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...

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

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

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.

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]. 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

Syftet med genomförandet av kontrollerade experiment är att utvärdera den prototyp som vi utvecklar för att sedan ha möjlighet att besvara vår forskningsfråga *Kan vårt bedömningssystem hjälpa studenterna att utveckla deras programmeringskunskaper*. Experimenten kommer att genomföras på studenter vid Malmö Högskola som har läst minst en introduktionskurs i programmering. För att komma i kontakt med studenter som vill genomföra experimentet kommer vi att kontakta elever som studerar datavetenskap antingen via epost alternativt presentera oss vid någon av deras föreläsningar. Vid experimenten kommer studenten att först läsa en text som presenterar oss och vårt examensarbete följt av övergripande information om hur experimentet kommer att utföras. I texten beskrivs även hur den insamlade informationen kommer att behandlas. Därefter får studenten besvara en enkät med inledande frågor[1]. Efter att enkäten besvarats kommer studenten att få instruktioner om hur användartestet kommer att genomföras följt av att studenten genomför användartestet. Testet går ut på att testa vår prototyp med en existerande uppgift som vi tagit från kursen *Introduktion till programmering* vid Malmö Högskola. Till denna uppgift har vi skrivit testfall som vi sedan har laddat upp till vår prototyp, det är dessa testfall som studenternas program sedan kommer att köras mot. När användartestet har genomförts kommer studenten att få besvara ytterligare en enkät[2]. Syftet med enkäten är att få en bra bild av hur studenten upplevde att använda vår prototyp.

Vid genomförandet av användartestet och den andra enkäten kommer vi ej närvara i rummet eller ta tid, detta för att undvika att skapa onödig press på studenten. Vi tycker även att det är viktigt att poängtera för studenten att all information som vi samlar in är helt anonym för att studentens åsikt ska vara sanningsenlig.

[1] Appendix A
[2] Appendix B

3 Resultat

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

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 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.

3.1.4 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.

3.1.5 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.

3.2 LEAP

3.2.1 Systemfunktioner

Informationen som införskaffades genom litteraturstudien och intervjuer med lärare låg till grund för LEAP, *LEarning and Assessment of Programming*, vilket är namnet på den prototyp som vi utvecklat. För att följa Nunamakers metod började vi med att utföra de två första stegen i metoden vilket innefattar att definiera funktioner och krav för prototypen. För att genomföra detta använde vi oss av den insamlade informationen och definierade följande funktioner och krav som vår prototyp ska uppfylla:

- En lärare ska genom ett webbgränssnitt kunna skapa en uppgift i systemet och ladda upp en tillhörande testfil med testfall.
- En lärare ska ha möjlighet att skapa ett tillhörande quiz till en uppgift.
- En lärare ska kunna få information om hur många som klarat en uppgift.
- En student ska genom ett webbgränssnitt kunna skicka in svar på en uppgift som sedan körs mot uppgiftens testfil.
- En student ska få information om lösningen av uppgiften godkändes eller ej.
- En student ska få ta del av den genererade utdatan från testfallen vid ett misslyckande.
- En student ska kunna se vilka uppgifter den klarat och ej klarat.
- Systemet ska stödja uppgifter definierade för Java programmering.
- Systemet ska exekvera studentens program i en säker miljö som skyddar servern.
- Systemet ska kunna hantera program med oändliga loopar.
- Systemet ska inte tillåta studentens program att utföra I/O operationer utanför den säkra miljön.

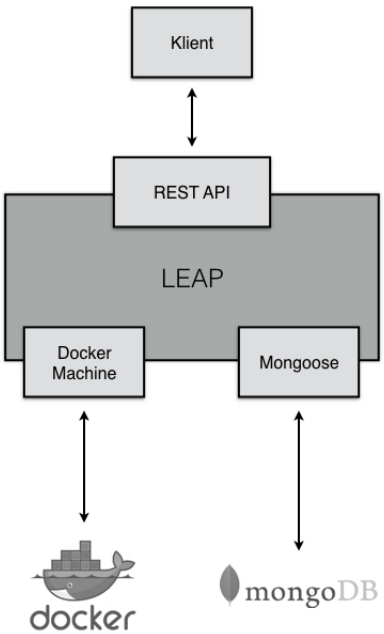
3.2.2 Systemkomponenter

Det andra steget i Nunamakers metod är att utveckla en systemarkitektur vilket innebär att definiera de olika systemkomponenterna för prototypen. De komponenter som vi anser behövs för att utveckla vår prototyp är en databas, en klient, en server, ett API-gränsnitt och en säker miljö samt en komponent för hantering av användare. Servern är nödvändig för att hantera kommunikationen med databasen och för att klienten ska kunna kommunicera med servern behövs ett API-gränsnitt. Klienten behövs för att studenter och lärare ska kunna utföra de ovannämnda funktionerna. För att lösa problemet med att studenternas program kan vara skadlig och därmed utgöra ett hot mot systemet anser vi att någon typ av komponent som kan exekvera programmen i en säker miljö är väsentlig. Komponenten för att hantera användare behövs för att servern ska kunna urskilja alla användare och deras uppgifter.

I tredje steget i Nunamakers metod identifierade vi olika lösningar, språk och tekniker som vi skulle kunna använda för att implementera vår prototyp. Vi analyserade och diskuterade olika lösningar och kom efter noga övervägande fram till att vår prototyp LEAP ska använda sig av följande tekniker och språk.

LEAP består i grunden av en HTTP-server utvecklad med Node.js [källa] som är en open source körningsmiljö byggd på Googles JavaScript-motor V8. Node.js har en händelsestyrd arkitektur som gör det möjligt att hantera asynkron I/O. För att interagera med servern har en webbaserad klient utvecklats som kommunicerar med servern via ett REST-API vilket har utvecklats med ramverket Express [källa]. Klienten är implementerad med ramverken Bootstrap [källa] och AngularJS [källa]. Bootstrap innehåller en mängd färdigutvecklade komponenter för utveckling av webbapplikationer. AngularJS används för utveckling av webbapplikationer och främjar utveckling av klienter som följer MVC arkitekturer. Datalagringen görs med hjälp av MongoDB som är en NoSQL-klassificerad dokumentdatabas. För att kommunicera med databasen från servern används Node.js modulen Mongoose. Tillsammans utgör MongoDB, Express, AngularJS och Node.js MEAN-stacken [källa]. För att hantera och skilja på de olika användare samt tillåta de att logga in använder LEAP sig av Googles API med OAuth 2.0.

LEAP består av ytterligare en komponent vilket är Docker, detta för att sätta upp en säker miljö där studenternas program exekveras med hjälp av Docker containrar. Samtliga komponenter presenteras visuellt i figur 1.



Figur 1: Systemarkitektur för LEAP

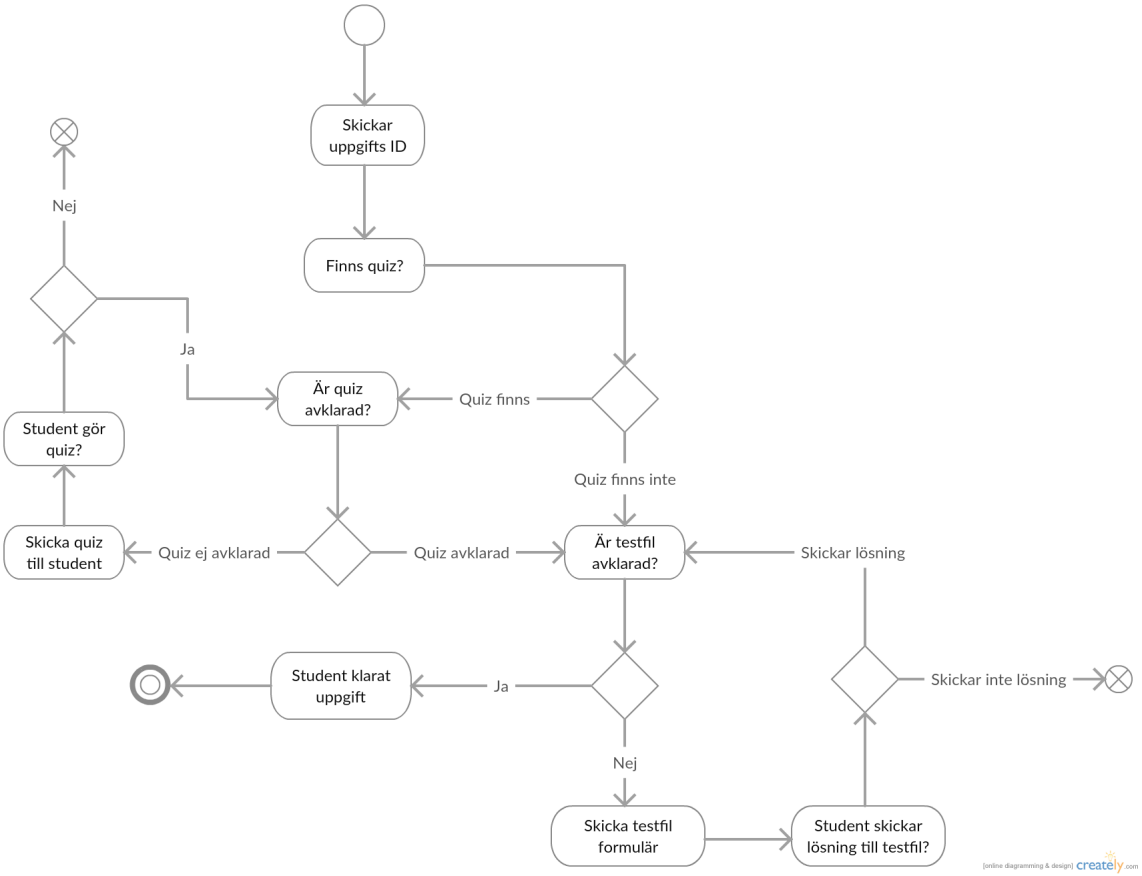
3.2.3 Förklarande av systemet vid uppladdning

När en lärare ska skapa en uppgift och ladda upp en tillhörande testfil hanteras detta av LEAP i följande steg:

1. Läraren loggar in i systemet genom webbklienten och skapar sedan en uppgift i en kurs och ger den ett namn.
2. Läraren väljer en komprimerad fil innehållandes en testfil och en tillhörande programfil.
3. Läraren får välja om den vill skapa ett tillhörande quiz till uppgiften och kan då skapa frågorna och svaren direkt i webbklienten.
4. Servern tar emot en förfrågan om att skapa en ny uppgift och verifierar först att det inte redan existerar en uppgift med det angivna namnet.
5. Servern skapar en unik mapp för att spara den uppladdade filen.
6. Servern packeterar upp den komprimerade filen och flyttar programfilerna samt testfilen till den unika mappen.
7. Servern skapar en Docker container och delar den unika mappen med containern för att container ska kunna exekvera lärarens testfil mot programmet.
8. Docker container startar exekveringen av testerna.
9. Servern letar i den delade unika mappen i containern efter en fil med namnet *output.txt* medans exekvering utförs.
10. När exekvering i container är klar skrivs resultat från testerna till filen *output.txt*.
11. Servern kommer då att hämta innehållet i *output.txt*.
12. Servern tolkar sedan innehållet i *output.txt*.
13. Om lärarens programfiler klarade testen sparas uppgiften i databasen och ges då ett unikt ID. Detta ID skickas sedan tillbaka till webbklienten och läraren kan då spara det och hänvisa sina studenter till detta. Om programfilerna inte klarar testen skickar servern tillbaka ett svar om att uppladdningen misslyckades tillsammans med ett felmeddelande om vad som gick fel.

3.2.4 Förklarande av systemet vid inlämning

För att underlätta beskrivningen av hur LEAP hanterar en inlämning av en student valde vi att använda oss av ett aktivitetsdiagram. Figur 2 nedanför visar flödet för när en student ska genomföra en uppgift.



Figur 2: Aktivitetsdiagram över inlämningsprocessen

Figuren visar hur genomförandet av en uppgift går till vilket kan involvera ett quiz som studenten får genomföra. En uppgift kommer dock alltid att innehålla en testfil som testar studenternas program. När LEAP testar studenternas program sker det i en Docker container för att skydda servern mot eventuell skadlig kod, exakt hur detta går till beskrivs nedanför.

1. Servern tar emot en förfrågan med uppgiftens ID och en komprimerad fil innehållande studentens programfiler.

2. Servern skapar en unik mapp för att spara studentens uppskickade filer och uppgiftens tillhörande testfil skapad av läraren.

3. Servern packeterar upp den komprimerade filen och flyttar programfilerna till den unika mappen.

4. Servern hämtar den tillhörande testfilen från databasen genom uppgiftens ID.
5. Servern flyttar testfilen till den unika mappen.
6. Servern skapar en Docker container och delar den unika mappen med containern för att container ska kunna exekvera testerna mot studentens program.
7. Docker container startar exekveringen av testerna.
8. Servern letar i den delade unika mappen i containern efter en fil med namnet *output.txt* medans exekvering utförs.
9. När exekvering i container är klar skrivs resultat från testerna till filen ?output.txt?.
10. Servern kommer då att hämta innehållet i ?output.txt?.
11. Servern tolkar sedan innehållet i ?output.txt?.
12. Beroende på om studentens program klarade alla tester eller ej skickar servern antingen ett svar om att testerna gick igenom alternativt att de ej gick igenom tillsammans med ett felmeddelande om vad som inte godkändes.

3.3 Kontrollerade experiment

Här presenteras resultaten från de kontrollerade experimenten som vi utförde på studenter vid Malmö Högskola. Vi kommer att sammanfatta och lyfta fram de åsikter som vi anser är relevanta för vår studie, dock finns samtliga åsikter och svar från den andra enkäten i Appendix C. I tabell 1 ges en översikt över testpersonerna i form av ålder, kön och vilket program som de läser.

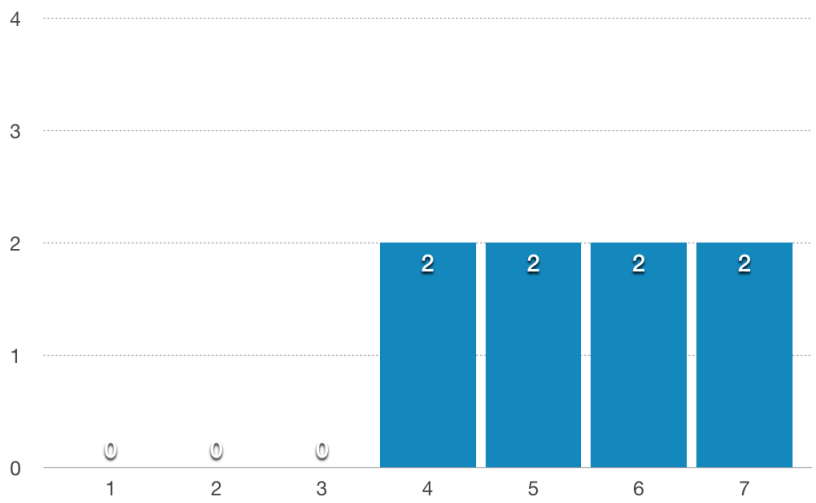
Tabell 1: Överblick av testpersonerna

Testperson	Ålder	Kön	Program	Termin
1	26	Man	Datavetenskap och applikationsutveckling	Andra
2	28	Man	Systemutvecklare	Sjätte
3	32	Kvinna	Systemutvecklare	Fjärde
4	22	Man	Informationsarkitekt	Sjätte
5	23	Man	Systemutvecklare	Sjätte
6	23	Man	Systemutvecklare	Fjärde
7	39	Kvinna	Systemutvecklare	Sjätte
8	39	Man	Spelutveckling	Fjärde

Av de åtta testpersonerna är två stycken kvinnor och fem av deltagarna läser till Systemutvecklare vid Malmö Högskola.

3.3.1 Användarupplevelse

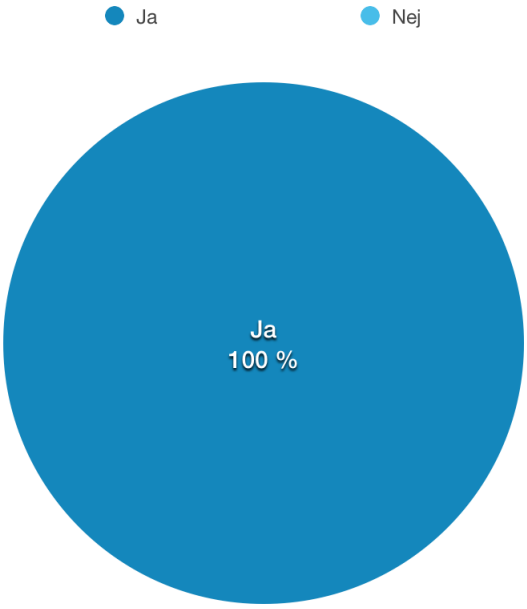
Efter att testpersonerna utfört användartestet fick de ge sina synpunkter på användarupplevelsen. I figur 3 presenteras resultatet på frågan *Hur skulle du bedöma den allmänna användarupplevelsen (UX) av vår prototyp?*.



Figur 3: Allmäna användarupplevelsen av prototypen

Som diagrammet visar är åsikterna från testpersonerna lite olika speciellt när det kommer till detaljrikedomen i återkopplingen. Tre personer upplevde att återkopplingen tydligt visar vad felet i programmet är. Dock upplevde tre personer att återkopplingen som vår prototyp gav var svårtydlig och svårförstådd. De flesta av testpersonerna upplevde att vår prototyp var enkel att använda eftersom navigeringen var tydlig och att det var tydligt hur man skickade in en uppgift för bedömning. Vi hade även en fråga där vi ville att testpersonerna skulle nämna förbättringar för vår prototyp. Där nämner samtliga att återkopplingen kan bli bättre på olika sätt. De nämner dels att den kan bli tydligare och mer strukturerad för att det ska vara lättare att veta vad felet i programmet är men även att det hade varit bra att få tips eller exempel på hur felen kan lösas.

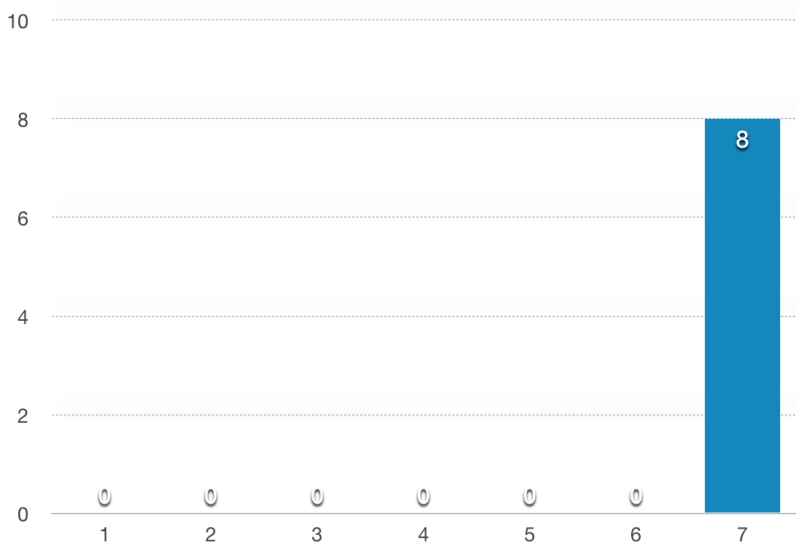
Vi ställde även en fråga om testpersonerna tror att det skulle vara användbart att använda vår prototyp vid programmeringskurser. Figur 4 visar att samtliga studenter ställer sig positivt till att använda vår prototyp vid programmeringskurser. De flesta av testpersoner menar att direkt återkoppling är anledningen till att de anser att vår prototyp kan användas vid programmeringskurser. En av testpersoner anser dock att återkopplingen som ges från en lärare eller en labbhandledare är mer givande. Två av testpersonerna påpekar att med vår prototyp skulle studenterna bli mer självständiga. En av dem menar att vid laborationer finns det inte alltid utrymme att få sina uppgifter bedömda eftersom antalet studenter är markant fler än antalet lärare och labbhandledare. Vidare menar testpersonen att det finns få redovisningstider vilket medför att studenter har svårt att få sina uppgifter redovisade och bedömda samt att stressen för lärare att bedöma uppgifterna minskar med användningen av vår prototyp.



Figur 4: Använda vår prototyp vid programmeringskurser

3.3.2 Återkoppling

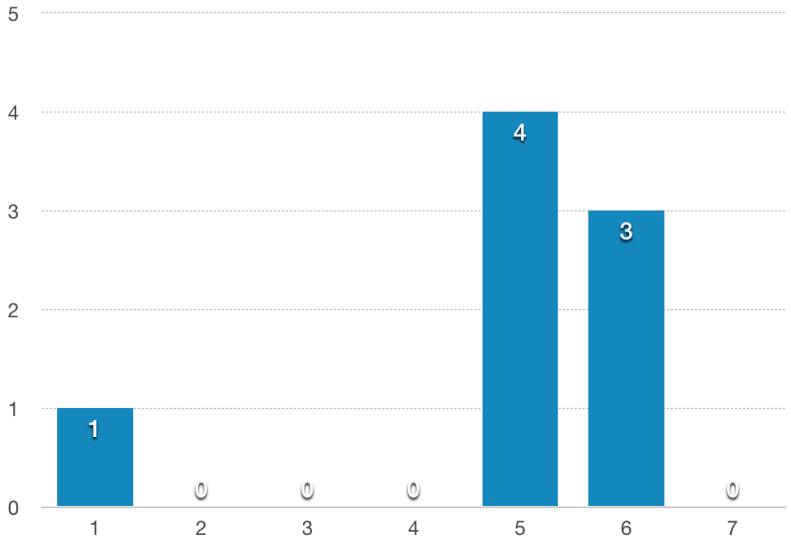
Vi ställde även frågor till testpersonerna om hur de upplevde att få direkt återkoppling och hur de upplevde detaljrikedomen i återkopplingen. I figur 5 presenteras hur testpersonerna upplevde att få direkt återkoppling på en skala från ett till sju.



Figur 5: Upplevelsen av direkt återkoppling

Som figur 5 visar upplevde samtliga testpersoner att direkt återkoppling var väldigt bra. De påpekade att det var skönt att slippa vänta på att få en uppgift bedömd eftersom testpersoner tycker att återkoppling ska ges när de är inne i tänket för uppgiften. En av testpersonerna menar att det är ineffektivt med bedömning som involverar en lärare när det gäller direkt återkoppling eftersom det kan dröja upp till två veckor innan en uppgift blir bedömd. En annan testperson menar även att med direkt återkoppling kan missuppfattningar kring en uppgift upptäckas tidigare jämfört med bedömning som involverar en lärare. En av testpersonerna menar även att uppgifter kan bli enklare vid direkt återkoppling, men att det kan lösas genom att sätta en begränsning på antalet inlämningar.

Testpersonerna fick även besvara frågan *Hur upplevde du detaljrikedomen i återkopplingen?* vilket presenteras i figur 6

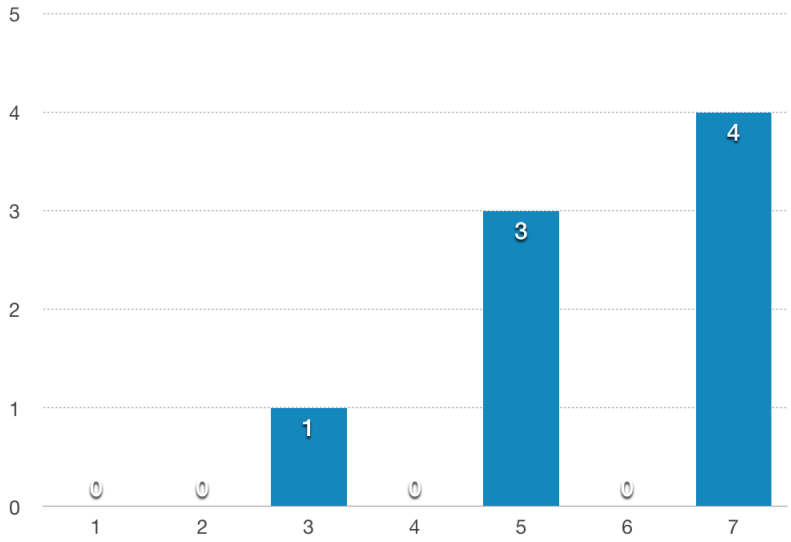


Figur 6: Upplevelsen av detaljrikedomen i återkopplingen

Samtliga testpersoner upplevde att detaljrikedomen i återkopplingen var hög men vissa menar att den var svårtolkad och kunde presenterats och strukturerats på ett enklare och mer konkret sätt. En av testpersonerna menar att prototypens återkoppling inte är lika givande jämfört med återkopplingen från en lärare eftersom en lärare även kan ge en förklaring på varför felet uppstod. De menar även att det kan vara bra att ge tips eller exempel för att lösa felen.

3.3.3 Quiz

Eftersom vår prototyp även gör det möjligt att använda ett quiz vid uppgifterna, ställde vi frågor relaterat till quizzet. I figur 7 visas resultatet på frågan *Hur mycket tror du att ett quiz hjälper studenter i deras programmeringsutveckling och förståelse?*



Figur 7: Betydelsen av quiz för utvecklingen

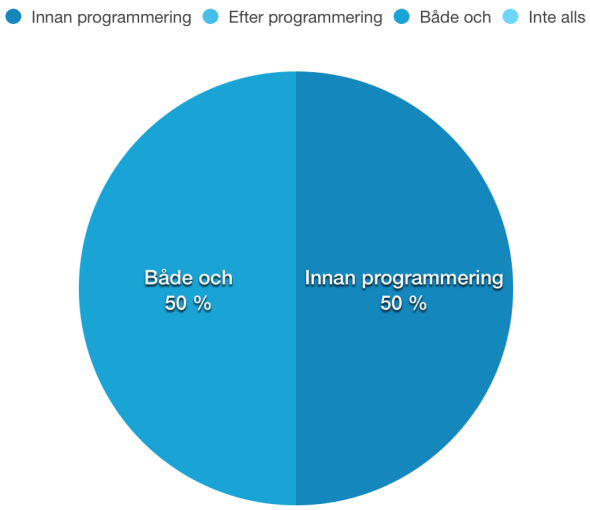
Sju av testpersonerna ställer sig positivt till att använda quiz för att förbättra studenternas förståelse och kunskap. Det är en person som ställer sig negativt till detta. Den personen tror inte att det hjälper studenterna med ett quiz utan menar istället att det är bättre med praktiska moment som programmeringsuppgifter eller laborationer för att utveckla studenternas förståelse och kunskaper. Personen menar att det istället är till mest nytta för läraren då den kan få en bättre bild av studenternas faktiska förståelse.

Två av personerna menar att det är viktigt med teori och begrepp för att förstå helheten och att det utan viss förståelse är svårt att programmera eller ha förståelse för varför lösningen ser ut som den gör.

Vidare ställde vi frågan *När tycker du att ett quiz ska användas?* och där testpersonerna kunde välja mellan (1) *Innan programmeringsuppgift* (2) *Efter programmeringsuppgift* (3) *Både och* (4) *Inte alls*. Resultatet från frågan presenteras i figur 8.

De personer som tyckte att ett quiz bör användas innan programmeringsuppgiften menar att det kommer hjälpa studenten att genomföra programmeringsuppgiften eftersom det kräver att studenten har en viss grad av förståelse innan programmeringen påbörjas. De påpekar även att ett quiz hjälper studenterna att inse inom vilka områden som de behöver förbättra sin förståelse.

De personer som tyckte att ett quiz bör användas både innan och efter programmeringsuppgiften har liknande åsikter om varför de anser att det ska vara innan. Anledningen till att de anser att ett quiz även borde finnas efter en programmeringsuppgift är för att verifiera att studenten har erhållit de förväntade kunskaperna från programmeringsuppgiften.



Figur 8: Användning av quiz

3.3.4 Traditionellt eller automatiserat?

För att kunna jämföra traditionell och automatiserad bedömning ställde vi frågor till testpersonerna där de fick lyfta fram fördelar och nackdelar om respektive sätt följt av att de fick göra en bedömning av vilket sätt de tror är bäst för deras utveckling som programmerare.

Fördelar med automatiserade bedömningssystem som testpersonerna lyfte fram är liknande delar som beskrivits innan som snabbare återkoppling och att studenterna blir mer självständiga. En av testpersonerna påpekar att de studenter som har problem kan få mer hjälp av lärare och labbhandledare eftersom de andra studenterna blir mer självständiga och kräver därav inte lika mycket av lärarens resurser. Personen menar även att med ett automatiserat bedömningssystem kan uppgifterna bli bättre genom åren eftersom ett sådant system kan användas av flera högskolor och på så sätt kan de olika högskolorna samarbeta för att göra uppgifterna bättre för studenterna. En av testpersoner menar även att med automatiserade bedömningssystem kan studenter rätta sina egna fel och undviker därmed att skämmas över sina inskickade program eller kodfiler som de inte är nöjda med.

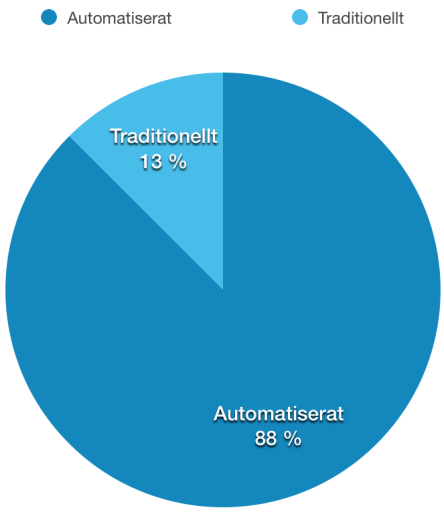
Nackdelar som testpersonerna lyfte fram med automatiserade bedömningssystem är är att återkopplingen kan vara svårtolkad, vilket testpersonerna har nämnt vid tidigare tillfällen. Två av testpersonerna menar att det är svårt att verifiera kodkvalitén i studenternas program. En av dem menar att ett problem går att lösa på många olika sätt och där vissa är bättre än andra. De menar även att det är svårare för ett bedömningssystem än för en lärare eller labbhandledare att ge förbättringsförslag som studenterna kan göra i sina program vilket exempelvis kan förbättra läsbarheten och effektiviteten. En person påpekar att lärare kan frästas av att konstruera uppgifter av enklare karaktär då de medför enklare testning.

Tre av testpersonerna ansåg att det inte finns några fördelar med att använda ett traditionellt bedömningssätt som involverar en lärare. De andra testpersonerna menar att med en lärare eller en labbhandledare kan en mer kvalitativ återkoppling ges. En av

testpersonerna menar att en student då kan få svar på varför något är fel i deras kod istället för vad som är fel. En annan testperson påpekar att läraren kan granska programmet utifrån ett läsbarhetsperspektiv vilket inte ett bedömningssystem kan göra.

Nackdelar som testpersonerna lyfter fram med traditionell bedömning är att det dels är tidskrävande för lärare och labbhandledare men även att det tar tid för studenterna att få återkoppling och bedömning. En av testpersonerna menar att bara för att det är en lärare som bedömer en uppgift behöver inte det automatiskt betyda att återkopplingen kommer hjälpa studenten att genomföra uppgiften. Vidare menar testpersonen att återkopplingen, från en lärare, i en del fall enbart kan uppge att studenten inte klarat uppgiften. En annan testperson är inne på samma spår, där testpersonen menar att de krävs att det finns bra labbhandledare som kan ge korrekt återkoppling, vilket enligt testpersonen inte alltid är fallet.

Vi avslutade med att ställa frågan *Vilket av traditionellt och automatiserat sätt tror du är bäst för din utveckling som programmerare att använda vid programmeringskurser?*. Resultatet på denna fråga presenteras i figur 9.



Figur 9: Traditionellt eller automatiserat

Det är sju av åtta testpersoner som anser att automatiserad bedöming är bättre för deras utveckling som programmerare. Motivering till deras val är fördelarna som nämnts ovanför som snabbare återkoppling och ökad självständighet för studenterna. En testperson menar att ett automatiskt bedömningssystem inte behöver utesluta en handledare eller lärare helt utan mer fungera som ett stöd till dessa. En annan testperson menar att automatiskt och traditionellt sätt uppfyller olika syften. Testpersonen menar att ett traditionellt sätt är bättre vid introduktionskurser eftersom det kan krävas mer hjälp och bättre återkoppling. Vidare menar testpersonen att ett automatiserat system är mer lämpligt att använda i kurser med avancerat innehåll eftersom fokus i dessa kurser är problemlösning av specifika problem istället för utlärning inom ett specifikt ämne vilket är fokus i introduktionskurser.

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.