screencast van een voorbeeld van een redelijke omvang kan zeker een bijdrage leveren aan de leerbaarheid van Rascal; daarnaast kan een dergelijke screencast gebruikt worden voor voorlichting en PR doeleinden op presentaties. De kosten van het maken zijn beperkt tot de te besteden tijd.
In het huidige onderzoek is afgezien van het maken en toetsen van een screencast voor Rascal. De reden daarvan is dat door de ontwikkelaars werd aangegeven dat op het moment van afronding van het onderzoek de gebruikersinterface nog aan veranderingen onderhevig zou kunnen zijn, waardoor de uitkomsten van deze toetsing geen relevantie zouden hebben. Ook zou de inspanning van het maken van de screencast dan geen rendement opleveren.

Ontwikkelomgeving.
De voor Rascal gebruikte IDE (Eclipse) heeft zichzelf al in de praktijk bewezen met talen als Java en C++. Het is zinvol om naar deze talen te kijken om te inventariseren welke functionaliteit geïmplementeerd zou kunnen worden om de leerbaarheid van Rascal te ondersteunen.
Hierbij kan gedacht worden aan zaken als:

- automatische module import;
- code completion;
- context-sensitieve help;
- dynamisch aangeven van syntax fouten en
- een geïntegreerde debugger.

Dit zijn mijns inziens de punten waarmee maximaal gescoord kan worden.

Bovendien mag de onderliggende infrastructuur geen belemmering voor het gebruik van Rascal zijn, hetgeen betekent dat er Rascal implementaties beschikbaar moeten komen voor Linux, Windows en Apple. Hierdoor heeft de gebruiker de keuze om in een vertrouwde omgeving te werken, zonder ook nog door de leercurve van een nieuw platform te moeten gaan.

Mindmapping.
Mindmpping is een interessante techniek, die weinig toegepast wordt als didactisch hulpmiddel. De student die een mindmap moet maken wordt gedwongen als het ware afstand te nemen van de details, en op een hoger abstractieniveau te kijken naar concepten en relaties tussen de concepten. Door hiervan een grafische representatie te maken wordt het begrip verhoogd.

Indien ook mindmapping als techniek gebruikt gaat worden kan dit ondersteund worden door Freemind, gezien het aantal voordelen dat dit hulpmiddel heeft.

Toetsing.
Blackboard kan gebruikt worden om tussentijds kennis te toetsen door middelen van periodieke tests. Het bleek in de 2e enquête, waarin een aantal kleine opgaven waren verwerkt, dat het begrip eenvoudig op deze wijze getoetst kan worden. De studenten zullen wel al hands-on ervaring met Rascal moeten hebben; op strikt theoretische basis (vanuit hoorcolleges en/of documentatie alleen) zullen de resultaten van deze toetsen waarschijnlijk teleurstellend zijn (zowel voor de studenten als voor de docenten!).
Dit blijkt ook uit de in enquête 2 gemaakte uitwerkingen.

Inzet van multimedia.
Multimedia zoals audio en video is ook nu al te integreren in het onderwijsmateriaal; colleges en workshops kunnen op een eigen site of via Blackboard aangeboden worden om off-line (nogmaals) te worden bestudeerd.

Het verdient aanbeveling om een website te maken die aan Rascal gewijd is. Een belangrijk onderdeel hiervan zou een forum kunnen zijn, waarbij gebruikers onderling en met het ontwikkelteam kunnen communiceren.
Hierdoor krijgt het ontwikkelteam regelmatig feedback van de gebruikerspopulatie, en kan Rascal gericht evolueren.

Interactieve leeromgeving.
Hoewel er al op enkele plaatsen in deze scriptie aan gerefereerd is, is het toch zinvol om op deze plaats nogmaals de bevindingen van het onderzoek naar computerondersteund onderwijs samen te vatten, met name vanwege de teleurstellende resultaten.
Bij aanvang van het onderzoek had ik de verwachting dat ik een keuze zou hebben uit een aantal (al dan niet commerciële) hulpmiddelen, waarin ik de Rascal syntax samen met een aantal oefeningen kon pluggen, waarna de studenten de gekozen omgeving als geautomatiseerd leermiddel zouden kunnen toepassen.
Idealiter zou het tool de door een student gegeven oplossing vergelijken met een standaardoplossing en op basis van eventuele verschillen een score en feedback verschaffen.

Bovenstaande hoop werd al snel de bodem ingeslagen. De literatuur die experimenten met dergelijke systemen beschreef was gebaseerd op een inventarisatie van oude experimenten [Wit01], een aantal systemen die gebaseerd zijn op LISP [Col01, Goe01, Sch01, Web01-1, Web01-2]of weliswaar nieuwere (multimedia en webbased) systemen zoals VIOPE [Car06] die geen mogelijkheid bieden om een willekeurige programmeertaal “in te pluggen”.
Het is dan ook niet mogelijk gebleken om via computerondersteund leren de leerbaarheid van Rascal te toetsen.

Dit alles leidt tot de conclusie dat de kosten van een interactieve leeromgeving (CAL) op dit moment nog te hoog zijn om inzet in het onderwijsprogramma van Rascal te realiseren en justificeren.
Met name bij het ontwikkelen van cases en de functionaliteit die op intelligente wijze de antwoorden verwerkt en relevante feedback geeft is te arbeidsintensief en (dus) te duur.
Als in de toekomst geautomatiseerde, generieke en multifunctionele hulpmiddelen hiervoor beschikbaar komen kan eventueel een heroverweging worden gemaakt.

Op dit moment worden de voorlopers van Rascal (ASF+SDF en RScript) nog redelijk klassiek gedoceerd: hoorcolleges, gevolgd door werkcolleges en practica.
Door een uitbreiding hiervan met bijvoorbeeld screencasts, video-colleges (eerder opgenomen hoorcolleges die online bekeken kunnen worden), en online individuele toetsen met behulp van Blackboard kan de student meer zijn eigen leerroute uitstippelen. Zoals eerder geconstateerd vinden studenten dit de prettigste en meest effectieve wijze van kennis vergaren. Hiervoor moet dan nog wel het nodige materiaal ontwikkeld worden, en zal het leertraject dit ook moeten ondersteunen.
De kosten hiervan zijn echter niet dusdanig dat het niet mogelijk of zinvol is om hier verder uitvoering aan te geven.

Bevindingen uit de 2e enquete.

- Het manual wordt over het algemeen als voldoende aangemerkt.
- De EASY concepten en voorbeelden worden positief ervaren.
- Ook de Rascal concepten en voorbeelden worden positief beoordeeld, maar hier ligt de gemiddelde waardering wel iets lager, met name over de voorbeelden. Vooral de concepten en gebruik van data structuren, constraint solving en execution en flow of control worden als onduidelijk ervaren.
- Daar staat tegenover dat de andere concepten duidelijk zijn, waarbij het comprehension deel zelf 100% scoort voor uitleg van concept en gebruik. Gezien het feit dat de ondervraagde studenten uit alle jaren van het HBO komen is dat een positieve ervaring!
- De klassieke voorbeelden scoren hoog op het gebied van begrip en toegevoegde waarde.
- Het Mystery Box voorbeeld lijkt daarentegen toch (te?) complex te zijn, gezien de lage scores op begrip.

7.3 Evaluatie

Weten we nu dingen die voor dit onderzoek niet bekend waren?

Algemeen gezien zijn er geen onverwachte uitkomsten. Het feit dat het exogeen constructivisme de meest geëigende onderwijsvorm is wekt geen verbazing; dit didactisch model is al geruime tijd in gebruik voor de meest uiteenlopende onderwerpen, en de voor- en nadelen zijn bekend.

Interessant is wel dat er een bruikbare lijst is met ontwerpcriteria / requirements voor programmeertalen. In de literatuur over requirements engineering wordt niet expliciet ingegaan op het ontwikkelen van programmeertalen; in mijn optiek is dit een omissie, omdat het een specifiek type toepassing betreft.

Het blijkt dat Rascal op de meeste punten voldoet aan de kwaliteitseisen die aan de leerbaarheid van een programmeertaal mogen worden gesteld. Uitbreiding kan nog plaats vinden door inzet van nieuwe hulpmiddelen.

Studenten kunnen (blijkens de ingeleverde uitwerkingen) niet alleen op basis van het usermanual Rascal leren. Leren programmeren kan dus niet alleen op basis van (zelf-)studie van theoretisch materiaal; ook praktische oefening speelt een grote rol in het verwerven van expertise.
Waarschijnlijk was het succes (en de succesbeleving) van de betrokken studenten groter geweest als ze ook een Rascal implementatie tot hun beschikking hadden gehad, omdat ze dan (directe) feedback op hun oplossingen hadden gekregen.

De meerderheid van de ondervraagde studenten geeft aan dat er zeker behoefte is aan theorie in de vorm van hoorcolleges en oefening in de vorm van practica. Dit zou mijns inziens uitgebreid moeten worden met mindmapping en kleine, wekelijkse voortgangstoetsen (pass/fail) in Blackboard. Deze toeten kunnen dan de plaats innemen van de huidige pass/fail labtoets in de afronding van het vak Software Evolution.

7.4 Eventueel toekomstig onderzoek

Binnen een onderzoek als dit is het helaas niet mogelijk om alle interessante paden in te slaan; de tijd als beperkende factor dwingt tot het maken van keuzes. Dat betekent dat er nog een aantal aspecten open ligt voor toekomstig onderzoek.

Zo kan de leerbaarheid (en het gebruiksgemak) van Rascal bevorderd worden door te kijken wat voor ondersteuning Eclipse kan bieden voor de student en ontwikkelaar. Voorbeelden hiervan zijn te vinden door naar onder andere de ondersteuning van Java in Eclipse te kijken.

Ook de ontwikkeling van een (generieke) leeromgeving is een uitdaging. Het zou aanzienlijk helpen als er een leeromgeving zou zijn waarbij studenten programma's kunnen invoeren (of misschien importeren vanuit Eclipse) waarbij de juistheid van het programma geanalyseerd zou worden en feedback wordt gegeven met betrekking tot de kwaliteit van de oplossing.

Beide onderwerpen kunnen mijns inziens onderwerp zijn van een volwaardige en interessante afstudeeropdracht binnen de master SE opleiding.

Referenties.

[**And86**]Anderson, John R. & Skwarecki, Edward (1986)
The automated tutoring of introductory computer programming
Communications of the ACM, September 1986, Volume 29, number 9, pp 842-849

[**And87**]Anderson, John R. & Boyle, C. Franklin & Farrell, Robert & Reiser, Brian J. (1987)
Cognitive principles in the design of computer tutors
Modelling Cognition, edited by P. Morris, 1987 John Wiley & Sons Ltd., pp 93-133

[**Bar07**]Barak, Miri & Harrad, Judson & Kocur, George & Lerman, Steven (2007)
Transforming an Introductory Programming Course: From Lectures to Active Learning via Wireless Laptops
Journal of Science Education and Technology, Vol. 16, No. 4, August 2007

[**Ber07**]Berglund, Anders & Wiggberg (Eds.) (2007)
6th Baltic Sea Conference on Computing Education Research – Koli Calling 2006
Technical report 2007-006, Department of Information Technology, Uppsala University, Uppsala, Sweden

[**Bro77**]Brooks, Ruven (1977)
Towards a theory of the cognitive processes in computer programming
International Journal of Human-Computer Studies 51, pp 197-211, (1999)

[**Car06**] Carver, Jeffrey & Henderson, Lisa(2006)
Viope as a Tool for Teaching Introductory Programming: An Empirical Investigation
Proceedings of the 19th Conference on Software Engineering Education & Training (CSEET'06)

[**Col01**]Collani, Gernot v. & Schomann, Munira (2001)
The Process of Acquisition of a New Programming Language (LISP): Evidence for Transfer of Experience and Knowledge in Programming
Cognition and Computer Programming, Ablex Publishing Corporation, pp 169-191
Wender, Schmalhofer and Böcker, editors

[**Dal01**]Dalgarno, Barney (2001)
Interpretations of constructivism and consequences for Computer Assisted Learning
British Journal of Educational Technology 2001 Vol. 32, No.2, pp 183-194

[**Dee98**]Deek, Fadi P. & McHugh, James A. (1998)
A Survey and Critical Analysis of Tools for Learning Programming
Computer Science Education 1998, Vol. 8, No. 2, pp 130-178

[**Goe01**]Goebel, Rainer (2001)
The Role of Visual Perception, Selective Attention and Short-Term Memory for Symbol Manipulation: A Neural Network Model that Learns to Evaluate Simple LISP Expressions
Cognition and Computer Programming, Ablex Publishing Corporation, pp 107-139
Wender, Schmalhofer and Böcker, editors

[**Gom05**]Gómez-Albarrán, Mercedes (2005)
The Teaching and Learning of Programming: A Survey of Supporting Software Tools
The Computer Journal Vol. 48, No. 2, 2005

[**Had08**]Hadjerrouit, Said (2008)
Towards a Blended Learning Model for Teaching and Learning Computer

Programming: A Case Study
Informatics in Education, 2008, Vol. 7, No. 2, 181–210

[**Has05**]Hassapis, George & Pavlidou, Niovi (2005)
Teaching ICT courses to virtual classes by using e-learning environments that support collaborative work
World Transactions on Engineering and Technology Education, Vol. 4, No. 1, pp 35-38

[**Jen01**]Jenkins, Tony (2001)
Teaching Programming – A Journey form Teacher to Motivator
2nd Annual LTSN-ICS Conference, London

[**Kli09-1**]Klint, Paul & van der Storm, Thijs & Vinju, Jurgen (2009)
Rascal Requirements and Design Document
Centrum voor Wiskunde en Informatica

[**Kli09-2**]Klint, Paul & van der Storm, Thijs & Vinju, Jurgen (2009)
EASY programming with Rascal – Leveraging the Extract-Analyze-SYntesize Paradigm for Meta-Programming
Centrum voor Wiskunde en Informatica

[**Lah05**]Lahtinen, Essi & Ala-Mutka, Kirsti & Jarvinen, Hannu-Matti (2005)
A Study of the Difficulties of Novice Programmers
ITiCSE'05, June 27–29, 2005, Monte de Caparica, Portugal, pp 14-18

[**Mal04**]Malan, Katherine & Hallan, Ken (2004)
Examples that can do Harm in Learning Programming
OOPSLA '04, Oct. 24-28, 2004, Vancover, British Columbia, Canada

[**Man06**]Manilla, Linda & de Raadt, Michael (2006)
An Objective Comparison of Languages for Teaching Introductory Programming
6th Baltic Sea Conference on Computing Education Research – Koli Calling 2006
Technical report 2007-006, Department of Information Technology, Uppsala University, Uppsala, Sweden, pp 32-37

[**May76**]Mayer, Richard E. (1976)
Some Conditions of Meaningful Learning for Computer Programming: Advance Organizers and Subject Control of Frame Order
Journal of Educational Psychology 1976, Vol. 68, No. 2, pp 143-150

[May81]Mayer, Richard E. (1981)
The Psychology of How Novices Learn Computer Programming
Computing Surveys, Vol. 13, No. 1, 1981

[**McD04**]McDougall, Anne & Boyle, Martin (2004)
Student Strategies for Learning Computer Programming: Implications for Pedagogy in Informatics
Education and Information Technologies 9:2, 109–116, 2004

[**Mil02**]Milne, Iain & Rowe, Glenn (2002)
Difficulties in Learning and Teaching Programming—Views of Students and Tutors
Education and Information Technologies 7:1, 55–66, 2002

[**Por04**]Porter, Ron & Calder, Paul (2004)
Patterns in Learning to Program – an Experiment?
Australasian Computing Education Conference (ACE2004), Dunedin, New Zealand. Conferences in Research and Practice in Information Technology, Vol. 30. Raymond Lister and Alison Young, Eds.

[**Rob03**]Robins, Anthony & Rountree, Janet & Rountree, Nathan (2003).

Learning and Teaching Programming: A Review and Discussion.
Computer Science Education 2003, Vol. 13, No. 2, pp 137-172

[**Sch01**]Schomann, Munira (2001)
Knowledge Organization of Novices and Advanced Programmers: Effect on Previous Knowledge on Recall and Recognition of LISP-Procedures
Cognition and Computer Programming, Ablex Publishing Corporation, pp 192-218
Wender, Schmalhofer and Böcker, editors

[**Wal01**]Walszek, Gerd (2001)
GLUE – A Graphical Lisp Environment for Beginners
Cognition and Computer Programming, Ablex Publishing Corporation, pp 317-351
Wender, Schmalhofer and Böcker, editors

[**Web01-1**]Weber, Gerhard & Bogelsack, Alexander (2001)
Representation of Programming Episodes in the ELM Model
Cognition and Computer Programming, Ablex Publishing Corporation, pp 1-26
Wender, Schmalhofer and Böcker, editors

[**Web01-2**]Weber, Gerhard & Mollenberg, Antje (2001)
ELM Programming Environment:A Tutoring Sysm for LISP Beginners
Cognition and Computer Programming, Ablex Publishing Corporation, pp 373-408
Wender, Schmalhofer and Böcker, editors

[**Wit01**]Witschial, Peter (2001)
TRAPS – An Intelligent Tutoring Environment for Novice Programmers
Cognition and Computer Programming, Ablex Publishing Corporation, pp 287-315
Wender, Schmalhofer and Böcker, editors

[**Wul05**]Wulf, Tom (2005)
Constructivist Approaches for Teaching Computer Programming
SIGITE'05, October 20–22, 2005, Newark, New Jersey, USA, pp 245-248

[**Yue06**]Yuen, Allan H.K. (2006)
Learning to program through interactive simulation
Educational Media International, Vol. 43, No. 3, September 2006, pp. 251–268

Appendix A: websites.

Blackboad Academic Suite
http://www.blackboard.com

Dokeos ELO
http://dokeos.org

Freemind mindmapping tool
http://freemind.sourceforge.net/wiki/index.php/Main_Page

Moodle – open source community-based tools for learning
http://moodle.org/

Guided tour: playing with Booleans (ASF+SDF, Paul Klint)
http://wwwmeta-environment.org/doc/books//getting-started/guided-tour-booleans/guided-tour-booleans.htm

Wink –screencast creatie software
http://www.debugmode.com/wink

AviScreen
http://www.bobyte.com/AviScreen/index.asp

CamStudio
http://camstudio.org

Copernicus
http://danicsoft.com/projects/copernicus

Viope
http://www.viope.com/products

Appendix B: verklarende woordenlijst

Declaratief (functioneel) programmeren.
Programmeerparadigma waarbij de logica van het algoritme vastgelegd wordt zonder dat er sprake is van besturingsconstructies. Het uitgangspunt is om te beschrijven wat er gedaan moet worden (functioneel) in plaats van hoe dat moet gebeuren.

Imperatief (procedureel) programmeren.
Programmeerparadigma waarbij het algoritme vastgelegd wordt in statements, die de state van een programma wijzigen.

BlueJ.
Een IDE voor het programmeren van Java, ontwikkeld voor onderwijsdoeleinden. Kan (ook) gebruikt worden voor kleinschalige software ontwikkelprojecten. (Voor meer informatie zie http://www.bluej.org)

Eclipse.
Een omgeving voor het ontwikkelen van software, bestaande uit een geïntegreerde ontwikkelomgeving en een gestructureerd systeem om uitbreidingen hierop te maken. Initieel begonnen als VisualAge van IBM, voor Java ontwikkeling, nu ook geschikt voor andere talen. Dit is de IDE voor Rascal.

Netbeans.
Een (met Eclipse concurrerende) omgeving voor het ontwikkelen van software, bestaande uit een geïntegreerde ontwikkelomgeving en een gestructureerd systeem om uitbreidingen hierop te maken. Geschikt voor meerdere talen (o.a Java, C++, Ruby). Wordt actief ondersteund door Sun. Netbeans en Eclipse nemen in een soort haasje-over de positie als leading/bleeding edge ontwikkeltool. Netbeans kent wat meer integratie in packaging van complete ontwikkelkits dan Eclipse, waar de gebruiker meer vrijheid heeft om een eigen unieke omgeving samen te stellen via beschikbare plugins.

Screencast.
Een screencast is een digitale opname van het computerbeeldscherm door gebruik te maken van software die vastlegt wat er op het beeldscherm gebeurt. Een screencast kan vaak voorzien worden van commentaar en extra informatie.

Appendix C: de vragen van enquête 1

Bij (technische) informaticastudenten van de Hogeschool van Amsterdam is een enquête afgenomen om in kaart te brengen wat tijdens het leren programmeren als moeilijk c.q. ondersteunend ervaren wordt.
De enquête bevatte de volgende vragen.

Ik had al programmeerervaring of kennis toen ik aan deze opleiding begon

- Ja
- Nee

Als je op de vorige vraag JA geantwoord hebt, hoe lang had je dan al ervaring?

- Minder dan 1 jaar
- Tussen 1 en 2 jaar
- Tussen 2 en 5 jaar
- Meer dan 5 jaar

Mijn ervaring met de volgende programmeertalen is als volgt te omschrijven

	Geen	Beginner	Ervaren	Expert
C				
C++				
PHP				
Visual Basic				
C#				
Java				
Anders				

Als je een niet met name genoemde programmeertaal beheerst, welke is dat dan?

Als je programmeert, ben je dan bezig met:

- Zelf leren programmeren
- Anderen leren programmeren
- Software ontwikkelen

Hoe effectief zijn voor jou de volgende leermethodes en hulpmiddelen?

	Zeer effectief	Effectief	Een beetje effectief	Niet effectief	Niet van toepassing
Hoorcolleges					
Zelf coderen in de					

klas					
Zelf coderen thuis					
Voorbeeldcode van de docent					
Tutorial doorwerken in kleine groep					
Hulp van practicum-assistent					
Hulp van docent					
Alleen studeren (thuis)					
Met een vriend studeren (thuis)					
Geintegreerde ontwikkel-omgeving (IDE)					
Voorgeschreven boek					
Website vak					
Website internet					
Powerpoint presentaties					
Voorbeeldprogramma's en -uitwerkingen					

Geef aan hoe vaak je tijdens practica en/ of werkcolleges het volgende doet:

	Zeer vaak	Vaak	Regelmatig	Niet zo vaak	Nooit
Je probeert een voorbeeld uit te werken					
Je probeert een opgave uit te werken					
Het lukt je een opgave uit te werken					

Ik vind/vond je moeilijk bij het leren programmeren:

- De ontwikkelomgeving (IDE) leren kennen
- De structuren van de taal (lussen, voorwaarden enz.) leren kennen
- De syntax (sleutelwoorden,haakjes, accolades, puntkomma's) leren
- Een programma bedenken om een bepaald probleem op te lossen
- Functionaliteit opdelen in procedures/functies/methodes

- Compilatiefouten vinden in je eigen programma
- Logische fouten vinden in je eigen programma
- Ik vind / vond niets moeilijk

Welke concepten vind/vond je moeilijk?

- Elementaire datatypes
- Abstracte / complexe datatypes
- Variabelen (definitie, initialisatie, levensduur, scope)
- Keuzestructuren (if/else, case)
- Lusstructuren (do / while, while, for)
- Recursie
- Arrays
- Referenties
- Pointers
- Parameters (formeel, actueel, doorgeven, by value, by reference)
- Objecten en klassen
- Interfaces
- In- en uitvoer
- Voorkomen en afhandelen van fouten (try / catch constructies etc.)
- Het gebruik van algemene bibliotheken (begrijpen van APIs etc.)
- Overerving
- Objectorientatie
- Ik vind / vond niets moeilijk

Ik zou graag nog het volgende willen opmerken:

Appendix D: de vragen van enquête 2

Er is onderzoek gedaan naar de nieuwe gebruikersdocumentatie voor Rascal (EASY programming with Rascal, [Kli09-2]). Na lezing van het document is aan de studenten die meededen aan het onderzoek de volgende vragenlijst voorgelegd.

1 Ik geef de algemene beschrijving van het EASY concept op een schaal van 1 -10 het volgende cijfer:
2 Ik geef het voorbeeld over "security breaches" op een schaal van 1 - 10 het volgende cijfer:
3 Ik geef het voorbeeld over "forensic DSL compiler" op een schaal van 1 - 10 het volgende cijfer:.
4 Ik geef het voorbeeld over "renovation financial software"op een schaal van 1 - 10 het volgende cijfer:
5 Ik geef het voorbeeld over "concurrency errors"op een schaal van 1 - 10 het volgende cijfer:
6 Ik zie de aangegeven voordelen van Rascal in de context van het EASY concept
7 Het is duidelijk voor mij wat het doel van het manual is en voor wie het bedoeld is
8 Ik heb de volgende opmerking(en) over dit deel van het manual:

9 Einde EASY concepten, start Rascal concepten

10 Ik begrijp wat values zijn binnen Rascal en hoe ze gebruikt worden
11 Je moet een Rascal programma schrijven dat als invoer sourcecode krijgt. Maak je eigen Rascal datatypes aan de hand van de volgende beschrijving:

- AuthNaam bevat de naam van de schrijver van een fictief programma
- LinesOfCode bevat het aantal regels in bovengenoemd programma

12 Ik begrijp wat datastructuren zijn binnen Rascal en hoe ze gebruikt worden
13 Ik geef het voorbeeld over datastructuren (pag. 8) op een schaal van 1 - 10 het volgende cijfer:
14 Ik begrijp wat pattern matching is binnen Rascal en hoe het gebruikt wordt
15 Ik geef de voorbeelden over pattern matching (pag. 8) op een schaal van 1 - 10 het volgende cijfer:
16 Ik begrijp wat parsing is binnen Rascal en hoe het gebruikt wordt
17 Ik begrijp wat enumerators zijn binnen Rascal en hoe ze gebruikt worden
18 Ik geef het voorbeeld over enumerators (pag. 9) op een schaal van 1 - 10 het volgende cijfer:
19 Maak een enumerator die de woorden "Aap", "Noot", "Mies" bevat open
20 Ik begrijp wat comprehensions zijn binnen Rascal en hoe ze gebruikt worden
21 Ik geef de voorbeelden over comprehensions op een schaal van 1 - 10 het volgende cijfer: open
22 Schrijf een comprehension voor het volgende reeks voor de getallen 1 tot 10, die voldoen aan het volgende voorbeeld:
1:
1 +1 = 2

als de uitkomst van de optelling, gedeeld door 3, niet 0 is moet dit getal in de de uiteindelijke relatie opgenomen worden (dus 1 voldoet aan deze eis, de som (2) komt in de resultaatset)
23 Ik begrijp wat control structures zijn binnen Rascal en hoe ze gebruikt worden
24 Ik geef het voorbeeld over control structures (pag. 9) op een schaal van 1 - 10 het volgende cijfer:
25 Ik begrijp wat switching is binnen Rascal en hoe het gebruikt wordt
26 Ik begrijp wat visting is binnen Rascal en hoe het gebruikt wordt
27 Ik begrijp wat functions zijn binnen Rascal en hoe ze gebruikt worden
28 Ik geef het voorbeeld over functions (pag. 10) op een schaal van 1 -10 het volgende cijfer:
29 Maak op basis van de volgende body de functie SUM, die voor het ingevoerde getal de som van 1 t/m dat getal teruggeeft:

```
int sum(int N)
{
......
}
```

30 Ik begrijp wat rewrite rules zijn binnen Rascal en hoe ze gebruikt
31 Ik geef het voorbeeld over rewrite rules op een schaal van 1 - 10 het volgende cijfer:
32 Ik begrijp wat constraint solving is binnen Rascal en hoe het gebruikt wordt
33 Ik begrijp wat type checking is binnen Rascal en hoe het gebruikt wordt
34 Ik begrijp wat execution is binnen Rascal en hoe de flow of control werkt
35 Op basis van het onderdeel over de Rascal concepten heb ik nog de volgende opmerking(en):

36 Einde **Rascal concepten, start classic examples**

37 Ik begrijp het Hello voorbeeld en de uitwerking
38 Het Hello voorbeeld heeft voor mij toegevoegde waarde om Rascal te leren
39 Ik begrijp het Factorial voorbeeld en de uitwerking
40 Het Factorial voorbeeld heeft voor mij toegevoegde waarde om Rascal te leren
41 Ik begrijp het Colored Trees voorbeeld en de uitwerking
42 Het Colored Trees voorbeeld heeft voor mij toegevoegde waarde om Rascal te leren
43 Ik begrijp het Word Replacement voorbeeld en de uitwerking
44 Het Word Replacement voorbeeld heeft voor mij toegevoegde waarde om Rascal te leren
45 Ik heb nog de volgende opmerking(en) over de voorbeelden:
46 Einde classic examples, begin mystery box example
47 Het hoofdstuk over Problem Solving Stategies (pag. 14 - 20) heeft toegevoegde waarde voor mijn begrip en toepassing van Rascal
48 Het hoofdstuk over Problem Solving Stategies (pag. 14 - 20) geef ik op een schaal van 1 - 10 het volgende cijfer:
49 Ik heb nog de volgende opmerking(en) over het hoofdstuk over Problem Solving Stategies (pag. 14 - 20):

50 **Einde classic examples, begin mystery box example**

51 Ik begrijp het mystery box voorbeeld (de Call Graph Analysis) helemaal (alle bewerkingen en voorbeeldprogramma's)

52 Ik begrijp het mystery box voorbeeld deels (de volgende voorbeeldprogramma's):
53 Ik geef het mystery box voorbeeld (de Call Graph Analysis)op een schaal van 1 - 10 het volgende cijfer:
54 Ik begrijp het Component Structure voorbeeld helemaal (alle bewerkingen en voorbeelden)
55 Ik geef het Component Structure voorbeeld op een schaal van 1 - 10 het volgende cijfer:
56 Ik begrijp het Analyzing the Structure of Java Systems voorbeeld helemaal (alle bewerkingen en voorbeelden)
57 Ik geef het Analyzing the Structure of Java Systems voorbeeld op een schaal van 1 - 10 het volgende cijfer:
58 Ik begrijp het voorbeeld over ongeinitialiseerde en ongebruikte variabelen helemaal (alle bewerkingen en voorbeelden)
59 Ik geef het voorbeeld over ongeinitialiseerde en ongebruikte variabelen op een schaal van 1 – 10 het volgende cijfer:
60 Ik begrijp het McCabe voorbeeld helemaal
61 Ik geef het McCabe voorbeeld op een schaal van 1 - 10 het volgende cijfer:
62 Ik begrijp het Dataflow Analysis voorbeeld helemaal (alle bewerkingen en voorbeelden
63 Ik begrijp het Dataflow Analysis voorbeeld deels (de volgende voorbeeldprogramma's):
64 Ik geef het Dataflow Analysis voorbeeld op een schaal van 1 - 10 het volgende cijfer

65 **Einde mystery box voorbeeld, begin algemeen Rascal deel**

66 Ik begrijp de (grotere) voorbeelden beter als de Rascal taalelementen (syntax) eerst besproken worden
67 Na het lezen van dit document begrijp ik de voordelen van Rascal en kan ik deze samenvatten
68 Ik zie nog andere toepassingen van Rascal dan de genoemde voorbeelden:
69 Ik denk dat ik op basis van dit document zelfstandig Rascal kan leren gebruiken en toepassen
70 Om Rascal te leren begrijpen en toepassen denk ik hoorcolleges nodig te hebben
71 Om Rascal te leren begrijpen en toepassen denk ik werkcolleges nodig te hebben
72 Om Rascal te leren begrijpen en toepassen denk ik practica nodig te hebben

73 **Einde algemeen Rascal deel, begin algemene kennis vragen**

74 Mijn eerder opgedane programmeerkennis helpt om Rascal te begrijpen
75 Voor ik dit document las wist ik wat de begrippen pattern matching en reguliere expressies inhouden
76 Mijn kennis van pattern matching en reguliere expressies is voldoende om de voorbeelden te kunnen volgen
77 Mijn kennis van pattern matching en reguliere expressies is voldoende om er in voorkomende gevallen zelf oplossingen mee te maken
78 Voor ik dit document las had ik kennis van verzamelingenleer (enumerators en comprehensions)
79 Mijn kennis van verzamelingenleer is voldoende om de voorbeelden te kunnen
80 Mijn kennis van verzamelingenleer is voldoende om in voorkomende gevallen zelf oplossingen met Rascal te maken

81 Mijn beheersing van het Engels is voldoende om de inhoud van het document te begrijpen
82 Ik zit in: 1e jaar / 2e jaar / 3e en 4e jaar

83 **Einde algemene kennis vragen**

84 Ik geef dit document op een schaal van 1 tot 10 het volgende cijfer:
85 Ik heb nog de volgende opmerking(en) over het document en/of Rascal:

In de enquête is de studenten gevraagd om op basis van het user manual een paar kleine voorbeeldopgaven uit te werken.
De letterlijke respons van de studenten is hieronder weergegeven.

Vraag 11
Je moet een Rascal programma schrijven dat als invoer sourcecode krijgt. Maak je eigen Rascal datatypes aan de hand van de volgende beschrijving:

- AuthNaam bevat de naam van de schrijver van een fictief programma
- LinesOfCode bevat het aantal regels in bovengenoemd programma

Antwoorden:

- sorry, ik snap deze vraag absoluut niet....
- rascal> AuthNaam;
- data AuthNaam = ...; data LinesOfCode = ...;
- data SourceCode = code(str AuthNaam, int LinesOfCode);
- data STAT = asgStat (AuthNaam, LinesOfCode)
- // Beetje zitten spieken in de manual. Heb het nog
  // zeker niet onder de knie.
  void countAssignments(PROGRAM P) {
  int LinesOfCode = 0;
  String AuthNaam = "";
  visit (P) {
  case amountLines(Id lines, _):
  LinesOfCode += 1;
  case idAuthor(id Author, _):
  AuthNaam = Author; }
  // Ik maak hier gebruik van de + operator
  // omdat de indruk gewekt dat println net als
  // in java gebruikt kan worden (op pagina 73) println("Author: " + AuthNaam);
  println("Lines of Code: " + LinesOfCode); }
  // geloof niet dat dit echt werkt, maar dit
  // lijkt me het idee.
- data AuthNaam = name(str n); data LinesOfCode = loc(int lines);
- Op dit punt in de manual heb ik nog geen flauw idee wat de notatie is van datatypes. Er staat een kort voorbeeld maar daar kan ik geen concrete notatie uit afleiden. Het ziet er behoorlijk anders uit dan in normale programmeertalen, ik denk dat ze dit punt meer aandacht hadden moeten geven in de manual.

Vraag 19:

Maak een enumerator die de woorden "Aap", "Noot", "Mies" bevat

- string sEnum <- {"Aap", "Noot", "Mies"}
- int x = {"Aap","Noot","Miels"};
- int e <- { "Aap", "Noot", "Mies" }
- str woorden <- {"Aap", "Noot", "Mies"}
- int x <- {"Aap", "Noot", "Mies"}
- str x <- {"Aap", "Noot", "Mies" }
- void enumTest() { String x <- {"Aap", "Noot", "Mies"} // x bevat nu de woorden "Aap", "Noot", "Mies" }
- str E <- {"Aap", "Noot", "Mies"};
- string s <- { “Aap”, “Noot”, “Mies” }

Vraag 22:
Schrijf een comprehension voor het volgende reeks voor de getallen 1 tot 10, die voldoen aan het volgende voorbeeld:
1:
1 +1 = 2
als de uitkomst van de optelling, gedeeld door 3, niet 0 is moet dit getal in de de uiteindelijke relatie opgenomen worden (dus 1 voldoet aan deze eis, de som (2) komt in de resultaatset)

- {x+x | int x <- [1 .. 10], (x + x) / 3 >= 1 }
- {1+1 / 3};
- { x / x | int x <- [1 .. 10], x % 3 != 0 }
- { x + x | int x <- [1 .. 10], (x + x) % 3 == 0 }
- { x + x | int x <- [1 .. 10], x % 3 != 0 }
- { x * x | int x <- [1 .. 10], (x + x) % 3 == 0 }
- void calc() { int relatie = 0; {x * x | int x <- [1 .. 10], x % 3 != 0, relatie <- x} }
- {x+x | int x <- [1 .. 10], x / 3 != 0 }
- { x | int x <- [1 .. 10], (x+x) % 3 != 0 }

Vraag 29
Maak op basis van de volgende body de functie SUM, die voor het ingevoerde getal de som van 1 t/m dat getal teruggeeft:

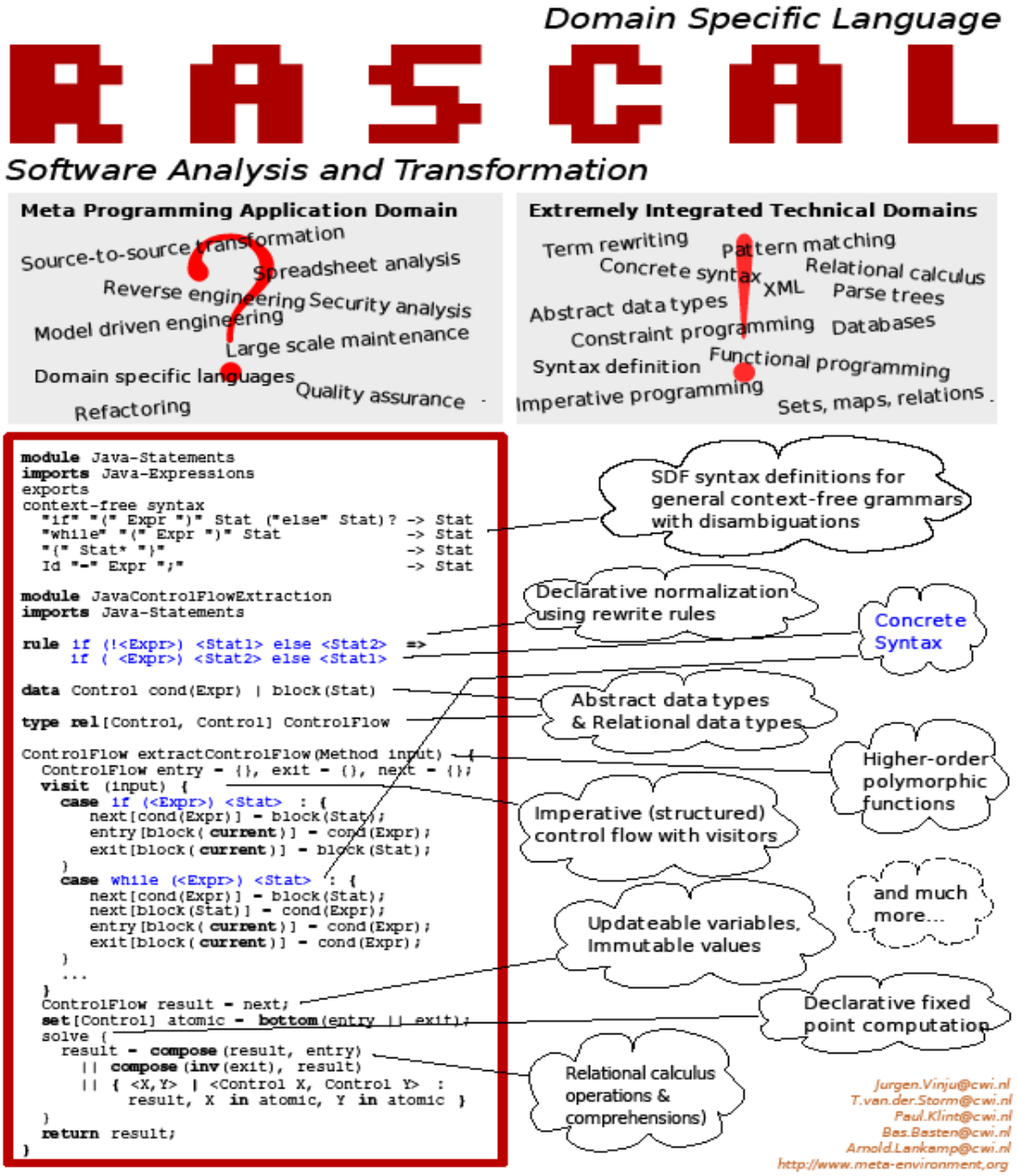
```
int sum(int N)
{
.....
}
```

- int i = 0; for ([0 .. N]) i + 1; return i;
- int z = 0; z += 1;
- int C = 0; int sum(int N) { for(C < N, C++){ N = N + C; } return N; }

- int sum(int N) { int c = 0; int r <- [1..N]; visit(r) { c = c + r; }; return c; }
- int sum(int N) int n = 0; { n += 1; } return n; }
- int sum(int N) { for (...) { } }
- int sum(int N) { int value = 1; for(value += N, N-= 1 < 1); return value; }
- int sum(int N) { if(n <= 0) return 0; else return n + sum(n-1); }
- int sum(int n) { m = 0; whileStat(m <= n) { n += (m++); } }

Appendix E: Rascal-at-a-glance

Deze poster is door het CWI gemaakt ter illustratie van de mogelijkheden en eigenschappen van Rascal. De achterliggende ideeën en concepten worden er op afgebeeld, evenals een voorbeeld van een Rascal programma.


Domain Specific Language
RASCAL
Software Analysis and Transformation
Meta Programming Application Domain
Source-to-source transformation
Spreadsheet analysis
Reverse engineering
Security analysis
Model driven engineering
Large scale maintenance
Domain specific languages
Quality assurance
Refactoring
Extremely Integrated Technical Domains
Term rewriting
pattern matching
Concrete syntax
XML
Relational calculus
Parse trees
Abstract data types
Constraint programming
Databases
Syntax definition
Functional programming
Imperative programming
Sets, maps, relations
module Java-Statements
imports Java-Expressions
exports
context-free syntax
"if" "(" Expr ")" Stat ("else" Stat)? -> Stat
"while" "(" Expr ")" Stat -> Stat
"{" Stat* "}" -> Stat
Id "=" Expr ";" -> Stat
module JavaControlFlowExtraction
imports Java-Statements
rule if (!<Expr>) <Stat1> else <Stat2> =>
if (<Expr>) <Stat2> else <Stat1>
data Control cond(Expr) | block(Stat)
type rel[Control, Control] ControlFlow
ControlFlow extractControlFlow(Method input) {
ControlFlow entry = {}, exit = {}, next = {};
visit (input) {
case if (<Expr>) <Stat> : {
next[cond(Expr)] = block(Stat);
entry[block(current)] = cond(Expr);
exit[block(current)] = block(Stat);
}
case while (<Expr>) <Stat> : {
next[cond(Expr)] = block(Stat);
next[block(Stat)] = cond(Expr);
entry[block(current)] = cond(Expr);
exit[block(current)] = cond(Expr);
}
...
}
ControlFlow result = next;
set[Control] atomic = bottom(entry || exit);
solve (
result = compose(result, entry)
|| compose(inv(exit), result)
|| { <X,Y> | <Control X, Control Y> :
result, X in atomic, Y in atomic }
)
return result;
}
SDF syntax definitions for general context-free grammars with disambiguations
Declarative normalization using rewrite rules
Concrete Syntax
Abstract data types & Relational data types
Higher-order polymorphic functions
Imperative (structured) control flow with visitors
and much more...
Updateable variables, Immutable values
Declarative fixed point computation
Relational calculus operations & comprehensions)
Jurgen.Vinju@cwi.nl
T.van.der.Storm@cwi.nl
Paul.Klint@cwi.nl
Bas.Basten@cwi.nl
Arnold.Lankamp@cwi.nl
http://www.meta-environment.org