Het detecteren van kopieën bij informatica-practica

Dick Grune
Matty Huntjens

Vakgroep Informatica
Faculteit Wiskunde & Informatica
Vrije Universiteit
De Boelelaan 1081
1081 HV AMSTERDAM

ABSTRACT

Bij ieder informatica-practicum, waarbij het ingeleverde werk van de studenten door meerdere begeleiders wordt nagekeken, is het voor een student mogelijk om werk, dat al door iemand anders bij een andere begeleider is ingeleverd, als eigen werk bij de eigen begeleider aan te bieden. Slechts bij hoge uitzondering en dan meestal ook nog bij toeval, zal zoiets ontdekt worden.
Dit leidt tot de onrechtvaardige situatie, dat bijvoorbeeld de ene student een 9 kan krijgen, zonder iets gedaan te hebben, en de andere een 7, na twee weken werken. Dit artikel beschrijft hoe bij een vakgroep Informatica van de VU gepoogd wordt om dit te voorkomen.

Inleiding

Een moeilijk probleem voor begeleiders bij computerpractica met veel deelnemers is: "Hoe kan ik zien dat het ingeleverde werk geen kopie is?" Dat het detecteren hiervan lang geen sinecure is, mag blijken uit het feit dat bijvoorbeeld bij het practicum Inleiding Programmeren van de Faculteit Wiskunde & Informatica van de Vrije Universiteit te Amsterdam, in het studiejaar 1988/1989 1 coördinator en 8 begeleiders nodig waren voor het assisteren van 181 deelnemers, die ieder 10 opgaven moesten inleveren. In dit artikel wordt een oplossing beschreven die werkt door alle ingeleverde opgaven automatisch met elkaar te vergelijken.

De similarity tester

De behoefte aan een programma dat overeenkomsten tussen twee andere programma's kan vinden, is in eerste instantie niet voorgekomen uit de wens om practicumopgaven op kopieën te kunnen kontroleren, maar uit de praktijkervaring dat grote software projecten "na voltooiing" veel op elkaar lijkende code bevatten. Bijvoorbeeld bij het schrijven van bedrijfssystemen, compilers, editors en dergelijke, heeft men geregeld stukken code nodig die veel lijken op stukken code die men al eerder in hetzelfde project geschreven heeft. Het is dan gebruikelijk de betreffende code met een editor te kopiëren en de kopie vervolgens bij te schaven. Vanuit software-engineering oogpunt zou men van zo'n stuk code waarschijnlijk een generiek moduul moeten maken, maar slechts weinig talen bieden deze mogelijkheid en het ontwerpen van een goed interface voor een generiek moduul is een intellectueel inspannende zaak, zodat toch veelvuldig van de kopieer-en-schaaf-techniek gebruik wordt gemaakt.

Is het project eenmaal in wat rustiger vaarwater gekomen, dan rijst de vraag of deze kopieën-paren teruggevonden kunnen worden, om te zien of ze op zinvolle wijze samengenomen kunnen worden; de kopieën zullen in het algemeen niet meer exact gelijk zijn en kleine of grote wijzigingen ondergaan hebben, maar kopieën die onherkenbaar zijn, zijn hierbij niet meer interessant. Voor dit doel heeft de eerste auteur een programma ontworpen en geïmplementeerd, een *software similarity tester*. Dit programma leest files met te onderzoeken code in, reduceert de tekst tot een string van "essentiële tokens" en zoekt vervolgens herhaaldelijk de langste meervoudig voorkomende niet-overlappende substring op. Van de bevindingen wordt dan een rapport afgedrukt. De wijze van reductie tot essentiële tokens is in principe instelbaar, maar houdt in de praktijk meestal in dat alle identifiers en getallen tot het token <idf> worden gereduceerd, alle strings tot <string> en alle karakters tot <char>, dat commentaar en lay-out worden weggegooid en dat keywords elk tot één token gereduceerd worden; de rest van de tokens blijft zichzelf. Van een file van zeg 2000 karakters blijft meestal een skelet van zo'n 100 tokens over.

Het is a priori helemaal niet duidelijk dat deze aanpak enige kans op succes heeft. Men kan zich heel goed voorstellen dat het skelet van een kopie van een stuk code zelfs door licht edit-en van de kopie al zo sterk gewijzigd wordt dat bovenstaande procedure het niet meer herkent; ook kan men zich aan de andere kant voorstellen dat de procedure gemakkelijk spoken ziet. In de praktijk blijkt het allemaal erg mee te vallen; de genoemde verschijnselen doen zich wel voor maar kunnen beteugeld worden door een minimum lengte in te stellen voor de meervoudig voorkomende substring. Een goede waarde is 24; voor waarden onder de 18 worden soms toevallige overeenkomsten in bijvoorbeeld declaratie-lijsten of expressies herkend, terwijl voor waarden boven de 30 bij voorbeeld significant gelijkende korte for-statements aan de aandacht ontsnappen. Met een goede definitie van "essentieel token" en een goede minimum substringlengte is de software similarity tester een bruikbaar hulpmiddel voor het vinden van op elkaar gelijkende stukken code; het ruime merendeel van de gerapporteerde overeenkomsten is zinvol.

Het testen op kopieën

Nadat de similarity tester goed bleek te voldoen, kwam al spoedig het idee op, dit programma te gebruiken voor het vinden van kopieën in door studenten ingeleverde werkstukken. De bovengenoemde twijfels kwamen hier echter in versterkte mate terug. De ingeleverde werkstukken beogen immers allemaal hetzelfde probleem op te lossen; zou dit niet automatisch een zo grote overeenkomst tussen de programma's veroorzaken dat de similarity tester overal kopieën zou zien? En zou niet de slinkse student, die uiteraard weet dat zijn werkstuk automatisch met dat van anderen vergeleken wordt en die weet hoe de similarity tester werkt, niet eenvoudig systematische veranderingen kunnen aanbrengen zodanig dat zijn gekopieerde werkstuk niet meer als kopie herkend wordt?

Wat is "lijken op"?

Om de eerste vraag goed te kunnen beantwoorden, moeten we het begrip "similarity" nauwkeuriger bekijken. We kunnen het kwantificeren door voor twee programmateksten A en B en een minimum stringlengte N een grootheid $S_N(A,B)$ als volgt te definieren. Stel we willen een kopie van A opbouwen door alleen stukken uit B te gebruiken, terwijl we geen stukken met minder dan N essentiële tokens mogen gebruiken. Afhankelijk van de gelijkenis tussen A en B kunnen we zo een kleiner of groter gedeelte van A opbouwen. Het percentage essentiële tokens in A die we zo op hun plaats kunnen krijgen noemen we $S_N(A,B)$.

Over het algemeen zal $S_1(A,B)$ gelijk zijn aan 100%, want elk token in A zal ook wel ergens in B voorkomen; ook zal voor voldoende grote N, $S_N(A,B)$ gelijk zijn aan 0%, tenzij A een exacte kopie van B is. Door N vanaf 1 te laten toenemen kan men het herkende overeenkomstpercentage S verlagen. Voor niet-verwante programmateksten A en B zal het percentage sneller dalen dan voor wel van elkaar afgeleide programmateksten. Het blijkt dat voor N=24 niet-verwante oplossingen van een practicumopgave een S van gemiddeld 20% en maximaal 50% hebben, terwijl wel van elkaar afgeleide oplossingen minimaal een S van 75% à 80% hebben. Op grond hiervan kunnen we dus een duidelijk onderscheid maken.

Bekijken we bijvoorbeeld bij een opgave (opgave *j2*) uit het in de inleiding genoemde practicum alle gevonden percentages en zetten we in een staafdiagram uit hoe de aantallen percentages in de range 1-10, 11-20, ..., 91-100 zich verhouden, dan zien we bij deze opgave dat de meeste van de ingeleverde oplossingen (39) tussen de 1-10 procent op elkaar lijken. Uitschieters lopen door tot 41-50 procent (zie figuur I).

Als alleen de percentages bekeken worden die op één ingeleverde opgave betrekking hebben, dan valt een kopie op doordat er een programma is waarmee een afwijkend hoog percentage aan overeenkomst gevonden wordt (zie figuur II).

Mogelijke tegenacties

Voor het bepalen van de kwetsbaarheid van de test ten opzichte van de "handige" student, moeten we iets dieper ingaan op het gehele proces van inleveren en evalueren van studentenwerkstukken, waarbij we het genoemde practicum Inleiding Programmeren weer als voorbeeld zullen nemen. Het practicum bestaat uit elf opgaven van oplopende moeilijkheidsgraad, genaamd *a* t/m *k*. De opgaven *a* t/m *f* zijn triviaal, dienen als vingeroefeningen en worden niet nagekeken; de opgaven *g* t/m *j* vormen de hoofdschotel; opgave *k* is facultatief. Als een student klaar is met een opgave wordt deze via electronische post naar de begeleider gestuurd. Deze kijkt of de binnengekomen opgave werkt en slaat deze in een database op indien dit het geval is. Doorstaat het werkstuk de test niet, dan gaat de begeleider bij de student langs en vertelt wat er mis gaat, waarna de student het werkstuk corrigeert, weer inlevert, en het proces zich herhaalt.

De in de database opgeslagen programma's worden automatisch met elkaar vergeleken. Dit leidt bij n opgeslagen programma's van eenzelfde opgave tot een lijst van maximaal $n-1$ percentages voor ieder van de opgeslagen programma's; omdat percentages van 0% niet in de lijst voorkomen, zijn er echter meestal minder dan $n-1$ percentages. De begeleider kan deze lijsten opvragen (zie figuur III). Programma's met percentages

tussen 59% en 80% worden door de begeleider kritisch bekeken op originaliteit. Blijkt er sprake van kopiëren te zijn, dan wordt dit aan de coördinator van het practicum gemeld. Bij programma's met percentages groter dan 79% krijgt deze, van de daemon die de database van opgeslagen programma's beheert, automatisch hier al bericht van.

Gaat alles goed, dan wordt een afdruk van het ingeleverde werkstuk gemaakt, dat vervolgens door de begeleider met een cijfer gewaardeerd wordt op basis van de stijl, opbouw, het volgen van richtlijnen, en dergelijke.

De student die probeert gekopieerd werk in te leveren zonder betrapt te worden, ziet zich dus geplaatst voor een aantal moeilijkheden.

- Zijn programmatekst moet in de essentiële tokens geregeld van het origineel verschillen. Dit betekent dat het niet helpt om namen te veranderen, commentaar of lay-out toe te voegen of weg te halen, of code te verplaatsen. Het is hierbij van belang dat de similarity tester niet, zoals het UNIX-programma *diff* de longest common subsequence zoekt, die volgorde-afhankelijk is, maar de longest common sub*string*, die volgorde-onafhankelijk is.
- Zijn programma wordt door de begeleider getest en moet dus goed werken. Dat betekent dat als hij zijn werkstuk bij elkaar sprokkelt, door gedeelten van werkstukken van verschillende medestudenten te kopiëren, hij het geheel wel werkend moet maken. Dit is waarschijnlijk evenveel werk als het gewoon maken van de opgave. Ook maakt het testen van de opgave het onmogelijk om bijvoorbeeld een UNIX manual page als oplossing in te leveren. Dit zou waarschijnlijk tot het fraaie resultaat van 0% als maximum overeenkomst leiden, maar nooit door de compiler heenkomen, laat staan werkende code opleveren.

  Overigens gebeurt het wel eens dat een begeleider een opgave verkeerd opslaat, bijvoorbeeld een ingeleverde *j2*-opgave wordt opgeslagen als een oplossing van opgave *j3*. In dat geval is de maximale gevonden overeenkomst inderdaad 0%, wat dan gelijk de reden is dat dit soort fouten ontdekt worden, omdat dit soort percentages net zo "verdacht" zijn als percentages van 100% .
- Zijn programma wordt door een practicum-begeleider beoordeeld en moet dus goed ogen. Dat betekent dat hij zich niet kan permitteren overal dummy-opdrachten tussen te zetten om de similarity tester te misleiden. Ook het toevoegen van nutteloze parameters aan procedures zou de practicum-begeleider opvallen.

Ervaringen

Hoewel we uiteraard niet zeker weten dat er geen strategie bestaat om een gekopieerd werkstuk over de hier genoemde barrières heen te krijgen, kennen we zo'n strategie niet. Het lijkt niet onredelijk te veronderstellen, dat zo'n strategie, zo deze al zou bestaan, minstens evenveel werk zou vergen als het zelf maken van de opgave. Bijvoorbeeld vervang alle FOR-statements door WHILE-statements, splits grote procedures in kleinere op, kies een iets andere datastruktuur, verander de teksten en lay-out van de uitvoer en dergelijke Maar bij een dergelijke aanpak wordt er uiteindelijk ook geen kopie meer ingeleverd. Het breken van het systeem is dan alleen als sport nog interessant.

Het blijkt dat studenten slechts bij uitzondering kopiëren. De similarity tester detecteert slechts zo tussen de één en de tien pogingen om gekopieerd werk in te leveren per practicum (afhankelijk van de grootte en opzet van het practicum). Pogingen om de similarity tester te misleiden zijn nooit waargenomen. Over het aantal geslaagde, en door ons dus nooit ontdekte pogingen hebben we uiteraard geen gegevens.

Door de grens voor een kopie op 80% te stellen, is bij alle op die manier gevonden gevallen gebleken dat er inderdaad sprake van een kopie was. Bij het afdrukken van de twee overeenkomende programma's is bovendien tot nu toe altijd gebleken dat de overeenkomsten groter waren dan de similarity tester gevonden had. Oorspronkelijk hadden we de grens op 60% gesteld, maar dat bleek te laag te zijn. Ten eerste bleek in deze gevallen bij korte programma's (programma's van één à anderhalve bladzijde) regelmatig dat het percentage te hoog uitviel en ten tweede, als het percentage korrekt was, dan is het bij een kort programma moeilijk om te zeggen dat het programma een kopie is, als het voor 40% verschilt met de "kopie". Bij een gevonden overeenkomst van $\geq 80\%$ bestaan dit soort twijfels niet meer.

Alle studenten die gekonfronteerd werden met ons vermoeden van het inleveren van gekopieerd werk, hebben dit (soms na enig praten) bevestigd. Het grootste probleem was vaak dat ze er niet aan wilden dat ze hetzelfde programma als iemand anders hadden ingeleverd, want ze hadden er immers heel veel werk in gestoken om het programma te veranderen. Daar deze veranderingen echter over het algemeen bestonden uit het veranderen van alle identifiers en de volgorde van de declaraties (iets waar de similarity tester niets van merkt, omdat hij daar helemaal niet op let), werd uiteindelijk toch hetzelfde programma ingeleverd. De 10% à 15% verschil die er dan wel waren werden meestal veroorzaakt doordat de statements die de uitvoer gaven,

herschreven waren. Het algoritme was nooit gewijzigd.

De doelstellingen van de twee eerste-jaars practica waarbij de in dit artikel beschreven procedure gebruikt wordt, is overigens niet om de studenten te "tentamineren", maar om ze iets te leren. We vinden het daarom niet erg als ze bij het maken van de programma's ook geholpen worden door medestudenten. Of ze nu iets leren van het college, van het gebruikte boek, van de begeleider of van een medestudent, in alle gevallen hebben ze iets geleerd, en dat is het enige waar het om gaat. Selectie vindt wel tijdens het tentamen plaats. Deze houding betekent echter niet dat ze zelf het practicum niet meer hoeven te maken. Er is een groot verschil tussen van iemand horen hoe een probleem het beste kan worden aangepakt en dat dan zelf uitwerken, of de uitgewerkte oplossing in de vorm van een electronische kopie aangereikt krijgen en deze dan als eigen werk inleveren.

Conclusie

Onze (niet bewezen) indruk op het moment is, dat het enige dat helpt tegen de similarity tester het schrijven van andere code is. Dit betekent dat iedereen die code van een ander inlevert, ook als eerst veranderingen gemaakt zijn in commentaar, identifiers, lay-out en declaratie-volgordes, als fraudeur te kijk staat.