

Nyttan av Extreme programming

Christian Davén
2I1402 Processes for IT Project Management
KTH

15 december 2003

Sammanfattning

Efter en kortfattad bakgrund och beskrivning av lättrorliga metoder beskrivs de värderingar som är grunden till Extreme programming, samt de principer metoden består av. Därefter diskuteras för- och nackdelar med metoden, med både teoretiska resonemang och resultat från praktiska studier. Bland annat nämns att metoden känns naturlig och entusiasmerar medarbetarna, men att bristen på dokumentation kan vålla problem.

Introduktion

I vilka sammanhang och på vilka sätt ger Extreme programming (XP) goda resultat? XP är en ung metod som det skrivits mycket om de senaste åren. Denna uppsats försöker reda ut vad metoden innebär och vad nyttan av den är genom att sammanställa befintlig litteratur. Uppsatsen försöker ge insikt i hur XP kan fungera i olika sammanhang liksom vad som är dess allmänna styrkor och svagheter.

Även om det skrivits en hel del om XP saknas det en grundlig genomgång av vad metoden är bra till och vad den har för svagheter. Många artiklar och rapporter redovisar erfarenheter från enskilda projekt, eller ger en generell helhetsbild av lättrorliga metoder. Som konstaterades av Abrahamsson et al. (2002) fokuserar de flesta uppsatser om XP på praktiska erfarenheter inom diverse användningsområden.

Nästa sektion innehåller en bakgrund till XP som tar upp ur vilka behov det sprang fram. Sedan ges en kort introduktion till metoden följt av en diskussion kring några av dess för- och nackdelar.

Bakgrund

Metoder för programvaruutveckling har funnits länge, men ändå är utveckling av programvara oftast en kaotisk aktivitet som kan sammanfattas med frasen "koda och fixa" (Fowler 2000). Utvecklingen sker utan en underliggande plan och systemets design växer fram genom kortsiktiga beslut. Detta fungerar när ett system är litet, men med ökande komplexitet blir det allt svårare att underhålla källkod och lägga till ny funktionalitet, samt att allt fler buggar introduceras och blir svårare att åtgärda (Fowler 2000). I sin tur leder detta till långa testfaser som spräcker tidsramarna.

De olika metoderna är tänkta att göra utvecklingsprocessen mer förutsägbar och mer effektiv. De försöker åstadkomma detta med detaljerade processmodeller som betonar planering, men de kritiseras ofta för att vara alltför byråkratiska (Fowler 2000).

Enligt Abrahamsson et al. (2002) används inte sådana metoder i praktiken – de sägs vara för mekaniska och kräver så mycket av utvecklarna att själva utvecklingen blir eftersatt (Fowler 2000).

Lättrörliga metoder

För att komma till rätta med alltför tunga metoder å ena sidan och odisciplinerad utveckling å andra sidan har mer lättrörliga (eng. *agile*) metoder tagits fram. Enligt en undersökning utförd av Charette (2001) är de mest använda XP, Feature-driven development och Adaptive software development. Representanter för dessa och andra metoder grundade tillsammans Agile alliance år 2001 och publicerade ett manifest tillsammans med de värderingar och principer som ligger till grund för metoderna. I kortet säger de att regelbundna och frekventa leveranser av fungerande programvara har högsta prioritet, att enkelhet ska genomsyra arbetet och att det bästa sättet att förmedla information är ansikte mot ansikte (Beck et al. 2001).

Abrahamsson et al. (2002) kom fram till att en lättrörlig metod är en där processen är inkrementell, samverkande (mellan kund och utvecklare), okomplicerad och adaptiv. Enligt Fowler (2000) är den stora skillnaden mellan lättrörliga och tunga metoder just att de förra anpassar sig efter förändringar istället för att försöka förutse dem samt att de är människoorienterade istället för processororienterade.

Extreme programming

XP växte under 1990-talet fram ur det nära samarbetet mellan Kent Beck och Ward Cunningham (Fowler 2000). Det första projekt där metoden formellt användes och som också anses vara metodens födelseplats, var utvecklingen av dåvarande DaimlerChryslers nya lönesystem som Beck, Cunningham och Ron Jeffries ledde från 1996 (Kaltermo & Rissanen 2002).

Kärnan i XP är fyra värderingar, som känns igen från de lättrörliga metodernas manifest, nämligen kommunikation, enkelhet, feedback och mod.

Kommunikation

Orsaken till de flesta misslyckade projekt kan hänföras till kommunikationsproblem (Newkirk 2002) och XP vill förbättra kommunikationen mellan alla inblandade parter. Samtal ansikte mot ansikte är det absolut mest effektiva sätt att kommunicera och tekniska hjälpmedel såsom CASE-verktyg hämmar kommunikation (Beck 1998). Istället för att huvudsakligen kommunicera genom systemets dokumentation manar XP till att minimera den – ingen dokumentation som inte omedelbart behövs och som är betydelsefull ska produceras (Martin 2001).

Rent konkret vill Beck (1998) att alla i gruppen ska jobba på samma kontor och ha ungefär samma arbetstider för att möjliggöra den mänskliga kontakten utan mellanhänder och för att förbättra den informella kommunikationen vid exempelvis kafferaster (Maurer 2002).

Enkelhet

För varje bit av komplexitet som införs i ett system ökar riskerna för projektet, bland annat är det svårare att undvika att införa buggar vid förändringar och dessutom svårare

att upptäcka och åtgärda buggarna. I XP eftersträvas enkelhet på alla plan; både krav, källkod och själva metoden ska vara så enkel som möjligt (Beck 1998).

Detta bygger delvis på antagandet att vi inte fullt ut kan förutsäga framtida krav från beställaren och att krav som redan fastslagits kan komma att förändras. Antaganden om framtiden kan leda till att arbete läggs ned på saker som aldrig kommer att behövas, men naturligtvis kan det också avsevärt underlätta införandet av ny funktionalitet – om antagandet är riktigt. Dock kan framtiden utvecklas på helt oförutsägbara sätt, varför det oftast blir alltför dyrt att vara förutseende (Newkirk 2002). För att därför minska riskerna med mjukvaruprojekt och istället lita på att man i framtiden kan förändra, implementeras enbart det som behövs för stunden (Beck 1998; Maurer 2002).

Feedback

Specialbyggd programvara kan inte beställas som en handelsvara där kunden beskriver det önskade systemet, betalar och går sin väg (Martin 2001). Elssamadisy och Schalliol (2002) jämförde detta med hur människor köper kläder hos en skräddare – man måste bland annat ägna tid åt måttagning – och för att få kvalitativ programvara måste kunden resonera på liknande sätt.

För att veta om utvecklingarna och beställaren har förstått varandra behövs kommunikation i form av feedback. I XP delas ett system upp i många små *deliverables*, som vart och ett ska testas och godkännas av kunden. Dessa släpps mycket ofta, i perioder om allt från dagar till ett fåtal månader. Här finns goda möjligheter att få och ge feedback.

Dessutom har varje XP-projektgrupp en kund som hela tiden befinner sig på samma plats som utvecklingarna bland annat för att göra det möjligt att direkt ställa frågor och få feedback utan dröjsmål.

Mod

Mod är resultatet av de tre ovanstående värderingarna – om utvecklingsgruppen och kunden kommunicerar effektivt, enkelheten genomsyrar arbetet och kunden ger rikligt med feedback får alla parter självförtroende att fortsätta processen på ett energiskt sätt (Beck 1998; Highsmith 2002).

Med mod vågar man kasta bort källkod som inte fungerar bra, göra enkel design som inte är anpassad efter hur framtiden kanske utvecklas och man vågar förlita sig på utvecklingsmetoden.

Principer

XP består av ett antal principer eller tekniker som bygger på ovanstående värderingar. Dessa balanserar och förstärker varandra och ger en synergistisk effekt (Newkirk 2002; Fowler 2000). Principerna var kända sedan länge när metoden först presenterades (Beck 1998, 1999; Highsmith 2000), men nu hade de för första gången smälts samman i en formell metod.

- Planering i XP sker i form av ett planeringsspel (eng. *planning game*), där kunden bestämmer hur mycket som ska ingå i den kommande iterationen och över hur lång tid den ska löpa, baserat på utvecklingarnas uppskattningar.

- För att snabbt kunna få feedback frisläpper man små delar av systemet i taget och man gör det ofta. Varje iteration slutar med en ny version av systemet som kunden får testköra och godkänna.
- Systemets arkitektur definieras av en metafor, eller en uppsättning metaforer, som både kund och utvecklare känner till. Denna princip är en av de svåraste att använda och lära ut (Ambu & Gianneschi 2003).
- I varje givet ögonblick ska systemet ha en enkel design, vilket innebär att man enbart designar för de aktuella behoven och håller antalet klasser och metoder så lågt som möjligt (Beck 1999).
- Enligt Beck (1999) är den mest centrala tekniken i XP att alltid skriva tester och före varje ny funktionalitet införs i systemet ska minst ett test bifogas. Om inte alla tester körs med lyckat resultat räknas systemet som ej fungerande.
- För att kunna vidareutveckla källkod då nya behov uppstår krävs *refactoring*, som är ett styrt sätt att förbättra strukturen på källkoden utan att påverka dess funktion.
- All källkod skrivs av två programmerare i par vid samma arbetsstation. Den ene producerar koden medan den andre observerar och letar efter misstag, tänker ut alternativa lösningar etc. Man byter roller med jämna mellanrum.
- Nyskriven och förändrad källkod integreras i systemet flera gånger om dagen.
- Ingen äger den källkod som han eller hon har skrivit och varje programmerare i gruppen kan göra ändringar överallt i systemet, när som helst.
- Kunden finns hela tiden på plats för att svara på frågor.
- Varje utvecklare arbetar 40 timmar i veckan och om övertid är nödvändigt ska det inte ske två veckor i rad.
- Gruppen arbetar i ett stort rum med öppen yta. För parprogrammerarna finns speciella arbetsstationer uppställda i mitten av rummet.
- Då alla programmerare delar den samlade källkoden och de kan göra förändringar överallt är det viktigt att man har gemensamma standarder för hur källkod skrivs.
- Dessa principer är inte rigida och kan anpassas efter behov, så länge som alla inblandade är överens.

Som den sista principen antyder är XP inte fullständigt färdigutvecklat – principerna kommer fortsätta att utvecklas, och en del koncept från de mer traditionella, tunga, metoderna kommer förmodligen också dyka upp (Beck 1998). Det ger också goda möjligheter att skräddarsy metoden efter omständigheterna.

För- och nackdelar

XP togs fram som en metod för små grupper och att använda metoden i mycket stora grupper kan vara en utmaning (Kalermo & Rissanen 2002). Med små grupper menas de som har färre än tio personer (Highsmith 2000), men Taber och Fowler (2000) rapporterade om ett lyckat XP-projekt med 50 inblandade.

Taber och Fowler (2000) upplevde många fördelar med XP:

- uppskattningarna av hur mycket som kunde åstadkommas under en iteration var väldigt exakta vilket förenklade planeringen och gjorde den realistisk,
- varje månad gjorde de demonstrerbara framsteg och det var aldrig några problem med att se hur långt projektet nått, samt att
- kommunikationen inom gruppen och mot kunden förbättrades avsevärt och alla visste hela tiden vad de närmaste målen var.

Mänsklig process

Beck (1998) menade att programvaruutveckling var en mänsklig process och att XP manar till att se utvecklarna som människor istället för maskiner. I en undersökning utförd av Kalermo och Rissanen (2002) såg de att när en grupp duktiga utvecklare arbetar utan att hålla sig till en viss process och istället förlita sig på sin erfarenhet, använde de sig instinktivt av samtliga principer ur manifestet för de lättroliga metoderna. Detta tyder på att de lättroliga metoderna har uppnått sitt mål att utforma en process anpassad efter utvecklarna.

Ett projekt utfört vid ett universitet i Valencia visade att studenter utan tidigare erfarenheter från liknande processer uppfattat XP som en mycket enkel metod och principerna som intuitiva och sunda (Letelier et al. 2003).

Taber och Fowler (2000) rapporterade från ett projekt, som sedan åtta månader drivs med en något modifierad form av XP, bland annat att lagandan hos utvecklarna hade skjutit i höjden. Ett annat projekt som utfördes av Ambu och Gianneschi (2003) visade att XP skapade stor entusiasm i utvecklingsgruppen.

ISO 9000

Ett sätt att locka kunder är att certifiera sin utvecklingsprocess enligt en kvalitetssäkrande specifikation och en sådan är ISO 9000.

På många punkter stämmer XP väl överens med specifikationen, exempelvis att metoden minimerar risker, är kundorienterad och säkrar kvaliteten genom intensiv testning – däremot saknas dokumenterade kravspecifikationer och andra utförligt beskrivande dokument (Nawrocki et al. 2002).

Företag som använder sig av XP kan således få svårt att erhålla en ISO 9000-certifiering. Dock anses den standarden kräva för mycket dokumentation och vara för byråkratisk (Nawrocki et al. 2002). Det finns andra certifieringar som bättre passar ihop med XP, exempelvis CMM.

Forskarutveckling

Långsiktig forskning ägnar sig oftast åt omogna tekniker som har långt kvar innan de blir kommersiellt gångbara. Då finns ingen kund att utveckla programvara för annan än

forskaren själv. Wood och Kleb (2002) utvecklade programvara som enbart de själva skulle komma att använda och de turades om att agera kund och svara på frågor. De fann att ett framtvingat användarorienterat perspektiv hjälpte till att fokusera arbetet, även om det var svårt att tänka på krav utan att samtidigt beakta kostnaden för att uppfylla dem.

För att kunna använda XP i en forskningsmiljö krävs att ett flertal principer modifieras. Att exempelvis arbeta heltid med programvaruutveckling är oftast ett problem då forskare har andra uppgifter och ansvar. Korta iterationer är ofta en omöjlighet, då mycken forskning arbetar långsiktigt och utvärderas väldigt sällan.

I mycket små utvecklargrupper krävs ännu mer att man anpassar XP, då man exempelvis kanske inte har en projektledare. Det är också oklart vilka uppgifter som kan utföras på egen hand med gott resultat och vilka som ger dåligt resultat (Wood & Kleb 2002).

Trots alla svårigheter lyckades Wood och Kleb (2002) att ensamma implementera sitt projekt genom att blint följa XP:s värderingar och så långt det var möjligt använda sig av alla dess principer.

Parprogrammering

Parprogrammering har undersökts bland annat av Williams et al. (2000), som kom fram till att 96% av de professionella programmerarna tyckte mer om sitt arbete när de programmerade i par än ensamma och att 90% av universitetsstudenterna föredrog parprogrammering. Genom sitt samarbete förbättrar programmerarna den gemensamma problemlösningsprocessen och tack vare trycket som uppstår höjs motivationen att slutföra uppgifterna i tid – resultatet blir att systemet blir färdigt snabbare och innehåller färre buggar (Williams et al. 2000).

Genom att man roterar och byter partner sprids kunskapen om hur systemets olika delar är uppbyggda och hänger ihop. På detta vis kommunicerar utvecklarna i detalj om aktuella val för design och implementation (Maurer 2002). Det kan dock vara otillräckligt vid mycket stora projekt om annan dokumentation saknas, vilket kan ge problem (Kaleramo & Rissanen 2002).

De flesta programmerare som inte har programmerat i par är till en början starkt kritiska till konceptet och anser att det är ett slöseri med resurser (Williams et al. 2000). Beck (1999) konstaterade också att XP är en social aktivitet som inte alla kan eller vill lära sig.

Distribuerade utvecklingsgrupper

Under 1990-talet dök olika virtuella organisationer för programvaruutveckling upp, där utvecklarna var utspridda över hela världen (Maurer 2002). Detta går inte att räkta av förena med XP, då metoden på många sätt utgår från att hela utvecklingsgruppen befinner sig på samma fysiska plats.

Det går exempelvis inte att på traditionellt sätt ägna sig åt parprogrammering på distans och det är orimligt att tro att kunden ska kunna ha representanter på alla de platser där utvecklarna befinner sig. Avståndet och fåtaliga möten ansikte mot ansikte kan också vara en grogrund till kommunikationsproblem.

Som en hjälp kan man använda sig av tekniken för att hålla ihop en grupp, exempelvis genom videokonferenser eller helt enkelt telefonsamtal. Visserligen kräver XP att samtliga utvecklare befinner sig på samma kontor, men då detta rimligen kan

antas vara enbart för att förbättra kommunikationen kan man byta ut detta mot tekniska hjälpmedel. Dock kan man inte avhjälpa bristen på informella samtal som uppstår spontant.

Underhåll

Underhåll av programvara kan definieras som processen att förändra ett tidigare levererat system (Sommerville 2001). I XP utgår man från att krav som ställts på systemet kommer att förändras så småningom och därför kan man säga att förändring är det konstanta tillståndet i ett XP-projekt. Det kan tolkas som att XP är anpassat för att underhålla programvara.

Ambu och Gianneschi (2003) skrev om sitt utvecklingsförsök med XP att varje bugg snabbt kunde hittas och åtgärdas, vilket drastiskt minskar kostnaderna. Dessutom gav i deras försök användandet av tester lägre underhållskostnader.

Dock finns ett problem med avsaknaden av dokumentation då utvecklarna lätt glömmer bort detaljer i kraven som tidigare ställts, vilket kan leda till att systemet blir sämre sett ur kundens perspektiv. Detta kan inträffa redan under den ursprungliga utvecklingen av systemet, men då man en tid efter leverans får i uppdrag att underhålla programvaran får man än större problem med att på detaljnivå sätta sig in i det.

Slutsats

Målet med denna uppsats var att försöka visa vad XP innebär och ge insikt i hur metoden fungerar i olika sammanhang samt att avhandla några av dess styrkor och svagheter.

Fyra grundpelare bygger upp XP; kommunikation, enkelhet, feedback och mod. På dessa vilar ett antal principer som mer konkret beskriver hur utveckling med metoden går till. Principerna är inte fixa och kan förändras om en utvecklingsgrupp så vill.

En för denna uppsats negativ följd av att principerna får förändras är att gränsdragningen mellan metoder suddas ut. Hur många av principerna kan åsidosättas eller gravt förvanskas innan man kan säga att det inte längre är XP? Faktum är att det inte finns några rapporter alls från projekt där metoden har följts fullt ut (Abrahamsson et al. 2002).

Den största fördelen med att använda XP är att det är en processmodell som verkar naturlig för många utvecklare, och att den genererar inspiration och entusiasm i gruppen. En del av detta kan parprogrammering sägas vara ansvarig för, som i sin tur minskar antalet buggar och snabbar upp utvecklingen.

På den negativa skalan dominerar farorna med underhåll av levererad programvara då dokumentationen som medföljer systemet är minimal. Efter en tid finns inte längre någon utvecklare som minns hur och varför källkoden ser ut som den gör. Detta gäller dock endast i teorin och XP har visats vara bra på att förhindra dessa svårigheter med hjälp av *refactoring* och ständiga tester – då det saknas praktisk erfarenhet är det svårt att uttala sig om hur det egentligen ligger till.

Bristen på dokumentation gör dock att det blir svårt för programvaruföretagen att certifieras enligt standarden ISO 9000, vilket kan ses som en förlorad konkurrensfördel.

Teoretiskt sett går XP inte att använda inom långsiktig forskning med väldigt få utvecklare, men en undersökning har visat att det faktiskt är möjligt och att det dessutom fungerar bra. Inte heller distribuerade utvecklingsgrupper ska gå att kombinera med XP, men med vissa tekniska hjälpmedel bör det gå i alla fall hjälpligt. Dock saknas det också här praktisk erfarenhet.

XP är en ung och lovande metod som också kritiserats för många av dess mer extrema drag. Men precis som tidigare nämnts är den inte färdigutvecklad och om den får mogna några år till kommer den säkerligen röna stora framgångar inom hela industrin.

Referenser

- Abrahamsson, Pekka, Salo, Outi, Ronkainen, Jussi & Warsta, Juhani (2002). *Agile Software Development Methods. Review and Analysis* [Elektronisk]. Espoo, Finland: VTT Electronics. (VTT Publications 478). PDF-format. Tillgänglig: <<http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf>>. [läst 2003-12-07].
- Ambu, Walter & Gianneschi, Fabrizio (2003). Extreme Programming at Work [Elektronisk]. *Lecture Notes in Computer Science*, vol. 2675, ss. 347–350. PDF-format. Tillgänglig: Springer/LINK. [läst 2003-12-07].
- Beck, Kent (1998). Extreme Programming: A Humanistic Discipline of Software Development [Elektronisk]. *Lecture Notes in Computer Science*, vol. 1382, ss. 1–6. PDF-format. Tillgänglig: Springer/LINK. [läst 2003-11-19].
- Beck, Kent (1999). Embracing Change with Extreme Programming [Elektronisk]. *Computer*, vol. 32:10, ss. 70–77. PDF-format. Tillgänglig: IEEE Xplore. [läst 2003-11-24].
- Beck, Kent, Beedle, Mike, van Bennekum, Arie, Cockburn, Alistair, Cunningham, Ward, Fowler, Martin, Grenning, James, Highsmith, Jim, Hunt, Andrew, Jeffries, Ron, Kern, Jon, Marick, Brian, Martin, Robert C., Mellor, Steve, Schwaber, Ken, Sutherland, Jeff & Thomas, Dave (2001). *Manifesto for Agile Software Development* [Elektronisk]. Tillgänglig: <<http://www.agilemanifesto.org/>>. [läst 2003-12-09].
- Elssamadisy, Amr & Schalliol, Gregory (2002). Recognizing and Responding to "Bad Smells" in Extreme Programming [Elektronisk]. Proceedings of the 24th international conference on Software engineering; 19–25 maj 2002; Orlando, Florida. ss. 617–622. Tillgänglig: ACM. [läst 2003-12-07].
- Charette, Robert (2001). The Decision Is In: Agile Versus Heavy Methodologies [Elektronisk]. *e-Project Management Advisory Service*, vol. 2:19. PDF-format. Tillgänglig: <<http://www.cutter.com/freestuff/apmupdate.pdf>>. [läst 2003-12-10].
- Fowler, Martin (2000). *The New Methodology* [Elektronisk]. PDF-format. Tillgänglig: <<http://www.thoughtworks.com/library/newMethodology.pdf>>. [läst 2003-12-07].
- Highsmith, Jim (2000). Extreme Programming [Elektronisk]. *e-Business Application Delivery*, februari 2000. PDF-format. Tillgänglig: <<http://www.cutter.com/freestuff/ead0002.pdf>>. [läst 2003-12-08].
- Kalermo, Jonna & Rissanen, Jenni (2002). *Agile Software Development in Theory and Practise* [Elektronisk]. University of Jyväskylä, Finland. PDF-format. Tillgänglig: <http://www.cs.jyu.fi/sb/Publications/KalermoRissanen_MastersThesis_060802.pdf>. [läst 2003-12-07].
- Letelier, Patricio, Canós, José H. & Sánchez, Emilio A. (2003). An Experiment Working With RUP and XP [Elektronisk]. *Lecture Notes in Computer Science*, vol. 2675, ss. 41–46. PDF-format. Tillgänglig: Springer/LINK. [läst 2003-11-24].

- Martin, Robert C. (2001). *Agile Development: Principles, Patterns, and Process* [Elektronisk]. Prentice Hall. PDF-format. Tillgänglig: <<http://www.objectmentor.com/resources/articles/agileProcess.pdf>>. [läst 2003-12-07].
- Maurer, Frank (2002). Supporting Distributed Extreme Programming [Elektronisk]. *Lecture Notes in Computer Science*, vol. 2418, ss. 13–22. PDF-format. Tillgänglig: Springer/LINK. [läst 2003-11-19].
- Nawrocki, Jerzy R., Jasiński, Michal, Walter, Bartosz & Wojciechowski, Adam (2002). Combining Extreme Programming With ISO 9000 [Elektronisk]. *Lecture Notes in Computer Science*, vol. 2510, ss. 786–794. PDF-format. Tillgänglig: Springer/LINK. [läst 2003-11-24].
- Newkirk, James (2002). Introduction to Agile Processes and Extreme Programming [Elektronisk]. Proceedings of the 24th international conference on Software engineering; 19–25 maj 2002; Orlando, Florida. ss. 695–696. PDF-format. Tillgänglig: ACM. [läst 2003-12-07].
- Sommerville, Ian (2001). *Software Engineering*, sjätte upplagan. USA: Addison-Wesley.
- Taber, Cara & Fowler, Martin (2000). An Iteration in the Life of an XP Project [Elektronisk]. *Cutter IT Journal*, vol. 13:11. PDF-format. Tillgänglig: <<http://www.thoughtworks.com/library/IterationXP.pdf>>. [läst 2003-12-14].
- Williams, Laurie, Kessler, Robert R., Cunningham, Ward & Jeffries, Ron (2000). Strengthening the Case for Pair Programming [Elektronisk]. *IEEE Software*, vol. 17:4, ss. 19–25. PDF-format. Tillgänglig: IEEE Xplore. [läst 2003-11-19].
- Wood, William A. & Kleb, William L. (2002). Extreme Programming in a Research Environment [Elektronisk]. *Lecture Notes in Computer Science*, vol. 2418, ss. 89–99. PDF-format. Tillgänglig: Springer/LINK. [läst 2003-11-24].