Master Thesis in Informatics

Studie av Brooks Lag och Kirurgteamsmetoden med hänsyn till öppen källkod

Victor Bergöö, Daniel Gedgaudas
Göteborg, Sweden 2007

Department of Applied Information Technology

REPORT NO. 2007:72

Studie av Brooks Lag och Kirurgteamsmetoden med hänsyn till öppen källkod

Victor E. Bergöö
Daniel M. Gedgaudas

Department of Applied Information Technology
IT UNIVERSITY OF GÖTEBORG
GÖTEBORG UNIVERSITY AND CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2007

A Study of Brooks Law and the Surgical Team in regard to Open Source

Victor E. Bergöö
Daniel M. Gedgaudas

Report no 2007:72
ISSN: 1651-4769
Department of Applied Information Technology
IT University of Göteborg
Göteborg University and Chalmers University of Technology
P O Box 8718
SE – 402 75 Göteborg
Sweden
Telephone + 46 (0)31-772 4895

[Department of Applied Information Technology]
Göteborg, Sweden 2007

Studie av Brooks lag och Kirurgteamsmetoden med hänsyn till öppen källkod

Victor E. Bergöö

Daniel M. Gedgaudas

Department of Applied Information Technology

IT University of Göteborg

Göteborg University and Chalmers University of Technology

ABSTRAKT

Uppsatsens behandlar de speciella fördelar och nackdelar som uppstår när utvecklare släpper kod under öppen källkod. Framförallt studeras problematiken med Brooks lag som säger att när fler personer läggs till i ett försenat utvecklingsprojekt blir projektet ännu mer försenat. Denna problematik studeras utifrån ett öppet källkodsperspektiv där utvecklare utanför projektet kan sända in kod och förslag på förändringar när som helst under projektets gång. Uppsatsen försöker att utifrån utvecklingsmetoden kirurgteamet, formulerad av Fredrick Brooks, finna en lämplig metod där utvecklingsarbetet går så smidigt som möjligt utan att försaka de fördelar det innebär att kunna ta emot hjälp utifrån. Uppsatsen är framförallt en kvalitativ litteraturstudie där fokus ligger på att studera och sammanställa befintlig litteratur för att dra generella slutsatser utifrån denna.

Rapporten är skriven på Svenska

Keywords:
Open Source, Free Software,
Brooks law, Surgical team, Collaboration.

Förord

I förordet vill vi tacka de personer som på olika sätt hjälpt till med att färdigställa denna uppsats genom att generöst dela med sig av sin tid och kunskap samt visat intresse för det arbete vi gjort. Det är tydligt att de ämne vi valt har väckt tankar och frågor hos många vilket har varit väldigt roligt.

Speciellt tack till vill vi ge till Kjell Engberg vår handledare som lagt ned mycket tid och varit till stor hjälp under hela arbetet med uppsatsen.

Göteborg 2007-05-28

Victor Bergöö

Daniel Gedgaudas

Innehållsförteckning

1. Introduktion

1.1. Problemområde

Idag har öppen källkod blivit en kraft att räkna med på allvar och alla större företag tvingas idag att finna ett sätt att förhålla sig till detta. Vi antar även att detta gäller för mindre företag vilka även de borde se över hur ställer sig i den här frågan. Vilka olika krav och möjligheter innebär det att börja utveckla öppen källkod och på vilket sätt innebär det en förändring? Det är viktigt att dels förstå de olika krafter som verkat för att skapa den situation vi har idag där användandet av öppen källkod hela tiden ökar.

1.1.1. Vad innebär öppen källkod?

Vad är då öppen källkod eller fri mjukvara för någonting egentligen? Kort kan det beskrivas som att alla tillåts studera den kod som ligger bakom själva programvaran.

> *”Låt mig göra en jämförelse mellan program och matrecept. Båda två är listor över saker som skall göras med regler som avgör när man är klar eller skall upprepa något. I slutändan så får man ett resultat. Lagar man mat så brukar man byta recept med sina vänner och ofta ändrar man i dem också. Om man ändrar något och vännerna gillar maten så kanske man ger vännerna den nya versionen utav receptet. Tänk er en värld där det inte går att ändra i receptet för att någon har gett sig fanken på att det inte skall gå att ändra. Och om man delar med sig av receptet så blir man kallad pirat och man riskerar att hamna i fängelse.”* [The Code, 2001]

Detta är tanken bakom fri mjukvara beskriven av grundaren bakom den fri mjukvaraörelsen Richard Stallman. Stallman är även grundare av en organisation, Free Software Foundation, som arbetar för att föra ut idéen om användandet av fri mjukvara och är även de som tagit fram den licens som fri mjukvara oftast släpps under.

I motsats till att släppa sin kod fri kan utvecklare välja att bara göra binärerna tillgängliga. För att ett program ska gå att köras på en dator krävs det nämligen att det skrivs om från läsbar kod till binär sådan, vilken kan läsas och förstås av datorn. Detta görs med hjälp av en kompilator. Att bara släppa dessa binärer kallas för att släppa koden proprietär och då är koden alltså oläsbar för människor.

1.1.2. Free Software

Free Software, eller fri mjukvara, är en term som skapats av Free Software Foundation där

tanken är att vem som helst skall kunna skapa, ändra och distribuera mjukvara med få eller helst inga restriktioner om hur koden får användas. Free Software Foundation vill skapa en miljö där utvecklare och användare kan titta på och ändra i källkoden om de inte är nöjda med den eller saknar någon funktion. Om de sedan tycker att funktionen är bra så kan de skicka patchar eller förslag till dem som utvecklar mjukvaran för utvärdering. Ifall ändringarna godkänns så kommer de att inkluderas i en kommande version. Om dessa inte skulle tas med, men utvecklaren fortfarande tycker att de ändringar som gjorts var användabara och bra kan denne själv släppa dem och låta andra ta del av dessa. Det går även att ta en produkt och på egen hand vidareutveckla.

För att garantera att mjukvaran förblir fri har Free Software Foundation tagit fram en speciell licens, GNU General Public License (GPL), som ger vem som helst rätt att ändra i koden, men ställer samtidigt krav på att det som utvecklas från en produkt släppt under GPL också släpps under denna licens. I praktiken innebär det att det inte går att ta den fri mjukvara och göra den stängd.

1.1.3. Open Source

Termen Open Source, eller öppen källkod, skapades 1998 för att vissa tyckte att Free Software Foundations tänkande om att ”fri som i yttrandefrihet, inte fri som gratis öl” inte riktigt verkade tilltala företag och andra att använda sig av öppenheten. De tyckte inte att detta uttryck gagnade utveckling där källkoden släpptes fritt så att vem som helst kan ta del av den, utan snarare hindrade detta. Till stor del handlade det om att termen ”fri programvara” lätt blev associerad med fientlighet mot intellektuell äganderätt och ett allt för dogmatiskt tänkande, något som avskräckte många företag. Därför skapade de Open Source Innitiativ vilka ville klä tankesättet i nya kläder och ge det ett bättre rykte än vad Free Software Foundation hade gjort. Tanken var att skapa något som klingade bra i både chefers och investerares öron. Open Source Innitiativ ville få bort den stereotypa bilden av den fria programvarans användare och bredda dess målgrupp, men samtidigt ville de att fördelarna med den öppna källkoden skulle tas till vara. Open Source Innitiativ ville frångå fokuset på filosofin och fokusera på de praktiska fördelarna. [Raymond, 2001]

Detta lyckades de också med och det var delvis tack vare att Eric S. Raymond skrev sin uppsats Katedralen och Basaren som gjorde att Open Source Innitiativ grundats och att användandet av öppen källkod började ta fart bland företagen. Idag ser vi att mjukvara utvecklad och släppt under öppen källkod verkligen blivit en kraft att räkna med.

1.1.4. Katedralen och Basaren

För tio år sedan, 1997, skrev Eric S. Raymond uppsatsen Katedralen och Basaren. Med denna gjorde han ett försök till att vända upp och ned på hur världen ser på mjukvaruutveckling och samtidigt slå ett slag för öppna standarder. Raymond hävdar att det är bra att kunna engagera många människor i ett projekt och att de som driver projektet hela tiden ska uppmuntra människor att engagera sig det. Han menar i uppsatsen att på detta vis skrivs det bättre program med färre brister. Något han visar genom att ge exempel på hur utveckling med en öppen källkod går till och vilka fördelar han anser detta ha. Han tar främst upp ett eget projekt för att hantera e-post som han drivit. Även utvecklandet av Linux-kärnan tar han upp som ett exempel på hur det går att skapa bra kod genom att engagera många i projektet. I boken sätter han upp en rad regler för hur arbete med öppen källkod bör genomföras för att bli framgångsrikt.

1.1.5. Brooks lag

Raymonds påstående om att det är en stor fördel att kunna ha en väldigt bred bas av utvecklare verkar dock gå stick i stäv med en av de äldsta sanningarna inom mjukvaruutveckling. Nämligen den som formulerats av Fredrick Brooks och som kommit att kallas för Brooks lag. Fredrick Brooks sammanfattar sin lag på följande vis.

> *”Att lägga till manskap till ett redan försenat mjukvaruprojekt kommer att göra det ännu mer försenat” - Fredrick Brooks*

Denna lag formulerade Fredrick Brooks för första gången i The Mythical Man-Month, 1975. Brooks lag säger alltså att om det läggs till mer manskraft till ett försenat mjukvaruprojekt tar det ännu längre tid innan det blir färdigt. Det tar nämligen mycket tid i anspråk att få de nya att komma in och lära sig förstå projektet, vilket de måste göra innan de kan vara produktiva och lönsamma för organisationen. Dels kommer de roller som projektdeltagarna redan har delvis att stöpas om då de nya ska ta sin plats i projektet. De nya måste läras upp och förstå det arbete som redan gjorts i projektet. Därför måste de eventuellt avvara en av de äldre projektdeltagarna och sätta denne på att lära upp de nya. Detta leder till att projektet under en period inte har tillgång till de nya projektdeltagarna, dessutom så går de även miste om en redan kunnig projektdeltagaren som nu får agera som lärare. [Brooks, 1995]

Det Raymond däremot säger är att det är bra att kunna engagera många i projektet vilket alltså verkar gå stick i stäv med vad Brooks lag säger. Därför bör många av de projekt som idag existerar

under öppen källkod och som är framgångsrika ha en bred bas med utvecklare som bidrar med tid och kunskap. Dessa projekt borde ha funnit en modell för utvecklingsarbetet där de på ett bra sätt hanterar problematiken med Brooks lag samtidigt som de kan ta till sig av all den hjälp som de erbjuds från utvecklare runt om i världen. Vi vill därför studera Brooks lag och de möjligheter för utveckling som öppen källkod för med sig i syfte att finna en metod där båda dessa faktorer faller på plats.

1.1.6. Kirurgteamsmetoden

Brooks diskuterar även flera olika metoder för utveckling som går att använda sig av för att försöka göra programutvecklingen så effektiv och bra som möjligt. En av de metoder som han tar upp handlar om att centerera den absolut bästa programmeringskunskapen i en liten kärna och allra helst i en enda programmerare. Denne har som uppgift att stå för den största delen av utvecklingen medan de andra hjälper till med saker runt omkring. Detta kallar Brooks för kirurgteamet då det påminner om hur kirurger arbetar när de opererar.

1.2. Syfte

För att bestämma målet och innehållet har vi utifrån det problemområde vi definierat valt att formulera ett ämne att fördjupa oss i. Då vi i uppsatsen antar att öppen källkod är någonting som har fått en verklig genomslagskraft och blivit något de flesta utvecklare måste förhålla sig till väljer vi att tillåta en viss bredd i frågan. Detta för att vi anser väldigt många faktorer spelar in och därför vill vi ge utrymme för att kunna skapa en rättvisare och bredare bild. Vi vill se hur de argument som fördes fram i början för öppen källkod står sig idag 10 år senare för att få en fördjupad bild av de möjligheter och risker som öppen källkod kan innebära samtidigt som vi studerar vårt område.

Vi antar i ett inledande skede att den metod som Brooks kallar för kirurgteamet är en lämplig metod och väljer därför att fokusera våra studier kring denna. Vi vill alltså finna en utvecklingsmetod som minimerar den problematik som Brooks lag eventuellt ställer till med.

Vi vill med vårt arbete studera och utvärdera de påståenden som kom från förespråkarna av öppen källkod nu tio år senare. Framförallt rörande hur problematiken med Brooks lag hanteras. Vi vill också försöka komma fram till i vilken grad den gamla sanningen med Brooks lag om utveckling har förändrats eller om den fortfarande består. Detta gör vi utifrån att studera problematiken och söka efter en metod som hanterar denna på ett bra sätt.

1.3. Avgränsningar

Öppen källkod är ett begrepp som används tämligen slarvigt idag och det finns många olika licenser med varierande grad av öppenhet. Det är ingenting som direkt bör påverka området vi valt att studera, men vi kommer att titta på de vedertagna definitionerna som helhet då licensformerna i sig inte påverkar utvecklingsmetoden. Däremot behöver vi en övergripande bild av vad öppen källkod innebär.

Vi vill lägga vårt fokus på hur utvecklandet av mjukvara påverkas av den öppna källkoden och försöka att undvika att inte fastna i rent ekonomiska fördelar eller nackdelar. Vi kommer dock i den mån de ekonomiska frågor är intressanta för själva utvecklingsarbetet att ta upp dessa då den mest kostnadseffektiva metoden för utveckling naturligtvis är högst intressant och relevant. Vårt fokus syftar dock till att ligga på utveckling av koden och programmerarnas arbetsmetoder och vi ska i den mån det är möjligt försöka att inte blanda in ekonomiska fördelar och nackdelar.

2. Metod

Här förklarar vi och motiverar det vetenskapliga angreppssätt vi valt för att genomföra vår studie. Vi förklarar även hur vi gått till väga i syfte att möjliggöra återupprepning av den process vi använt oss av.

Backman ställer upp tre olika former av rapporter. Det är dels den traditionella rapporten där undersökarens uppgift är att vara en observatör som betraktar och mäter skeenden i objektet utifrån. Objektet som studeras ska här betraktas objektivt. Här kan det sägas att forskaren ska vara som en fluga på väggen. Dels tas den kvalitativa rapporten upp. Denna betraktar objektet mer subjektivt som en social konstruktion. Här studeras hur människan uppfattar att objektet beter sig snarare än hur objektet beter sig. Den sista formen av rapport är forskningsöversikten som syftar till att sammanfatta existerande forskning och göra generaliseringar utifrån denna. Detta angreppsätt försöker ha ett helhetsperspektiv och inte begränsas av ett enstaka fall. Syftet med denna form av rapport kan vara olika. Det kan vara att beskriva eller förklara ett fenomen, eller vara undersökande. [Backman, 1998]

De undersökande studierna har till syfte att få grundläggande förståelse för ett fenomenets natur. Då vi i denna rapport försöker studera öppen källkod och påståenden om dess natur och har vi valt denna typ av angreppssätt med en kvalitativ grundsyn.

För att kunna studera det problemområde som vi har identifierat har vi valt att göra en litteraturstudie över existerande forskning med ett kvalitativt angreppssätt. Vi valde den kvalitativa studien då vår tidsram för arbetet varit tjugo veckor. Vi ansåg därför att detta angreppsätt var det mest sannolika för att få ett intressant och bra resultat. En kvantitativ studie hade sannolikt varit svårare att genomföra och krävt betydligt mer tid. Vi anser att den kvalitativa metoden gav oss möjlighet att på ett bättre sätt fördjupa oss i vår frågeställning.

Det skulle ta avsevärd mycket mer tid i anspråk att identifiera de företag som har en intressant utvecklingsmetod och som också var beredda att avsätta tid för intervjuer. Därför valde vi att fokusera på att studera befintlig litteratur snarare än att genomföra några intervjuer.

Studien kan sägas vara hermeneutisk då vi valt att studera och tolka vald litteratur snarare än att skapa någon objektiv och absolut kunskap utifrån traditionellt positivistiska metoder.

För att kunna få en bra förståelse för det område vi valt att studera börjar vi vår undersökning

med den litteratur vi sedan tidigare känner till inom ämnet och som har haft stort inflytande på utveckling i allmänhet och på senare tid även inom öppen källkod. Dessa var framför allt Eric S. Raymonds Katedralen och Basaren och Frederick P. Brooks The Mythical Man-Month: Essays on Software Engineering. Utifrån denna grund har vi i sedan börjat leta oss vidare för att fördjupa oss i vår frågeställning.

När vi letat artiklar och böcker har vi framförallt använt oss av GUNDA som är Göteborgs Universitets gemensamma bibliotekskatalog. Vi har även till viss del använt oss av Google. De sökord vi framförallt använt oss av är ”open source”, ”free software”, ”Brooks Law”, ”collaborative”. En stor del av de artiklar vi läst kommer även från Massachusetts Institute of Technology Free/Open Source Research Community vilket är en stor samling vetenskapliga artiklar som på olika sätt berör öppen källkod. Vi har även genom Open Source Initiative funnit många artiklar.

Med hänsyn till rapportens omfattning på 20 veckor och syftet den fyller i utbildningen har ett urval av böcker och artiklar gjorts. Detta urval är subjektivt och färgat av den förförståelse av området som vi hade redan innan vi började rapporten.

3. Teori

3.1. Definitioner

3.1.1. Introduktion

Utveckling inom den öppna källkodsvärlden brukar skilja sig från den i den proprietära mjukvaruutvecklingen i och med att det oftast finns en kärna med utvecklare som uför huvudarbetet. Utöver detta kan projektet använda sig av frivilliga användare som hjälper till med arbetet genom att till exempel skicka in lösningar på fel eller rapportera om dem. De kan även skicka in kod som kanske skapar nya funktioner eller förbättrar redan existerande funktionalitet i programmet. Tanken är att alla skall fritt kunna använda och förbättra de program som de tycker är bra och är värda att spara. För att kunna göra detta måste användarna ha rätt att studera den källkod som ligger bakom programmet och på så sätt kunna finjustera det till att passa sina behov. Vidare kan de även dela med sig av sina ändringar till andra. På så sätt menar förespråkarna att användaren blir friare och även att koden blir bättre. [Raymond, 2001]

3.1.2. Öppen Källkod

The Open Source Initative definierar hur en öppen källkodslicens skall se ut genom att ställa upp tio olika punkter:

- Alla skall kunna ha rätt att återdistribuera ett program utan att behöva betala några avgifter till den ursprungliga skaparen. Licensen skall inte heller förbjuda att vem som helst får sälja programmet vidare i till exempel en distribution så som Red Hat eller Mandriva.
- Källkoden måste finnas tillgänglig för att kunna distribueras som både källkod och som kompilerad dito. Om källkoden inte skickas med programmet skall det finnas väl dokumenterat var det går att få tag på denna.
- Det måste gå att skapa nya versioner utav programmet och det måste kunna göras ändringar i programmet.
- Möjligheten finns att i licensen förbjuda att det inte skall gå att göra ändringar i källkoden, men då måste det tillåtas att användaren kan applicera patchar när programmet väl skall kompileras. Denna regeln finns för att det skall gå att värna om författarens

integritet.

- Diskriminering får inte förekomma gentemot en enskild person eller en grupp av personer.
- Licensen får inte göra det möjligt att förbjuda användning av programmet inom ett visst område. Det vill säga att det får till exempel inte förekomma några regler om att programmet får användas överallt förutom i undervisning i skolan.
- De rättigheter som licensen ger måste även ges till dem som programmet återdistribueras till utan att de lägger till nya licenser ovanpå den sagda.
- Licensen gäller det program samt de komponenter som programmet innehåller.
- Det måste gå att använda andra licenser runtomkring den som ens program använder sig av, det skall alltså inte gå att till exempel förbjuda användningen av proprietära program i samma miljö som ens eget program används.
- Licensen måste vara generell och får inte vara begränsad till en viss plattform eller teknologi. [OSI, 2006]

3.1.3. Fri mjukvara

Free Software Foundation definierar fri källkod genom fyra friheter:

- Fri mjukvara ger friheten att köra program för vilket ändamål de själv vill.
- En fri mjukvarulicens ger även friheten att studera och använda programmet efter hur användaren själv vill ha det. Att ha tillgång till källkoden är en förutsättning för att kunna göra detta.
- Friheten att återdistribuera kopior så du kan hjälpa din nästa.
- Genom att källkoden är tillgänglig så får användaren även friheten att ändra och göra förbättringar i koden.

Själva tanken med fri mjukvara är att koden skall vara:

> *”Fri som i yttrandefrihet, inte fri som i gratis öl” -Richard Stallman [The Code, 2001]*

All fri mjukvara är öppen källkod, dock så är inte all öppen källkod fri mjukvara. Det är alltså en skillnad mellan fri mjukvara och öppen källkod. Den största skillnaden brukar vara att öppen källkod oftast kan tillåtas att modifieras och sedan släppas som proprietär medan den kod som uppfyller kraven för att kallas fri mjukvara har en begränsning i licensen som gör att den aldrig får släppas som stängd.[FSF, 2007]

För vår uppsats är skillnaden mellan fri mjukvara och öppen källkod inte relevant då båda uppfyller de krav på öppenhet som ryms inom vårt problemområde. Däremot är det viktigt ur ett historiskt perspektiv för att förstå vad öppen källkod är för någonting.

3.2. Katedralen och Basaren

Eric S. Raymond menar i sin uppsats Katedralen och Basaren att om utvecklarna ser sina användare som sin största tillgång så kommer de även bli det. Det är därför utvecklaren av öppen källkods jobb att få användarna att skicka in både lösningar på buggar och felrapporter, men även förslag i allmänhet. Det är alltså viktigt att utvecklaren inte bara skriver ett bra program utan även lyckas engagera sina användare i processen. Raymond förtydligar detta genom att beskriva den klassiska synen på utveckling som en katedral där bara den utvalda gruppen med utvecklare har tillgång till koden för programmet som utvecklas. Denna modell kallar han därför katedralmodellen. I kontras till denna klassiska syn har vi basarmodellen där all kod som utvecklas är synlig för vem som helst under hela utvecklingsprocessen. Det finns även fall inom den fria mjukvarurörelsen där de använt sig av en katedralmodell. Visserligen var källkoden fri att läsas av alla, men det var bara i samband med att nya stabila versioner som den blev tillgänglig. Exempel på stora och välkända fri mjukvaruprojekt som utvecklades på detta traditionella sättet är editorn GNU Emacs och kompilatorn GCC. Även om dessa projekt varit framgångsrika leder metoden inte enligt Raymond till att användarna får samma möjlighet att engagera sig i projektet som de skulle kunna ha.

För att visa upp värdet av att ha engagerade användare tar Raymond upp Linus Torvalds arbete med Linux-kärnan. Istället för den traditionella katedralmetoden valde Torvalds att genast tillkänna ge att han påbörjat sitt arbete med kärnan och en månad senare när arbetet nått en nivå där kärnan var körbar la han ut den för att alla skulle kunna testa och hjälpa till. Detta sätt att hela tiden dela med sig av sitt arbete kallar Raymond för basarmodellen och han anser detta kan leda till ett ökat deltagande i projektet från användarna.

Anledningen till att det tidigare betraktades som dåligt att lägga upp koden innan den nått en stabil version var, enligt Raymond, att det på detta vis skulle tvinga på användarna versioner med väldigt många buggar vilket skulle skrämma bort dem från projektet. Tanken var istället att användaren skulle se så få buggar som möjligt. Linus Torvalds gjorde tvärtom. Han gav ut sin kod så tidigt som möjligt och så ofta som möjligt och uppmuntrade användarna att bli en del av utvecklandet. Detta var enligt Raymond inte helt unikt i sig men den utgivningstakt som Torvalds använda sig av var det däremot.

Denna metod för utveckling visade sig fungera väldigt bra. Enligt Raymond beror inte Linuxkärnans framgångar så mycket på att den var speciellt innovativt eller att Torvalds är en ovanligt duktig programutvecklare. Det beror istället på att han var en duktig projektledare som kunde engagera sina medutvecklare och få folk över hela världen att rapportera in ny kod och buggar de stött på.

Buggletandet och problemlösandet är enligt Raymond en av denna utvecklingsmetods starka sidor. Genom att fler människor läser koden och använder programmet anser han att chansen att buggar kommer upp till ytan ökar markant. Lika så kommer chansen att någon finner lösningen på ett problem att öka då fler personer är bekanta med koden. Med tusentals människor som studerar koden kommer det alltid att finnas någon för vilken lösningen på ett problem är uppenbar.

> *”Du skulle inte lita på en vetenskaplig artikel ifall den inte hade blivit kontrollerad av andra experter inom området /../ och du kan inte heller lita på mjukvara som inte har blivit kontrollerad av experter. /../ Ifall du använder dig av öppen källkod så får du din kontroll, du får trovärdighet. Ifall din kod inte är öppen så får du inte kontrollen och du får inte trovärdigheten.” - Eric S Raymond [Leonard, 1998]*

För att få användare att hjälpa till med ett öppet källkodsprojekt, oavsett om det är något som skrivits på kammaren eller om det ett företag som ligger bakom så måste utvecklaren ha en produkt att visa upp. Den behöver inte vara färdig, dock måste det gå att visa det finns en tanke bakom projektet och en vilja att jobba vidare på det. Linus Torvalds började med att presentera sin idé och lade sedan upp den första stabila versionen och på så sätt fick alla andra en möjlighet att ta del av projektet och få en känsla av hur det kan hjälpa dem och hur de kan engagera sig i projektet.

Eric S. Raymond lägger i Katedralen och Basaren fram en rad andra viktiga punkter för hur det öppna källkodsprojekt ska bli framgångsrika. Först och främst ska syftet med projektet vara att tillfredställa ett behov som utvecklaren själv har. Detta då det kan vara svårt att engagera andra i ett

projekt som denne själv inte brinner för. Det är därför också viktigt att den ansvarige för projektet lämnar över det till någon annan som både brinner för projektet och är en kompetent utvecklare om denne själv tappar intresset. Det är därför också viktigt att den med huvudansvaret hela tiden uppmuntrar de som lägger ned tid på att hjälpa till med projektet och behandlar dem som om de vore medutvecklare. Detta är väldigt viktig eftersom Raymond hävdar att en av de stora fördelarna med utveckling inom öppen källkod är just mängden med utvecklare. Ytterligare en viktig egenskap som ledaren av ett öppen källkodsprojekt ska ha är förmågan att kunna identifiera bra idéer. Raymond påstår att det bästa efter att ha egna bra idéer är att kunna identifiera andras bra idéer och tillvarata dem. [Raymond, 2001]

3.3. Brooks lag

1975 så skrev Fredrick Brooks sin bok The Mythical Man-Month, en bok som av många kommit att betraktas som en av de viktigaste böckerna inom mjukvaruutveckling. I The Mythical Man-Month beskriver Brooks den lag som han kallar för Brooks lag. Denna lag säger att ifall det läggs mer manskap till ett redan försenat mjukvaruprojekt så kommer det bli ännu mer försenat. Brooks menar att för varje ny person som läggs till projektet så måste denne tränas upp och genom att göra detta förlorar projektet dels den tid personen som tränas upp, men även den tid som de personer som tränar den nye projektdeltagaren. Tid som skulle ha kunnat läggas på projektet istället. På detta sätt så halkar projektet ännu längre efter och det blir svårare att komma ikapp.

Brooks tar som exempel ett projekt som är planerat att ta 12 manmånader, en manmånad motsvarar 160 arbetstimmar, och det är 3 personer som ska arbeta på det. Projektet har 4 milstolpar som planeras in att vara nådda vid slutet av varje månad. Projektet planeras alltså vara färdigt på fyra månader. Men om det av någon anledning tar två månader att nå den första milstolpen, då måste projektledaren ta ställning till hur de ska fortsätta med projektet. Ifall det så skulle antas att det nästkommande milstolparna kommer att vara korrekta blir projektet antingen försenat eller så måste det då anställas två nya personer för att hinna med den uppsatta deadlinen. Detta ger att de har nio manmånaders arbete framför sig, men bara två månader. Antas det istället att alla milstolparna kommer ta två månader kommer projektet att kräva arton manmånader till och för att hinna detta på två månader behövs nio nya personer anställas.

Det uppstår dock ett problem. Oavsett hur kompetenta och duktiga de nyanställda må vara så kommer de behöva en inlärningsperiod och någon som lär upp dem hävdar Brooks. I fallet med att

anställa två nya personer antar Brooks att det krävs en månad för att komma in i projeketet. Då har projektet förlorat 3 manmånader på att lära upp dessa två nya, då de behöver en lärare. Till på köpet krävs det att arbetet som tidigare var uppdelat mellan tre personer delas upp mellan fem. Det kommer kanske leda till att delar av det som redan gjorts måste kastas för att passa in i den nya uppdelningen. Så vid slutet på den tredje månaden skulle det med Brooks exempel fortfarande finnas sju manmånaders arbete kvar att göra på fem personer. [Brooks, 1995]

3.4. Tredjepart

Fördelen med att använda öppen källkod är att det blir mycket lättare att få support på de program som användarna nyttjar menar Carolyn Kenwood. Då koden till programmen är öppen kan väldigt många fler företag läsa dem och därför också leverera support till programmen. Detta i sin tur leder till en konkurrens på support för produkten vilket ökar kvaliteten på den support som finns och även sänkta kostnader för support. Kvaliteten på supporten ökar också då företaget som säljer support kan få en djupare kunskap om hur programmet fungerar tack vare att de kan studera källkoden. På så sätt levereras det bättre support då deras förståelse för programmet blir större.

Kenwood gör antagandet att om det finns en efterfrågan att använda en produkt kommer också någon att leverera support på denna. Även om utvecklarna själva inte ger support till programmen längre kan efterfrågan på produkterna leva vidare och då kommer även andra supportföretag att kunna leverera support. På så vis skulle, enligt Kenwood, produkter som är släppta under öppen källkod kunna ha en längre livslängd än proprietära motsvarigheter. Supportföretag och andra skulle dessutom kunna fortsätta arbeta på koden till produkten vilket ytterligare kan förlänga dess livslängd. [Kenwood, 2001]

Med hjälp av öppen källkod så slipper användaren till stor del de inlåsningseffekter som stängd källkod kan skapa. Detta då denne själv har tillåtelse att gå in och ändra och modifiera källkoden vilket gör att om produkter som användes slutas utvecklas så kan andra ta vid och vidareutveckla koden. Det leder till en trygghet för framförallt företag och andra grupper som lägger ner mycket tid och pengar på att skapa en miljö där en viss typ av mjukvara används. [Statskontoret, 2003]

Brian Behlendorf menar att om ett företag som utvecklar ett program bestämmer sig för att släppa detta proprietärt så kan det bidraga till att det går emot programmets naturliga utveckling. Detta då den stängda koden inte kan läsas och utvecklas av lika många. Ett i grunden bra program

som används av många skulle kunna bli mycket bättre om fler förstod koden bakom programmet. Inlåsningen av koden som proprietär kan alltså vara till dålig gagn för andra företag även om det kan rent kortsiktigt kan vara bra för det utvecklande företaget då de får ett monopol på detta programmet. Detta skulle kunna ställa kunderna i positioner där de kan vara tvungna att byta plattform för att tjäna pengar i långa loppet då de hamnat i en beroendesituation till företaget som utvecklar programmet. Det kan även leda till att säkerhetshål och andra fel i programmet uppdateras senare då användarna är beroende av ett enda företags patchar. [Voices, 1999]

Runt många produkter som släpps som öppen källkod växer det ofta upp en community där användare och utvecklare delar med sig av tankar och erfarenheter kring produkten. De tenderar även att vara hjälpsamma mot nybörjare och skapar ofta guider och lägger ned tid på att hjälpa andra. I en studie av webbservern Apache från år 2000 kom Karim Lakhani och Eric von Hippel fram till att en av de viktigaste anledningarna till varför det finns så många personer som svarar på frågor och hjälper till att lösa andras problem var att dessa personerna själva fick den hjälpen när de var nybörjare. Nu när de inte själva längre är nybörjare vill de ge tillbaka den erfarenhet och kunskap som de fått till den communityn som de har fått hjälpen från. [Lakhani, von Hippel, 2000]

Under mars 2005 så avslutade Microsoft sin fria support för Visual Basic 6 till förmån för Visual Basic.NET. Detta gjorde att många utvecklare blev missnöjda med Microsoft då Visual Basic.NET inte är helt kompatibelt med sin föregångare. Resultatet av detta blev att många utvecklare som tidigare arbetade i Visual Basic valde att inte gå över till .NET-versionen av språket utan istället valt att gå till andra språk så som Java och C#. Detta berodde bland annat på att de kände en viss oro över att råka ut för liknande saker igen. [Taft, 2003][LaMonica, 2005]

I kontrast till detta kan 3d-animerings programmet Blender ges. År 2002 köptes det upp av dess community från de investerare som då ägde programmet när företaget som utvecklade Blender gick i konkurs. Blender har sedan släppts under den öppna licensen GPL för att vidareutvecklas med hjälp av frivilliga runt om i världen. [Blender]

3.5. Felgranskning

Genom att fler personer tittar på den kod som ingår i projektet så kommer fler buggar och säkerhetsrisker att komma upp till ytan då fler personer granskar koden. Det kommer att öka säkerheten markant menar Bruce Perens i en artikel.

Det kommer alltid att finnas människor som gör sitt bästa för att finna säkerhetsbrister i program och använda dessa för att ställa till problem på olika sätt. De som släpper koden stängd hoppas på att färre säkerhetshål ska hittas. Detta leder dock till att bara utvecklarna och skaparna av skadlig kod kommer ha kännedom om bristerna, menar Perens. Att istället släppa koden öppen skulle enligt honom göra att även folk med goda avsikter kan granska koden och upptäcka fel. På så vis kan fler fel komma upp till ytan och rättas till snabbare. [Perens, 2002]

Richard Stallman säger att alla människor gör fel och så även utvecklarna av mjukvara. Alla program som inte är väldigt banala kommer därför att innehålla buggar och säkerhetsrisker. Dock har användaren av proprietära program väldigt liten makt att själv identifiera dessa risker utan är mer utlämnad till utvecklarens förmåga att identifiera och rätta till brister. [O'Riordan, 2006]

I en artikel av Natalie Whitlock så pratar Eric S. Raymond om samma sak som Perens och Stallman är inne på. Raymond menar att inom den öppna källkodsvärlden så kan buggar och säkerhetbrister rättas till snabbare än inom de proprietära då fler undersöker koden. Det leder även till, enligt Raymond att det skapas en jämlikhet mellan dem som granskar koden med onda avsikter och dem som gör det för att göra koden säkrare, om koden släpps som öppen källkod där vem som helst kan titta på koden.

> *”Stängd källkod ger endast de elaka en chans att hitta säkerhetshålen. Med öppen källkod så får även de snälla en chans att leta efter dem med.” - Eric S. Raymond*

Dock så finns det även de som menar att den påstådda säkerheten som fås genom öppen kod inte är så bra som dess förespråkare försöker påstå. Det stora problemet enligt dessa är att det inte går att vet vilka det är som tittar på koden och ofta är det till största delen amatörer som egentligen inte vet så mycket om programmering och säkerhet.

Företag som släpper koden som proprietär mjukvara kan ha allvarliga säkerhetsrisker utan att användarna har en aning om dem och kan hålla dem hemliga. I April 2000 så upptäcktes det att det fanns en bakdörr, ett odokumenterat sätt att ta sig in i ett system, i Microsoft FrontPage. Denna bakdörr hade funnits där i fyra år utan att återgärdas. Detta kunde ske därför att koden släpptes som binärer utan möjlighet för användaren och andra att studera den. [Whitlock, 2001]

När Borlands databas Interbase blev öppen källkod så upptäcktes det att det fanns en bakdörr, med hårdkodat lösenord i databasen. Den bakdörren har funnits i databasen i minst sju år. [Poulsen, 2001]

Sedan går det inte alltid att veta ifall de som tittar på koden faktiskt letar efter säkerhetsfel, de kan i själva verket bara vara ute efter att plocka en viss bit kod från programmet. John Viega menar att de som skriver proprietär programvara oftast, när det väl görs säkerhetstester, har bättre och säkrare testmetoder än det som den öppna källkodsvärlden kan stoltsera med. John Viega menar att det finns en oro hos många att den öppna källkoden ofta utvecklas av ”hackare” snarare än ingenjörer som använder sig av vedertagna utvecklingsmetoder med ordentliga kravspecifikationer och analyser.

Det är också så att många av dem som läser och sätter sig in i koden till program är släppta som öppen källkod inte läser den i jakt på säkerhetsbrister utan är ute efter en möjlighet att bygga in ny funktionalitet. De som däremot letar säkerhetsbrister tenderar att göra det i de större programmen vilka ger större uppmärksamhet, då ett fel i ett mer välkänt program förmodligen ger detta snarare än ett fel i ett mindre känt program. Viega menar även att dessa tenderar att leta efter stora och uppenbara säkerhetsrisker och ignorerar mindre självklara fel. [Viega, 2004]

3.6. Kommunikation

Eric S. Raymond menar att om det finns ett medie som utvecklare av mjukvara kan använda för att kommunicera genom och skapar möjlighet för dem att kommunicera med personer utanför den närmsta gruppen av utvecklare, så bör de använda sig av denna möjlighet. Det mest uppenbara mediet för denna kommunikation är, enligt Raymond, Internet. Alla möjligheter som finns bör användas för att hålla kontakt med användarna. Det är enligt Raymond alltid bra att försöka få utomstående att bidraga med tankar och kod till mjukvaran. Dock så måste utvecklarna kunna ta tillvara på denna hjälp på ett så bra sätt som möjligt. För att detta ska vara möjligt måste de som leder projektet känna att de vill göra det och inte göra det av tvång, för annars blir det inte något bra resultat.

Linus Torvalds och Linux-kärnan är Raymonds exempel på värdet av att skapa ett community kring det som utvecklas för att få så mycket hjälp som möjlig med produkten. Det är enligt Raymond till väldigt stor hjälp för projektet om de utomstående på ett bra sätt kan engageras att bidraga till projektet.

Raymond menar vidare att det skulle vara svårt att för ett företag som utvecklar proprietär mjukvara att kunna betala samma antal personer som alla de människor som hjälpt till med hans

egna program Fetchmail, men ändå har de hjälpt honom. År 2000 så hade runt 800 olika personer skickat in buggfixar och funktioner till Fetchmail. Detta beror till en stor del av att det har skapats en community runt omkring programmet och att Raymond på ett bra sätt tagit tillvara de kunskaper som folk varit villiga att ställa till hans förfogande. Det är dock omöjligt att få någon att ställa upp om de inte känner sig uppskattade och att de får något av värde tillbaka för sitt arbete. Det är viktigt att förstå att det är människors tid och kunskap som ställs till ens förfogande och därför är det viktigt att dessa personer känner sig värdesatta. Annars kommer de inte vilja deltaga i projektet. Inte utan ekonomisk ersättning. Därför kan det vara bra att skapa ett utrymme där alla som bidragit uppmärksammas. [Raymond, 2001]

För att ett projekt ska kunna genomföras krävs det dock att alla kan kommunicera på ett bra sätt Det finns därför flera saker som bör tänka på som projektledare. David William Brown skriver att det är viktigt att alla som är involverade i ett projekt förstår innebörden av vad de andra i projektet menar när de diskuterar olika problem. Då förkortningar kan betyda olika saker så är det viktigt att de andra förstår vad det är som sägs. Detta är extra viktigt när det finns personer i projektet med olika bakgrunder, såväl tekniska som etniska. För att hålla alla uppdaterade så är det då viktigt att ofta hålla både formella och informella möten där det tas upp och gås igenom vad som händer i projektet för tillfället, vad alla gjort och vad de ska göra. Detta bör göras för att se till att alla är på samma sida så att alla tillsammans kan se till att de problem som kan uppstå skapar så liten skada som möjligt.

För att lösa kommunikationsproblem så kan projektet använda sig av modeller så som UML för att övervinna dessa hinder. Om det finns en standard för att kommunicera angående hur programmet fungera, så kan alla enklare deltaga i uvecklingsprocessen och förre missförstånd uppstår. Detta i sin tur kan leda till att projektet kan skapa bättre mjukvara då alla parter kan kommunicera på ett sätt som alla kan förstå, även om projektmedlemmarna inte har fullständig förståelse för det språk som huvudsakligen pratas av de resterande projektdeltagarna. [Brown, 2002]

Det är även viktigt att kunna finna information och dela med sig av denna till medarbetare som inte sitter i samma byggnad som en själv. Detta är ofta extra sant inom den öppna källkodens värld där utvecklare kan vara spridda över hela världen. Genom Internet så skapas ytterligare sätt att presentera och göra information tillgänglig på ett enkelt sätt. Detta genom att skapa gränssnitt som gör det lätt att komma åt data och information i olika medier så som text, bilder och film. Internet

skapar ytterligare möjligheter att hitta kunskap som annars skulle vara svårt att få tag på eller kosta mycket pengar att tillförskaffa sig. [Turban, 2005]

3.7. Produktivitet

I sin bok Peopleware så menar Tom DeMarco och Timothy Lister att kavliteten på produkten riskeras när det sätts upp mål som inte går att nå. Detta för att det finns en övertygelse om att projektmedlemmarna arbetar snabbare och bättre när de arbetar under tidspress. Att jobba under tidspress ger inte bättre arbete utan det gör bara att projektmedlemmarna jobbar fortare menar författarna. DeMarco och Lister menar dock att det istället leder till att brister i produkten ignoreras för att utvecklarna upplever att de inte har tid att ta hand om felen nu utan måste fokusera på större och mer akuta problem som måste lösas för att nå målet. Genom att lägga en för snäv deadline för när ett projekt skall vara klart så skapas det en extra påtryckning på medarbetarna. Detta kan bland annat leda till att kvaliteten blir lidande när projektet skall bli klart i tid. Författarna menar att om utvecklarna själva tillåts sätta gränserna för kvaliteten så går produktiviteten upp gentemot ifall kunden tillåts sätta gränsen för hur kvaliteten skall vara.

DeMarco och Lister tar bland annat upp en studie som gjorts utav Michael Lawrence och Ross Jefferey där de har kommit fram till att de utvecklare som är mest produktiva är de som inte har någon tidpress, eller har en väldigt löst satt tidpress, för att bli klara med ett projekt. I undersökningen kunde det ses att även om utvecklarna själva fick bestämma och sätta upp ungefärliga tider, då de trodde att projektet skulle bli klart så blev de även mer produktiva än när de fick jobba mot ett schema som redan satts av ledningen. [DeMarco och Lister, 1999]

Dag Ingvar Jacobsen och Jan Thorsvik menar dock, i sin bok Hur moderna organisationer fungerar, att om projektet använder sig av mål med klara tidsgränser så leder detta till bättre motivation än när ett mål sätts upp utan att använda sig av definierade tidsbegränsningar. Emellertid så behöver de anställda acceptera och sluta upp kring dessa mål för att det skall kunna skapas motivation kring dem. Om målen upplevs som orimliga eller på annat sätt dåliga, kommer de anställda inte alls att arbeta bättre.

Båda författarparen menar dock att genom att ha utmanade mål så skapar det bättre motivation än mål som är betydligt lättare att nå. De anställda vill uppleva att målen innebär en utmaning samtidigt som de känns meningsfulla. Meningen med en projektgrupp är inte att uppnå målet utan

att alla är inriktade på det uppsatta målet hävdar DeMarco och Lister. [Jacobsen och Thorsvik, 2002]

Linux-kärnan kan här ges som exempel på detta. Nya inslag i kärnan och buggfixar till gamla inslag släpps endast när de är färdiga och det ligger inte någon sista deadline på när de skall bli färdigställda, utan de läggs in i den nästa stabila versionen av kärnan när utvecklarna känner att de är färdiga och koden har blivit buggtestad och godkänd av de som färdigställer de nya versionera av kärnan. [Raymond, 2001]

Genom att använda sig av ett högnivåspråk kan programmeraren bli fem gånger mer produktiv än när han hade använt sig utav ett lågnivåspråk. Att använda sig av ett högnivåspråk ger även en annan fördel, koden blir oftast betydligt kortare än om den hade skrivits i ett lågnivåspråk. Detta är bland annat en av andledningarna till att den som utvecklar blir mer produktiv när denne kodar i språk så som Python eller Perl. Med hjälp av Lisp går det att i vissa fall ersätta upp till 20 rader kod som är skrivet i C med en enda rad. Detta leder även till att koden blir enklare att förstå för andra som läser den. [Brooks, 1995][Graham, 2002]

3.8. Kirurgteamet

Så som inom de flesta områden så varierar graden av kunskap mellan individer. I en studie gjord av Sackman, Erikson och Grant kom de fram till att de bästa programmerarna inte sällan var upp till tio gånger så snabba som de sämsta och producerade kod fem gånger så snabbt. Studien visade även att det inte fanns någon relation mellan erfarenhet och hur snabbt programmeraren arbetade. Detta är någonting som organisationen måste hantera på ett så bra sätt som möjligt. [Sackman, Erikson, Grant, 1968]

Fredrick Brooks presenterar i The Mythical Man-Month en modell utarbetad av Harlan Mills för programutveckling som kallas för kirurgteamet. Tanken bakom denna är genom att organisera programmerarna efter den modell som ett kirurglag använder sig av kommer de att tillvarata alla inblandades kunskaper på bästa sätt. Programmeringsuppgifterna skall enligt Mill delas upp i mindre uppgifter som utförs av grupper av programmerare. Dessa skall vara organiserade så som kirurglag. De mest centrala delarna i ett kirurglag är förstås kirurgen och dennes närmsta medarbetare som utför själva operationen men även personerna runt denne som gör det möjligt för kirurgen att arbeta. Den bästa programmeraren ska vara kirurgen som tar hand om att utföra själva

programmeringen. Denne måste vara erfaren och kunna dokumentera väl och även ha en bra miljö att arbeta i där programmet kan testas och gamla versioner sparas. Till sin hjälp har kirurgen sin högra hand som kan hjälpa till med alla delar av utvecklingen, men har inte samma erfarenhet som kirurgen. Denna persons främsta uppgift är att vara kirurgens bollplank att diskutera och utvärdera idéer tillsammans med. [Mills 1971]

Genom att dela upp arbetet på detta sättet kan projektledarena försöka att styra in projektdeltagarna på det som de klarar av bäst. Kirurgen är chefen som gör det mesta av jobbet medans de andra hjälper till med olika delar för att underlätta kirurgens arbete så mycket som möjligt. På detta sätt menar Brooks att kirurgteamet skulle kunna lösa problemet med att programmerare är olika duktiga på att programmera och styra in dem på det som de gör bäst. [Brooks, 1995]

Utveckling inom öppen källkod går enligt Paul Jones ofta till på detta sättet. Oavsett om utvecklarna är medvetna om det eller inte. Det är oftast en liten grupp utvecklare som står själva för den största delen av arbetet medan väldigt många hjälper till med det som de kan bidra med. Det ställer dock frågan om det verkligen går att betrakta alla som utvecklare även om de bara bidragit med väldigt lite kod eller kanske bara letat buggar. Om 80% av arbetet utförs av en grupp på fem personer medan 100 andra bidrar med resten bör man fråga sig om detta verkligen är ett projekt med 105 utvecklare.

Jones menar därför att eftersom de som arbetar med öppen källkod oftast arbetar efter kirurgteamsmodellen, så kan de alltid komma till nya personer och bidra med det som de är bra på. Detta betyder dock inte att det för den sakens skull påverka projektet negativt då de bara kommer till nytta när kirurgerna eller chefsutvecklarna behöver deras hjälp. De utgör därför inte någon flaskhals i arbetet utan blir bara en resurs som kan utnyttjas vid behov.

Att arbeta utifrån teorin om kirurglaget kräver dock att projektledarna har delat upp arbetet på ett bra sätt och har rätt mängd med kirurgerteam. Jones menar att om ett projekt kan delas upp i tre delprojekt, men har två kirurger skulle det vara väldigt fördelaktigt att tillföra en tredje sådan och projektet skulle sannolikt vinna mycket på detta. Att där efter tillföra en fjärde till samma projekt med tre delprojekt skulle inte alls tillföra samma fördelar även om det kanske inte kommer att sinka det heller. [Jones, 2000]

Trots att vem som helst som kan skicka in och bidraga med kod till ett öppet källkodsprojekt så

är det oftast en handfull människor som gör det huvudsakliga arbetet med utvecklingen på ett projekt. Andrew Morton har sagt, 2004, att det är ungefär tusen personer som skickar in kod till Linux-kärnan. Sammanlagt har dessa bidragit med ungefär 38000 olika ändringar och förbättringar till kärnan. Av dessa 38000 ändringar och förbättringar så har cirka 37000 av dem gjorts av uppemot 100 utvecklare som är anställda vid olika företag och har som arbetsuppgift att bland annat förbättra och hålla Linux-kärnan under konstant utveckling. [Jackson, 2004]

3.9. Sammansvetsade team

DeMarco och Lister diskuterar värdet av att gruppen av utvecklare är ett sammansvetsat team. De menar att för att få utvecklarna att verkligen vilja arbeta mot företagets mål på ett djupare plan måste organisationen skapa en gruppkänsla där alla vill arbeta mot det gemensamma målet och identifierar sig med målet. I ledningen på ett företag är människorna ofta väl införstådda med och arbetar mot målet, men på en lägre nivå är det ofta inte så enligt DeMarco och Lister. Chefer och andra på högre poster identifierar sig ofta som chef och en del av företaget, medan en databasexpert kanske snarare identifierar sig som pappa, ordförande i sportfiskeklubben eller som medlem i en församling i första hand istället för sin position på företaget.

Det är därför viktigt enligt DeMarco och Lister att försöka motivera de anställda till att vilja identifiera sig med företagets värderingar och känna en lojalitet mot företaget. Detta skulle leda till en ökad produktivitet menar de.

Det är därför önskvärt att försöka skapa grupper eller team med gemensamma värderingar och en bra stämning där individerna i laget känner att de är en del av något större som de uppskattar och kan identifiera sig med. Ett sådant team kommer enligt DeMarco och Lister att arbeta mer målmedvetet och effektivare och även trivas bättre i sin yrkesroll.

Det som identifierar ett väl sammansvetsat team är att det har en låg omsättning på personal. DeMarco och Lister menar att om de anställda känner sig som en del av ett bra team där de trivs kommer lön, status och chanser att avancera i karriären att bli mer sekundära. De anställda kommer bli mer fokuserade på att arbeta och möta de krav som ställs på dem. De känner att de kan identifiera sig med teamet och identifierar sig själva som en del av teamet. Tecken på att gruppen kommit långt med att skapa ett väl sammansvetsat team är till exempel när internskämt börjar utvecklas och teammedlemmarna börjar umgås på fritiden, enligt DeMarco och Lister.

Ett ytterligare tecken på att teamet är väl sammansvetsat är att en viss kaxighet och elitism uppstår när medlemmarna av teamet upplever att de är del av någonting viktigt och unikt. Samtidigt så har de en känsla av delägarskap i den slutgiltiga produkten. Teammedlemmarna är stolta över att gemensamt stå som utvecklare av mjukvaran. [DeMarco och Lister, 1999]

3.10. Kritik och kommentarer på Brooks lag och kirurgteamet

Steve McConnell menar dock att även om projektet blev försenat på grund av att det togs in nya projektmedlemmar så går det inte att veta huruvida projektet skulle bli försenat om dessa inte lagts till. Vad blev projektet egentligen försenat jämfört med? Var det jämfört med den första optimistiska uppskattningen om ur lång tid projektet skulle ta? McConnell anser även att Brooks exempel där han beskriver hur Brooks lag fungerar är helt fel, då de personer som är med i projektet inte har hunnit göra tillräckligt mycket för att tillsfredställa en hela mansmånads träning, vilket gör att Brooks beräkningar inte är helt sannolika.

Det som är viktigt enligt McConnell är att projektledaren har god insyn i projektet och en god förståelse för den aktuella typen av projektarbete i stort. Om projektledaren är så pass duktig och medveten om hur projektet löper kan denne i ett tidigt stadie se om projektet kommer att bli försenat och kan då sätta in nya medarbetare i projektet i ett mycket tidigare skede och på så sätt minska de eventuella problemen med Brooks lag. [McConnell, 1999]

I Recommended Approach to Software Development från NASA så menar de att projektstorleken, bemödandet och schemat skall ombedömas varje månad. NASA anser att det inte är något fel i att se över hur mycket folk det behövs till ett projekt och att varje fas i utvecklingen kräver olika mängder med personer som är involverade. Det anses även av NASA att varje projekt skall påbörjas med en liten grupp med personer som sedan kommer att bli gruppledare och ta hand om och lägga upp stora delar av projektet för att sedan plocka in nya människor allt eftersom att det behövs. Projektet bör bara plocka in ny personal när det verkligen behövs. [NASA, 1992]

Tom DeMarco och Timothy Lister är inne på samma linje som NASA i sin bok Peopleware, de menar att det bättre att design och analys sköts av mindre grupp med personer. Anledningen till detta menar de är att det värsta som en chef kan göra är att slösa på sin personals tid. Har de ingenting att göra i ett projekt så är de inte produktiva vilket gör att företaget förlorar pengar.

Som exempel ger DeMarco och Lister ett två års projekt där den stora flertalet av de anställda inte kommer in i själva projektet förrens det gått sex till åtta månader. Detta för att om de hade lagts till tidigare så hade de inte gjort någon nytta, vilket hade gjort att företaget hade förlorat pengar. Under den första tiden som de som är aktiva i projektets inledande faser håller på med analys och design kan de resterande projektdeltagarna som kommer in i senare faser jobba med andra projekt. Detta gör att företaget inte behöver förlora pengar under den här initiala perioden och de anställda kan hela tiden hållas med arbete. Dock så är det inte bra ifall de anställda hamnar i flera olika projekt vilket gör att deras tid splitras. Detta kan göra att projektmedlemmarnas effektivtet blir sämre då de måste lägga tid på flera olika projekt. Ifall detta förekommer så kommer det i slutänden drabba företaget negativt. [DeMarco och Lister, 1999]

Det finns alltid nya delar av projektet som är bättre lämpade att ges till de nyanställda, skriver Scott Berkun i en kommentar om sin bok The Art of Project Management. Han kommenterar även att om en deltagare plockas bort från sitt projekt så kan detta lämna rum för att lägga till en ny person. För att få in nya personer i projekt så skriver han att projektledaren kan bygga upp lagkänslan och skapa sammahållning för att få in de nya personerna. Detta kan göras genom att skapa olika sociala händelser. Exempel på detta skulle kunna vara att ta med de anställda ut i skogen för att spela paintball eller paddla kanot. [Berkun, 2006]

R. D Stutzke kommenterar Brooks Lag med att alla personer som läggs till sent i ett projekt måste vara lagspelare och jobba utefter de satta gränserna för projektet och inte försöka ändra eller förbättre själva processen. Om de inte är lagspelare utan försöker ändra på arbetssättet i projektet kommer risken för förseningar att öka. [Stutzke, 1994]

4. Diskussion

4.1. Öppen källkod

Eric Raymond och hans likasinnade hävdar att ifall ett företag eller en organisation använder sig av öppen källkod så kommer detta gagna dem i slutändan. Vidare så hävdar förespråkarna av öppen källkod att användarbasen blir större, men även att användarna blir mer engagerade i projektet ifall det är öppet. Fördelen med öppen källkod är just öppenheten vilket gör att vem som helst kan ändra i kodbasen utan några speciella restriktioner, men även friheten att vidaredistribuera program eller ändra i dem för att programmet skall göra de saker som kan anses fattas.

Raymond beskriver två olika utvecklingsmetoder, dels katedralen och dels basaren. Båda två använder sig av kärnor som har huvudansvaret för utvecklingen av projektet. Kärnorna tar i båda fallen hand om själva huvudarbetet, men skiljer sig i huvudsak åt när det kommer till utgivningstakten. Då katedralmodellen inte släpper nya versioner lika ofta då de vill hålla nere på antalet fel som användarna skall se i programmet, det kan leda till att användarna inte får lika stor insikt i vad som händer i projektet. Fördel med detta är att det släpps stabila och väl fungerande program med denna metoden. Dock så medför det även att många nya inslag och funktioner kan komma på en och samma gång i stället för att rullas ut allt eftersom så som det kan ske i basarmodellen. Detta förhindrar dock inte att det kan komma många nya funktioner eller ändringar i nya stabila versioner av system som utvecklas under basar metoden. Båda modellerna tillåter dock att användare kan modifiera och skicka in kod för att förbättra den redan existerande kodbasen.

Raymond beskriver till exempel att Linux-kärnan använder sig av ett system för att stoppa in ny funktionalitet först när de delsystem eller drivrutiner som utvecklas är färdiga. Samma mentalitet bör användas när det gäller nya inslag och funktioner till andra öppna projekt med. Genom att då sätta upp ett datum där nya funktioner slutas tas emot inför nästa större stabila version så ger detta att deltagarna har ett mål att arbeta mot. Dock så är det viktigt att poängtera att de inte skall behöva försaka kvalitet för att nå detta mål, utan istället fokusera på nästkommande större version ifall det känns som om den nuvarande uppsatta deadline inte kommer att nås.

Även om de deadlines projektet har är löst satta så är det ändå viktigt att det finns sådana här mål som alla kan sluta upp kring. Genom att ha utmanade mål så hävdar både DeMarco och Lister såväl som Jacobsen och Thorsvik att de inblandade arbetar bättre. Dock så kan det vara viktigt att

poängtera här att det inte behöver vara mål som till exempel att få klart hela systemet på två månader utan snarare att tillverka de bästa produkterna i branschen. Tidsaspekten bör inte vara den viktiga faktorn här, deltagarna bör istället satsa på kvalitet då de flesta utvecklare blir mer produktiva och gör bättre arbete när de har lösa deadlines eller får sätta dem själva, menar DaMarco och Lister.

Ifall ett projekt är öppet så kan detta leda till att det ger större säkerhet för användarna. Ifall vi tittar på exemplen från resultatet så innehöll både Microsoft Frontpage och Borland Interbase bakdörrar. Skillnaden mellan dessa två program är dock att Frontpages bakdörr upptäcktes av en slump, medan Interbase bakdörr hittades när koden började studeras när databasen släpptes under öppen licens. Då vem som helst kan titta på källkoden och gå igenom den så leder detta till att utvecklarna inte kan lägga in bakdörrar eller skicka data som användarna inte vill att andra skall få reda på, till exempel öppna portar eller vilken hårdvara som används. Tyvärr så finns inte den här kontrollen inom proprietär programvara utan här får användarna förlita sig på att utvecklarna faktiskt inte har några onda avsikter efter att programmet har installerats på datorn. Det finns ingenting som säger att det inte finns ytterligare bakdörrar i Frontpage. Då koden till Frontpage är stängd så kan inte utomstående experter inom området studera koden till programmet. Detta gör att Frontpage inte kan få den trovärdighet som Raymond menar att utvecklarna får ifall koden är släppt under öppna förhållanden.

I och med att vem som helst kan läsa och studera den kod och dokumentation som de öppna projekten innehåller så ger detta möjligheter för företag och privatpersoner att ge support till de program och system som finns. Ifall ett program har en bra bas för support, antingen genom egna metoder eller via tredjepart så får programmet en längre livslängd än ett likvärdigt privatägt program som kanske inte har samma utgångsläge. Detta leder bland annat till ett program som är släppt under öppna källkodslicenser kan få större kommersiell spridning ifall organisationen bakom, eller tredjepartsföretag, tillhandahåller en erkänt bra support för programmet.

Runt de flesta stora öppna program, men även mindre, så finns det oftast en community med frivilliga användare runt omkring som hjälper till med support för personer som precis börjat använda programmen. Då de själva fick denna hjälpen när de var nybörjare så vill de ge tillbaka den kunskap som de själva fått genom liknande hjälp, hävdar Lakhani och von Hippel. Detta kan leda till att det ofta finns gratis support att tillgå i öppna projekt då många användare samlas i olika

kanaler så som e-postlistor och forum för att diskutera projektet och hur det kan vidareutvecklas eller bara svara på frågor som ställs.

Detta kan bero på att det ofta skapas snöbollseffekter i öppna projekt där användarna i communityn får hjälp av utvecklarna så att de till slut känna att de vill hjälpa till och ta över en del av ansvaret i att hjälpa andra användare med support. I slutändan så kan detta leda till att utvecklarna får mer tid över till att vidareutveckla deras program istället för att behöva lägga ner lika stor tid på att hjälpa nybörjare med support. En annan fördel med att det växer upp en sådan här community runt omkring projekt är att utvecklarna kan få hjälp med att dokumentera och se över redan existerande dokumentationen, tror vi. Den fria supporten hindrar dock inte organisationen bakom projektet eller tredjepartsföretag att starta egen support för att kunna tjäna pengar på de öppna program som finns. På detta vis så går det att dra ekonomisk fördel på koden även om vem som helst kan använda den utan några större restriktioner.

Tyvärr så leder detta dock till problem. Ju större ett projekt blir desto mer människor börja strömma till för att de vill hjälpa till med projektet. Även om projektet inte är försenat från första början så kan det nu bli försenat då utvecklarna måste börja hjälpa olika personer som har problem eller måste läsa och studera den kod som skickas in till projektet. När det sedan strömmar till personer som hjälper till med supporten så växer den grupp med deltagare som är engagerade i projektet och Brooks lag börjar träda i kraft. Det största problemet med att ett projektet är öppet är just att projektet är öppet. Vi antar att allt eftersom olika personer börja skicka in mer och mer kod till projektet så kan detta leda till problem för de som utvecklar då de blir tvungna att sätta sig in i den kod som skickas in för att studera om det kan finnas problem eller fel med den. Detta leder dem bort från den huvudsakliga utvecklingen och mer in i administrativa roller. Allt eftersom projektet växer behövs fler och fler läras upp och tas hand om vilket tar tid ifrån det huvudsakliga utvecklingen.

4.2. Brooks lag

Brooks lag säger nämligen att när det tillkommer nya människor så tar det längre tid att färdigställa projektet. För varje ny person som väljer att engagera sig i projektet så kommer det innebära en belastning. Personen blir dock inte en belastningen förren utvecklarna faktiskt bekräftar att personenen finns och försöker tillföra något till projektet. I och med att utvecklarna måste ta sig tid att kontrollera vad det är som skickas in och om det är användbart. Detta tar då tid från själva

utvecklandet av systemet i fråga. De nya personerna som väljer att skicka in kod eller dokumentation kan ses som en bra vidare resurs, i dessa fall kan de behöva vägledning så att de kan göra ett bättre framtida jobb. Detta kan leda till att personerna tar upp mer tid i anspråk från utvecklarnas sida, men i långa loppet kan bli en god resurs för projektet. Dock så finns problematiken med Brooks lag fortfarande kvar då projektet fortskrider långsammare ju fler personer som skickar in resurser.

Då de flesta öppna projekt arbetar utefter icke existerande deadlines eller väldigt löst satta deadlines så kan detta leda till projektledarna inte behöver koncentrera sig lika mycket på att beräkna huruvida projektet kommer bli försenat eller inte. Detta betyder dock inte att arbetsbördan blir mindre bara att Brooks lag blir mer otydlig. Till exempel kommer det behövas kontrolleras ifall vissa delar och delmål inom projekten kommer att bli klara inför kommande versionssläpp. Det är även lika viktigt inom öppna projekt som i stängda att se till att de delprojekt som är beroende av varandra ligger i fas.

Ifall det är en liten grupp med personer som har huvudansvaret för att ett program eller systems utveckling skall fortskrida så kan det ställa till med problem om deras projekt blir allt för intressant för utomstående att bidra med kod och dokumentation. Detta skulle kunna leda till att i perioder så skulle utvecklingen av programmet eller systemet gå väldigt långsamt eller stå helt stilla. Anlednigen till detta är att alla de resurser som skickas in till projektet kan bli överbelasta utvecklarna med ny information och data att ta hand om. Därför måste projektet finna en utvecklingsmetod där de kan ta vara på hjälpen utan att det belastar resten av arbetet.

4.3. Kirurgteamet

Brooks lag skulle alltså kunna skapa stora problem för dem som väljer att släppa sin mjukvara under en öppen källkodslicens. De fördelar som utvecklarna hoppas kunna vinna med att släppa mjukvaran fri bör ställa till problem med Brooks lag. Detta för att det hela tiden tillkommer nya människor som bidrar med vad de kan. På detta vis kan det som utvecklarna strävar efter bli en belastning för projektet. Därför måste de aktiva i projektet försöka finna ett lämpligt arbetssätt där problematiken med Brooks lag minimeras.

Med utgångspunkt i den metod som Brooks kallar för kirurgteamet bör det gå att finna en metod som lämpar sig för att hantera just den problematik vi studerat. Paul Jones skriver att de flesta stora

projekt använder sig av en form av kirurgteam. Projektet delas upp i många små delprojekt med en liten kärna av utvecklare som skriver det mesta av programmet. Detta kräver alltså att de som driver projektet kan dela upp det på ett bra sätt om det behövs. Mindre projekt har naturligtvis inte samma behov av uppdelning som ett större. Detta är dock en ständigt pågående process och allt eftersom att projektet växer sig större måste organisationen ses över. Vid en viss punkt kan det bli aktuellt att dela upp ett delprojektet i två nya delprojekt och tillsätta ett nytt delprojekt och tillsätta nya kärnor med kirurger. NASA menar i sin rapport att det är meningsfullt att regelbundet följa upp projektet och se hur många personer som behöver avvaras, vilket går hand i hand med det Jones säger om värdet av att ha precis rätt mängd kirurger i projeketen. Vinsten av att ha en extra kirurg kan vara obefintlig, medan ett kirurgteam för lite kan kan leda till förseningar.

För att på ett bra sätt kunna dela upp projekt i mindre projekt krävs det att någon ska sätta sig in i vad som redan gjorts och detta kan skapa ännu fler problem med Brooks lag. Det är därför väldigt viktigt att allt som sker i projektet är väl dokumenterat för att avlasta de redan existerande kirurgerna i arbetet med att engagera den nya.

Den lilla kärna av utvecklare som är satta att sköta programutvecklingen i projektet ska kunna ta in hjälp utifrån när de behöver för att deras arbete ska gå så smidigt och bra som möjligt. Använder projektet sig då av öppen källkod kan denna lilla kärna av utvecklare få hjälp från människor över hela världen som valt att engagera sig i projektet. Till sin hjälp för att finna denna kompetens och även att hjälpa utvecklaren bör, precis som Mills säger, kärnan inte bara bestå av utvecklare utan även av deras närmsta medhjälpare. Dessa kan fungera som en länk mellan kärnan och resten av utvecklarna och styra utvecklarna till rätt utomstående kompetens. Detta bör leda till att arbetet skulle kunna ske så smidigt som möjligt och mycket av problematiken med Brooks lag kan på så vis undvikas.

Det är även viktigt att de som finns i dessa små kärnor av utvecklare och medhjälpare inte bara är duktiga på utveckling utan att de kan engagera andra. Precis som Raymond säger måste de få andra människor att ställa sin tid och sitt kunnande till projektets förfogande, när projektet behöver. Det skulle förmodligen vara lämpligt att detta arbete i större projekt sköttes av medhjälparna då de redan har en roll där de ska veta vilken kompetens som finns tillgänglig. Även arbetet med att organisera personer som kan hjälpa till med support bör medhjälparna kunna utföra.

Att dela upp utvecklingsarbetet på detta sätt skiljer sig från andra då det mesta av arbetet görs av

en väldigt liten grupp människor som utvecklar medan det kan finnas väldigt många som hjälper dessa utvecklare. Genom att arbeta på detta sätt kan organisationen hantera Brooks lag på ett bra sätt samtidigt som de tillgodogör sig fördelarna med öppen källkod. Däremot borde en stor sårbarhet skapas. Även om det går bra att när som helst byta ut dem i den yttre delen av projektet så som Jones säger så borde det ställa till stora problem om någon ur kärnan försvinner från projektet.

Den kärna av utvecklare, medhjälpare och eventuella andra som ska utgöra grunden för projektet är väldigt viktig och måste fungera väl. De som utgör denna kärntrupp bör därför känna starkt för projektet och även för gruppen som sådan. Det skulle därför vara lämpligt att, som DaMarco och Lister säger, försöka få kärnan till en tätt sammansvetsad grupp. Denna gruppen skulle då arbeta bättre och effektivare än en grupp som inte är lika sammansvetsad.

Det sammansvetsade kirurgteamet av utvecklare och medhjälpare kommer visserligen att vara mer effektiv än ett mindre sammansvetsat team. Däremot måste gruppkänslan och den eventuella elitism som DaMarco och Lister skriver om hanteras på ett bra sätt. Det får inte uppstå en vi mot dem attityd inom kärnan utan de måste acceptera och välkomna nya personer. Det skulle därför kunna vara bra att försöka styra denna eventuella elitism till att gälla för hela projektet eller företaget, snarare än gruppen. Enligt NASA kan de som arbetar inom ett företag röra sig inom organisationens olika projekt för att hjälpa till där de behövs mest. På så sätt behövs inte kunskapen bindas i ett enskilt projekt utan många i personalen kan röra sig mellan de olika projekten och vara till hjälp där de behövs för tillfället. Det blir då bara själva kirurgteamen som är bundna till ett specifikt projekt. Detta skulle inte fungera om elitismen var allt för stark inom de olika kirurgteamen. Ett kirurgteam som är inte är öppna för nya människor och dessas idéer skulle få det väldigt svårt att tillvarata den.

5. Slutsats

I vår studie har vi försökt att finna en metod där det på ett så bra sätt som möjligt går att hanterar de eventuella problem som kan uppstå. Det största problemet var Brooks lag vilken vi valde att fokusera studierna kring.

Vi anser det rimligt att anta att Brooks lag definitivt är ett problem inte bara inom den traditionella proprietära utvecklingen utan även inom utveckling av öppen källkod. Problemet kan dock te sig lite annorlunda då projekten inom öppen källkod inte alltid engagerar nya personer på samma sätt. Det är inte alltid så att den nya blir någon som tas in som en helt fast roll i projektet utan det kan ofta vara en temporär roll som denne får. Problematiken med Brooks lag blir dock inte mindre för detta utan snarare större.

Däremot såg vi snabbt att det sällan är så att projekten under öppen källkod har fasta deadlines. Detta leder till att vissa delar av problematiken med Brooks lag inte blir synlig och kanske därför inte hanteras trots att projektet skulle gått smidigare att genomför om problemet hade hanterats. Det är därför viktigt att vara medveten om Brooks lag, trots att vissa hävdar att problemet inte finns inom öppen källkod.

Den utvecklingsmetod vi valde att studera för att se om den kunde användas till att hantera problematiken med Brooks lag anser vi fungera bra med de justeringar som vi tar upp i diskussionen. Det är dock viktigt att de personer som fungerar som medhjälpare till kirurgerna inte bara är bra på kod utan kan ta tillvara och organisera den hjälp som erbjuds projektet från personer utanför det. Kirurgerna ska helst inte själva behöva lägga ned tid på att uppmuntra de utomstående eller leta efter dem med rätt kompetens utan ska kunna fokusera på utvecklande medan medhjälparen sköter detta.

Ett projekt inom öppen källkod som lyckas med att engagera många utomstående och samtidigt är medvetna om problematiken med Brooks lag och hanterar denna på ett bra sätt genom att arbeta utifrån en metod liknande kirurgteamsmetoden bör ha goda förutsättningar att fungera bra.

6. Förslag till vidare studier

Vidare studier inom området skulle kunna vara att studera problemet med att vem som helst kan välja att sända in kod till ett öppet källkodsprojekt. Det finns förmodligen väldigt många som vill bidra med kod till större och kändare projekt vilket kan göra att väldigt stora mängder kod skickas. Kod som någon måste sätta sig in i eller på annat sätt sortera för att finna de bra bidragen.

Ett annat område som skulle kunna studeras är hur ett projekt bör dokumenteras och hur dokumentationen bör göras tillgänglig för att på ett enkelt sätt göra det möjligt för utomstående att sätta sig in i det med så liten belastning på dem i projektets kärna som möjligt.

7. Referenslista

Backman, 1998: Jarl Backman, [Bok] *Rapporter och uppsatser*, 91-44-00417-6

Berkun, 2006: Scott Berkun, [WWW-dokument] *Exceptions to Brooks' LawExceptExceptions to Brooks' Lawions to Brooks' Law*, http://scottberkun.com/blog/2006/exceptions-to-brooks-law/

Blender: Blender Foundation, [WWW-dokument] *Blender History*, http://www.blender.org/blenderorg/blender-foundation/history/

Brooks, 1995: Frederick Brooks, [Bok] *The Mythical Man-Month*, 0201835959

Brown, 2002: David William Brown, [Bok], *An Introduction to Object-Oriented Analysis, Objects and UML in Plain English*, 0-471-37137-8

DeMarco och Lister, 1999: Tom DeMarco, Timothy Lister, [Bok] *Peopleware: Productive Projects and Teams*, 978-0-932633-43-9

FSF, 2007: Free Software Foundation, [WWW-dokument] *The Free Software Definition*, http://www.gnu.org/philosophy/free-sw.html

Graham, 2002: Paul Graham, [WWW-dokument] *Revenge of the Nerds*, http://www.paulgraham.com/icad.html

Jackson, 2004: Joab Jackson, [WWW-dokument] *Linux now a corporate beast*, http://www.gcn.com/online/vol1_no1/26641-1.html

Jacobsen och Thorsvik, 2002: Dag Ingvar Jacobsen, Jan Thorsvik, [Bok] *Hur morderna organisationer fungerar*, 91-44-02276-X

Jones, 2000: Paul Jones, [WWW-dokument] *Brooks' Law and open source: The more the merrier?*, http://www-128.ibm.com/developerworks/linux/library/os-merrier.html

Kenwood, 2001: Carolyn A. Kenwood, [Rapport] *A Business Case Study of Open Source Software*

Lakhani, von Hippel, 2000: Karim Lakhani och Eric von Hippel, [Uppsats] *How Open Source Software Works: Free user-to-user assistance*

LaMonica, 2005: Martin LaMonica, [WWW-dokument] *Microsoft walks VB tightrope*, http://news.com.com/Microsoft+walks+VB+tight+rope/2100-1007_3-5620821.html

Leonard, 1998: Andrew Leonard, [WWW-dokument] *Let my software go!*, http://archive.salon.com/21st/feature/1998/04/cov_14feature.html

McConnell, 1999: Steve McConnell, [WWW-dokument] *Brooks' Law Repealed?*, http://www.stevemcconnell.com/ieeesoftware/eic08.htm

Mills 1971: Harlan Mills, [Rapport] *Cheif programmer teams, principles, and procedures*

NASA, 1992: Linda Landis, Sharon Waligora, Frank McGarry, Rose Pajerski, Mike Stark, Kevin Orlin Johnson, Donna Cover, [Rapport] *Recommended Approach to Software Development*, http://www.construx.com/File.ashx?cid=1211,

O'Riordan, 2006: Ciarán O'Riordan, [WWW-dokument] *The Free Software Movement and the Future of Freedom; March 9th 2006*, http://www.sweden.fsfeurope.org/documents/rms-fs-2006-03-09.en.html

OSI, 2006: Open Source Initative, [WWW-dokument] *The Open Source Definition*, http://www.opensource.org/docs/osd

Perens, 2002: Bruce Perens, [WWW-dokument] *Open source advocate says, "We're better at security"*, http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2859555,00.html

Poulsen, 2001: Kevin Poulsen, [WWW-dokument] *Borland Interbase backdoor exposed*, http://www.theregister.co.uk/2001/01/12/borland_interbase_backdoor_exposed/

Raymond, 2001: Erik S. Raymond, [Bok] *The Cathedral and The Bazaar*, 0-596-00108-8

Sackman, Erikson, Grant, 1968: H. Sackman, W. J. Erikson, E. E. Grant, [Uppsats] *Exploratory Experimental Studies Comparing Online and Offline Programming Performance*

Statskontoret, 2003: Statskontoret, [Rapport] *Free and Open Source Software - a feasibilty study*

Stutzke, 1994: Richard D. Stutzke, [Rapport] *A Mathematical Expression of Brooke's Law*

Taft, 2003: Darryl K. Taft, [WWW-dokument] *Survey: Java, C# Draws Visual Basic Developers*, http://www.eweek.com/article2/0,1759,1655796,00.asp

The Code, 2001: Hannu Puttonen,User-defined1, [Dokumentär] The Code, http://www.code.linux.fi/

Turban, 2005: Efraim Turban, Jay E. Aronson, Ting-Peng Liang, [Bok] *Decision Support Systems and Intelligent Systems*, 0-13-123013-1

Viega, 2004: John Viega, [WWW-dokument] *Open Source Security: Still a Myth*, http://www.onlamp.com/pub/a/security/2004/09/16/open_source_security_myths.html

Voices, 1999: Brian Behlendorf, [Bok] *Open Source as a Business Strategy*, 1-56592-582-3

Whitlock, 2001: Natalie Whitlock, [WWW-dokument] *The security implications of open source software*, http://www-128.ibm.com/developerworks/linux/library/loss.html?open&I=252,t=gr,p=SeclmpOS