Testdriven utbildning – strukturerad formativ examination

Författare: Emma Enström, Gunnar Kreitz, Fredrik Niemelä och Viggo Kann

Lärosäte/organisation: Kungliga Tekniska Högskolan, Skolan för datavetenskap och kommunikation

Kontaktuppgifter: emmaen@kth.se, gkreitz@kth.se, niemela@kth.se, viggo@kth.se

Presentationsform: presentation NU2010

Tema 2 Verktyg för lärande

Abstract
KTH har sedan 2005 arbetat med det automatiserade och webbaserade rättningssystemet Kattis som stöd vid undervisning på programmeringskurser. Detta verktyg möjliggör en ny metodik kring laborationer: testdriven utbildning. Vi har sett stora pedagogiska fördelar med metodiken, och tror att andra högskolor kan dra nytta av våra erfarenheter.

Syftet med att införa testdriven utbildning är dels att förbättra kvaliteten på lärartiden och dels att öka tillgängligheten till stöd och feedback för studenterna. Metodiken uppmuntrar också studenterna till fördjupat lärande och erbjuder en mer objektiv och noggrannare bedömning av korrekthet i studenternas lösningar än vad som tidigare skedde.

Det automatiserade rättningssystemet (Kattis) fungerar genom att studenterna skriver program som löser en uppgift och skickar in sin programkod via Kattis hemsida. Med hjälp av testfall (konstruerade av kurslärare) kontrolleras korrektheten och effektiviteten hos studenternas program, och studenten får omedelbart veta resultatet. När en uppgift är godkänd av Kattis gör studenten en muntlig redovisning av uppgiften. Läraren som tar emot redovisningen kan fokusera på att ge återkoppling på programmeringsstil, samt att undersöka och förbättra studenternas förståelse av lösningen. Utan ett automatiserat system måste lärarna ägna tid åt att manuellt testköra och inspektera programmen, vilket ofta blir fokus för redovisningen. Det innebär dåligt använd lärartid – summativt istället för formativt.

För att metodiken ska kunna tillämpas krävs att ett korrekt svar objektivt och automatiskt kan kännas igen. Många laborationsuppgifter inom datalogi har den egenskapen.

Inledning

En stor del av kurserna för civilingenjörsutbildningen i datateknik vid KTH innehåller programmeringsövningar. Studenterna behöver lära sig olika språk, protokoll, algoritmer och angreppssätt som är användbara när man vill styra beteendet hos eller kommunicera med en dator. Själva programmeringen är ett hantverk, men den backas upp av kunskaper i matematik och av kunskaper om och erfarenhet av möjliga gränssnitt mot en tänkt användare. Civilingenjörer i datateknik ska i sin yrkesroll ha förståelse för vilka typer av problem som kan lösas av datorer och hur. Det finns många aspekter av programmering. I den här artikeln koncentrerar vi oss på systemet Kattis som granskar program som löser algoritmiska problem.

Syftet med att införa Kattis var att befria lärare och assistenter från mekaniska uppgifter som går lätt att automatisera, för att de ska hinna med pedagogiskt sett viktigare saker. När studenter skriver program som löser specifika uppgifter är en viktig aspekt av bedömningen av deras program att det verkligen löser uppgiften, det vill säga beter sig korrekt (Bailey, 2005). Detta kontrolleras i första hand genom att man testar programmet på lämplig mängd olika indata, och i en ideal värld har studenterna redan gjort det när de anser sig vara klara med uppgiften. I verkligheten brukar det ofta finnas saker som de har glömt att testa eller fel som de hoppas att läraren inte kommer att upptäcka. En annan viktig aspekt är prestanda. Att programmet löser uppgiften på ett effektivt sätt kan också testas med hjälp av testfall, som då bör vara tillräckligt stora för att programmets effektivitet ska bli synlig. Studenterna har sällan provat så stora testfall, alla har inte ens gjort minimiansträngningen att kontrollera att programmet hanterar exempel från uppgiftslydelsen på korrekt sätt.

Tidtagning, minnesmätning och korrekthetstestning är vid användning av automatiserad rättning inte en uppgift för läraren vid redovisningen utan något som studenterna själva

kontrollerar i förväg - assisterade av Kattis. Det finns både praktiska och pedagogiska fördelar med detta, och det möjliggör en ny metodik.

Algoritmisk problemlösning

Det är viktigt att skilja mellan att använda algoritmer, vilket kan göras mekaniskt och utan att förstå dem, och att kunna konstruera algoritmer, vilket den som skriver program för datorn måste vara kapabel till. Algoritmkonstruktion för olika problemklasser är ett forskningsområde, och algoritmer är en nödvändighet för att en dator ska kunna utföra någonting alls. Det finns ett stort antal algoritmer som är välkända och som löser vissa typer av problem, men det finns även en oändlig mängd uppgifter eller situationer som man kan vilja hantera med ett program. Det räcker då inte att kunna använda färdiga algoritmer, utan man behöver kombinera och modifiera kända algoritmer och/eller konstruera nya algoritmer.

Vid algoritmisk problemlösning måste problemet och dess matematiska struktur vara i fokus, och alla dess eventuella specialfall tas i beaktande. Det program som skrivs eller den metod som tänks ut måste lösa problemet i alla dess tänkbara skepnader. Det finns några saker som är särskilt intressanta att öva på när det gäller sådana program och som går alldeles utmärkt att testa automatiskt: att få dem att ge ett rätt svar, att de går snabbt att exekvera och att de kräver rimligt mycket av datorns arbetsminne. Andra aspekter av programmering är till exempel planering av ett användargränssnitt, sätt att strukturera koden, designmönster, hur väl programmet är dokumenterat och hur läsbart det är för en människa. Kattis ger ingen hjälp med att granska dessa aspekter; där behövs en mänsklig lärare.

Formativ utvärdering och automatiserad rättning

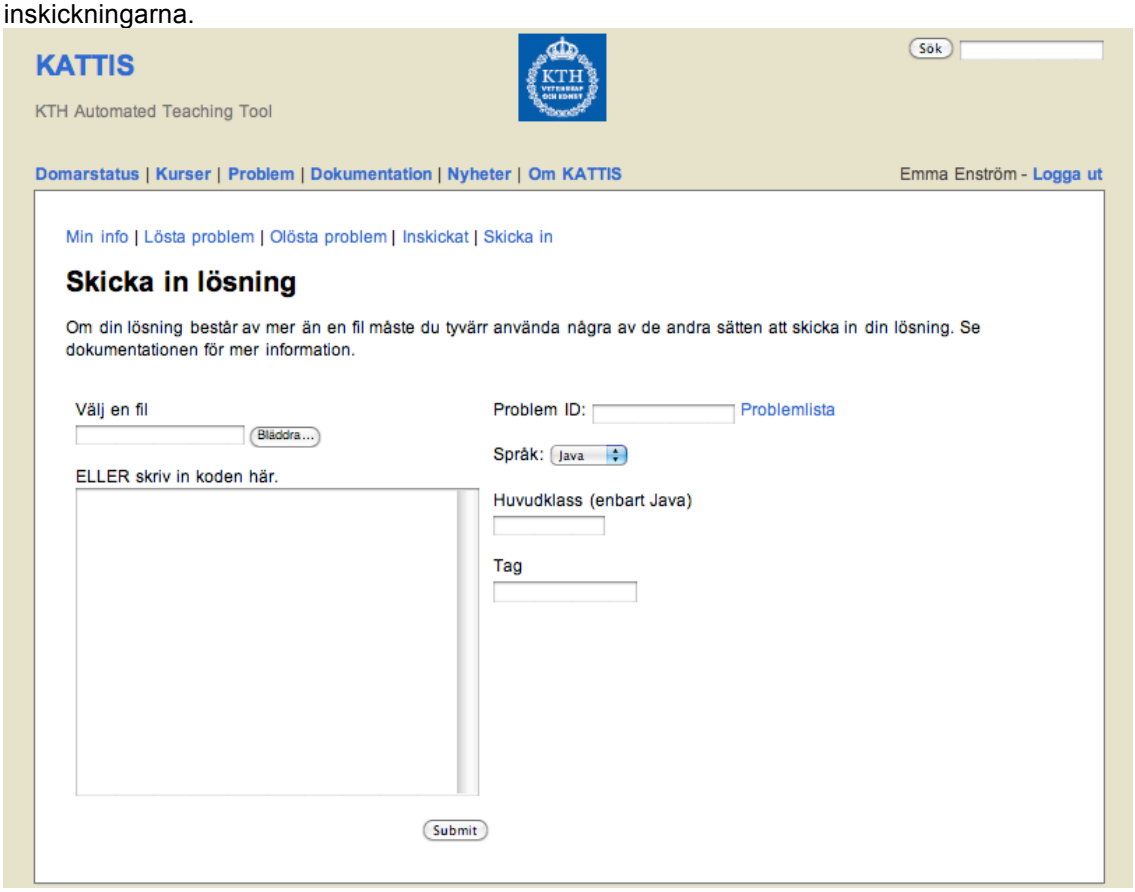
Utvärdering av kunskaper kan ske på olika sätt och i olika syften. En viktig skiljelinje brukar gå mellan formativ och summativ utvärdering (Lindström & Lindberg, 2005). Den summativa utvärderingen sätter punkt, bedömer vad som uppnåtts och betygsätter, medan den formativa diagnosticerar och ger möjlighet att planera studier och undervisning. Det är inte en unik tanke att slå ihop formativ utvärdering, det vill säga utvärdering av kunskaper under lärandeprocessens gång och i diagnostiskt syfte, med automatiserad rättning. Att ge möjligheter till självdiagnos och att tillhandahålla dem utan lärarmedverkan kan bland annat betraktas som kostnadseffektivt (Higgins & Bligh, 2006) och mera rättvist och spårbart (Suleman, 2008). Att i egenskap av att vara den som lär sig något ha tillgång till verktyg för utvärdering utan att vara beroende av en lärare ger större frihet och flexibilitet.

Automatiserad rättning är tillämpbar i sammanhang där det går att känna igen ett rätt svar objektivt och algoritmiskt. Som en jämförelse skulle ett system som assisterar vid språkstudier typiskt sättas in för glostester men inte för att bedöma essäer. Uppgiften behöver inte vara av faktakaraktär för att kunna rättas automatiskt. Det behöver inte bara finnas ett rätt svar, men det brukar ändå vara just svaret som bedöms, det vill säga bedömningen är i någon mån indirekt. Det är svårare att förstå en process eller till exempel ge delpoäng, eftersom det är svårt att säga *varför* något inte är rätt. Ju mera av kreativitet eller hänsyn till sammanhang som krävs i bedömningen, desto svårare blir det att göra bedömningen automatiskt. Just i de uppgifter som kräver algoritmisk problemlösning, och där det är essentiellt att algoritmen som används faktiskt fungerar, kan problemet göras nästan obegränsat komplicerat medan svaret fortfarande går att bedöma som korrekt eller inte korrekt. Uppgifterna kan också vara av betydligt enklare karaktär, till exempel att implementera en sorteringsalgoritm. Det som studenterna övar när de löser sådana uppgifter är i första hand hantverksskicklighet.

Kattis

Kattis fungerar på så sätt att studenterna skickar in sina program via e-post eller via en uppladdningssida på webben, se figur 1. De tester som bestämts av den som konstruerade uppgiften kommer då att köras, och när det är klart får studenten reda på resultatet, också detta via e-post och/eller via webbsidan. Den minsta information som studenterna får av systemet är huruvida programmet blev godkänt, om det kraschade, om det gav upphov till kompileringsfel, om det tog för lång tid eller om det inte gav riktiga svar för något av testerna. Själva testfallen presenteras inte, eftersom det bara är själva uppgiften som studenterna ska utgå ifrån när de konstruerar sin lösning. Beroende på vad det är för ett problem kan också annan information lämnas, under förutsättning att uppgiftskonstruktören har skrivit ett program som granskar utdata från studenternas program och ger lämpliga kommentarer.

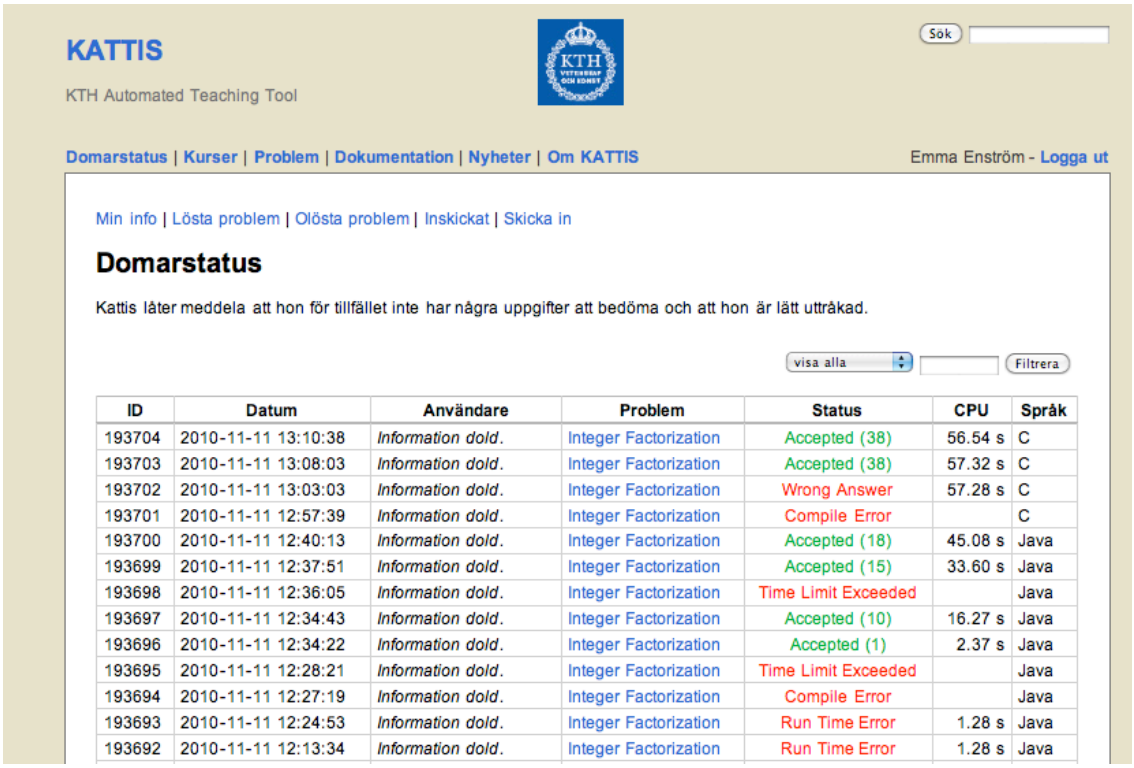
Det finns inga begränsningar för när eller hur ofta studenterna kan skicka in en lösning till Kattis. De får så många försök de vill ha. Figur 2 visar webbgränssnittet för Kattis pågående och nyligen avslutade aktiviteter, där studenter och lärare kan se bedömningen av de senaste inskickningarna.



Figur 1: Webbgränssnittet för inskickning av program till Kattis.

Kattis kan användas så att de uppgifter som studenterna måste lösa i någon kurs läggs in i Kattis och läraren därmed sparar tid. Läraren har också möjlighet att påverka hur modulärt studenterna löser uppgiften genom att dela upp den i flera mindre problem och låta dessa testas separat av Kattis. Då påverkas indirekt även strukturen på det program som studenten skriver. Eftersom Kattis är tillgänglig dygnet och året runt, kan läraren även gå steget längre och erbjuda rättning av fler uppgifter än ett par stora laborationer. Annat som gås igenom i kursen kan också presenteras som programmeringsproblem. Detta medför att studenterna får praktisk erfarenhet av fler saker, istället för att berätta om dem teoretiskt på tentamen. Den omedelbara återkopplingen gör att studenterna inte hinner glömma hur de tänkte om en lösning visar sig vara felaktig. För studenten fungerar Kattis som en ständigt beredd "lärare" som kan ge omedelbar återkoppling dygnet runt.

Att lägga till nya problem i Kattis är inte lika automatiserat som bedömningsprocessen. Den som konstruerat ett problem behöver också konstruera en exempellösning som är rimligt snabb och en mängd testdata som kan användas för att avgöra om problemet är löst eller inte. Problemet måste också testköras och förses med lämpliga tidsgränser innan det läggs in i problemdatabasen.



ID	Datum	Användare	Problem	Status	CPU	Språk
193704	2010-11-11 13:10:38	*Information dold.*	Integer Factorization	Accepted (38)	56.54 s	C
193703	2010-11-11 13:08:03	*Information dold.*	Integer Factorization	Accepted (38)	57.32 s	C
193702	2010-11-11 13:03:03	*Information dold.*	Integer Factorization	Wrong Answer	57.28 s	C
193701	2010-11-11 12:57:39	*Information dold.*	Integer Factorization	Compile Error		C
193700	2010-11-11 12:40:13	*Information dold.*	Integer Factorization	Accepted (18)	45.08 s	Java
193699	2010-11-11 12:37:51	*Information dold.*	Integer Factorization	Accepted (15)	33.60 s	Java
193698	2010-11-11 12:36:05	*Information dold.*	Integer Factorization	Time Limit Exceeded		Java
193697	2010-11-11 12:34:43	*Information dold.*	Integer Factorization	Accepted (10)	16.27 s	Java
193696	2010-11-11 12:34:22	*Information dold.*	Integer Factorization	Accepted (1)	2.37 s	Java
193695	2010-11-11 12:28:21	*Information dold.*	Integer Factorization	Time Limit Exceeded		Java
193694	2010-11-11 12:27:19	*Information dold.*	Integer Factorization	Compile Error		Java
193693	2010-11-11 12:24:53	*Information dold.*	Integer Factorization	Run Time Error	1.28 s	Java
193692	2010-11-11 12:13:34	*Information dold.*	Integer Factorization	Run Time Error	1.28 s	Java

Figur 2: Webbgränssnitt med resultat av testkörningar. Det är nära deadline för kursuppgiften Integer Factorization, därför jobbar många med den. Uppgiften har ett speciellt granskningsprogram som beräknar poäng utifrån hur stora/svåra tal programmen lyckas faktorisera, vilket gör att man kan bli godkänd med olika poäng. Studenterna kan välja om deras namn ska vara synligt eller inte.

Laborationer

Det finns olika typer av laborationer i olika ämnesutbildningar. De kan vara allt ifrån en övning som ska illustrera en teori till ett självständigt projektarbete. I en översikt av laborationer i naturvetenskapliga ämnen 2000 ger Håkan Hult (s 48-52) svidande kritik mot hur laborationer slentrianmässigt används motiverade av argument som inte är tillämpbara i sammanhanget. Medan läraren tror att laborationen illustrerar en teori eller kopplar ihop resultat med praktiken kan studenten uppfatta den som en helt isolerad övning som går ut på att följa ett protokoll och skriva in svaren på några frågor, eller mätresultat från något delsteg, på anvisade platser. Hult talar också om en indelning av laborationer i kategorierna “torra” och “våta”, där de torra laborationerna endast simulerar den verklighet som de handlar om.

I datalogi är det lite svårare att dra en gräns mellan torra och våta laborationer, eftersom simulering i sig kan vara stoff i kurser och själva aktiviteten under arbetet med laborationen i mycket högre grad är tätt kopplad till det som laborationen är avsedd att öva – laborationerna är mera direkta. Det finns inte en fysikalisk verklighet som ska simuleras, utom som rekvisita i uppgifterna. Denna annorlunda relation mellan ämnet och laborationen gör att vi anser att flera av argumenten för laborationer som Hult kritiserar – att de motiverar, ger hantverksskicklighet och stöder meningsfullt lärande – faktiskt är tillämpliga i det här fallet. En fjärde punkt listas bland de saker som brukar åberopas: att laborationer ger insocialisering i den vetenskapliga världen. I vårt fall rör det sig snarare om insocialisering i den professionella rollen som ingenjör, där problemlösning och programmering är viktiga delar.

Vår definition av laborationer

Laborationsuppgifter i datalogikurser är vanliga på KTH och går normalt till så att studenterna får uppgifter att lösa genom att skriva program, två och två. Ibland förekommer förberedelseuppgifter som ska redovisas före arbetet med laborationen. Det finns schemalagda tillfällen då studenterna arbetar med uppgifterna och har möjlighet att fråga om hjälp när de kör fast, och ofta någon deadline för redovisning av uppgiften. Redovisningen har flera syften: för eleven gäller det att visa att man är klar med uppgiften, att visa att man har förstått och själv löst uppgiften, att diskutera programmeringsstil och de val man gjort vid implementationen, och

för läraren att godkänna att eleven klarat av uppgiften och förstått vilka val och alternativ som ingått. Vid redovisningen är det studentens prestation som ska examineras vilket innebär att det är studentens kunskaper som står i fokus, inte programmet i sig. Läraren granskar lösningen för att se att den uppfyller de krav som ställs. När automatisk rättning används så består granskningen i huvudsak av en kort kontroll att koden är rimligt strukturerad och kommenterad, sådant som är svårt att bedöma automatiskt. Utan automatisk rättning ingår det även i lärarens roll att vara den som testar att programmet fungerar, det vill säga att det löser rätt problem och att den uppfyller kraven, något som tar tid att göra noggrant och som lätt blir fokus för redovisningen. Redovisningen sker oftast muntligt och vid datorn, för en lärare eller en labbassistent.

Testdriven utbildning

De kurser som använder Kattis fungerar oftast som vanligt i det att studenterna arbetar med enskilda uppgifter, laborationer och teori. Kontinuerlig examination tillämpas vanligtvis, så det finns olika kontrollstationer längs vägen mot en färdig uppgift. De uppgifter som rättas av Kattis har det gemensamt att studenterna direkt får veta om deras program kan anses vara färdigt eller inte. Det är själva utlåtandet från Kattis som påverkar studentens uppfattning om hur färdigt programmet är, inte en oavsiktlig missuppfattning eller förenklad tolkning av uppgiften. På så vis jobbar studenterna (åtminstone) tills deras lösning nått den nivå som läraren hoppats på. Denna metodik kallar vi testdriven utbildning. Den har beröringspunkter med programutvecklingsmetoden testdriven utveckling som används i delar av programvaruindustrin (Beck, 2001). Om studenterna själva utvecklar testfall för sitt program är det enklare att sedan klara Kattis granskning.

Det bästa sättet att arbeta med Kattis är att dela upp laborationer i mindre bitar som rättas enskilt av Kattis. Bitarna byggs ihop till en lösning på den större laborationsuppgiften, som också rättas av Kattis. På det viset får studenterna hjälp att systematiskt testa och felsöka sina lösningar. Dock får man flera av fördelarna redan av att ta en befintlig laboration och lägga in den utan modifieringar i Kattis, vilket möjliggör en gradvis övergång till metodiken.

Metodiken och Kattis har hittills främst använts på förhållandevis avancerade datalogikurser.

Exempel

Här beskrivs tre kurser som använder Kattis i olika stor utsträckning. Dessa kurser exemplifierar olika sätt att bedriva undervisning med stöd av automatisk rättning.

Obligatorisk kurs där Kattis granskar labbarna

Kursen *Algoritmer, datastrukturer och komplexitet* (ADK) är obligatorisk för studenterna på civilingenjörsprogrammet i datateknik på KTH och följs av cirka 130 studenter. Det är en fortsättningskurs i algoritmkonstruktion, algoritmanalys och datastrukturer, och introducerar komplexitetsteori. I kursmålen står det bland annat att studenterna efter kursen ska kunna

- utveckla och implementera algoritmer med datastrukturer och analysera dem med avseende på korrekthet och effektivitet
- jämföra alternativa algoritmer och datastrukturer med hänsyn till effektivitet och pålitlighet,

och studenterna får lära sig att motivera att deras algoritmer löser de problem de är avsedda att lösa och får öva på att skriva effektiva program, sådana som inte tar för lång tid. Laborationerna är omfattande och arbetskrävande, och det finns många sätt som ett program kan göra fel på utan att det märks vid en enstaka testkörning på slumpat indata. Redovisningarna i denna kurs skulle ta mycket längre tid om lärarna måste testköra programmen med alla lämpliga indata, och lärarens roll skulle bli motståndarens, domarens eller opponentens.

I den här kursen har lämpliga testfall och granskningsprogram skrivits för de befintliga uppgifterna, och Kattis har tagit över de delar av arbetet som passar att automaträtta. Kvar blir redovisningar med lärare och elever som kan samtala om problemet och den valda lösningen utan att eleven behöver känna sig orolig att något inte har tagits om hand eller läraren behöver oroa sig för att missa något och godkänna felaktiga lösningar. I en av labbuppgifterna skickas lösningar in till Kattis inte mindre än tre gånger. Den är nämligen uppdelad i två delproblem och en sammansättning av delproblemen.

Lärarna i kursen vittnar om att deras tid vid redovisningarna nu används pedagogiskt sett mycket effektivare.

Valfri kurs med labbar som tränar testdriven utveckling
Programsystemkonstruktion i C++ är en valfri kurs som läses av ca 100 studenter varje år. Bland kursmålen står att studenterna ska lära sig att skriva testkod och riktig C++-syntax. Den har flera uppgifter som portionerats ut i deluppgifter som kan testas separat i Kattis. Detta möjliggör för studenterna att träna på ett sunt testningsbeteende med separata tester av olika komponenter och tester av hur komponenterna fungerar ihop. Laborationerna utnyttjar här möjligheter som av ekonomiska skäl inte skulle finnas utan Kattis: att alla studenter gör flera preliminära versioner av varje uppgift och får dem bedömda separat. Därmed får studenterna feedback på mera av sitt arbete och dessutom tidigare än vad som hade varit fallet med manuell rättning.

Kurs som bygger helt på Kattis
Kursen *Problemlösning och programmering under press* (POPUP) använder Kattis i betydligt högre grad än andra kurser. De flesta av målen med denna kurs är sådana som kan testas effektivt med Kattis:

- analysera effektiviteten hos olika lösningsmetoder för att avgöra vilka som i ett givet sammanhang är rimligt effektiva,
- jämföra givna problem med avseende på svårighetsgrad,
- använda och anpassa grundläggande algoritmer inom områden som grafteori, talteori, geometri på givna problem,
- använda algoritmkonstruktionsmetoder som giriga algoritmer, dynamisk programmering, dekomposition och kombinatorisk sökning för att konstruera algoritmer för att lösa givna problem,
- givet en specifikation av en algoritm eller datastruktur, implementera den korrekt i ett programmeringsspråk.

Kursen examineras med upprepade hemläxor, laborationsuppgifter och "prov", dvs. tidsbegränsade sessioner där studenterna löser så många som möjligt av 8-12 problem under fem timmar. Under laborationerna skriver studenterna programkod som det är meningen att de ska ha kvar och återanvända under resten av kursen och även i andra sammanhang. De implementerar olika datastrukturer och algoritmer. Hemläxorna är ett antal problem som rättas av Kattis. "Proven", eller problemsessionerna, rättas också av Kattis. Laborationer och problemsessioner görs i grupper om två. Totalt löser studenterna mellan 25 och 75 uppgifter under kursens gång.

Domare i programmeringstävlingar
Kattis används också för att bedöma bidrag i programmeringstävlingar, som liknar problemsessionerna i kursen POPUP men också räknar poäng och rankar de tävlande utifrån hur många problem de löst och hur fort det går. Felaktiga lösningar som föregår en korrekt lösning av något problem ger tidspåslag. Kattis utformning och språk är mycket präglat av tävlingskontexten, med Kattis som "domare" och bedömningarna av olika lösningar som "domslut".

Det finns många fler system som används i programmeringstävlingar, se till exempel (Leal & Silva, 2003). Som framgår av exemplen ovan går det dock utmärkt att använda Kattis i kurser som inte alls har någon programmeringstävlingskontext eller likhet med programmeringstävlingar.

Utvärderingar av Kattis i kurser
Vid utvärdering av kurser som använt Kattis brukar studenternas åsikter falla inom två kategorier: att Kattis är väldigt bra att ha respektive att Kattis är värdelös för att man inte får veta vilka testfall som används. Efter ett par års användning av Kattis i en kurs tenderar de negativa åsikterna att minska och de positiva att öka.

Till tävlingssammanhanget hör, att det är i de tävlandes eget intresse att skicka in lösningar som inte innehåller några fel. I undervisningssammanhang är vi inte intresserade av begränsningar och av att använda straff för felaktiga lösningar, utan tvärtom är idén att studenterna ska kunna göra så många försök som de anser sig behöva utan repressalier. Detta medför möjligen en annan syn på ansvarsfördelningen mellan student och bedömare. På ADK-kursen, som läses av en hel årskurs, kritiseras ofta lärarna och Kattis för att vara snåla med sin kunskap. Studenterna undrar varför de inte kan få se testfallen så att de enklare kan åtgärda fel i sina program. Eftersom lärarnas ambition är att studenterna ska öva sig på att testa sin kod skulle det vara kontraproduktivt att ge dem alla testfall. Uppgiftslösandet får då enbart karaktären av snitslad bana. Istället för att analysera den matematiska strukturen på problemet som ska lösas kan studenterna fokusera på en i taget av en uppsättning testdata, utan att fundera på vilka data det är och varför dessa är viktiga. Det blir då fråga om att praktiskt hantera specifika, givna datakombinationer och problemet blir oviktigt för den aktiviteten.

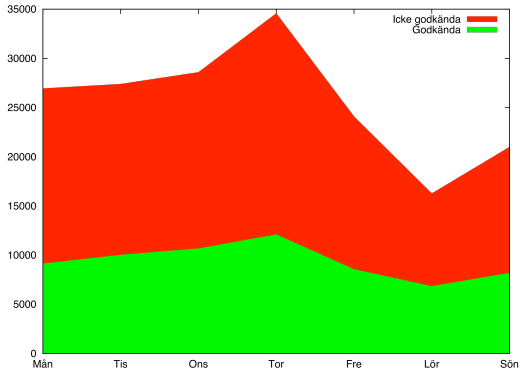
Diskussion

I kurserna har vi använt systemet på båda sätten som beskrivs i avsnittet Testdriven utbildning. Att använda den anpassade metodiken kräver naturligtvis ett ibland omfattande arbete då labbarna och eventuellt delar av kursen måste omarbetas, men resultaten har varit väldigt positiva.

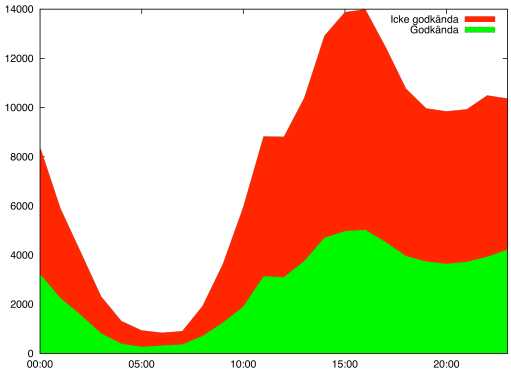
Att systemet går att använda på båda dessa sätt innebär att det är lätt att införa systemet, man kan alltid helt enkelt överföra befintliga kurser och labbar, men man kan dessutom omarbeta dessa för att till fullo utnyttja fördelarna.

För varje problem med tidsgräns finns en “highscorelista” av samma typ som i till exempel datorspel. Studenterna får möjlighet till visst erkännande genom att vara författare till en av de snabbaste lösningarna. Detta får vissa studenter att anstränga sig än hårdare. Det gäller vanligtvis dem som är duktigast på att programmera, och som också kunde ha valt att bara glida genom kursen. Genom att de sporras att anstränga sig lär de sig också något.

Figur 3 och 4 visar statistik över studenternas användning av Kattis hittills. Möjligheten att skicka in lösningar dygnet runt utnyttjas flitigt.



Figur 3: Fördelning av icke godkända (rött) och godkända (grönt) lösningar över olika veckodagar.



Figur 4: Fördelning av lösningar över dygnet.

Begränsningar

För att Kattis ska kunna tillföra något krävs att det går att testa om ett svar är korrekt. De flesta laborationsuppgifter i programmering och datalogi har den egenskapen, och det är viktigt att programmen faktiskt gör rätt. Minsta bugg kan i tillämpningar orsaka stora problem och skador för hälsa och egendom, då mjukvara ingår i navigationssystem, signalsystem, sjukvårdsutrustning och skyddsteknik i industrin, så förmågan att skriva korrekta program är helt nödvändig hos den som utvecklar sådana system. Det är lätt att skriva program som *oftast* gör rätt, men inte klarar vissa krångliga specialfall.

Kattis kan inte användas till all bedömning av studenternas program. Den muntliga redovisning som brukar ingå i våra programmeringsuppgifter är inte oviktig. Om man inte tillämpar ett sådant arbetssätt (till exempel genom att ersätta redovisningar med automaträttning) kommer andra viktiga aspekter ur fokus.

Kattis gör inte i sig själv så mycket för att främja ett sunt testbeteende bland studenterna, men laborationsuppgifter kan utformas så att studenterna får övning i att testa ofta och systematiskt.

Man kan inte bedöma all typ av kunskap inom alla ämnen genom Kattis, men man kan inte heller enbart bedöma färdigheter på tillämpningsnivån i Blooms taxonomi. Det är istället typen av stoff som är begränsad. Uppgifterna kan gå ut på att implementera kända algoritmer, men också på att modifiera dem för att passa på nya problem, kombinera dem till lösningar på mera komplexa problem, eller konstruera helt nya algoritmer för något problem. Detta kräver förmåga till analys och till att återknyta till vad man tidigare lärt sig och går att göra så svårt som situationen kräver.

Slutsatser

Avslutningsvis återvänder vi till artikelns titel ord för ord: *Strukturerad* – får studenterna att bygga modulära, genomtestade, effektiva, återanvändbara program. *Formativ* – ger självdiagnos och omedelbar återkoppling, fortlöpande och vid behov. *Examination* – labbarna ingår normalt i kurskraven. *Testdriven* – snabba resultat för att motivera till ytterligare ansträngningar och topplista som ger tävlingsassociationer. Sammanhanget för detta är övningar i algoritmisk problemlösning, på olika kognitiva nivåer.

Arbetet med uppgifter på kurser som använder Kattis drivs mer eller mindre av de resultat som Kattis meddelar. Kattis är rättvis, objektiv, noggrann och snabb, eftersom alla lösningar utsätts för exakt samma behandling. Det är inte fråga om att slumpmässigt välja några vanliga tester från en större lista, utan allt testas. Lärare som använder Kattis behöver inte oroa sig för att de oavsiktligt godkänner allvarliga misstag för någon grupp av studenter men hittar dem hos någon annan, och studenter behöver inte redovisa sina lösningar under stor stress över att det kan finnas fel.

Relationen mellan lärare och studenter i labbsammanhang har därför bättre förutsättningar för att vara positiv, eftersom läraren inte i samma utsträckning behöver utöva makt. Kattis har tagit över rollen av motståndare, och de roller som läraren kan tilldelas betonar mera hjälp och stöd.

Referenser

Bailey, M. W. (2005). IronCode: Think-Twice, Code-Once Programming. I *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, St. Louis, MO, USA. (s. 181-185). ACM.

Beck, K. (2001). Aim, fire (test-first coding). *IEEE Software*, 18(5), 87-89.

Higgins, C. A. & Bligh, B. (2006). Formative Computer Based Assessment in Diagram Based Domains. I *The Eleventh Annual Conference on Innovation and Technology in Computer Science Education*, Bologna (s. 98-102). ACM.

Hult, H. (2000). Laborationen – myt och verklighet. En kunskapsöversikt över laborationer inom teknisk och naturvetenskaplig utbildning (CUP:s Rapportserie Nr 6). Linköpings Universitet, Centrum för universitetspedagogik.

Leal, J. P & Silva, F. (2003). Mooshak: a web-based, multi-site programming contest system. *Software – Practice & Experience*, 33, 567-581.

Lindström, L. & Lindberg, V. (Red.). (2005). *Pedagogisk bedömning. Om att dokumentera, bedöma och utveckla kunskap*. Stockholm: HLS förlag.

Suleman, H. (2008). Automatic Marking with Sakai. I Proceedings of Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, Wilderness, South Africa. (s. 229-236). ACM.