Fakulteten för teknik och samhälle
Datavetenskap

Examensarbete
15 högskolepoäng, grundnivå

Bedömningssystem

Assesmentsystem

Felix Alhbin
Mattias Pernhult

Examen: Kandidatexamen 180 hp
Huvudområde: Datavetenskap
Program: Systemutvecklare
Datum för slutseminarium: 2016-05-30

Handledare: Ulrik Eklund
Andrabedömare: John Doe

Sammanfattning

Text på svenska...

Abstract

Text in English...

UEk ►*Det är bara en kommentar för att se att det funkar*◄

Innehåll

1 Inledning

1.1 Bakgrund

Ett sätt att lära sig att programmera är genom introduktionskurser på universitetet. I sådana kurser ges studenter olika uppgifter som kan hjälpa studenten att bli familjär med moderna programmeringsspråk, lära sig viktiga verktyg och ge insikter om hur systemutveckling bedrivs []. Utvärdering och bedömning av dessa programmeringsuppgifter utförs vanligtvis manuellt av lärare och övningsassistenter. När kurser utförs på detta traditionella sätt och de innehåller ett högt antal studenter skapar det tunga och tidskrävande uppgifter för lärare och övningsassistenter vilket tär på universitetets resurser. Det innebär även att återkopplingen till studenterna fördröjs vilket har en negativ påverkan på studenternas utveckling jämfört med direkt återkoppling [1]. Eftersom programmeringsuppgifters potentiella lösningar är av likartad karaktär kan utvärdering och bedömning automatiseras genom automatiserade bedömningssystem. Därav kan lärarens resurser användas på ett mer effektivt/givande sätt och på så vis förbättra kvalitén i kursens andra moment samtidigt som det ger en objektiv bedömning och direkt återkoppling till studenterna []. Automatiserade bedömningssystem för programmeringsuppgifter nämns redan första gången 1960 av Hollingsworth [2]. Studenterna använde sig av hålkort med program skrivna i assembly som de skickade in för bedömning. Sedan dess har automatiserade bedömningssystem vidareutvecklats.

1.2 Relaterat arbete

UEk ►*Det finns två principiella sätt att strukturera relaterade artiklar. Aningen per papper (som ni gör här) eller per ämne som är relevant. Oavsett så tycker kjag att ni borde skriva om hur ni har fått fram dem (sökning i vilka databaser och vilka söktermer som ni ansett vara relevanta).*◄

Daly och Horgan [4] presenterar i sin artikel hur de utvecklade det automatiserade lärningssystemet RoboProf [4]. I artikeln utförde de även en studie där de testade att använda RoboProf i en introduktionskurs för programmering. Resultaten från studien visade att de studenter som utförde kursen på ett traditionellt vis presterade betydligt sämre än de studenter som använde RoboProf. Vidare presenterar författarna även en lösning för plagieringsproblemet vilket bygger på en idé av Plauger [5].

I artikeln [] presenterar Edwards och Pe?rez-Quin?ones deras erfarenheter av att använda testdriven utveckling med Web-CAT []. Web-CAT är unikt bland automatiserade bedömningssystem eftersom det låter lärare bedöma hur bra studenterna testar deras egen kod istället för att studenternas kod testas gentemot fördefinierade testfall [].

Higgins m. fl. [3] påpekar i sin studie om det automatiserade bedömningssystemet CourseMarker [12] att studenter ofta vill ha väldigt detaljerad information om vilken del av deras kod som inte godkändes. Författarna hävdar dock att alltför detaljerad information i återkopplingen kan ha negativ påverkan på studenters resultat. CourseMarker löser problemet genom att låta läraren själv ange hur detaljerad återkopplingen till studenten ska vara [3] [7][11].

En alternativ lösning presenteras av Reek som begränsar antalet försök en student har på sig att genomföra en uppgift. Detta för att tvinga studenten att tänka över deras lösning

mer noggrant innan de lämnar in den [6]. Denna lösning presenteras även av Ihantola m. fl. [].

Caiza och Del Alamo tar även de upp problem kring återkoppling till studenterna [7]. De nämner trial and error problemet vilket betyder att studenter prövar sig fram genom att göra mindre förändringar i deras program istället för att reflektera över vad problemet beror på. Författarna påstår att de inte upplevt *trial and error* beteendet bland de medverkande studenterna.

Ihantola m. fl. [Ihantola] lyfter fram olika lösningar för att förhindra *trial and error* beteendet. En alternativ lösning som de presenterar är att begränsa mängden återkoppling till studenterna. Dock kan detta frambringa förvirring hos studenterna eftersom de inte förstår varför deras program bedömdes som inkorrekt [26]. Vidare nämner författarna en lösning där studenter får en tidsbestraffning vid ett misslyckande, de påpekar även att tidsbestraffningen kan öka exponentiellt. De presenterar även en lösning där studenten får en slumpmässigt utvald uppgift vid varje nytt försök vilket förhindrar *trial and error* beteendet.

Hollingsworth påpekar att studenters program kan innehålla skadlig kod vilket kan förstöra bedömningssystemet, studenterna utgör därmed ett hot mot systemet [2]. Detta problem adresseras av artiklarna [13][8][Itanhola] där författarna presenterar två alternativa lösningar som bygger på grundtanken att kompilera och exekvera studenternas program i en säker miljö för att skydda bedömningssystemet. I artikeln [8] presenterar Malan bedömningssystemet CS-50 vilket utvecklades som en del av en internetbaserad kurs vid Harvard Universitetet. För att säkert kompilera och exekvera opålitlig kod använder CS-50 SELinux [9] och PAM begränsningar [10]. ?pa?ek m. fl. [13] hävdar i sin studie att flertalet av dagens bedömningssystem saknar stöd för exekvering av opålitliga program i en säker miljö. Författarna har i sin lösning valt att använda den virtuella containerbaserade plattformen Docker [14] för att kompilera och exekvera studenternas program. APAC verifierar studenternas program genom att analysera och validera programmets utdata.

1.3 Syfte

UEk ►*Jag hade flyttat syfte och frågeställning till slutet av inledningskapitlet. Då blir det också mer förståeligt till varför just de relaterade arbetena är relaterade.*◄

Studien ämnar att implementera ett bedömningssystem som använder Docker för att exekvera studenternas program i en säker miljö samt bedömmer programmen genom fördefinierade testfall tillhandahållna av en lärare. Bedömningssystemet är tänkt att användas för introduktionskurser i programmering som bedrivs på Malmö Högskola. Syftet är dels att undersöka om vårt bedömningssystem kan underlätta och ge stöd åt lärare i deras bedömningsprocess av programmeringsuppgifter samt undersöka om vårt bedömningssystem är ett bra hjälpmedel för att utveckla studenters programmeringskunskaper. UEk ►*Om ni har dels för de första syftet måste ni även inleda det andra syftet med dels. Annars skriv att ni har två syften: 1) och 2)*◄

1.4 Frågeställning

Studien avser ge svar på följande frågor:

1. Kan vårt bedömningssystem minska tidsåtgången för lärares bedömningsprocess av programmeringsuppgifter?
2. Kan vårt bedömningssystem hjälpa studenterna att utveckla deras programmeringskunskaper?

2 Metod

Följande avsnitt beskriver den metodik som valts för utförandet av studien.

2.1 Metodbeskrivning och metoddiskussion

Eftersom studiens syfte är att konstruera, implementera och utvärdera en systemartefakt valdes en metodik som presenteras av Nunamaker m. fl. [Nunamaker] eftersom det är en erkänd och välbeprövad process för forskning inom informationssystem [källa]. Författarna definierar en iterativ femstegsprocess för hur sådan forskning bör bedrivas vilken innefattar att: *(1) konstruera ett konceptuellt ramverk (2) utveckla en systemarkitektur (3) konstruera och analysera systemet (4) implementera systemet* och *(5) utvärdera systemet*. För att ytterligare säkerställa studiens kvalité ämnar den att följa de sju riktlinjer för forskning inom informationssystem definierade av Hevner m.fl. [Hevner].

2.1.1 Konstruera ett konceptuellt ramverk

Syftet med att konstruera ett konceptuellt ramverk är att definiera en relevant forskningsfråga och systemkrav. För att identifiera och definiera dessa kommer vi att genomföra en litteraturstudie av liknande system och utföra intervjuer med lärare och studenter vid Malmö högskola. Vi har valt ut tre lärare som alla har lång, dokumenterad erfarenhet av att lära ut programmering till högskolestudenter i introduktionskurser. Valet av intervjupersoner anser vi vara befogat eftersom personerna i fråga är representativa för den huvudsakliga målgruppen och verksamma inom den miljö som systemet ska tillämpas. Därav kommer vi få en djupare förståelse i ämnet vilket hjälper oss att omsätta den insamlade informationen till krav för systemet samt definiera forskningsfrågan.

Anledningen till att vi inte väljer en kvantitativ undersökningsmetod som exempelvis enkäter med fördefinierade svar beror på att de endast ger en ytlig förståelse i ämnet till skillnad från intervjuer [seminarieboken]. UEk ►*Ni behöver inte argumentera för/emot metoder ni inte använder*◄ Vid intervjuer finns det även utrymme för att ställa följdfrågor till respondenten samt att det går att förtydliga frågorna vid eventuella missuppfattningar, något som inte är möjligt vid fördefinierade enkätundersökningar [seminarieboken]. Informationen som vi samlar in från litteraturstudien kommer att användas för att urskilja de befintliga systemen. Därigenom kan vi identifiera funktioner som vi anser saknas i dessa system vilket gör att vi, tillsammans med informationen vi erhåller från intervjuerna, kan definiera krav som systemets ska uppfylla.

2.1.2 Utveckla en systemarkitektur

Enligt Nunamaker m. fl. innefattar följande steg att bestämma systemets komponenter och definiera förhållandet mellan komponenterna samt att identifiera och definiera systemets funktionalitet []. Vi kommer att använda den insamlade information från föregående steg för att definiera komponenterna och systemfunktionerna. Nunamaker m.fl. påpekar betydelsen av att definiera mätbar funktioner för systemet eftersom det underlättar vid systemutvärderingen [].

2.1.3 Konstruera och analysera systemet

I detta steg är syftet att vi ska att identifiera möjliga lösningar, tekniker och språk för att kunna implementera systemet i kommande steg []. Möjliga lösningar kommer att konstrueras vilka vi sedan analyserar innan vi slutligen väljer det bästa alternativet att gå vidare med och implementera.

2.1.4 Implementera systemet

Tanken med följande steg är att implementera en prototyp som ska utvärderas i kommande steg. Prototypen kommer att ge insikter angående konceptets genomförbarhet och dess möjlighet att besvara frågeställningen.

2.1.5 Utvärdera systemet

Hevner presenterar i sin tredje riktlinje *Design Evaluation* olika metoder för utvärdering av informationssystem. Den metod som vi anser är lämpligast för vår studie är en experimentell utvärderingmetod, mer specifikt kontrollerade experiment, eftersom det innebär att studera en implementerad prototyp i en kontrollerad miljö. Vi kommer att utföra experiment på lärare och studenter vilka är de två huvudgrupperna i studien. Målet med experimenten för lärare är att besvara frågeställningen *Kan vårt bedömningssystem minska tidsåtgången för lärarens bedömningsprocess av programmeringsuppgifter?*, medans experimenten för studenter har som mål att besvara *Kan vårt bedömningssystem hjälpa studenterna att utveckla deras programmeringskunskaper?*.

Vi hade kunnat använda en analytisk utvärderingsmetod där artefakten studeras analytiskt eftersom det saknas en fungerande prototyp, men eftersom vi kommer ha en fungerande prototyp anser vi att denna utvärderingsmetod inte lämpar sig för vår studie.

Hevner presenterar även observationsbaserade utvärderingmetoder som fallstudier där artefakten studeras i den tilltänkta miljön under en viss tidsperiod. Denna utvärderingsmetod hade varit ett bra val eftersom vi då hade kunnat observera hur systemet fungerar i den tilltänkta miljön. Men eftersom studien skrivs under vårterminen finns det inga pågående introduktionskurser i programmering vid Malmö Högskola vilket innebär att en fallstudie ej är genomförbar. Därav anser vi att den lämpligaste utvärderingsmetoden är kontrollerade experiment.

2.2 Intervjuer

Målet med de intervjuer vi tänker genomföra är att samla in information om lärarnas åsikter och erfarenheter inom undervisning av introduktionskurser i programmering. Vi vill skapa oss en uppfattning om hur mycket av lärarnas resurser som läggs på att rätta programmeringsuppgifter, deras åsikter om automatiska bedömningssystem för programmeringsuppgifter samt undersöka om lärarna upplever plagiering bland studenterna som ett problem och i vilken omfattning som det sker. Vi vill dessutom ta reda på lärarnas ställningstagande till att använda system som upptäcker plagiering av källkod och även deras åsikter om att använda versionshanteringssystem i introduktionskurser. Vi kommer att ställa frågor till lärarna om hur detaljerad återkopplingen till studenterna bör vara och om de tror att ett automatiskt bedömningssystem är ett bra stöd i studenternas utveckling.

2.3 Kontrollerade experiment

OBS!! Detta är bara nedskrivna tankar för att visa hur vi tänker

- Genomföra experimenten på studenter och som har läst introduktion till programmering
- Antal är inget som vi bestämt än men 10 känns rimligt, ungefär 3 personer för de 3 olika grupperna som vi tänker (en grupp som bara använder testfall, en grupp som använder både testfall och quiz, och en grupp där studenterna får pröva både med enbart testfall och med testfall och quiz)
- vid intervjuerna har samtliga lärare påpekat att flervalsfrågor i samband med uppgifter hade varit önskvärt för att kunna testa studentens huvudsakliga kunskaper, vi har därför implementerat ett quiz som studenten måste klara innan de kan skicka in sin uppgift
- Det vi tänker att vi vill ta reda på är hur studenter presterar beroende på om de har ett quiz innan eller inte, har de som gör quizzen innan färre försök innan deras uppgift godkänns?
- Vi tänker att det kanske är onödigt att köra kontrollerade experiment på lärare eftersom vi får nog inte den typ av data som vi vill ha. Vi vill ta reda på om vårt system kommer minska tiden för lärare att bedöma programmeringsuppgifter, men kommer vi få svar på det? Känns som att man måste testa det under en kurs för att kunna svara på den frågan på ett bra sätt Det vi kan ta reda på är ungefär hur lång tid som det tar att skriva testfall för en uppgift och sedan jämföra det med hur lång tid det tar för lärare och handledare att rätta uppgifter. Är på det sättet som vi kan svara på frågan. Och detta kanske vi skulle kunna göra själva? Det vill säga någon typ av analytisk studie/experiment.

3 Resultat

Här presenteras de resultat vi samlat in från de specificerade aktiviteterna i Nunamakers process.

3.1 Intervjuer

Vi genomförde tre intervjuer med Farid Naisan, Rolf Axelsson och Jonas Wahlfrid vilka alla är anställda som lärare vid Malmö Högskola. Resultaten från intervjuerna presenteras enligt de ämnesområden som intervjufrågorna berörde.

3.1.1 Resurser

En fråga som vi ville få besvarad av respondenterna var hur mycket resurser som läggs på att rätta programmeringsuppgifter. Från intervjuerna framgick det att olika faktorer har inverkan på resursanvändningen beroende på om kursen är platsbelagd eller distansbelagd, antalet studenter och storleken på uppgiften samt kunskapsnivån för studenten som bedöms. Respondenterna hade svårt att ge en exakt siffra för hur mycket tid som krävs för bedömning av en uppgift.

Farid uppskattade att bedömningen för en uppgift kräver mellan 20-60 minuter per student, dock påpekade han att det är väldigt beroende på uppgiftens storlek. Enligt Farid har han uppskattningsvis 300 studenter per termin i de distanskurser som han är ansvarig för.

Rolf uppskattade att bedömingen av uppgifter tar 180 minuter per student för en kurs på 15 högskolepoäng.

Jonas gav ingen exakt siffra på hur mycket tid som bedömningen av uppgifter tar men han påpekade att det inte finns tillräckligt med resurser för bedöma samtliga studenters uppgifter. Han poängterade dessutom att restredovisningar av uppgifter är resurskrävande. Jonas berättade att de prövat att använda ett upplägg där de endast kollade närvaron på labbtillfällen i förhoppningen om att studenterna skulle använda tiden till att arbeta. Han berättade även att studenterna genomskådade ett annat upplägg där enbart vissa uppgifter bedömdes.

Gemensamt för lärarna är att de delegerar bedömningen av studenternas uppgifter till lärarassistenter.

3.1.2 Automatiserade bedömningssystem

Respondenterna fick besvara frågan på vad de anser om att använda automatiserade bedömningssystem för programmeringsuppgifter. Både Farid och Rolf ställer sig negativa till användningen av dessa system och anser att de ej lämpar sig för bedömning av programmeringsuppgifter eftersom det enligt dem är svårt att bedöma om studenten har uppnått tillräcklig förståelse.

Farid påpekade att ett problem kan ha flera alternativa lösningar och han menar på att det inte är som vid en matematisk uppgift där det endast finns ett korrekt svar och därigenom är det svårt att göra en bedömning av inlämningens kvalité genom automatiserade system. Detta var något som även Rolf ansåg då han påpekade att det är möjligt att även en korrekt löst uppgift kan vara dåligt implementerad.

Jonas påpekade att det är viktigt att hålla studenterna sysselsatta men enligt honom ligger svårigheten i att balansera antalet uppgifter som ges ut till studenterna mot de resurser som finns till förfogande för bedömning. Därför skiljer sig Jonas åsikter från de övrigas då han anser att ett bedömningssystem kan vara ett bra sätt att bedöma programmeringsuppgifter eftersom det då är möjligt att hålla studenterna kontinuerligt sysselsatta med begränsade resurser. Men han menar dock att vid obegränsade resurser vore en mänsklig bedömning det optimala sättet att bedöma programmeringsuppgifter.

Något som samtliga respondenter lyfter fram som en uppskattad funktion är möjligheten att använda flervalsfrågor för att verifiera studenternas kunskaper.

3.1.3 Återkoppling

Under intervjun ställdes även frågor om respondenternas åsikter angående återkoppling till studenterna, frågorna berörde synpunkter kring detaljrikedom och tidsaspekt. Gemensamt för respondenterna är att de anser att återkopplingen till studenter bör ske utan fördröjning då det enligt dem är bra för studenterna att få direkt återkoppling. Angående detaljrikedomen i återkopplingen menar Jonas att studenterna bör få ta del av samma utdata som ett testramverk ger för testfall. Både Farid och Rolf delar Jonas åsikter där de även påpekar att återkoppling är en viktig del av studenternas programmeringsutveckling.

3.1.4 Plagiering

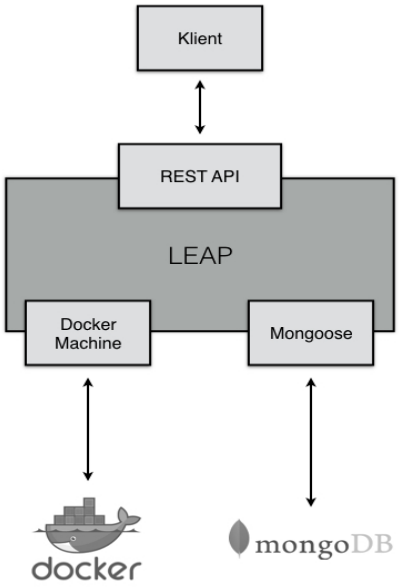
Plagiering är något som samtliga av de intervjuade anser vara vanligt förekommande bland studenter. Men de anser att ett verktyg för att upptäcka plagiering är svårt att använda i praktiken på grund av flera anledningar, dels uppmuntrar lärarna studenterna att samarbeta till en viss grad och samtidigt är en del av uppgifterna utformade på ett sätt som gör det svårt att avgöra om en lösning är plagierad. De menar att mindre uppgifter har färre potentiella lösningar vilket innebär att studenternas lösningar blir svåra att särskilja. Jonas påpekar att studenter som plagierar ofta fångas upp i andra moment som exempelvis vid redovisningar av inlämningar och tentamina.

3.1.5 Inlämningsmetod

Vi ville ta reda på hur respondenterna anser att inlämningen av studenternas lösningar ska ske, antingen via ett versionshanteringssystem som Git eller med hjälp av en webbapplikation. Samtliga respondenter var överens om att ett versionshanteringssystem är viktigt för en utvecklare att behärska. Men två av de intervjuade anser att versionshanteringssystem inte lämpar sig för användning i en introduktionskurs till programmering eftersom fokus ska ligga på att lära studenterna grunderna i programmering och inte vara ett onödigt hinder i deras utveckling.

3.2 Systembeskrivning

Detta är bara ett test.



Figur 1: Systemarkitektur för LEAP

3.3 Kontrollerade experiment

Detta är bara ett test

4 Analys

5 Diskussion

6 Slutsatser och vidare forskning

Referenser

[1] H. N. Djidjev, A. Lingas, J.-R. Sack, An $O(n \log n)$ algorithm for computing the rectilinear link center of a simple polygon. *Discrete and Computational Geometry* 8:131–152, 1992.

[2] Y. Ke, An efficient algorithm for link distance problems. In *Proceedings of the 5th ACM Symposium on Computational Geometry*, pages 69–78, 1989.

[3] W. Lenhart, R. Pollack, J.-R. Sack, R. Seidel, M. Sharir, S. Suri, G. Toussaint, S. Whitesides, C. Yap, Computing the link center of a simple polygon. *Discrete and Computational Geometry* 3:281–293, 1988.