

# Innovativ programmering - software craftsmanship, open source, design

## Förslag till ny kandidatexamen

Dokumentets författare: Lars Degerstedt

Innehåll: Mikael Kindborg, Kevin McGee, Rita Kovordanyi, Arne Jönsson, Jalal Maleki och Anders Haraldsson

Uppdaterad version: 2006-06-22

*I texten används genomgående det nya poängsystemet med 60 poäng per läsår.*

## Introduktion

Innovativ programmering (IP) är en ny utbildning på 180-p (kandidatexamen) som är tänkt att komplettera de existerande IT-utbildningar C/D/DI. Den stora nyheten ligger i betoningen av **hantverkskunnande** inom programmering. De grundläggande programmeringsämnena undervisas likartat med existerande linjer. Men i existerande utbildning ges dessutom utvidgad matematisk och allmänorienterad teoretiskt fördjupning. Här väljer IP istället att fördjupa den **individuella skickligheten** som programmerare och att kunna fungera väl ihop med andra i **projekt och community**.

Undervisning i IP består av programmeringsnära teori och omfattande egen programmering/systemhantering. Betoningen ligger på att omsätta en relativt begränsad men högst användbar teori genom omfattande utveckling av den egna hantverkarkompetensen. Det viktiga är att den kunskap som lärs drivs ända fram till yrkesmässigt användbar förmåga, på bekostnad av att en mindre mängd ämnen kan täckas in. Ämnena väljs baserat på vilka tillämpningar som bedöms som industriellt relevanta när studenten går ut i arbetslivet 3-5 år senare. Dessa ämnen och universitetets kompetens för IP kommer därmed förändras över tiden på ett kontrollerat sätt. Vi förutser också att det trots allt kommer finnas en grund som inte förändras så snabbt, vad gäller språk och plattformar och hantverksregler.

Tentativt tror vi att studenterna i den närmsta tidsrymden kommer få jobb bl a som: **programmeringskonsulter, tekniker, webb- och gränssnittsutvecklare**. Betoningen ligger på att snabbt kunna fånga en kunds krav och leverera en bra lösning. Det kan röra sig om snabb skriptprogrammering, mer avancerade former av system och arkitekturer, samt mer system-drift-orienterade jobb. Därutöver vill vi också uppmuntra till  **eget entreprenörskap**. Här tänker vi på det nya utrymme som finns för IT-baserade produkter och online-tjänster, där det  **kreativa och affärsmässiga** momentet är minst lika viktigt som det rent tekniska/formella utbildningsmässiga. Det är här viktigt att kunna paketera sin kompetens/tjänst/produkt och sälja den på ett bra sätt, t ex som underkonsult eller som online-företag.

Utbildningen har fem grundläggande teman som kurserna grupperas runt:

- Programspråk och plattformar
- Gränssnittsdesign
- Software craftsmanship
- innovativ egen tillämpning
- Open Source

Viktig **bok**: Software Craftsmanship av McBreen (kursbok i PP2-kursen/ETE247). Detta är en viktig grundläggande bok som betonar vikten av individuellt hantverkskap bland annat. Boken myntar begreppet Software Craftsmanship och kan användas som en standardreferens för de värderingar som bör genomsyra denna utbildning.

Förutom kursbetyg och den färdiga studentens ha skapat en **portfölj** av programsystem under utbildningens gång. Portföljen kommer vara strukturerad så att det framgår vilka olika tekniker som ingått i olika moment och projekt. Utseendet på denna portfölj kommer behöva uppdateras kontinuerligt, men i en hanterlig takt. Den relativt låga förändringstakten förutser vi eftersom IP fokuserar på den tekniska basen och det långsiktigt principiella.

Vi är också positiva till att utbildningen ges på **engelska** och erbjuds för både svenska och internationella studenter. För en programmerare är ju ändå arbetsspråket engelska, så det känns naturligt för utbildningen.

## Blockbaserad översikt

### År 1: Grundår

- språk+plattformar (15p)
- UI-programmering (10p)
- datateknik (5p)
- projekt + hantverk (30p)

### År 2: Tillämpningsområden

- informationssystem och databaser (10p)
- Internet (10p)
- media, konsumentprodukter och spel (5p)
- operativsystem och systeminstallation (5p)
- övr teori t ex matematik (5p)
- projekt + metodik och design (25p)

### År 3: Fördjupning

- programhantverk (10p)
- valfria kurser (20p)
- kreativt projekt + innovation (15p)
- industriellt projekt + industrikontakt (15p)

Vi ser ett antal olika utbildningsmoment som ska lyftas fram i utbildningen. Metodik för individuellt arbete och mindre grupp är en viktig del av software craftsmanship, t ex hantverksregler och agil metodik. Djup förtrogenhet med grundläggande utvecklingsverktyg och programmeringsspråk. Här kommer en större inventering av nuläget och framtid göras senare. Exempel på utvecklingsverktyg är texteditor, IDEer, byggverktyg, versionshantering. Egen erfarenhet att arbeta och lära sig av dominerande utvecklingsplattformar och teknikfronter med speciellt fokus på öppen källkod: exempel är Linux, Java, Apache. Kännedom och erfarenhet av dominerande tillämpningsområden (inkl målplattformar): exempel är enterprise computing, online-tjänster, dotnet, c# och produkter för Windows-plattformen.

## Väggkarta och roller

**Hösten 2007**: första kullen studenter börjar.

**Industrikoppling:** En viktig komponent för att det tillämpningsnära inslaget ska fungera är nära koppling till olika företag.

**Lärarkompetens för IP:** IP måste kontinuerligt förnya och vidareutveckla lärarnas egen programmerarkompetens. Att personer har egen erfarenhet av konkret programmeringsarbete bör premieras vid val av kurser och lärare.

Utbildningen var ursprungligen ett förslag från avdelningen HCS på IDA. Fortsatt utformande av utbildningen bör dock ske i en mer gemensam arbetsgrupp där intressenter inbjudas att delta i utformningen av olika detaljer.

## En innovations- och tillämpningsdriven utbildning

IP är en innovations och tillämpningsdriven utbildning. Teori-delen hålls relativt liten och driven av behov i tillämpningarna, fokus är på direkt omsättbart hantverksskunnande. Tillämpningarna **väljs med största omsorg**, extremt viktigt moment. Genomgång och utlärande av olika tekniker och verktyg är mer en form av grötrecepts-kunskap som dock måste läras ut/självläras i ganska stor omfattning (exakt hur kan man diskutera).

Den pedagogiska modellen för utbildningen är att ha en stor del projektarbete vid sidan av kurser. Projektarbetet sker till stor del enskilt eller i grupp beroende på uppgift.Handledning och avstämningar görs till stor del i stor grupp, t ex som halv/heldag-workshops. Samtidigt betonas och uppmuntras att arbete sker i utbyte med andras i studenternas egen community, inspirerat bl a av arbetssätt från Open-Source-rörelsen.

Det är **projekt delen** tredje året som kan ses som utbildningens examensarbete eller gesällprov. Projekten ska fokusera på att vara "riktiga" projekt och leda fram till fungerande system/programvara/produkt/etjänster. Projekten ska i största möjliga mån ankyta till existerande företag, brancher och tillämpningar där det finns intresse och arbetstillfällen.

En preliminär lista över lämpliga tillämpningsområden:

- skriptad filhantering och systemadm
- Program för intranät
- Enklare informationstjänster via gui
- enklare webb- och telefonitjänster
- spelprogram
- konsumentprodukter
- enterprise computing
- nätverksprogrammering
- etc.

Listan bör vara föränderlig och svara mot existerande marknadsbehov på några års sikt.

Projektet ger studenten en **portfölj** av system som man sen kan visa upp när man söker jobb, vid sidan av examensbetyg. Projektarbetet ger egen erfarenhet av att ha deltagit i "hela processen" till färdig programvara och personer får jobba mycket individuellt (om än lära av varandra). Alla ska programmera själva, inte bara sitta och titta på, vi har strikta regler som blir mer avancerade gradvis som de måste följa till punkt och pricka.

Områdena Software Craftmanship och Open Source kommer ses som vägledande för hur den grundläggande pedagogiken med hur man lär upp personer i hantverket att bli en bra programmerare. Där finns bl a metodik som betraktar den individuella utvecklingen i stil med det traditionella hantverket (därav namnet craftmanship). Här

ser man individens utveckling i tre steg, från **apprentice** (lärling), till **journeyman** (gesäll) till **master** (mäster). Dessa tankegångar har testas i IDA-kurser inom Pragmatisk programmering idag (ETE247 och ETE248).

## Att få igenom studenter

Det allra viktigaste ekonomiska målet är att få igenom studenter. Här får man fundera extra noga. Vi har en avvägning mellan samläsning med C/D/DI och det faktum att detta är en mer praktiskt orienterad utbildning som bör fokusera på att lära studenterna att arbeta med hög kvalitet praktiskt först och främst.

Utbildningen kommer tilltala **nya grupper** än de som redan täcks av C/DI/D. Att undersöka och gradvis utvidga utbildningen mot nya typer av personer och intressen bör ingå som del i utbildningens motto. En grundbult är att den ska lära ut teknik utan att förutsätta traditionell matematisk och analytisk färdighet. Istället bör den förlita sig **på andra sidor** hos de sökande, t ex **kreativa, praktiska, ekonomiska, etiska, estetiska** och **sociala** förmågor och intressen. Men syftet är fortfarande att vi utbildar programmerare, dvs alla moment baseras på riktiga implementerade system.

Erfarenhet från tidigare kurser inom pragmatisk programmering är att det finns mycket tid/pengar att spara genom att använda Internet-relaterade tekniker t ex forum för den tekniska delen av undervisningen. Där finns också många möjligheter att uppmuntra till stöd student-student. Undervisningen kan på så vis till stor del bestå av **självinläring** i en community som finns i en bra mix både på campus och online.

Exempel på studentgrupper vi **riktar oss till** speciellt och kommer stödja är:

- Programmeringsintresserade personer: personer som redan har ett intresse och en färdighet i det praktiska hantverket. Utbildningen kommer ge dessa personer tillfälle att möta gelikar och att bredda och fördjupa sin kompetens.
- Kvinnliga studenter: betoningen på interaktiva system, kreativitet, design och tillämpning kommer att passa även personer som är intresserade av att konstruera och utforma teknik för ett syfte, snarare än tekniken för sin egen skull.
- Yrkesorienterade personer: personer med liten programmeringsbakgrund med intresse mot "ett bra hantverksjobb" kommer få en bra konkurrenskraftig utbildning. IP tillgodoser denna grupp genom att på olika sätt erbjuda det som är eftersökt på arbetsmarknaden och en högkvalitativ professionell utbildning, samt att vi inte förutsätter att man ska kunna programmera innan man börjar IP.
- Kreativa personer: hantverksbetoningen och projektorienterade synsättet inom IP kommer tilltala personer som har en kreativ läggning. Även om de inte har ett tidigare programmeringsintresse kommer betoningen inom IP på den individuella kreativa problemlösnings- och designförmågan att tilltala denna grupp.

I en förlängning kan man även tänka sig att IP även görs tillgängligt för internationella studenter. Influensen från t ex Open Source visar tydligt att mångkulturellt inslag i communities ger en ytterligare styrka för alla inblandade. **Engelska** är ju dessutom det normala arbetsspråket för allt utvecklingsarbete, så tröskeln är relativt låg för denna typ av framtida utvidgning.

## Industriell relevans

Vi har valt termen innovation, istället för t ex kreativitet, för att betona både den problemlösande och den industriella aspekten av utbildningen. Inom IP betonas vi i första hand den tekniska betydelsen av ordet. Men det finns givetvis en koppling även till den ekonomiska användningen indirekt.

Våra industriella kontakter (främst genom utbildningsnämnd-D) har hittills varit positiva till förslaget. De har ofta spontant sagt att det låter som en typ av kompetensprofil de gärna anställer. Tanken är även att projektdelen av utbildningen ska ge utrymme för ett nära samarbete med industri och företag, inte minst företag med anknytning till Linköping. Dessa och flera andra kontakter kommer också att leda och styra innehållet i utbildningen så att IP blir lyhört för marknadens förändrade krav över tiden.

Förslaget till denna nya utbildning är även grundat i den rörelse som skrivit under The Agile Manifesto (<http://agilemanifesto.org>), en vägledande sammanslutning som vuxit fram inom progressiva mindre företag inom mjukvaruindustrin de senaste tio åren. I korthet kan man säga att den agila metodiken är relativt okontroversiell men ytterst användbar speciellt i mindre organisationer och grupper. Den summerar bland annat bra en lång epok av landvinningar inom datavetenskaplig forskning på ett pragmatiskt sätt.

Den agila rörelsen betonas snabb programutveckling med extremt stor anpassningsbarhet till förändrade villkor. Man betonas även vikten av individuell programmerarkompetens, att kunna arbeta disciplinerat och att kunna samarbeta och kommunicera för att uppnå detta.

Vi inser dock att även de agila idéerna till sist kommer ersättas av ytterligare förbättringar, men i dagsläget är de en viktig utgångspunkt. De agila tankegångarna är inte heller den enda utgångspunkten för IP givetvis. Även t ex tankar inom skildtagande design med fokus på användning och praktisk nytta är en viktig influens, samt den mer traditionella programmeringsläran som t ex representeras av Kernighan och Pike (grundare av bl a C och Unix).

Här är ett axplock av framstående personer inom programmeringsområdet som vi tagit intryck av vid formuleringen av IP-programmet och som exemplifierar hur IP passar in i ett större sammanhang av industriell teknisk utveckling som pågår just nu.

Kent Beck som drivit konsultföretag, känd objekt-orienterad profil, och skapare av metodiken Extreme Programming:

- “Optimism is an occupational hazard of programming: feedback is the treatment.”
- “Each trick comes with costs and benefits. The designer with perspective understands that. You're not a designer until you know the tricks, but you're not really a designer until you know when *not* to use them. “

Pete McBeen är självständig konsult, senior utvecklare, mentor och författaren till boken Software Craftsmanship som fått stort gehör:

- “The problem is that the schooling model employed by universities does not allow for enough coaching and mentoring.”
- “Overall, university courses provide a great theoretical background, but new graduates still have to learn the craft of software development.”

En annan viktig röst är Steve McConnell författare till boken Code Complete och en viktig person inom Microsoft. Han skriver om programvarukonstruktion och betonar vikten av det praktiska kunnandet och hur man kan lära sig detta:

- “Character is more important than intelligence”
- “The characteristics of a superior programmer have almost nothing to do with talent and everything to do with commitment...”

Inom Open Source är Eric S Raymond en viktig person som är en av grundarna av begreppet tillsammans med bland annat Tim O'Reilly. Raymond har bland annat skrivit en av del allra bästa mer praktiska böckerna om Unix (dvs indirekt även om GNU/Linux) som heter Art of Unix. Några viktiga poänger från honom är:

- “The tradition of code-sharing depends heavily on hard-won expertise about how to make programs cooperative and reusable. And not by abstract theory, but through a lot of engineering practice---unobvious design rules that allow programs to function not just as isolated one-shot solutions but as synergistic parts of a toolkit.”
- “People who actually write code generally warm to the open-source idea very quickly once they understand that they can still have jobs and pay their bills; open-source development is much more productive and more fun than the traditional closed-source mode”

Peter Norvig, forskningschef på Google och framstående person inom AI-forskning:

- “Get interested in programming, and do some because it is fun. Make sure that it keeps being enough fun so that you will be willing to put in ten years. “
- “Talk to other programmers; read other programs. This is more important than any book or training course. “
- “Program. The best kind of learning is learning by doing.”
- “Work on projects with other programmers. Be the best programmer on some projects; be the worst on some others. When you're the best, you get to test your abilities to lead a project, and to inspire others with your vision. When you're the worst, you learn what the masters do, and you learn what they don't like to do (because they make you do it for them).”

Sammanfattningsvis kan man säga att citaten betonar hantverksskicklighet och föreslår en pedagogik som betonar projekt med mentorer samt att programvaruutveckling övas på olika sätt enskilt, i par, i projekt och i community. Helt i linje med liggande förslag.

## Varför behövs en ny utbildning?

Den stora skillnaden mellan IP och C/DI/D/IT är större fokus på praktiskt kunnande inom programkonstruktion. Detta motiverat av att:

- det finns en stor stabil arbetsmarknad för personer med färdigt praktiskt kunnande inom programmering
- teoretiskt och matematiskt kunnande leder inte automatiskt till gott praktiskt kunnande, och är inte heller en nödvändig förutsättning för det (Se t ex Steve McConnells diskussioner i ämnet).
- personer med fallenhet för programmering behöver inte nödvändigtvis ha intresse av eller fallenhet för mer teoretiska ämnen. De personer som saknar sådant intresse bör

istället få utveckla sin hantverkskunskap som om det utvecklas kan bli mycket värdefullt för arbetsmarknad och den personens kärnkompetens.

Skillnaden mellan IP och en utbildning hos ett utbildningsföretag är att den grundläggande yrkesförberedande karaktären på utbildningen och att dess fokus på en konkurrensmässig kärnkompetens. Utbildningsföretagens kurser är ofta kortare och mindre sammanhängande. Deras fokus ligger på att ge korta konkurrensfördelar till personer och företag, med fokus på det senaste som efterfrågas på marknaden. Även IP bör beakta marknadsbehov men på en medellång sikt, snarare än kort. IPs fokus på kärnkompetens gör också att vi till stora delar kommer ta upp saker som ger en grundkompetens snarare än en konkurrensfördel. Dvs IP ger personer ett inträde till marknaden, snarare än fördelar mot konkurrenterna.

Skillnaden mellan IP och en programmeringsutbildning på gymnasienivå ("yrkesinriktad utbildning") är att den högre kognitiva mognad som studenten uppnått efter gymnasiet gör att vi kan börja arbeta på en högre nivå. Detta är troligen helt nödvändigt, eftersom programmering till skillnad från fysiskt orienterade hantverk (t ex snickare) kräver en stor dos av intellektuell förmåga vilket en gymnasieutbildning tillhandahåller som ett minimum.

## Exempel på existerande kurser för IP

Här följer en lista på hur **existerande kurser** anknyter till i olika block (poäng anges i detta avsnitt enligt det gamla systemet med 40 poäng per år).

- språk+plattformar:
  - Datorsystem och imperativ programmering, 3 p
  - Perspektiv på datateknik/datavetenskap, 4 p
  - Programmering, Lisp och funktionell programmering, 4 p
  - TDDC37 Programmering, Java och objektorienterad programmering
- övrig teori/matematik:
  - HKGB13 Formella och matematiska grunder för Kognitionsvetenskap, 5p
- UI-programmering:
  - HKGB11 Användbara system 2p
  - ETE 257 Interaktionsprogrammering 5p
  - TDDB91 Människa-datorinteraktion, 3 p
- operativsystem och systeminstallation:
  - TDDI05 Systeminstallation
- informationssystem och databaser
  - TDDB48 Databasteknik, 5 p
- media, konsumentprodukter och spel:
  - TDDC29 Design och programmering av datorspel, 5 p
  - TNM079 Modellering och animering, 4 p
- Internet:
  - TDDB64 Web programming and Interactivity
- Programhantverk/programvarukonstruktion:
  - ETE246-EET247 Pragmatisk programmering 5+5 p

Hur pass mycket dessa kurser behöver anpassas, bör utredas vidare och blir delvis en

ekonomisk avvägning.

## Förslag på viktiga nya kurser

Förslag på **nya kurser** som är speciellt centrala för IP:

- inom språk+plattformar:
  - fördjupning av kunskap om att arbeta via Linux och för Windows, samt att arbeta via/med öppen källkod
- Programhantverk/programvarukonstruktion:
  - inledande programmeringskunskap, förslagvis baserat på Steve McConnells bok Code Complete
- inom informationssystem:
  - Design av objekt-orienterade informationssystem, t ex baserat på Patterns of Enterprise Application Architecture Martin Fowler. i HIBC20 Programmering II (för statistiker) gjorde vi en sådan kurs i miniatyr.
- Projektserie
  - En genomtänkt projektkursorganisation som en sammanhängande tråd i utbildningen, vilket leder till en "studentportfölj". Här kan man ta fasta på lärlingsbegreppen från software craftsmanship som stöd. Erfarenheter med att undervisa i "praktisk programmering" från Pragmatisk programmeringskurserna och andra projektkurser som PUM ger en utgångspunkt att arbeta ifrån.

## Praktiskt teknikkunnande

(Läsare som inte är så programvarutekniskt orienterade kan läsa detta avsnitt översiktligt. )

Inom ramen för IP ska studenten lära sig dagens industriella teknik inom programmeringsområdet. Utbildningen måste här uppdateras kontinuerligt men plattformar och programmeringspråk ändras i långa cykler så det lär inte bli något praktiskt problem förutser vi.

Här skisseras den konkreta tekniska bas som alla studenter på utbildningen ska ha och som ska ingå i deras portfölj när de utexamineras (så som det kommer se ut för kullen som börjar 2007).

Vi lyfter fram denna konkreta tekniska del av utbildningen - språk, verktyg och plattformar etc – i detta dokument eftersom den är av stor vikt för IP. Det är också en viktig skillnad mot t ex den datavetenskapliga linjens grundupplägg. Inom IP kommer det betonas att **både principer** och vilka **konkreta industriella plattformar** som lärs ut är av stor vikt. Undervisningen kommer ske direkt i industriellt relevanta verktyg genomgående. Men IP kommer givetvis fokusera på långsiktigt hållbar teori och praktik, inom ramen för detta. Listan som följer är bitvis ofullständig vilket kommer kompletteras när kursplanerna tas fram. Den ger dock en övergripande blick över vilken teknik som kommer stå i fokus.

Från **år ett** får studenten ett baspaket av teknisk kompetens som är lika för alla. Här ingår viktiga/grundläggande/dominerande programmeringsspråk:

- webbklient: HTML/JavaScript och övriga klientspråk
- grafiska högnivåverktyg: Visual Basic

- dynamiska språk: Python, Ruby, LISP, etc
- imperativa språk: C
- statiska objekt-orienterade språk: C++, Java, C#

Plattformer:

- Windows
- GNU/Linux

Utvecklingsmiljöer:

- avancerade editorer: Eclipse och Emacs
- kommandoskalverktyg: sed, awk, grep, etc
- bygg- och testverktyg: JUnit, Ant och Make

Från **år två** får studenten ytterligare tillägg som också är lika för alla men här knutna till de olika tillämpningsområdena (förutom fördjupning i redan introducerade):

Programmeringsspråk:

- databaser: SQL
- webbserver: PHP/ASP/JSP etc
- linuxskript: bash, Perl
- spel: Smalltalk, Lua etc.

Plattformer:

- webb
- nätverk

Utvecklingsmiljöer

- informationssystem: dotnet, J2EE
- standardkodbibliotek och ramverk för olika plattformer

Därutöver ska finnas utrymme inom utbildningen att fördjupa sig i andra kända och/eller akademiskt viktiga tekniska milstolpar som t ex andra programmeringsspråk, operativsystem och olika ramverk/kodbibliotek. Under **år tre** kommer studenten mer fritt få välja vad man vill fördjupa sig i, vilket blir deras personliga tekniska profilering.

## Preliminär lärotimplan år 1 och 2

### ÅR 1

HT		
Programspråk och plattformar I, 5p	Datateknik, 5p	Projekt 1, 10p
Programspråk och plattformar II, 10p		
VT		
Programvaru-konstruktion, 10p	UI-programmering, 10p	Projekt 2, 10p

## ÅR 2

HT		
Informationssystem, 10p		Projekt 3, 10p
Spelprogrammering, 5p	Programutvecklings metodik, 5p	
VT		
Nätverk, 5p	Matematik, 5p	Projekt 4, 10p
Systeminstallation och Linux, 5p	Webbteknik, 5p	

## Projekt driven utbildningsform

För projekt delen av IP kommer vi ta intryck från hur man arbetar inom andra mer traditionella kreativa utbildningar, liksom andra liknande programmeringsutbildningar. Pedagogiskt handlar det bland annat om att hitta en bra mix mellan kurser och projekt.

Projekten kommer bedrivas löpande inom utbildningen inom en gemensam community som sträcker sig **mellan årskullarna**. Vi kommer att betona både att studenterna ska mötas **on-campus** men samtidigt utnyttja det senaste inom Internetbaserade kommunikationsverktyg för att stödja denna IP-community **online**, t ex i form av forum, wikis och bloggar. Den sociala dimensionen är speciellt viktigt för att stödja horisontell inlärning från student till student inom IP-communityn.

Projekten kommer anknyta till parallella eller tidigare lästa kurser. Uppgifterna kommer att följa de tillämpningsområden som vi pekat ut för utbildningen. Arbetsformerna kommer att variera så att man får tillfälle att arbeta både ensam, i par, och i grupp. Att lära sig lyssna på användare, kunna ta emot en beställning och att lära av och samverka med andra grupper betonas som extra viktigt. Projekten kommer följa den idagsläget bäst lämpliga metodiken för olika projekt, vilket idag återfinns bland annat inom den sk agila metodiken för mindre företag/grupper.

Projekt delen kommer ställa krav både på resultat och genomförande. För att säkerställa kvalitén på genomförandet kommer moment vara obligatoriska. T ex **obligatorisk närvaro** och att man gör ett **obligatoriskt antal timmar** med egen programmering i olika projekt.

Förutom projekten kommer kurserna ha egna laborationsmoment med mer avgränsade övningar. En slogan för utbildningen är "programmera i varje kurs" som vi tror kommer locka studenter.

Studenten kommer gradvis bygga upp sin egen portfölj av arbete. Vi kommer understödja detta genom att studenterna får bygga upp sin **egen miljö online** där de hela tiden får paketera och presentera sina resultat via sina portföljer. Så vid utbildningens slut är tanken att deras portfölj redan ska finnas online och vara väl presenterad. Detta är också en viktig pedagogisk poäng så att portföljen får bli den **röda tråden** rent konkret resultatmässigt i utbildningen.