Beginnende ontwikkelaar kan codekwaliteit verbeteren met slim tutorsysteem

BETERE CODE MET AUTOMATISCHE FEEDBACK EN HINTS

De kwaliteit van code van beginnende ontwikkelaars is vaak laag. Daar zijn natuurlijk tools voor, maar het gros daarvan blijkt niet geschikt voor beginners. Volgens Hieke Keunings kunnen studenten goed geholpen worden met intelligente tutorsystemen die geautomatiseerd zowel hints als feedback geven.

door Hieke Keunings beeld Shutterstock

EEN BEKEND PROBLEEM VOOR DOCENTEN IN HET PROGRAMMEERONDERWIJS: een student schrijft een functioneel correcte oplossing, maar de code is inefficiënt, onnodig complex of onleesbaar. Helaas is het binnen het onderwijs niet altijd mogelijk om voldoende tijd en aandacht te besteden aan de kwaliteit van code. Uit onderzoek blijkt dat zelfs de code van professionele programmeurs veel 'code smells' bevat. Dit kan tot gevolg hebben dat code moeilijk te onderhouden is, en diverse andere kwaliteitskenmerken in het gedrang komen. Een tutorsysteem met automatische feedback en hints kan studenten in een vroeg stadium helpen om code te leren verbeteren.

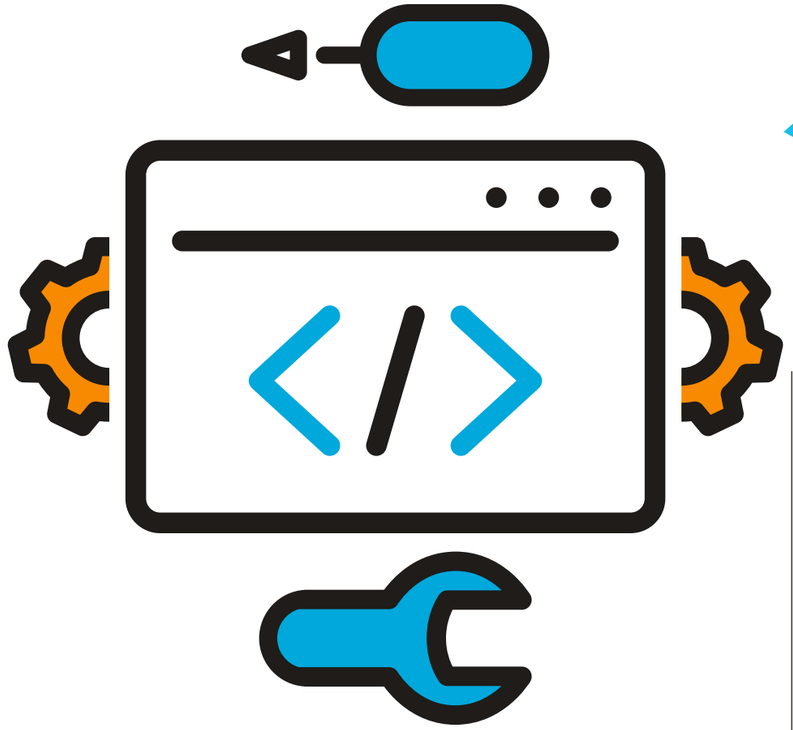
DE KWALITEIT VAN STUDENTCODE

Hoe moeilijk leren programmeren nu eigenlijk is, is al decennialang onderwerp van discussie. In elk geval staat vast dat studenten die leren programmeren talloze fouten maken.[1] Deze fouten kun je onderverdelen in (1) syntactische fouten, zoals het gebruiken van de toewijzingsoperator = in plaats van de vergelijksoperator ==, (2) conceptuele fouten, zoals loops die één keer te veel of juist te weinig worden uitgevoerd, en (3) strategische fouten, zoals het onjuist opdelen van een programmeerprobleem in subproblemen.

Over de kwaliteit van studentcode is daarentegen veel minder bekend, hoewel dit onderwerp steeds meer aandacht krijgt in onderzoek. Een analyse van meer dan 2,5 miljoen codefragmenten geeft een beeld van de kwaliteitsissues in studentcode.[2] De fragmenten zijn automatisch geanalyseerd met de statische analysetool PMD, waarvan checks zijn geselecteerd die relevant zijn voor studenten. Deze checks signaleren kwaliteitsissues op het gebied van de flow van code, keuze voor programmeerconstructies, duidelijkheid van expressies, decompositie en modularisatie. De studentcode bevatte diverse problemen, zoals veel te grote klassen en hoge cyclomatische complexiteit, die vaak niet werden opgelost in latere versies van dezelfde code.

CODEKWALITEIT BINNEN HET ONDERWIJS

Codekwaliteit krijgt binnen het onderwijs niet altijd de aandacht die het verdient. In een onderzoek geven studenten, docenten en ontwikkelaars aan dat ze tijdens hun opleiding weinig hebben geleerd over codekwaliteit. Docenten vinden het belangrijk, maar het speelt een beperkte rol in het summatief beoordelen (bijvoorbeeld met een cijfer) van code. Daarnaast lopen de meningen over de kwaliteit van code sterk uiteen. Dit werd zichtbaar in een onderzoek[2] waarin informaticadocenten studentcode werd voorgelegd van matige kwaliteit, met de vraag welke hints ze zouden geven. Docenten gaven vergelijkbare hints over het verminderen van algoritmische complexiteit en het opruimen van clutter. Over andere aspecten gaven ze echter wisselende combinaties van hints. Er was een grote diversiteit te zien in hoe ze de programma's zouden verbeteren.



ZELFS CODE VAN PROFESSIONELE PROGRAMMEURS BEVAT VEEL 'CODE SMELLS'

PROFESSIONELE TOOLS

Er zijn talloze tools voor professionele ontwikkelaars die helpen bij het opsporen en verbeteren (refactoren) van problematische code, maar het is discutabel of deze geschikt zijn voor beginners. Statische analysetools zoals SonarQube, linters en het eerdergenoemde PMD rapporteren over overtreden regels die 'code smells' detecteren. Voor beginners kunnen de lange lijsten van meldingen met technische terminologie ontmoedigend werken. Verder zijn de meldingen niet altijd relevant in de context van kleine programmeeropgaves. Daarnaast bieden veel IDE's de mogelijkheid om automatisch refactorings door te voeren, waarbij de student weinig inzicht krijgt in de veranderingen. Kortom, het is waarschijnlijk beter om deze tools pas in een later stadium en onder begeleiding in het onderwijs in te zetten.

TOOLS VOOR HET ONDERWIJS

Al sinds de jaren zestig, toen de eerste programmeurs werden opgeleid, worden er tools specifiek voor het onderwijs ontwikkeld die ondersteunen bij het leren programmeren.[3] Dit zijn vaak (intelligente) tutorsystemen, die studenten helpen een programmeeropgave te maken door middel van feedback op stappen en het geven van geautomatiseerde hints. Volgens Kurt VanLehn is het doel van een tutorsysteem om de hulp die een docent zou geven zo goed mogelijk na te bootsen.[4]

Ook bestaan er diverse automated assessment systemen die feedback geven op studentcode. Deze feedback richt zich meestal op het aanwijzen van fouten, bijvoorbeeld door het tonen van gefaalde testcases en studentvriendelijke meldingen over compileerfouten. In mindere mate geven de tools aan hoe je verder kunt als je vastzit, of hoe je code van lage kwaliteit kunt verbeteren.

PROGRAMMEERTUTOR MET FEEDBACK EN HINTS

Een tutorsysteem kan studenten helpen met het vroeg leren hoe ze hun code kunnen verbeteren.[2] Het systeem (zie figuur 1) biedt refactoropgaves aan, waarin de student kleine programma's die al functioneel correct zijn, moet herschrijven naar een versie die beter, leesbaarder of efficiënter is.

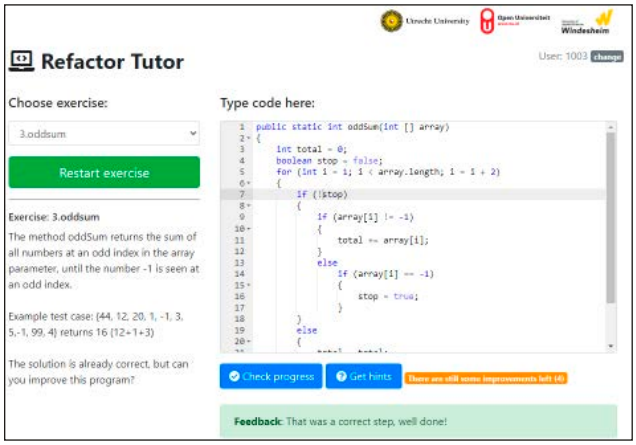
De tutor geeft automatische feedback op de stappen die studenten nemen en geeft hints als ze vastlopen. Bij elke controlestap worden testcases uitgevoerd om te kunnen waarborgen dat het programma nog steeds juist functioneert.

Hints worden dynamisch gegenereerd en gegeven op verschillende niveaus: van een globale aanwijzing tot een concreet codevoorbeeld.

Het systeem is ontwikkeld met behulp van het Ideas software framework[5], dat gebruikt wordt om tutorsystemen voor diverse onderwijsdomeinen te bouwen. De kern van het systeem bestaat uit herschrijfregels die vastleggen hoe een codefragment kan worden vervangen door een ander codefragment met dezelfde semantische betekenis.

De regels zijn gebaseerd op suggesties van docenten uit de eerdergenoemde studie, regels uit professionele tools die geschikt zijn voor studenten, en wiskundige regels.

De kern van het systeem bestaat uit herschrijfregels



Figuur 1: interface van het tutorsysteem

AUTEUR

HIEKE KEUNING
is universitair docent bij de Universiteit Utrecht, waar ze onderzoek doet naar tutorsystemen voor programmeren binnen de onderzoeksgroep Software Technology for Learning and Teaching. Daarvoor werkte ze bij hogeschool Windesheim als docent software engineering bij de ICT-opleidingen, waarnaast ze promotieonderzoek deed bij de Open Universiteit met ondersteuning van een NWO-promotiebeurs voor leraren.

Een eenvoudig voorbeeld is de volgende constructie met een onnodige conditie:

```
if (x > y) {…}
else if (x <= y) {…}
```

Een herschrijfregel specificeert dat de conditie in de else if mag worden weggelaten als deze de ontkenning is van de voorafgaande if. Als de student om een hint vraagt, wordt 'Can you find an else-if condition that is not necessary?' getoond. Klikken op 'Explain more' geeft vervolgens: 'We don't need the check in the else-if, because we know it will be true'. De laatste hint is een codevoorbeeld waarin de suggestie is doorgevoerd.

Het systeem is uitgeprobeerd met ruim honderd tweedejaarsstudenten software engineering van een hogeschool. Hierbij is geanalyseerd hoe ze zich gedragen in het systeem, en wat hun eigen bevindingen zijn. Uit de analyse blijkt dat ze over het algemeen goed uit de opgaves kwamen, maar moeite hadden met bepaalde aspecten, zoals het versimpelen van complexe codestructuren. De hints konden ze hierbij helpen. Studenten gaven aan dat ze codekwaliteit een belangrijk onderwerp vinden, waaraan zeker aandacht moet worden besteed. Er wordt weleens gedacht dat studenten niet verder kijken dan functioneel werkende code, maar het tegendeel blijkt dus waar. In de toekomst wordt het systeem uitgebreid met nieuwe opgaves en regels. Er zal worden gekeken naar ondersteuning voor refactorings op een hoger niveau, zoals klassen en interfaces. Ook is het wenselijk dat docenten het systeem naar eigen keuze kunnen inrichten, omdat iedere cursus andere leerdoelen heeft. Ten slotte is het belangrijk om aanvullende instructie te geven, om zo beginnende programmeurs bewust te maken van het belang van code van goede kwaliteit.

REFERENTIES

1 *Students' misconceptions and other difficulties in introductory programming: A literature review. Yizhou Qian and James Lehman. ACM Transactions on Computing Education, 2017. https://doi.org/10.1145/3077618*

2 *Automated feedback for learning code refactoring. Hieke Keuning. Proefschrift, 2020. http://www.hkeuning.nl/PhD%20Thesis%20Hieke%20Keuning.pdf*

3 *A systematic literature review of automated feedback generation for programming exercises. Hieke Keuning, Johan Jeuring and Bastiaan Heeren. ACM Transactions on Computing Education, 2018. https://dl.acm.org/authorize?N676758*

4 *The behavior of tutoring systems. Kurt VanLehn. International Journal of Artificial Intelligence in education, 2006.*

5 *Ideas software framework. http://ideas.cs.uu.nl*