IT University of Göteborg
CHALMERS | GÖTEBORG UNIVERSITY

(Examensarbete/Magisteruppsats inom Systemvetenskap och informatik)

REPORT NO. 2008:033
ISSN: 1651-4769

Department of Applied Information Technology or
Department of Computer Science

Application lifecycle management utvärdering av komponenter

Application lifecycle management a component evaluation

Tobias Björk
(gusbjtob@student.gu.se)

CHALMERS | UNIVERSITY OF GOTHENBURG

IT University of Göteborg
Göteborg, Sweden 2008

Innehållsförteckning

Abstrakt

Uppsatsen ger en överblick av vad synsättet Application Lifecycle Management (ALM) innebär. En utvärderingsmodell beskriver de analyserade verktygens duglighet gällande ett antal funktionella och icke-funktionella krav. Modellen jämför kommersiella verktyg och uppsättningar med en motsvarande uppsättning av Open Source verktyg. De huvudsakliga problemen idag är att de som nyttjar ALM-synsättet ofta blir begränsade till en kommersiell plattform, vilket får till följd att de inte kan bearbeta sådan data som inte stöds i plattformen. Ett annat stort problem är verktygens kostnad. Målet är att ta fram en uppsättning Open Source verktyg med motsvarande funktionalitet som de ledande kommersiella verktygen på marknaden. Open Source verktygen ska innebära en lägre kostnad i avseende på licens, underhåll och support samt dra ner på begränsningarna i datahanteringen.

Uppsatsen har har tagits fram genom att undersöka vad som har skrivits i ämnet tidigare och även genom egna experiment. Slutligen redogörs om Open Source verktygen når upp till samma användbarhet som de kommersiella verktygen.

1. Inledning

Detta avsnitt har för avsikt att klargöra uppsatsens ämne och även vilket problem som låg till grund till dess utförande. Förutom bakgrund kommer även problemområde, frågeställning och syfte samt avgränsningar att beskrivas.

1.1. Bakgrund

Detta examensarbete har utförts tillsammans med ett företag i Stockholm, som heter Signifikant. Företaget räknades under uppsatsen utförande som ett litet företag (5-100 anställda) med en inriktning mot anpassning av IT-applikationer och support.

Min kontaktperson och handledare under arbetet har under ett antal år intresserat sig av systemutveckling och specialiserat sig på ALM-synsättet. Han har under en tid arbetat för Borland (Borland.com) som kan räknas till en av de största leverantörerna av metoden. Under de år som tillbringades på detta företag så upplevdes dock ett antal problem med synsättet som kunde optimeras på många sätt. I takt med att intresset för synsättet har ökat och även då mjukvara och system har blivit allt mer komplicerad så ville han ha fram en alternativ lösning som kunde ge ännu mer underlag för synsättets allmänna intresse.

Systemutveckling är ett mycket brett begrepp vars innebörd delvis kommer beskrivas utifrån ett synsätt som heter Application Lifecycle Management (ALM). Synsättet går ut på att med hjälp av ett antal komponenter bygga upp en uppsättning som kan stödja hela utvecklingsprocessen av ett projekt.

1.2. Problemområde

Många IT-organisationer har heterogena systemlösningar eftersom det är enklast att få ut flest fördelar per komponent med ALM på det sättet. Utvecklare inom samma projekt använder ofta verktyg som inte är designade för att fungera tillsammans. ComputerworldUK (2007) beskriver i en undersökning hur Borlands kunder använder ALM. Denna undersökning har kommit fram till att mer än 50% av Borlands kunder använder verktyg från fler än tre olika leverantörer, vilket enligt en Borland-artikel publicerad på Applicationsoftwaredeveloper.com (2007) tyder på svårigheten att täcka alla primära domäner inom ALMs beståndsdelar som bland annat innefattar kravhantering, testhantering, ärendehantering, konfigurationshantering och design. Gedney (2007) menar att användare av ”Java” och ”.Net”-plattformar tillsammans med både ”Waterfall”- och ”Agile”-processer har en viss typ av arbetsflöden som kan vara problematiska att koppla ihop om man inte har rätt verktyg som kan hantera de varierande arbetsflödena. Ett annat företag använder istället ”Eclipse”, ”Visual Studio”, ”.Net” och ”Ivy” som har en annan typ av arbetsflöden och bundenheter. Har man då en stängd ALM-lösning som är anpassad till en viss typ av processer och arbetsflöden så finns det ingen möjlighet att använda dessa processer. Kan inte organisationen använda sina processer på önskat sätt så kommer de med största sannolikhet fortsätta att använda olika utvecklingsverktyg inom samma projekt.

1.3. Frågeställning och syfte

Syftet med detta examensarbete är att hitta en fungerande kombination verktyg inom Open Source och även att ta fram en utvärderingsmodell som ger en övergripande beskrivning av kommersiella- samt Open Source-verktyg för systemutveckling. Verktygen kommer väljas ut baserat på deras funktionalitet, marknadsposition, förmåga att bli del av en helhetslösning och även icke funktionella krav (såsom prestanda, installation, säkerhet m.m). Jag har för avsikt att undersöka hur sambanden

mellan de olika komponenterna fungerar och hur man ska gå tillväga för att få dem att samarbeta så bra som möjligt. Vad som också är intressant är att undersöka vad var och en har för uppgift i utvecklingsprocessen. Jag kommer främst att koncentrera mig på min uppdragsgivares synsätt på projektet, men kommer även att presentera andra projekt av liknande karaktär med en snarlik definition för att utifrån detta kunna göra jämförelser och hitta för- respektive nackdelar mellan de olika synsätten. Målet är att uppsättningen ska underlätta för intressenten både gällande installation och användning.

Jag är intresserad av att undersöka följande frågor:
Vad är syftet med ALMs olika komponenter och hur kan det kopplas ihop?

Hur bedömer man Open Source verktygens kompatibilitet med övriga verktyg?

1.4. Avgränsningar

Jag avser att avgränsa problemområdet genom att välja ut ett antal verktyg, såväl inom Open Source som kommersiellt, som jag tittar närmare på. Huvudmålet är att se om de Open Source verktyg som väljs ut har motsvarande funktionalitet som de verktyg som finns till förfogande inom den kommersiella marknaden. Jag vill undersöka om slutanvändaren har intresse av att välja Open Source verktyg eftersom de ofta innebär mycket lägre kostnader gällande licens, installation och även underhåll. De verktyg jag kommer jämföra med inom den kommersiella marknaden är de som anses ha störst marknadsandel just nu och som därmed bör ha mycket av den funktionalitet som slutanvändaren förväntar sig. Open Source verktygen kommer väljas ut baserat på antal aktiva användare och även dess funktionalitet i jämförelse med de kommersiella verktygen.

En utvecklingsprocess kommer att beskrivas, och den kommer innehålla följande delar: krav, konfigurationshantering, test, portföljshantering, design, ärendehantering samt hur de kan kopplas ihop. I ett senare exempel på hur processen kan gå till i verkligheten, med de verktyg som kommer analyseras, kommer dock fokus läggas på krav, konfigurationshantering, test och ärendehantering samt hur dessa kan kopplas ihop och få nytta av varandra.

2. Metod

Detta avsnitt kommer att behandla vilka metoder som använts för att framställa detta arbete, och även i vilket sammanhang jag ämnar använda de olika metoderna.

2.1. Litteraturstudie

För att få fram den information som ämnet avser så kommer mycket tid läggas på att söka på olika ordkombinationer på söktjänster som Google. Synsättet som ska undersökas har funnits i ett antal år och det finns därför en del material att titta igenom. Då termen innehåller så många olika begrepp så finns det med all säkerhet också artiklar som specialiserar sig på ämnets olika delar. Det kommer därför vara intressant att söka på både stora och små områden inom ämnet för att få fram ett budskap som är lättförståeligt och övergripande. Mycket av informationen kommer alltså hämtas från allmänna artiklar och studier. Det går även enkelt att få fram ännu mer information genom att undersöka artiklarnas källor.

Av erfarenhet vet jag att komplicerade begrepp och termer ofta beskrivs på ett övergripande sätt i den studentlitteratur som jag har läst under utbildningens lopp. Jag kommer därför i vissa fall använda mig av den, och även undersöka övrig relevant litteratur som finns i ämnet.

2.2. Kvantitativ och kvalitativ metod

Jag har valt att titta på ett urval av metoder inom det kvantitativa respektive det kvalitativa synsättet, för att analysera hur de skulle kunna komma till användning i mitt examensarbete. Analysen gick ut på att undersöka allmän litteratur i ämnet för att sedan välja ut de metoder som verkade vara mest lämpade. Till att börja med beskrivs alla metoderna för att ge en förståelse av dess innebörd, för att vidare göra en redogörelse hur de skulle kunna komma till användning. Då urvalet metoder och litteratur är väldigt begränsat så finns det ett antal metoder som inte har analyserats, syftet är dock att titta närmare på vilka huvudsakliga metoder som finns och hur de kan användas.

2.2.1. Kvantitativ metod

Patel et. al (1994) menar kvantitativt inriktad forskning syftar på att den använder sig av statistiska bearbetnings- och analysmetoder. Med hjälp av statistik och matematik så kan mätningar och därmed kvantifiering tas fram. Detta syftar alltså till sådana metoder som resulterar i numeriska observationer eller som kan bli omvandlade till det. Exempel på vad metoden kan innebära är empirisk observation, experiment, statistik, enkäter och frågeformulär.

”Företrädare för den kvantitativa forskningen skulle dock tillägga att, oavsett hur man kommer fram till resultatet och oavsett vilken metod man använder sig av i forskningsprocessen, måste man kunna göra empiriska generaliseringar av det uppnådda resultatet” Backman (1998); Barbosa de Silva et. al (1994), hämtat ur Starrin et. al (1994).

2.2.1.1. Experiment och hypoteser

Backman (1998) menar att den traditionella inställningen man har till den valda inriktningen, förhållningssättet eller synsättet är att det finns någon form av objektiv verklighet som skiljer sig från människan. Objekt, tillstånd och händelser existerar i sig själva utan människors inverkan. Den information som vi får från omgivningen tas emot av våra sinnen och ger oss den verklighet vi lever i, och levererar kunskap om den. Vi vill gärna förklara vår omvärld utifrån hur vi själva upplever den, vilket man enklast gör genom att även beskriva de människor som finns runt omkring, och på

så sätt hitta allmänna principer eller lagar. Det finns några olika sätt att framföra detta, och det kan exempelvis gå ut på att vi ”gissar” oss till hur saker och ting fungerar (genom teorier och antaganden). Dessa antaganden kan läggas fram genom att man testar och provar dem (hypoteser eller implikationer) med hjälp av ett antal olika premisser och omständigheter som realiseras (så som experiment).

Metoden kan användas då man vill ha reda på hur ett fenomen fungerar. Finns det inga tidigare studier inom samma ämne, eller om ett förhållande är svårt att förstå utan att prova sig fram så krävs det att man antar händelseförloppet och experimenterar. En sådan situation kan uppstå i samband med att jag provar att interagrera olika komponenter med varandra. De komponenter som åsyftas kommer beskrivas i ”2. Utvecklingsprocessen (Application lifecycle)”. Anledningen till att komponenterna ska kunna integrera med varann beror på att detta är önskvärt då många idag använder sig av heterogena lösningar med diverse olika komponenter från olika leverantörer. Intressenten önskar sig snarare en helhetslösning från en och samma leverantör.

2.2.1.2. Empiri

Patel et. al (1994) beskriver empirisk vetenskap som den kunskap som observeras utifrån verklighetsbaserade händelser. En ungefärlig översättning av empiri kan beskrivas med ordet erfarenhet. Man kan alltså säga att den kunskap människan tar in genom erfarenheter och observationer av omvärlden är empirisk kunskap. De flesta vetenskaper är empiriska, t ex fysik, kemi, ekonomi och sociologi. Barbosa de Silva et. al (1994), hämtat ur Starrin et. al (1994) skriver att empiriska förhållanden exempelvis kan uppnås via litteraturgranskning, som även visar hur begrepp inom området definierats, preciserats och använts inom tidigare forskning. En litteraturgranskning brukar ofta föra med sig ett antal fördelaktiga funktioner. En av de viktigaste funktionerna som den kan hjälpa till med är en formulering av problemställningen. Den kan även peka på tidigare brister i den kunskap som tagits fram och som beskriver problemställningens relevans. Tidigare dokument ger oss vidare upplysning om vilken metodik som använts, och ger även en bild av hur man tidigare lagt upp sina undersökningar (procedurer och design), hur man bearbetat sina data (kvalitativa och/eller kvantitativa tekniker), och hur man tolkat metodernas resultat.

Empirisk undersökning bör alltså användas då man vill beskriva något vetenskapligt, vilket kommer att bli nödvändigt för att exempelvis förklara hur olika samband fungerar. Den egna erfarenheten kan då kopplas ihop med tidigare forsknings erfarenhet, för att på så sätt få en bättre trovärdighet i det man skriver.

2.2.1.3. Enkäter

Dahmström (1991) skriver att ordet enkät härstammar från franskan och kan översättas till (vittnes)förhör. Detta förhör syftar numera till att man slumpmässigt väljer ut ett antal personer eller företag som tilldelas ett frågeformulär att fylla i och skicka tillbaka. I många år skickades dessa frågeformulär ut via vanlig post, med idag används främst Internetbaserade enkäter som för det mesta skickas ut med hjälp av e-post, där en länk bifogas till ett elektroniskt formulär.

En enkätundersökning kan komma till användning i samband med en informationsinsamling från ett urval personer som besitter en viss erfarenhet. Denna erfarenhet kan exempelvis syfta till att de har använt de olika verktygen som ska analyseras, vilket kan hjälpa mig att förstå dess komplexitet. Har de använt verktygen under en period kan de upplysa om dess fördelar och brister som kanske inte leverantören av verktygen ger så mycket information om.

2.2.2. Kvalitativ metod

Backman (1998) menar att den kvalitativa filosofin syftar mera till individen, vilket betyder att man i stället för att utgå från en objektiv verklighet ställer frågan hur individen tolkar och formar sin verklighet. De begrepp som får central betydelse i detta sammanhang är innebörd, kontext och process. Med innebörd åsyftas hur individer upplever, tolkar och strukturerar den verklighet de omges av i relation till sina tidigare kunskaper och erfarenheter. Kontexten avser det förlopp då man studerar människans händelser i det verkliga livet. Processer karakteriserar den kvalitativa aspekten snarare än resultatet, vilket är vad som återfinns i det traditionella synsättet.

Barbosa de Silva et. al (1994), hämtat ur Starrin et. al (1994) menar att det handlar om egenskaper eller beskaffenhet hos något. Med detta något avses företeelser man söker kunskap om, och inte själva metoden. Det vill säga det sätt man går till väga på, när man tar fram data, information och/eller empiri med avseende på egenskaper hos ett visst fenomen.

2.2.2.1. Hermeneutik

Barbosa de Silva et. al (1994), hämtat ur Starrin et. al (1994) påpekar att ”Hermes” ursprungligen var en gud i den grekiska mytologin, som agerade gudarnas sändebud. Hans uppgift var att översätta gudarnas budskap från gudarnas språk till det mänskliga. Hermeneutik betyder alltså att tolka, att översätta, att förtydliga, att klargöra, att förklara, att säga.

Metoden kan användas i viss utsträckning då man behöver förtydliga eller klargöra vissa invecklade situationer som uppstår. Detta är en grundläggande del av metodarbetet eftersom en stor del av projektet kommer att bestå i att tolka och förklara olika fenomen. Uppsatsen kommer att ta upp ett antal begrepp vars innebörd måste analyseras och utvärderas då dess innebörd inte alltid är självklar. Exempelvis kan det handla om när Open Source-synsättet ska beskrivas och tolkas utifrån ALM's olika komponenter.

2.3 Sammanfattning av metoder

De kvantitativa metoderna empiri och experiment är de metoder som kommer vara till störst användning i uppsatsen. All fakta om systemutvecklingssynsättet ALM och dess innehållande komponenter kommer beskrivas utifrån artiklar och litteratur, då jag inte besitter någon kunskap om synsättet sedan tidigare. De verktyg som kommer analyseras och som kan stödja synsättets innehållande komponenter har jag inte heller någon större erfarenhet av, vilket leder till att ett antal experiment kommer att krävas för att ta reda på deras funktion och användningsområden. Synsättet kan anses vara i en inledande fas, vilket betyder att det fortfarande finns flera olika sätt tolka det på, varpå hermeneutik också får en viktig roll. Slutligen kan det även vara intressant att undersöka om det finns utvecklare som besitter nyttig kunskap för min undersökning av verktyg. Denna kunskap kan jag få del av genom att skicka ut enkäter med ett antal relevanta frågor, om exempelvis användbarheten i de olika verktygen, till företag och chefer inom området.

3. Teori

Avsnittet har för avsikt att beskriva ämnets huvudsakliga termer och vad var och en innebär i en utvecklingsprocess. Den ska ge en förståelse av hur ALM-synsättet används idag och vilka problem detta kan innebära. Problemet analyseras även utifrån hur Open Source skulle kunna hantera det. Slutligen ska avsnittet även förklara varför det kan vara till stor hjälp att använda verktyg för att utföra synsättets olika delar.

3.1. Att använda Open Source

Weber (2004) skriver att de som använder Open Source-mjukvara traditionellt sett inte ses som kunder. Varför de inte ses som kunder beror på att användaren enkelt kan tanka hem mjukvaran från Internet och även har rätt att kopiera mjukvaran i ett oändligt antal utan extra kostnad. Han menar även att användarna interagerar med själva produktionsprocessen på ett grundligt sätt. På samma sätt som man kan betala för en licens av Microsoft Windows, så finns det också möjlighet att köpa sin Open Source mjukvarukopia. Med Open Source är det dock oftast frivilligt om man vill betala eller inte (ibland kan det handla om symboliska summor som ska täcka kostnaderna som uppstod vid leverans och liknande).Teknologin och licenseringsbestämmelserna runt Open Source stödjer på ett positivt sätt användare att vara aktiva deltagare i utvecklingsprocessen. Det är många som är det och på många olika nivåer. Det grundläggande målet med Open Source intellektuella egendomsregim är att maximera det aktuella användandet, storleken, utvecklandet och spridandet av fri mjukvara. Vidare är den huvudsakliga förutsättningen bakom Open Source att människor vill vara kreativa och originella och att det inte krävs för mycket inspiration för att uppnå detta sätt att konstruera på. Den enda gången nyskapandet kan bli otillfredsställt är när nyskapande människor förhindras från att komma åt det grundläggande materialet och de verktyg som behövs för att utveckla.

Raymond (2001) beskriver vidare att det sätt som Open Source framställs på ofta leder till att program utvecklas snabbare och att resultatet många gånger blir bättre än med traditionella avtalsenliga metoder. En följd av att många fler programmerare får tillgång till att inspektera koden, för att rätta till brister, blir att kvaliteten ökar eftersom antalet fel minskar. Att koden ideligen kontrolleras påverkar inte bara feltoleransen, utan även den generella designen. Ett program med dålig design blir ganska snart uppmärksammat och kommer antingen blir omgjort eller också ersättas av ett annat med en bättre utgångspunkt.

3.1.1. Supporttjänster inom Open Source

Weber (2000) förklarar att om en person vill tjäna pengar på Open Source mjukvara så paketerar och sprider de ut den på lätthanterlig media (såsom CD-ROM) och erbjuder sig sedan att hjälpa till med teknisk support och annan hjälp som kunden kan tänkas behöva. Att kunden betalar för att få denna form av support beror på den komplicerade proceduren som krävs för att få ett fungerande system. Oftast krävs det att man laddar hem produkten från Internet, för att sedan anpassa den till det egna systemet, installera den anpassade produkten på ett antal maskiner, skriva en utförlig dokumentation om användandet inom organisationen och även kontinuerligt uppdatera systemet. Att anställa en specialist som utför denna arbetsuppgift är oftast mycket mer effektivt än att försöka själv, även om det blir ”gratis” om man gör det själv. Den tid som går åt och de problem som uppstår vid egen installation kan till och med bli dyrare.

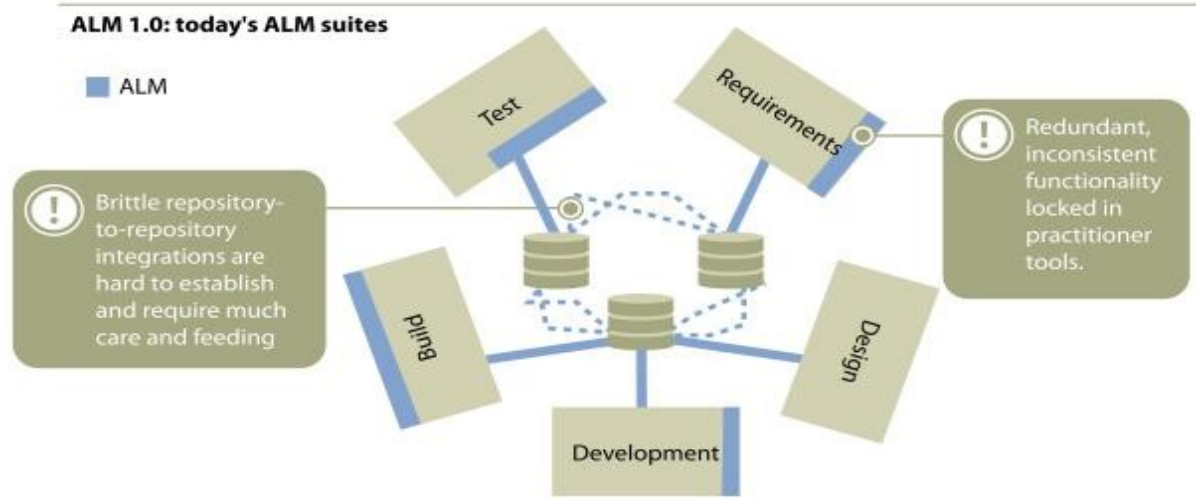
3.2. Utvecklingsprocessen (Application lifecycle)

Det IT-företag som examensarbetet utfördes på heter Signifikant (Signifikant.se), vars syn på ALM

innebär planering, definiering, design, utveckling, testning, driftsättning och ledning. Dessa begrepp kan tolkas på ett flertal olika sätt och de kan inkludera många processer. Vilka dessa processer är kan skilja sig en aning beroende på källa, men de flesta är överens om att krav är en grundläggande faktor som de övriga processerna delvis eller fullständigt är beroende av. Målet med detta avsnitt är att förklara hela livscykeln och även diskutera hur de olika komponenterna kan interagera med varann och varför de är beroende av varandra för att hjälpa till att utveckla system och mjukvara.

3.3. Application Lifecycle managment (ALM)

Techexcel som är en ledande leverantör av ALM-tjänster skriver att ALM är ett synsätt på en uppsättning komponenter som ofta sträcker sig över flera affärsenheter och uppgifter och som tillsammans kan vara till stor hjälp vid utveckling av system och mjukvara. De innehållande komponenterna har funnits sedan tidigare, men kombinationen gör det enklare för utvecklaren att ta hänsyn till alla steg som inkluderas i en utvecklingsprocess. Borland (2007b) menar att dagens ALM har ännu inte nått sin fullständiga potential. Detta beror på att utvecklarna av dessa system låser programmen till ett speciellt format eller utvecklingsmiljö som binder konsumenterna till kommersiella IT-plattformar. Ofta leder dessa lösningar till att konsumenterna inte kan interagera med existerande utvecklingsprocesser, verktyg och plattformar. Dessvärre så blir ofta resultatet att mjukvaruutvecklarna får ifrånkopplade processer med stora mängder ALM-data, vilket leder till att slutanvändaren inte kan få ut all den funktionalitet som ALM erbjuder.



***Figur 1**: Det övergripande problemet med dagens ALM-lösningar (Schwaber et. al 2006)*

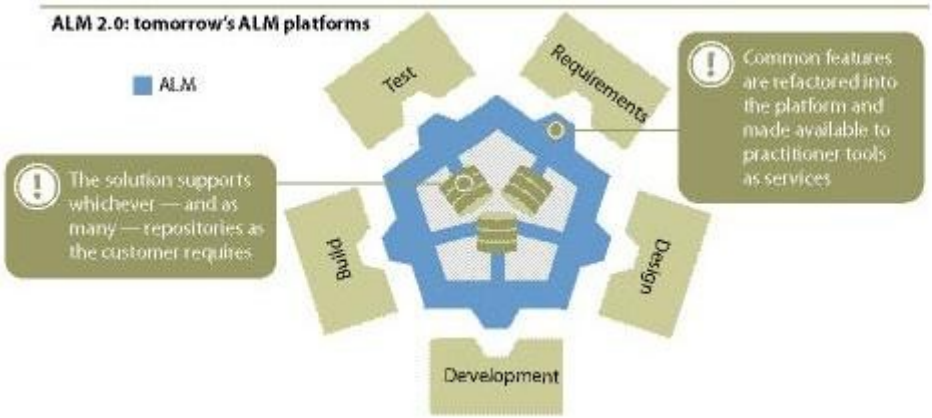
The hidden costs of ALM 1.0

Characteristic	Cost
A single tool for each role	• Low productivity due to tool complexity • Role misalignment leads to overspending on licensing
Redundant and inconsistent ALM features locked in practitioner tools	• Lack of cross-life-cycle transparency • Ossification of functional silos
Microprocesses embedded in tools and macroprocesses in tool integrations	• Effort spent building and maintaining synchronizations • No single source of truth
Integration through brittle repository synchronization mechanisms	Process assets are opaque and difficult to manage

37653 Source: Forrester Research, Inc.

***Figur 2**: Kostnader som kan uppstå i dagens ALM-lösningar (Schwaber et. al 2006)*

Schwaber et. al (2006) skriver att IT-organisationer årligen spenderar miljarder dollar på utvecklingsverktyg som inte kan samarbeta speciellt bra. Resultatet för de flesta är att ALM, som ska vara den uppsättning som samkör alla processer, blir ett manuellt bearbetningssätt. Dagens ALM-lösningar erbjuder därmed inte speciellt stort stöd jämfört med vad ALM-lösningarna kan uppnå genom lätthanterliga integrationer i verktygen. Morgondagens syn på ALM-plattformar har som mål att stödja grundläggande funktionalitet i utvecklingsverktygen. Dessa lösningar kommer att vara enklare att implementera, underhålla och använda, de kommer även ge en bättre möjlighet för utvecklingsföretag att bygga bättre mjukvara. ALM kan dock inte garantera framgångsrik framställning av mjukvara i sig själv och utan annat stöd. Stora delar av ALM's värde kan kopplas till uppdelade men relaterade operationer inom ”project portfolio management (PPM)”. Mer om vad PPM innebär kommer tas upp i avsnitt 3.3.4 ”Portföljshantering”, men dess största fördel är att den kan behandla ALM-data på ett sätt som stödjer beslutsfattande. Den nya ALM-plattformen ska underlätta samkörning och härledning av utvecklingsaktiviteterna genom att livscykelverktygen är oberoende och olåsta till någon speciell utvecklingsplattform.



***Figur 3**: Morgondagens ALM-lösning (Schwaber et. al 2006)*

ALM 2.0 is better

Characteristic	Benefit
Practitioner tools assembled out of plug-ins	• Customers pay only for the features they need • Practitioners find the features they need faster
Common services available across practitioner tools	• Easier for vendor to deploy enhancements to shared features • Ensures correspondence of activities across practitioner tools
Repository neutrality	• No need to migrate old assets • Better support for cross-platform development
Use of open integration standards	Easier for customers and partners to build deeper integrations with third-party tools
Microprocesses and macroprocesses governed by externalized workflow	• Processes are versionable assets • Processes can share common components

37653 Source: Forrester Research, Inc.

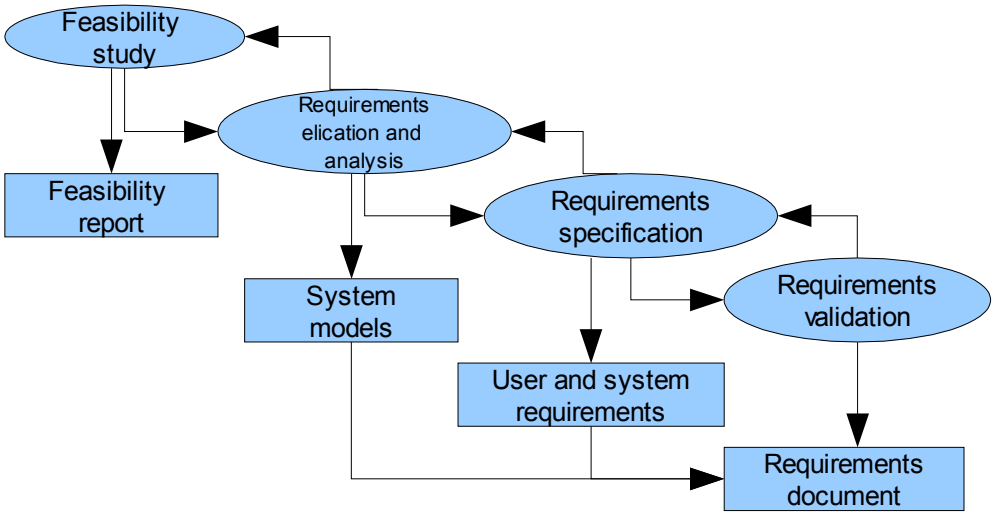
***Figur 4**: Morgondagens förbättringar av ALM-verktyget (Schwaber et. al 2006)*

3.3.1. Krav

Smith (2003) menar att krav definierar syftet med ett projekt och är grunden till en projektplan. De uppsatta kraven bestämmer vad som ska framställas och uppnås. Avsnittet nedan som beskriver en kravspecifikation, redogör hur processen kan se ut vid framtagande och utvärdering av krav. Vidare beskrivs även hur ett kravverktyg kan stödja processen samt vad som är karaktäristiskt i ett allmänt

3.3.1.1. Kravspecifikation

”Requirements engineering processes” innehåller enligt Sommerville (2004) stegen realiserbar studie, framtagande och analys av krav, kravgiltighet samt kravhantering. Den mest relevanta delen är kravhanteringen, men eftersom det är lämpligt att förstå hela processen så kommer även de övriga delarna i processen att diskuteras. Nedan beskrivs hela processen i en figur:



***Figur 5**: ”Requirements engineering processes” (Sommerville 2004)*

Realiserbar studie

Sommerville (2004) menar att en kravspecifikation bör inledas med en realiserbar studie vid formgivning av ett nytt system. Studien ska innehålla preliminära affärskrav, en övergripande beskrivning av systemet och hur systemet kommer att stödja organisationens processer. Resultatet från studien ger en rapport som beskriver huruvida det är någon mening att fortsätta med kravspecifikationen och systemutvecklingsprocessen eller inte. En realiserbar studie ska vara kort, fokuserad och bör svara på följande frågor:

1. Stödjer systemet de övergripande målen i organisationen?
2. Kan systemet implementeras med nuvarande teknologi och inom den givna kostnads- och tidsramen?
3. Kan systemet integreras med andra system som redan finns tillgängliga i organisationen?

Aspekten huruvida systemet kan stödja företagets mål är direkt avgörande; om ett system inte stödjer affärs/företagsmålen så får det inget egentligt värde för organisationen. Detta kan framstå som uppenbart men många organisationer utvecklar system som inte stödjer affärsmålen eftersom de inte har en begriplig förklaring av dem, vilket ofta beror på att kraven är dåligt definierade eller andra politiska eller verksamhetssamma faktorer som påverkat framtagandet. I den realiserade studien bör man även beakta informationskällorna vilket kan göras genom ledningen som vet var systemet ska användas, mjukvarutekniker som har koll på systemets syfte samt systemets slutanvändare.

Framtagande och analys av krav

Nästa steg i kravspecifikationen handlar enligt Sommerville (2004) om framtagande och analys av krav. I denna process jobbar mjukvaruutvecklarna mot kunder och systemets slutanvändare för att hantera användningsområdet, vad systemet ska stödja, systemets prestandakrav,

hårdvarubegränsningar och liknande. Processen bör vidare ta hänsyn till en varierande användarbas inom organisationen. Termen *intressent* kan användas för att hänvisa till en person eller grupp (exempelvis en slutanvändare) som, direkt eller indirekt, inverkar i systemet. När kraven har tagits fram bör de även klassificeras och organiseras för att upptäcka överlappande krav från olika intressenterna och även för att gruppera relaterade krav. Det enklaste sättet att gruppera kraven är genom att använda en modell som beskriver hur systemet är uppbyggt, för att på så sätt upptäcka undersystem och sedan koppla ihop kraven med varje undersystem. Intressenterna har ofta olika syn gällande betydelse och prioritet av kraven, vilket kan leda till konflikter. Det är därför lämpligt att genomföra en förhandling så att man kommer fram till en vettig kompromiss. En lämplig avslutning av denna process är att dokumentera de krav som tagits fram på ett sätt som kan vara användbart för att upptäcka ännu fler krav. I detta steg framställs ett utkast av hur kravdokumentationen kommer att se ut, vilket innebär att vissa krav kommer att vara bristfälliga.

Kravgiltighet

Sommerville (2004) menar att kravgilighet är den del som ska visa att kraven faktiskt levererar det som kunden vill ha, genom att beakta analysen och på så sätt hitta problem i kraven. Dess betydelse är väsentlig eftersom bristfällighet i kraven kan innebära att man får göra om mycket av arbetet om det upptäcks senare i utvecklingsprocessen eller till och med efter att systemet implementerats. Kostnaderna som uppstår vid en omfattande systemändring blir mycket större än om bara designen eller koden måste göras om. Anledningen till detta är att ändringar som uppstår i kraven leder till att designen och implementeringen också måste ändras innan systemet kan testas igen. Mer om konsekvenser och fördelar kommer att tas upp i nästa avsnitt ”Kravhantering”.

Kravhantering

Johnsson (2002) skriver att kravhantering är den vanligaste anledningen till att projekt blir ”misslyckade”. Det finns dock ett antal olika sätt att se på huruvida ett projekt är misslyckat eller inte, som inte syftar till huruvida det har blivit slutfört. Det som åsyftas är då bland annat förseningar och överskridningar i budgeten.

Enligt Bergvall et. al (2003) så innefattar kravhantering ett antal aktiviteter inom områdena insamling, kontroll, analys, filtrering och dokumentation av systemkrav. En viktig detalj enligt Smith (2003) är att krav ska specificera *vad* som ska göras och inte *hur* det ska utföras. Bergvall et. al (2003) förklarar vidare att implementering av kravhantering ger organisationer ett antal fördelar. Den mest grundläggande fördelen som tas upp är att arbetet får klarare riktlinjer vilket brukar leda till att dess beståndsdelar får mindre brister redan från början, vilket ofta är tidsbesparande eftersom man då inte behöver göra om lika mycket under senare delar av utvecklingsarbetet. Johnsson (2002) skriver att ju tidigare man upptäcker fel desto mildare brukar konsekvenserna bli. Ett fel som upptäcks sent i processen kan bli kostsamt eftersom många andra delar måste göras om som bygger på samma grunder. Smith (2003) påstår att kostnaderna kan öka med tre till tusen gånger så mycket om felet upptäcks i senare faser, jämfört med vad det hade kostat att upptäcka det redan vid kravhanteringen (illustreras även nedan i ”Figur 6: Kostnad relaterad till utvecklingsprocessen).

Kraven som finns i stora mjukvarusystem ändras kontinuerligt, skriver Sommerville (2004), vilket bland annat beror på oförklarliga eller konstiga problem (wicked problems). Dessa konstiga problem är ofta väldigt komplexa och har så många relaterade enheter att det inte finns någon bra specifikation av problemet. Eftersom problemet inte kan definieras fullt ut så blir vissa krav ofullständiga. Intressenternas förståelse för problemen ökar dock under utvecklingsprocessen, och kraven måste följaktligen beskriva den förändrade synen på problemet.

Phase in Which Fixed	Relative Cost
Requirements	1
Design	3 – 6
Coding	10
Development Testing	15 – 40
Acceptance Testing	30 – 70
Operations	40 – 1000

***Figur 6**: Kostnad relaterad till utvecklingsprocessen (Smith 2003)*

Enligt Bergvall et. al (2003) så tror många att kravhanteringsprocessen är överflödig eftersom det fördröjer produktens implementering, men kravinsamling ger utvecklarna en bättre bild av vad som måste ingå i systemet och även vad marknaden förväntar sig av det vilket är en viktig faktor för att få ett gott resultat. Göhlman et. al (2002) menar att man redan från början tar hänsyn till de intressenter som är aktuella genom att beakta deras krav och önskemål. Bergvall et. al (2003) tycker det är viktigt att man har en förståelse för vad kunden förväntar sig av systemet för att på så sätt undvika överflödig och oönskad funktionalitet och även för att man förhoppningsvis inte ska missa någon funktion som ska ingå. Göhlman et. al (2002) påstår vidare att kravhantering även kan vara till nytta vid exempelvis beslutsfattande. De konstaterar att en väl strukturerad kravspecifikation ger ett bra underlag för projektbeslut och kan fungera som en bas för hur utvecklingsprocessen ska se ut.

Finkelstein et. al (2000) skriver att företag kan utföra kravhanteringsprocessen på ett antal olika sätt. De kan för det första sätta upp sina krav i ett neutralt formulerat språk och sedan direkt övergå till designen. Ett annat alternativ som många använder sig av är att skriva kraven i ett neutralt formulerat språk samtidigt som de tillämpar en modellbaserad analys för att sedan gå vidare till designen. Endast ett fåtal använder sig av en ursprunglig modellmetod när de ska gå från analys till design. Enligt Wiegers (1999) så brukar utvecklare beskriva alla krav, med ett så neutralt språk som möjligt, i ett dokument som kallas för ”software requirements specification (SRS)”. Det finns dock ett antal begränsningar när man dokumenterar enligt SRS-standarden, exempelvis är det svårt att ständigt ha ett uppdaterat dokument vilket också leder till svårigheter att förmedla vidare vilka ändringar som har gjorts till de övriga utvecklarna i samma projekt. Det kan även vara svårt att på ett smidigt sätt ha med all information om alla krav, när de är invecklade eller om en längre förklaringar krävs för att inte missförstånd ska uppstå. Till sist kan det även vara svårt att definiera sambanden mellan funktionella krav och motsvarande användarfall, design, kod och projektuppgifter.

3.3.1.2. Kravverktyg

Att endast utgå ifrån metoder kan dock bli ett problem vid större utvecklingsprojekt, enligt Göhlman et. al (2002), då kravens omfattning växer och det blir svårt att få en överblick. Vid sådana omfattande projekt krävs verktygsstöd som kan ge ett bättre helhetsperspektiv. Verktyget ger även möjlighet att utnyttja kravinformationen så att det är enklare att styra och kontrollera utvecklingen. Wiegers (1999) menar att även om man har gjort ett utmärkt jobb beträffande projektets krav, så kan automatiserad assistans hjälpa till att behandla dem i en utvecklingsprocess. Nedan beskriver Wiegers (1999) ett antal olika angelägenheter presenteras som verktyg kan hjälpa till att genomföra.

- **Behandla versioner och förändringar**: Projektet bör ha en definierad utgångspunkt av uppsatta krav, som alltså ska innehålla en specifik samling krav som ska gälla för en viss

utgåva. Det finns även en historik om hur förändringarna sett ut sen föregående utgåva och vilka beslut som ligger till grund för dem.

- **Lagra kravkännetecken**: Det är inte sällan som det dyker upp mycket information om ett krav som syftar till hur det bör behandlas. All denna information eller kännetecknande delar bör lagras. Detta gör det enklare för alla som jobbar med projektet eftersom de har möjlighet att se vilka kännetecken som är aktuella, då vissa personer måste uppdatera vissa värden kontinuerligt. Kravhanteringsverktyg kan generera ett antal definierade kännetecken om systemet, såsom vilket datum den skapades och versionsnummer, och de ger en möjlighet att definiera ett flertal kännetecken i olika dataformat. Kännetecknen kan innebära författare, ansvarig person, grundläggande uppkomst, nummer på utgåva, status, kostnad, svårighetsgrad, stabilitet och risk.
- **Krav som kopplar ihop systemets olika enheter:** Genom att spåra individuella krav till andra systemkomponenter kan man gardera sig mot att någon i utvecklingslaget missar något krav vid en implementering. Man kan koppla ihop olika typer av krav och krav mellan olika undersystem. När vidare effekten av förändringarna utvärderas i ett specifikt krav, kommer man även se vilka förändringar som kommer ske i de övriga systemkomponenterna.
- **Status:** Går ut på att man kan se i vilket tillstånd kraven befinner sig under utvecklingsprocessen. Om exempelvis projektledaren vet om att 55 % av alla lokaliserade krav kommer att verifieras i nästa utgåva, och att 28 % inte kommer bli verifierade, medan 17 % inte kommer implementeras fullt ut, så har projektledaren en bra insyn i projektets status.
- **Granska kravens underansatser:** Det finns möjlighet att sortera, filtrera och ge databasen förfrågningar för att få upplysning om kravens underansatser med specifika kännetecknande värden.
- **Rättigheter:** Man kan ge ut olika rättigheter till en individ eller en grupp användare. Åtkomst via Internet ger även möjlighet att dela kravinformationen med alla lagmedlemmar, även om man inte befinner sig på samma plats.
- **Kommunicera med arbetskamrater:** De flesta kravhanteringsverktyg ger även möjlighet att kommunicera elektroniskt med andra lagmedlemmar angående kravutfärande. E-post meddelande kan ge upplysning till berörda parter om det har uppkommit en ny diskussion eller om ett krav har ändrats.

Finkelstein et. al (2000) menar att diskussionen ovan behandlar nyckelfunktionaliteten i kravhanteringsverktyg och att den till stora delar styrs av allmänna aktiviteter inom dokumenthantering.

3.3.1.3. Kravhantering och Open Source

Scacchi (2001) skriver att i jämförelse med den traditionella mjukvaruutvecklingen (software engineering) så använder inte ”open software development communities” modern mjukvaruutveckling eller kravhanteringsprocesser på samma sätt. Dessa ”communities” utvecklar mjukvara som är värdefull, allmänt pålitlig, oftast trovärdig och användarvänlig inom dessa kretsarna. Det som är intressant är alltså att undersöka vilka processer och metoder som används för att utveckla krav för ”open software”-system. Främst kommer vikten läggas på Internetbaserade Open Source projekt. Scacchi et. al (2005) menar då att ett projekts hemsida innehåller ett nätverk bestående av dokument eller artefakter som kan nås via hyperlänkar. Med artefakter menas exempelvis hemsidor, e-post diskussioner, felrapporteringar, källkodsfiler och liknande.

Scacchi (2001) skriver vidare att det förekommer att ”open software”-krav finns tillgängliga på Internet på ett tillfredsställande sätt. Det finns även andra fall då kraven påträffas i ett e-post meddelande eller i ett diskussionsträd (forum) som finns tillgänglig i anslutning till eller på

projektets hemsida. Dessa krav kan då förekomma utan referens till andra dokument, källor eller standarder; de är krav eftersom utvecklarna önskar dessa möjligheter. Möjligheterna kategoriseras vidare som funktionskrav som redan är implementerade i systemet. De inblandade utvecklarna rättfärdigar sina krav genom programmeringskoden och gör möjligheterna funktionsdugliga. Möjligheten blir vidare en del av systemets distribution genom att projektets äldre medlemmar eller dess kärnutvecklare röstar eller samtycker genom diskussion. Den historiska berättelsen bör finnas, i e-post meddelande eller i forumsdiskussionen, för att kunna få en överblick om vem som krävde vad, när, varför och hur.

Scacchi et. al (2005) skriver att de studerat ett antal olika utvecklingsprojekt inom Open Source och att det är vanligt, precis som Scacchi (2001) skriver, att kraven framträder och försäkras efter att de har blivit implementerade snarare än innan de blivit implementerade. Dessa utvecklingsprojekt skiljer sig även från traditionell ”software engineering”eftersom de inte behöver tänka på budgetbegränsningar, schemaläggning, och projektledningsrestriktion på samma sätt. Det är även så att Open Source-utvecklare dessutom kan agera slutanvändare eller administratör till projektet de utvecklar, snarare än att det är åtskilda som utvecklare och användare som brukar vara fallet.

3.3.2. Konfigurationshantering

Konfigurationshantering (CM) är enligt Asklund (1999) det område inom mjukvaruutveckling som kontrollerar och hanterar projekt för att hjälpa utvecklarna att synkronisera sina arbeten med varandra. Detta möjliggörs genom att definiera metoder och processer som ska gälla, sätta upp en plan som ska följas och genom att använda CM-verktyg som hjälper utvecklarna och projektlederna med sitt dagliga arbete. Alla projekts huvudmål är att förändra någonting menar Smith (2003). I tidigare avsnitt om kravhantering så beskrevs Sommerville's (2004) syn på att kraven alltid förändras under utvecklingen och användandet. Dessa förändringar bör följaktligen bli en del av den nya systemversionen.

Smith (2003) skriver vidare att ett system exempelvis kan bli uppgraderat eller utbytt för att uppnå bättre funktionalitet, användarvänlighet eller för att minska driftkostnaderna. Inom mjukvaruutveckling och andra projekt så måste förändringsförslag utvärderas för att få förståelse för dess bidrag till projektmålen. Det är viktigt att se om de leder till förbättringar av projektet eller om det direkt hindrar eller försämrar kvaliteten. Alla förändringar bör beaktas, även de som visar ultimata fördelar vid introduktionen och implementeringen. Förser man exempelvis ett flygplan med en större och kraftfullare motor kan man se klara förbättringar, men den kan inte tas i drift förrän strukturen på flygplanet med säkerhet kan intyga att en den är kapabel, eller att den har uppgraderas, för att klara av den nya vikten och hastigheten.

CM har fått en allt mer central roll under de senaste åren anser Asklund (1999). En anledning till det är ”SEI Capability Maturity Model (CMM)” där CM är en av nyckelprocesserna i modellens aktivitetscykel. Mer om CMM och dess betydelse kommer tas upp i avsnitt ”3.3.6 Capability Maturity Model Integration CMMI”. En annan väldigt viktig anledning, som Asklund (1999) beskriver, är att mjukvara blir allt större och mer komplex vilket leder till att den får större behov av CM-stöd. Vad som exempelvis ofta gör saker mer komplicerade är större begäran gällande den tid det tar att få ut produkten på marknaden, som kräver en stegrande och samstämmig utveckling, vilket i sin tur ökar behovet av CM-stöd. Vad som också gör CM mer och mer användbart är det faktum att utvecklarna i allt större utsträckning sprider ut sig rent geografiskt, men att de ändå arbetar med samma projekt.

Asklund (1999) skriver vidare att CM kan sägas inrikta sig på två olika målgrupper som har skilda behov och krav, nämligen ledningen och utvecklare. Ledarperspektivet innebär att CM styr och

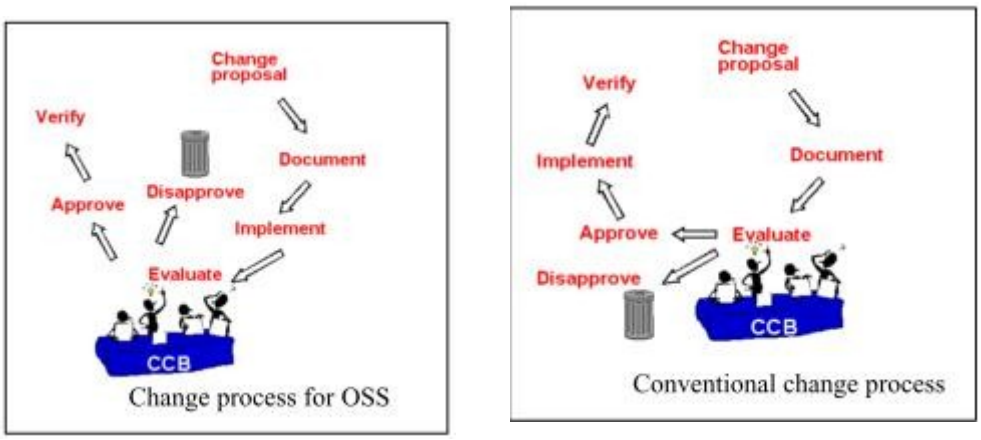
kontrollerar produktutvecklingen genom att identifiera produktens komponenter och se över deras fortsatta förändring. Målet är att dokumentera en produkts komponentuppsättning och status, som sedan kan användas som en utvecklingsbas för att få ett tillfredsställande resultat. I det utvecklingsbaserade perspektivet (som behandlar verktygsstöd), så underhåller CM produktens aktuella komponenter, lagrar dess historik, erbjuder en stabil utvecklingsmiljö och kopplar ihop förändringar av liknande karaktär i produkten. Målet är att göra utvecklingsgruppen så effektiv som möjligt med hänsyn till deras aktuella bearbetning av produkten. CM inkluderar både produkten (konfiguration) och även arbetssättet (metoder).

De viktigaste funktionerna som CM erbjuder är enligt Asklund et. al (2002) versionskontroll, utvecklingshantering, konfigurationsval, arbetshantering sammanställningskontroll, förändringshantering och utgåvehantering. Med versionskontroll åsyftas möjligheten att lagra olika versioner och även olika former av dokument för att hela tiden ha möjlighet att jämföra dem. Utvecklingshantering är mekaniska processer för att samla all data i ett system och utveckla systemet genom att uppdatera de genererade filerna. Konfigurationsval är en funktion som väljer de dokumentversioner som tillsammans utgör ett komplett och överensstämmande system. Arbetshantering handlar om att utvecklare ofta vill jobba med förändringar, utan att samtidigt behöva tänka på versionen eller att se de förändringar som andra gör i samma process. Sammanställningskontroll stödjer den flerfaldiga användningen och åtkomsten genom att antingen hindra den eller stödja den, vilket i båda fallen hjälper till att synkronisera utvecklarnas arbete. Förändringshantering är den process som behandlar förändringsförfrågningar, felrapporteringar, implementering av dessa förändringar, dokumentering av problemet och en lösning, samt när det är genomförbart. Utgåvehantering handlar om identifiering och organisering av alla dokument och resurser som ingår i en utgåva. Utvecklingsledaren är ansvarig för att produkten förses med rätt konfiguration och egenskaper.

3.3.2.1. Konfigurationshantering och Open Source

Mjukvaruprojekt inom Open Source har enligt Asklund et. al (2002) ett anarkistiskt sätt att organisera projekt på och de menar även att en så stor uppsättning utvecklare gör det svårt att hantera konfigurationshanteringsområdet. Trots detta vill Open Source utvecklare producera mjukvara som åtminstone har likvärdig kvalitet som konventionella producenter av mjukvaruutveckling.

Som nämnts tidigare så finns det ett antal anledningar till att ändra saker i ett system, som ofta kan kopplas ihop med perfektionism, korrekthet och anpassningsbara ändringar. För att få en klar dokumentation av ändringarna så krävs verktyg och processer som kan strukturera och beskriva det som har hänt från projektets begynnelse och fram till den aktuella implementeringen av källkod.



***Figur** 7: Skillnaden på Open Source och konventionell förändringsprocess (Asklund et. al 2002)*

I Open Source mjukvara så behöver inte ändringsförslagen vara klara, om det ens finns något förslag. Vem som helst kan ändra och oftast så granskas inte förslagen innan de implementeras. Ändringsförslagen kan uttryckligt eller underförstått få olika prioritet, men det brukar inte förekomma någon utdelning av uppdrag eller problemlösning till utvecklarna; alla jobbar med det som den önskar. Vidare kan två olika scenarion inträffa angående huruvida nya inlägg behöver passera en administratör eller om personens rättigheter räcker för att implementera förändringen. I båda fallen så inträffar dock samma sak i nästa process, som innebär att ändringsförslaget/inlägget tas emot, implementeras och testas och slutligen utvärderas genom testning, recension och diskussion. Den slutgiltiga utvärderingen resulterar vidare till en patch som antingen förkastas eller också leder till en ändring som förvaras och infogas av en koordinator. Vanligtvis ger man bara skriv- och ändringsrättigheter till utvecklare med ett visst förtroende, så att inte ändringsförslag behöver förkastas så ofta.

3.3.3. Testhantering

Whittaker (2000) beskriver en situation som de allra flesta utvecklare har upplevt, nämligen när deras mjukvara får en bugg/felrapportering från missnöjda användare. Den allmänna frågan som dyker upp är hur dessa buggar kunde undgå testningen. Det som kan ha uppstått är att någon har lagt till kod som inte har testats, vilket inte är ett ovanligt fenomen i samband med tidsbrist. Ett annat scenario som kan ligga bakom felet är att testarna inte har provat alla olika kombinationer av funktioner som tusentals användare kan skapa. Vissa saker är väldigt svåra att testa och man måste följaktligen besluta hur testningen ska gå till utifrån en lämplig rimlighetsnivå, och ibland tar man felaktiga beslut. Vad som också kan vara fallet är att slutanvändarens operationsmiljö inte testades. Detta kan bero på svårigheten att testa en exakt kombination av hårdvara, kringutrustning, operativsystem och andra applikationer.

Testhantering är enligt Kumar (okänt årtal) en metod som hjälper till att ta fram de artefakter som ska testas. Veenendaal et. al (1997) skriver vidare att hur man än utför sina tester så handlar det alltid om planering, förberedelse och exekvering. Man brukar göra en uppdelning mellan dessa aktiviteter som exempelvis kan se ut på följande sätt: 20% planering, 40% förberedelse och 40% verkställande exekvering. Genom att specificera dessa funktioner så har man skapat en ”master test plan”. Denna plan beskriver vem som utför vilken typ av test och när det utförs. Planen bör innehålla alla olika testtyperna. Ibland gör man dock begränsningar vilket kan innebära ”black-box testing” (system- och acceptanstestning via specifikationsbaserade tekniker) eller till ”white-box testing” (programbaserade tekniker). Chang Liu beskriver i en artikel på freshmeat.net (2000) de programbaserade tekniker som en process som utvecklar testfall (test cases) utifrån programstrukturen utgår ifrån. Testfallens syfte är att hjälpa till att beskriva programmets kontroll- och datastruktur samt dess olika beteenden. Det enklaste sättet att få en bra överblick av vilka testfall som är relevanta är genom problemspecifikation eller liknande beskrivningar. Syftet med metoden är alltså att om programmets uppgift är att lösa ett problem så spelar inte dess uppbyggnad så stor roll så länge problemet blir löst.

Veenendaal et. al (1997) menar att de olika testtyperna agerar inom olika områden, och att det därför blir nödvändigt att alla målen, uppgifterna, skyldigheterna och inriktningarna beskrivs tydligt. Efter att ha definierat testplanen, som även inkluderar att sätta upp en specifikation, design och programmeringsprocess, så kan man börja utveckla testfall och teststruktur. Planerar man sin testprocess genom att fundera ut en bra strategi hur testerna ska utföras och även gör en riskbedömning så kan kostnaderna minska betydligt. Praktisk tillämpning visar att design av testfall tillsammans med en kravspecifikation kan avslöja en stor del av existerande felaktigheter, vilket i stort sätt alltid är värt de kostnader som lagts ut på metoden. Misstag och oriktighet är enklare och mindre kostsamt att reparera desto tidigare de upptäcks, vilket har konstaterats tidigare.

3.3.3.1 Testare och testprocessen

För att planera och utföra test så är det enligt Whittaker (2000) viktigt att testerna har en bra överblick på den mjukvara och de funktioner som ska undersökas och även vilken information som tas emot av de olika funktionerna och hur informationen kan hanteras. Det är också viktigt att kontrollera den miljö som mjukvaran ska arbeta emot. Det är en svår och tidskrävande process som kräver planering och hög insikt i den föreliggande tekniken. Testarna måste ha bra erfarenhet av utveckling, eftersom testningen ofta kräver undersökning av kod, och även viss kunskap i formella språk, grafteori och algoritmer. De brukar dock inte bry sig så mycket om någons specifika sätt att skriva kod på, utan snarare hur helheten fungerar. Vidare angående inkommande information (input) så undersöker de inte heller individuella lösningar utan snarare mjukvarans huvudsakliga funktioner och funktionalitet. Testarna strävar hela tiden efter den metod som är bäst och lämpligast för att hitta så många fel som möjligt.

3.3.3.2. Testverktyg

Velusamy skriver i en artikel på freshmeat.net (2004) att nästan alla system har alldeles för många testfall och att man bara har tid till att utföra en del av dem, vilket leder till att man väljer de som förväntas hitta flest fel i mjukvaran. Det är i stort sätt omöjligt att upptäcka alla fel, och därför är det lämpligt att använda testverktyg. Det finns åtskilliga testverktyg som kan inrikta sig på olika områden och mål med testerna. Det är viktigt att man väljer ett väl anpassat verktyg till just den utvecklingsmiljö som hanteras för att testningen ska bli lyckad. Vilket testverktyg som är lämpligast kan bestämmas genom testets mål.

För att beskriva varför automatiska tester många gånger kan vara mer effektiva än manuella eller ”mänskliga” tester, så följer här ett exempel baserat på ett systems stresskänslighet. Vid ett manuellt test skulle en grupp testare kunna logga in på systemet samtidigt för att på så sätt undersöka maximala laddningstider. Detta kan dock inte sägas vara ett tillfredsställande sätt att undersöka situationen på, eftersom den innebär många begränsningar, exempelvis kan inte prestandan mätas precist eller upprepningsvist. Ett anpassat verktyg för laddningstid kan användas som en simulering av flera virtuella användare under reglerade stressförhållanden för att på så sätt mäta prestandan mer exakt.

Velusamy har även i sin artikel på freshmeat.net (2004) beskrivit ett antal aktiviteter som ett verktyg kan hjälpa till att genomföra:

1. **Definiera ett fullständigt testkriterium:** Testförsöket bör ha specifika fastställda mål. Testningen är avslutad först när alla mål är uppnådda (exempelvis att testningen är klar när testet har identifierat samtliga systemfunktioner och exekveringen går igenom).
2. **Designa testfall:** Logiska testfall definieras utifrån tre parametrar: Systemets tillstånd innan testningen påbörjas, vilka testfunktioner som ska utföras samt de förväntade resultatet.
3. **Utveckla testfall:** Ta fram nödvändig information och utforma tekniska komponenter för att automatisera testexekveringen.
4. **Genomför testerna:** Exekvera testfallen mot systemet som ska testas och dokumentera resultaten.
5. **Kontrollera testresultaten:** Kontrollera testresultaten så att det förväntade resultatet är uppnått samt att testfallen når upp till testkriteriet.
6. **Kontrollera testets täckningsgrad:** Undersök hur många fler funktioner som kan tillämpas vid varje utfört test (så att verkligen testet gör någon skillnad, och vilka skillnader som tillkommer)
7. **Gör ett testbibliotek:** Testbiblioteket underhåller relationerna mellan testfallen och programmen som har testats. Biblioteket bör även hålla reda på vilka test som utförts och inte, och även huruvida de exekverade testen har lyckats eller inte.

3.3.3.3. Testhantering och Open Source

Andrews (2000) menar att ledningen inte alltid har en bra översikt gällande produktens flöden; företagschefer utgår ifrån att alla funktioner är bristfria och fungerar tillfredsställande. Det är dock väldigt ovanligt, för att inte säga obefintligt, med mjukvara som inte innehåller några brister och som är helt säkra. Det kommer alltid finnas fel/brister i mjukvara; huruvida de upptäcks är en annan sak. Open Source gör det enklare att peka ut fel och flöden i koden, och genom aktiv säkerhetskontroll via allmän beskådan skapas en stabilare mjukvara. Zwartjes et. al (2005) skriver att sättet som Open Source mjukvara testas på är unikt. Genom att erfarna slutanvändare kan rapportera in fel så blir kontrollen mer omfattande. Andrews (2000) skriver vidare att detta är en anledning till varför kommersiella produkter ibland misslyckas med testningen; deras produkt kan inte få något stöd utifrån vilket gör det svårare att utforma en pålitlig och flexibel mjukvara eftersom de har begränsade möjligheter till programmeringsforum utanför organisationen som kan hjälpa till att optimera koden. Raymond (2001) skriver att den hjälp de kommersiella utvecklarna kan få av människor utanför den egna organisationen är genom felrapporteringar, men de måste för det mesta lösa problemen själva. Med sluten källkod får inte slutanvändaren någon överblick av vad problemet är; den beskrivning som användaren ger baseras på vad personen har upplevt och är ofta formulerat med ett språk som utvecklaren (programmerare eller testare) inte riktigt förstår. I en öppen programutvecklingsmodell blir detta problem inte lika omfattande, eftersom det blir enklare för användaren att peka ut grunder till problemet i källkoden och man kan som slutanvändare skapa sig en gemensam bild med utvecklaren om hur problemet kan lösas. Utvecklare inom Open Source kan därför sägas ha större nytta av felrapporterna eftersom det finns möjlighet att koppla problemet direkt till källkoden, vilket dock inte alltid behöver vara fallet eftersom alla slutanvändare inte har sådan erfarenhet. En felrapport för sluten källkod kan dock inte peka på så mycket annat än synliga symtom i programmet, vilka ofta är problematiska att beskriva i detalj då det många gånger är svårt att komma ihåg precis vad man gjorde när felet uppstod.

3.3.4. Portföljshantering

Gareis (2002) menar att projektinriktade organisationer utför ett antal olika projekt och program samtidigt, vilket ofta samlas ihop i en projekt- och programportfölj. Denna portfölj kan alltså definieras som en uppsättning projekt och program som en organisation hanterar parallellt under en angiven tidpunkt och där relationerna tydligt framgår mellan de olika projekten och programmen. Det blir mer och mer komplext desto fler projekt och program som företaget implementerar samtidigt. Komplexiteten kan bero på att projektens och programmens antal och storlek konstant förändras, både ordinarie och provisoriska källor läggs till, och samband fastställs genom gruppering. Vidare inrättas strategiska förbindelser och förhållanden till olika allmänna miljöer där projekten och programmen behandlas

3.3.4.1. Portföljshantering i projekt

Hughes (2007) och Korpiaho (2007) menar att hanteringen av självverkande projektinriktade organisationer kräver fokus (strategisk planering), en utvärderingsfas (projektets utvärderingsfas), val av de mest passande projekten (portföljshantering) och även en målmedveten utveckling (projekthantering). Allt detta krävs för att få fram den rätta portföljen som ska generera en framgångsrik portföljshantering i projektet (project portfolio management).

Strategisk planering

Korpiaho (2007) rekommenderar att en organisation har en klar strategisk riktning innan de börjar beakta vilka olika projekt som ska ingå i portföljen. Riktning och fokus ska beskriva organisationens starka och svaga sidor samt den nuvarande positionen och var man vill befinna sig i framtiden. Det är viktigt att redogöra hur strategin tas fram och utvecklas, det är dock inte

nödvändigt att beskriva vilka metoder som finns för att utveckla en strategi om portföljhanteringen utgår ifrån en existerande strategi. Strategin implementeras genom riktlinjer och uppsatta affärsmål.

Projektets utvärderingsfas

Utvärderingsfasen innefattar enligt Korpiaho (2007) de projektförslag och pågående projekt som har nått upp till ett visst delmål och följaktligen blir utvärderade med ett antal beslutskriterium. Detta möjliggör utformning av en kombinerad portfölj, vilket syftar till lämpliga projektförslag, gamla projekt, källor och möjliga förändringar i miljön som man arbetar mot. Genom att förena beslutskriterierna så kan man enkelt jämföra gamla och nya projekt. Då de nya och gamla projekten konkurrerar om samma knappa resurser, så är det bara de bäst lämpade som tilldelas resurser.

Portföljhantering

Korpiaho (2007) menar att denna fas huvudsakligen innebär att konstruera en tänkbar portfölj och analysera dess stabilitet. Dessa två steg repeteras tills en acceptabel portfölj, som uppnår resurskraven, upprättas. Den tänkbara portföljen kan konstrueras utifrån olika typer av metoder och ansatser, exempelvis ”scoring models” och/eller monetära förväntningar. ”Scoring models” syftar till att rangordna projekt och därigenom föreslå en portföljsammansättning utifrån de bäst klassificerade projekten, medan det monetära synsättet konstruerar en tänkbar portfölj utifrån en analytisk metod som utgår ifrån den ekonomiska situationen.

Projekthantering

Korpiaho (2007) menar att de flesta metoder vars syfte går ut på att hitta den mest lämpade portföljen används inte på grund av komplexiteten, då de ofta kräver så mycket information, de ger en otillräcklig bild av risker och pålitlighet, och de kan inte identifiera tvetydiga förbindelser och oväsentliga kriterium. De brukar bara vara svåra att använda och förstå, eller också används de inte i målinriktade processer. Det är därför ofta lämpligare att implementera processen utifrån de kriterium som karaktäriserar företagets kultur. Beslutsfattarna tar åt sig den nya operationsmodellen på ett naturligare sätt om den innehåller bekanta och otvungna komponenter.

I dessa projekt är det också viktigt att komma ihåg att projektförslagen inte alltid bör utföras. Man bör också göra beslut där projektförslagen får statusen ”utför inte” ,eller för att uttrycka det på ett mildare sätt ”tillfälligt hålls tillbaka” eller ”planeras om”, som alltså är en mycket viktig del i en framgångsrik projekthantering. Det strategiska projekthanteringssynsättet ser inte ”utför inte”-beslut som misslyckanden.

3.3.4.2. Portföljhanteringsverktyg

Cardin (2007) menar att det många gånger kan det vara svårt att framställa en planering som stämmer överens med vad som händer i verkligheten. Då det idag blir allt högre förväntningar på företags kapacitet gällande leveranstid och kvantitet så blir planeringsprocessen allt mer komplex att utföra manuellt. De allt mer komplexa processerna leder till att fler och fler har börjat implementera portföljhanteringsverktyg som kan vara till stöd vid urval, övervakning och ledande av IT-projekt. Ett verktyg kan enligt Borland (2007c) hjälpa till med följande:

- Effektivisera val av portfölj
- Gynna och underhålla enighet gällande de bästa interna tekniska investeringarna
- Öka gynnsamheten på IT-investeringar genom att fokusera på de mest fördelaktiga källorna
- Mer verklighetsbaserad planering

3.3.5. Design

Waldo (2006) menar att all mjukvara har någon form av design, vilket dock inte behöver betyda att

mjukvaran är designad. Att designa ett system kräver en väl utformad strategi, där all funktionalitet bryts ner till mindre komponenter, vilket bland annat kan innebära att forma enkla objekt som kan återanvändas och omformuleras på olika sätt för att uppnå den önskade funktionaliteten. Design är den process som beskriver hur delarna av något större ska se ut, och hur dessa delar passar ihop.

Sommerville (2004) menar att mjukvarudesign är en beskrivning av mjukvarans struktur som ska implementeras, den data som ska ingå i systemet, gränssnitten mellan systemkomponenter och ibland även algoritmer som används. Designutvecklare kommer inte fram till en slutgiltig design med en gång utan utvecklar den i flera steg, som specificeras i flera versioner. Dessa versionsspecifikationer bör vara detaljerade eftersom designen utvecklas genom att undersöka tidigare versioner och senare rätta till de fel som uppstått. Nästa steg består i att beskriva varje designaktivitets effekt. Denna beskrivning kan utformas som en sammanfattning, en formell beskrivning som klargör alla krav, eller också beskrivs alla realiserbara delar av systemet. I takt med designprocessens fortlöpande formgivning så blir beskrivningen allt mer detaljrik. Det slutgiltiga resultatet av denna process leder till en konsekvent beskrivning av algoritmer och datastrukturer som ska implementeras. Den naturliga designprocessen innebär att utveckla programmen följt av att implementera dem. Det finns dock system som måste designas på ett detaljerat sätt innan de kan implementeras, vilket bland annat gäller i system där säkerhet är en viktig del. Det vanligaste är emellertid att lägga till förhållanden när de uppmärksammas under senare delar av design- och programutvecklingen. Enligt Mayfield (2005) så bör även designdelen innehålla en prototyp av alla gränssnitt som mjukvaran ska visa. Designfasen med alla användargränssnitt beskriver varje synligt datafält, exempelvis hur en knapp ska se ut och vad som händer när användaren trycker på knappen. Designen bör även betrakta vilken typ av förfrågningar som ligger till grund för varje knapp (alltså hur den unika funktionen som ska utföras).

Sommerville (2004) menar att det inte finns någon perfekt metod, och de metoder som finns är lämpade till olika områden. Exempelvis är objektorienterade metoder ofta användbara i interaktiva system men inte i system med logiskt följdriktiga krav. Alla metoder baseras på konceptet att utveckla modeller av ett system som beskrivs grafiskt och kan användas som systemspecifikation eller design. Nedan beskrivs två mycket använda metoder.

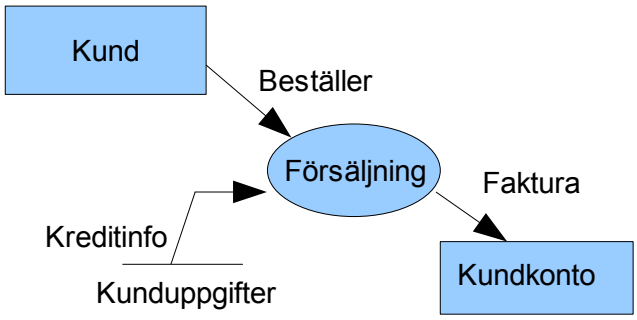
3.3.5.1. Strukturerad design

Enligt Fernandez et. al (1994) så ser strukturerad design varje system som en funktion som omvandlar inkommande information till utdata. Den grundläggande uppgiften blir att designa denna omvandlingsfunktion. Smith (2003) beskriver att en strukturerad design skulle göra det enklare att dokumentera, underhålla och även att designa mjukvaran. De två huvudsakliga reglerna i metoden innebär för det första att programmen delas in i funktioner och enklare slentrianer, och för det andra så ska det endast vara en unik avslutningspunkt för varje funktion eller slentrian. Fernandez et. al (1994) skriver vidare att designen utgår ifrån att den viktigaste beståndsdelen inom ett system alltid är funktioner medan övrig data prioriteras lägre.

Designens metod består, enligt Fernandez et. al (1994), av en strukturerad tabell (structure chart) som beskriver alla funktioner och även vilka transaktioner som sker emellan dem. Metoden kan sägas innehålla fyra steg:

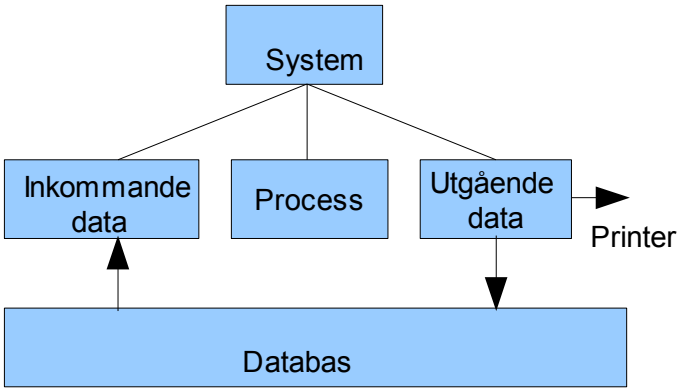
1. Formulera om problemet som ett dataflödesdiagram. Ett dataflödesdiagram är enligt Brown (2002) ett diagram som visar vilken person eller program som kommer utföra varje process. Det visar precis hur data tar sig från ena aktiviteten till nästa och om det skrivs ut på ett papper eller skickas ut som en elektronisk signal eller något annat format eller utförande. Det visar även vilket format all information sparas som. Figuren nedan visar hur ett

dataflödesdiagram kan se ut för en kund som ska utföra en beställning och hur ett företag kan behandla information om denna beställning.



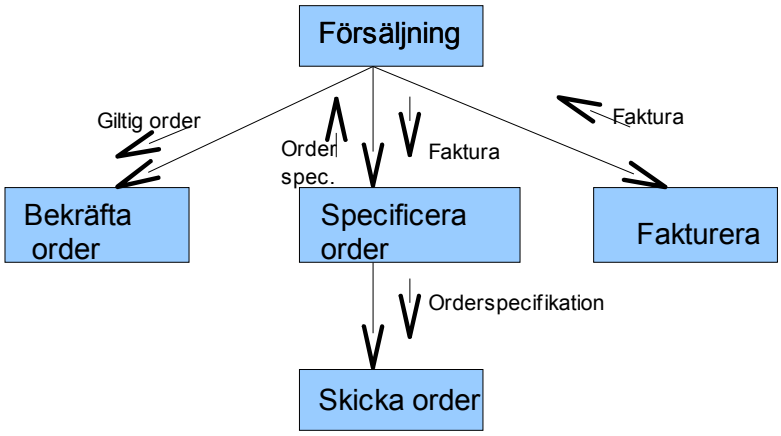
***Figur 8**: Dataflödesdiagram (Guo 1997)*

2. Hitta de inkommande och utgående dataflödena. Sommerville (2004) menar att en komponent läser inkommande data (input) från en fil eller en databas, undersöker datans validitet och korrigerar eventuella felaktigheter (process), sedan skickas den giltiga datan till databehandling (utgående data eller output). Figuren nedan visar hur dataflödet ser ut mellan ett system och en databas, och även hur den utgående datan kan skickas till en printer för utskrift.



***Figur 9**: Input-process-output modell (Sommerville 2004)*

3. Gör om dataflödesdiagrammet till en strukturerad tabell. Figuren nedan beskriver hur försäljning av en produkt kan hanteras av en leverantör, vilket även är en delaktivitet i dataflödesdiagrammet ovan.



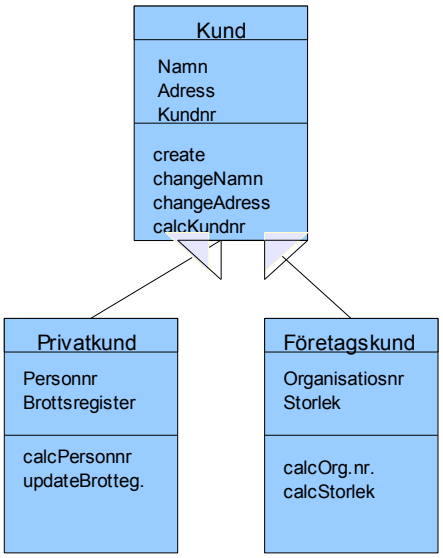
***Figur 10**: Strukturerad tabell (Guo 1997)*

4. Skapa förhållanden mellan inkommande, utgående och omvandlade moduler. Brown (2002) beskriver att denna process bland annat handlar om att testa så att all input frambringar den förväntade utgående datan (output).

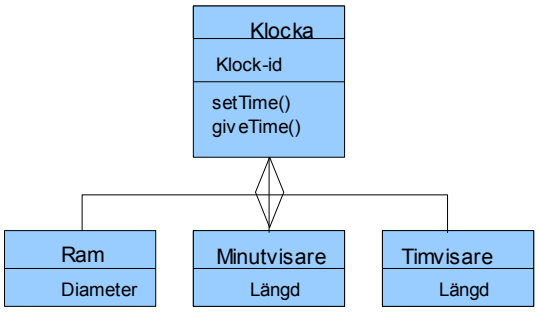
3.3.5.2. Objektorienterad design

Smith (2003) menar att objektorienterad design (OOD) utvecklades för att komma underfund med de problem som inte blev lösta genom strukturerad design. Metoden introducerade ett helt nytt sett att undersöka och utveckla mjukvara på. I OOD så kan man enkelt relatera till verkliga händelser vilket gör det lättare att testa designen innan den utvecklas.
Designens metod kan sägas innehålla fem grundläggande steg:

1. Hitta all objekt och tillhörande attribut. Attribut innebär, enligt Brown (2002), saker som beskriver en entitet och registrerar dess tillstånd. Attribut är alltså saker som är nödvändigt att veta *om* en entitet
2. Hitta objektets olika beteenden. Beteenden är, enligt Brown (2002) något som en entitet *gör*.
3. Associera objekten, inkluderande generalisering, aggregering och relationer. Brown (2002) skriver att entiteter med samma beteenden och samma attribut skapar en egen klass. Klassen används för att beskriva en grupp av objekt som utför samma saker. Ett exempel skulle kunna vara en kund. Alla kunder har då ett namn, en adress och ett kundnummer etc. Generalisering (se även ”Figur 11: Generalisering” nedan) innebär att man kan skapa underklasser för att på så sätt minska redundans i systemet. Exempelvis så kanske kunderna måste delas in i företagskunder respektive privatkunder. Superklassen (den överordnade klassen) kan då kallas kund och innehålla alla attribut och beteenden som privat- och företagskunder har gemensamt. Aggregering (se även ”Figur 12: Aggregering” nedan) innebär att det finns en relation mellan två klasser där instanserna är komponenter, medlemmar eller innehåll av en helhet. En Relation innebär, vilket har antytts tidigare, att innehållande delar har något gemensamt som kan koppla ihop dem.



***Figur 11**: Generalisering (Brown 2002)*



***Figur 12**: Aggregering (Brown 2002)*

4. Utveckla en dynamisk modell av systemet. Sommerville (2004) menar att dynamiska modeller beskriver den aktiva systemstrukturen och visar interaktioner mellan systemobjekten (inte objektklasserna). Interaktionerna, som bör dokumenteras, inkluderar objektens förfrågningar och dess tillstånd i systemet.

3.3.5.3. Designverktyg

Smith (2003) menar att alla programmerare har sin egen stil gällande lösningar och uppbyggnad av kod. Detta blir ett centralt problem när programmeringsutvecklarna för ett projekt blir alt fler och projekten blir större och mer komplexa. Problemen brukar uppstå i underhållsfasen då mjukvaran ska uppdateras och den ursprungliga programmeraren inte längre finns tillgänglig i projektet. Detta problem leder ofta till att det är enklare att skapa ny mjukvara än att underhålla den existerande versionen. En rekommendation är därför att vid implementering och utformning av design använda programmeringsmetoder, för att få en konsekvent och lättförståelig plattform. Dessa metoder finns tillgängliga i designverktyg och kan bland annat stödja framställningen av struktur och gränssnitt samt avgränsningar mellan olika objekt.

3.3.5.4. Design och Open Source

Zwartjes et. al (2005) skriver att designen är en ofrånkomlig process som ingår i implementeringsprocessen. Inom Open Source är det inte speciellt vanligt att utveckla designen utifrån en sammanställd dokumentation, men för det mesta finns det någon form av design i mjukvaran även om den inte är specificerad. En rik design är alltså inget vanligt förekommande, koden och implementeringen tenderar att vara den del som beskriver designen i Open Source mjukvara. Bristen på dokumentation inom Open Source får till följd att koden kan bli svår att återanvända eftersom funktionaliteten är ”gömd” (användaren kan inte förstå vad den är till för eftersom det inte finns någon beskrivning). API-dokumentationssystem är ett bra sätt att beskriva de olika processer som koden består av, men även dessa system utnyttjas sparsamt inom Open Source. Open Source handlar väldigt ofta om implementeringsfasen. Möjligheten att skriva kod är den mest väsentliga drivkraften för i stort sätt all Open Source mjukvaruutveckling.

3.3.5. Ärendehantering

Altcom (2005), som är ett företag som bland annat är verksamma inom systemutveckling och drift, beskriver ärenden som en bristande egenskap eller ett fel i en utvecklad programvara, ett fungerande system eller en samarbetande grupp människor vilket även kan kopplas till hur stor riskmedvetenhet det finns i projektet. Noor (2001) menar att risk- och ärendehantering är en kontinuerlig process som upprepas under hela projektets livstid. Under tiden som projektet utvecklas, så redigeras och anpassas riskerna och ärendena till den nya miljön. Riskerna och ärendena bör prioriteras högt så att de observeras i ett tidigt stadium, det är också viktigt att granska om besluten som avläggs är ändamålsenliga. Processens syfte innebär alltså att hantera riskerna och ärendena fortlöpande genom hela projektet. Enligt Fitzgerald (2005) så har ärenden och risker lite olika innebörd. Ärenden kan beskrivas som motsägelser, olägenheter eller frågor som påverkar vissa aspekter i ett projekt. Vidare anknyts ärendena ofta till två olika saker, nämligen ”rutin-” och ”politiska” ärenden. De flesta ärendehanteringssystem som finns idag fokuserar på rutinärenden vilket för det mesta resulterar i att affärsprocessen får en viss informationsbrist eller brist på tydlighet eller formulering. Politiska ärenden innebär främst personlighet och makt, vilket många gånger resulterar i oenighet om hur eller om någonting ska utföras och av vem. Ärendena kan också innebära en eller flera medlemmars förmåga (eller brist på förmåga) i ett stort projektlag vilket ofta skapar förrädiska känslor, samt att gruppen ofta förlitar sig för mycket på projektledaren.

Dokumentationens språkrör är enligt Fitzgerald (2005) den tekniska ärendeavhandlingen (technical

issue memorandum). En sådan avhandling är en namngiven belägen samling av projektets metadata (sökning via beteckning på ärende), beskrivande text, traditionell text, lösningsförslag och även vilka relationer som finns till andra ärenden. De egenskaper som avhandlingen frambringar om projektet kan analyseras för att bedöma de olika problemens effekt under utvecklingsarbetet och för att värdera projektets kvalitet, samt att det kan användas som ett underlag för att lösa problemen.

3.3.5.1. Ärendehanteringsverktyg

El-Najjar (2006) skriver att ett ärendehanteringsverktyg är en typ av informationssystem med förmågan att behandla både intern och extern information. Informationen kan vara till hjälp för verksamheten i besluts- och handlingsärenden. Dess tillämpningsområde består i att automatisera kartlagda processer, rutiner och arbetsflöden. Vektyget brukar även innehålla funktioner som underlättar beskrivning och övervakning av ärenden. Callahan et. al (1998) menar att en databas med ärenderapporter som utvecklats och underhållits under utvecklingen kan vara till stor hjälp för att se vilken information som ligger bakom ett speciellt beslut eller varför ett specifikt fel uppstod. En sådan lagringsplats skapar en dialog mellan olika grupper vilket minskar oförenlighet i projektet. El-Najjar (2006) skriver vidare att verktyget kan underlätta processhantering genom förenkling och effektivisering, vilket är möjligt då det kan ge en grundläggande överblick och kontroll över projektets mål.

3.3.5.2. Ärendehantering och Open Source

Fogel (2005) menar att det finns ett annorlunda ansvar inom Open Source eftersom det är både utvecklarnas och observatörernas (slutanvändarens) uppgift att få processen att fungera. Detta gäller även kommersiella produkter men är desto viktigare i Open Source, eftersom kommersiella produkter ofta kan testas på ett mer övergripande sett. Ofta är en felrapporteringsdatabas ett gott tecken på att projektet är tillförlitligt och seriöst. Vidare så brukar även de projekt med många felrapporter accepteras i större grad än de som har få felrapporter. Att mjukvaran har många inrapporterade fel kan tyckas vara negativt, men då ska man även komma ihåg att antal upptäckta fel kan kopplas till tre olika saker: Det totala antalet fel som finns/rapporterats i mjukvaran, antal användare som nyttjar mjukvaran, och hur enkelt det är för användarna att registrera nya fel. All mjukvara som är relativt stor har ett antal fel som som förr eller senare blir upptäckta. Hur väl de prioriteras är dock väldigt varierande. Ett projekt med en stor och väl underhållen felrapporteringsdatabas (vilket betyder att felrapporteringarna behandlas omgående, och att liknande fel kopplas samman osv.) ger därför ett bättre intryck än ett projekt utan databas, eller en nästan tom databas. Naturligtvis kommer det inte finnas speciellt många felrapporteringar i början av projektet, men om observatören enkelt kan se projektets begynnelsedag och även kan se att förändringarna i felrapporteringsdatabasen är färska så har de flesta överseende med detta och hakar inte upp sig på att databasen är så begränsad.

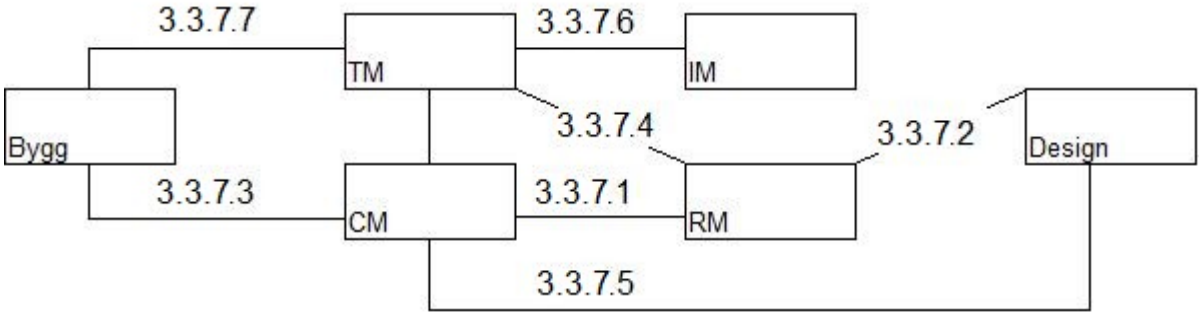
3.3.6. Capability Maturity Model Integration CMMI

CMMI Product Team (2002) skriver att en process är en aktivitet vars mål är att förbättra eller förenkla utvecklingsarbete. Syftet med CMMI är att stödja utvecklingsorganisationers förbättring av processer genom att ge en bättre möjlighet att hantera utveckling och underhåll av produkter eller tjänster. Modellen strukturerar ansatserna på ett sätt som hjälper organisationen att bedöma sin verksamma mognad och områdesmöjligheten för processerna, prioritera olika förbättringsmöjligheter och implementera dessa förbättringar. Hägglund (okänt årtal) skriver att organisationen kan få en bättre översikt gällande processernas förutsägelse, effektivitet och kontroll i takt med att verksamheten mognar. Ökad förutsägelse innebär att organisationen lärt sig att uppskatta tid- och kostnadsaspekter på ett mer tillfredsställande sätt, vilket ger dem bättre möjligheter att planera projektet. Vidare åsyftar ökad kontroll att organisationen kan garantera det

faktiska utfallet för ett projekt på ett mer tillförlitligt sätt, vilket kan vara till hjälp för att i ett tidigt stadium kunna avgöra om projektet är värt att satsa på eller inte. Ökad effektivitet åsyftar att organisationen får förbättrade resultat gällande utnyttjad tid och kostnad som erlagts för att ta fram ett visst system. Modellen kan vara till stor hjälp för hela ALM-utvecklingsprocessen.

3.3.7. Hur komponenterna möts

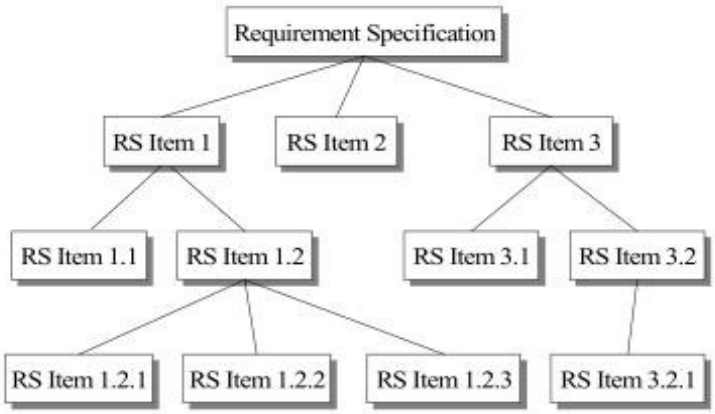
Detta avsnitt kommer behandla hur de olika komponenterna fungerar ihop. Alla verktygens funktion är i sig bidragande till utvecklingen, men det är även viktigt att förstå varför det är intressant att koppla ihop komponenterna till ett verktyg.



***Figur 13**: Hur komponenterna möts*

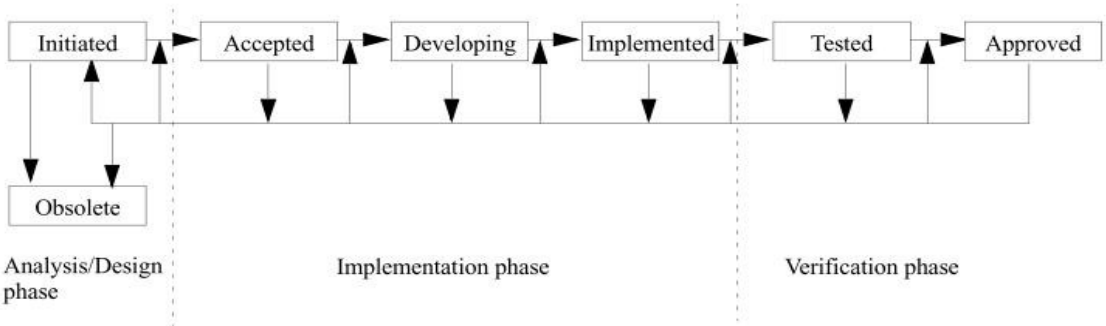
3.3.7.1. Hur konfigurationshantering kan användas för att utveckla krav (CM - RM)

Crnkovic et. al (1999) är en av många som bekräftar det faktum att krav förändras under hela utvecklingslivscykeln. När de definieras är de ofta bristfälliga och ofullständiga. Det är till stor hjälp att göra en förtydligande specifikation av alla krav som bör fortgå under hela utvecklingsprocessen för att på så sätt anpassa kraven efter den nya miljön och efterhand som kunskapen om systemet ökar. Kravspecifikationen ger en beskrivning av varje specifikt krav. Kraven beskrivs förslagsvis i en trädstruktur där den högsta nivån i strukturen innehåller de huvudsakliga kraven som även bearbetas under de lägre nivåerna i strukturen. Strukturens beståndsdelar (items) beskrivs ofta i ett enkelt och neutralt språk men kan även förklaras med hjälp av form, design, beskrivningsspråk, grafer eller formella specifikationer. Då de olika beståndsdelarna har lite olika karaktär så kan de delas upp i en versionskontroll. På så sätt blir varje kravbeståndsdel hanterat som ett konfigurationshanteringsobjekt. Detta gör det enkelt att skapa nya objekt eller att redigera eller ta bort gamla objekt. Varje förändring i ett objekt beskrivs i en ny version. Det är konfigurationshanteringsverktygets (CM-verktygets) uppgift att skilja på de olika versionerna av objekten, vilket enkelt kan utföras på följande sett: Varje objekt får en identitet, titel och ibland även andra kännetecken som kan relateras till varje version, såsom versionsidentitet, upphovsman, utgivningsdag, och ett antal andra attribut beroende på vilket CM-verktyg man använder. Nedan visas en figur över hur ett objekt kan få en identitet genom versionsidentitet. Det finns även andra sett att identifiera ett objekt på, men versionsidentifieing syftar alltså till att varje version får ett unikt namn som även återspeglar vilken version som förändringen utgått ifrån:



***Figur 14**: Versionsidentitet (Crnkovic et. al 1999)*

CM-verktyget kan vidare hjälpa till att skapa en hierarkisk struktur som utgår ifrån logiska samband mellan de olika objekten. Denna struktur är viktig för att se vilken uppsättning krav som tillsammans skapar utgångspunkten av hur systemet ska utvecklas. De olika versionerna av varje kravbeståndsdel hamnar i ett visst tillstånd baserat på i vilket stadium kravet befinner sig i. Ett krav som undersökts och som inte har några brister blir godkänt och kan tas i bruk. Den senaste godkända versionen av hur ett krav fungerar kan kopplas ihop med de övriga giltiga kravversionerna och tillsammans skapa en uppsättning krav som bildar en utgångspunkt (baslinje) av hur systemet ska utvecklas. Syftet är alltså att det ska finnas möjlighet att återgå till förändringar som skett under utvecklings lopp. Ett förslag på en förändring behöver inte alltid vara positiv, då det kan vara bra att kunna återgå till en version som möjligen känns mer tillfredsställande.
Tillstånden som kraven kan tilldelas används för att utvecklare inom samma projekt ska få vetskap om i vilken process kravet befinner sig i. Finns det exempelvis något krav som någon av utvecklarna håller på att utveckla så finns det möjlighet att ge kravet tillståndet ”under utveckling” och det brukar även finnas möjlighet att låsa kravet under den period som utvecklingen äger rum. I figuren som följer presenteras de tillstånd som brukar användas:



Figur 15: Kravtillstånd (Crnkovic et. al 1999)

Verktyget har även ett antal fler huvuddrag för att hantera kravförändringsprocessen. En CM-rapport ger möjlighet att undersöka:

- Vilka krav, eller vilka delar av dem, som har förändrats sedan den senaste baslinjen
- Vem som gjorde förändringen
- När förändringen ägde rum

3.3.7.2. Krav till design (RM - design)

Reed (2002) skriver att det är viktigt att vara medveten om vilka risker de uppställda kraven innebär. Dessa krav som ofta blir utfallet av att ledningen fattar riskfyllda beslut, exempelvis om projektarkitekturens riktlinjer, inte kan testas förrän kodningsarbetet börjar. Det är viktigt att vara medveten om detta eftersom man då kan börja undersöka hur riskerna kan förmildras. Man kan dock inte vara säker på att alla funktionella krav är identifierade, men oftast har man tillräckligt mycket kunskap om projektet för att kunna peka ut de krav som innebär störst risk för själva utformningen.

Reed (2002) skriver vidare att det är även viktigt att ta hänsyn till alla användarfall (use cases), då många av dem beskriver viktiga funktioner. Många gör misstaget att implementera de enklaste kraven först, som inte har någon större inverkan på riskbedömning av utformningsarbetet, vilket innebär att bedömningen inleds senare än nödvändigt. Det är också viktigt att koppla ihop kravens och designens informationsflöden, vilket exempelvis kan göras genom en användarfallsrealisering. I en användarfallsrealisering beskrivs ett användfall som ett designobjekt. Det vanliga användarfallet beskriver endast *vad* användaren eller aktören kan göra i systemet. En realisering innebär även en kungörelse över *hur* användarfallet ska implementeras. Ibland kan det även vara så att realiseringen innebär flera beskrivningar av samma användarfall, vilket beror på att det krävs flera tekniska lösningar för att implementera fallet. Om man tar en varubeställning via Internet som exempel, så kanske man önskar att användaren både ska kunna göra en vanlig beställning via en hemsida på Internet, men även genom ett trådlöst gränssnitt för mobiltelefoni och liknande utrustning. I detta fall blir det alltså nödvändigt att göra användarfallsrealiseringar för både den vanliga beställningen via hemsidan och även för den trådlösa implementeringen. Varför det i detta fallet krävs två realiseringar beror helt enkelt på att lösningarna skiljer sig åt så mycket, rent tekniskt, att det krävs en förklaring för båda fallen. Användarfallsrealisering stödjer, precis som det objektorienterade synsättet, en realistisk (real-world) designsyn av de gällande krav som utformats under inledningen. Att åskådliggöra dessa användarfall ger en bra bild av projektets största risker och ökar möjligheterna till framtida lyckade iterationer.

Reed (2002) menar att det tyvärr finn flera exempel på systemdokumentation där innehållande data definieras otydligt vilket leder till att stora delar av övergången från krav- till designprocessen är svårförståelig. Kraven beskrivs då ofta fåordigt vilket gör att de är svåra att peka ut i designdokumentet. Gör man inte en redogörelse för varje ordinarie krav följt av en översättning till designdelen, så döljs ofta användarens huvudsakliga syfte i systemet. Fastställ alltså inte sådana krav som inte kan översättas på ett tydligt sätt. Varje krav ska kunna urskiljas både under tidigare och senare delar av projektets livscykel. Översättningen från krav till design bör vara tydlig och inte ge upphov till några oklarheter. Det bör vara en naturlig process som är enkel att förklara och förstå sig på för samtliga berörda parter inom projektet.

I figuren nedan sammanfattas en uppsättning faktorer som brukar innebära hög risk för en organisation vid en övergång från krav till design:

Coverage Area	Risk Factors
1. New technology or frameworks not presently employed in other projectswithin the organisation	Identifies areas in which the organization is not yet adept using a new technology.
2. New or revised business processes that the customer is introducing via new technology	Exposes expectations the customer may have about workflows that have not been tested on a new technology. For example, if a bransch bank is switching its account data entry and retrieval processes to a thin client, Web-based application, then that application might not be nearly as flexible as an earlier, clientcentric solution when it comes to workflow and user interface possibilities.
3. Time based processing	There are very few robust, off-the-shelf products that facilitate time-based event processing. Many applications require either a customized solution or a combinationof pre-purchased components and custom code
4. Batch processing	Don't believe the myth that batch-oriented tasks have disappeared with newer generation technology choices. This is just plain wrong; batch processing can be a delicate part of the architecture. Sharing business logic between batch and online can be a tricky proposition.
5. Multi-panel interactions requiring state management throuchout the process (workflow mangement)	This is tarheted primary at Web-based solutions, given the stateledd nature of the Web. The way in which software vendors manage state when dealing with multi-page interactions ranges in complexity and affects availability.
6. Security, logging, and archiving demands	Given most customers' desire for single sign-on (SSO) and integration of security and archiving capability with pre-purchased solutions, this area alone can consume tremedous amounts of effort to integrate into the overall solution.
7. Presistence management	If the solution will be based on object-oriented techniques, then care must be given to the impedance mismatch when mapping objects to relational stores.
8. Quality of service	Performance is always a consideration. Although actual volume testing may not be feasible in the early stages of Elaboration, simulation tool can be applied to provide meaningful approximation of potential throughput.

***Figur 16**: Högrisk-faktorer (Reed 2002)*

3.3.7.3. Konfigurationshantering inom testning (CM - TM)

Ett grundläggande krav för mjukvarutestning innebär, enligt Crowther (2007), att kunna ge ett garanterat säkerhetsbaserat resultat. Vid planering av ett test bör man undersöka huruvida ett fel verkligen är ett fel samt hur det kan hanteras. För att tolka ett specifikt fel som någon har pekat ut så krävs många gånger samma utgångspunkter som var förutsättningarna när felet uppstod. Detta innebär att testet bör utföras med samma mjukvara, samma villkor och samma data. Det är med andra ord många gånger svårt att framställa samma fel om man har olika villkor. Detta blir ett problem när testare upptäcker ett fel som inte utvecklarna har möjlighet att återskapa, och ännu värre kan det bli om en slutanvändare rapporterar in ett ärende som varken testaren eller utvecklaren kan återskapa. Att mjukvara fungerar i ett system men inte i ett annat beror på att systemen inte har jämförts. Även om en grupp användare får sina nya mjukvaruversioner från samma plats, använder

samma testdata och versionskontrollsverktyg, så är det fortfarande en viktig detalj som saknas, nämligen den lokala testmiljön och hur den har utformats.

När man ska skapa en testmiljö är det viktigt att beskriva allt utvecklingsarbete som görs i koden, så att det är enkelt att återgå till hur det såg ut innan utvecklingsarbetet började. Vad som också kan vara till stor hjälp för eftervärlden är att specificera vilken hårdvara och mjukvara som använts vid testningen. Genom denna specifikation har man tagit hjälp av konfigurationshantering vid genomförandet av testet, och har därmed skapat en ”test configuration management approach”. Metoden är användbar eftersom både hårdvara och mjukvara kontrolleras, vilket ger en bra bild av produkternas kvalitet. Genom metoden finns det möjlighet att undersöka om förutsättningarna inom hårdvara och mjukvara verkligen var samma för de olika utvecklarna. Om testmiljöerna inte skulle vara likadana så är det enkelt att hitta källan till varför man får olika resultat. Det är på så vis enklare ett precisera upp de exakta kraven som en ny utgåva av ett system behöver, vilket förhoppningsvis ska minska antalet felrapporter från missnöjda slutanvändare.

3.3.7.4. *Testning baserat på krav (TM - RM)*

Aharonovitz (2006) menar att otillfredställande systemutveckling kan bero på två saker, nämligen bristande kravspecifikation och otillräckliga systemtester. Bach (1999) beskriver krav som en uppsättningen idéer som tillsammans ska beskriva en produkts kvalitet. Vidare beskriver han test som en utvecklingsprocess av produktens kvalitet.

Det är oftast viktigt att uppnå vissa krav, och testarens uppgift ligger i att utvärdera produkten utifrån kraven, vilket betyder att testaren behöver bli upplyst om dessa krav. En viss problematik uppstår när de uppställda kraven inte täcker all funktionalitet som systemet ska leverera. Detta är ett tecken på att kravspecifikationen är ofullständig och otydlig. Testningen bör alltså både fungera som ett utvärderingsverktyg, samt som en process som ska analysera kravens innebörd och delaktighet. Aharonovitz (2006) menar att det helt enkelt beror på att kraven är felaktiga redan från början.

Krav för komplexa mjukvaruapplikationer specificeras ofta i två huvudsakliga frågor, som genomströmmar hela projektets livscykel. Dessa frågor behandlar ”vad *behöver* vi bygga?” och ”vad *kan* vi bygga?” Hur väl man besvarar dessa frågor är ofta avgörande för den slutgiltiga kvaliteten för den konstruerade applikationen.

Pinto (2006) menar att kravbaserad testning utgör två väsentliga faser: En recension samt ”cause-effect graphing”. Recensionen utformas för att identifiera tvetydigheter i kravspecifikationen och följaktligen förbättra kravens kvalitet. ”Cause-effect graphing” är en designmodell över alla testfallen som erhåller det minsta antalet testfall som kan täcka samtliga funktionella krav.

3.3.7.5. Design och implementering av versionshanting (Design - CM)

Ramakrishnan et. al (1996) menar att stora system ofta är komplexa och innehåller ett antal olika komponenter (exempelvis så innehåller ett operativsystem olika program). Varje komponent kan i sin tur innehålla komponenter på en lägre nivå (exempelvis så kan ett program innehålla olika funktioner). Komponenterna vidareutvecklas under hela designprocessen. Med tanke på denna utvecklingprocess så är det fördelaktigt att bevara de olika versionerna av varje komponent, så att varje förändring, eller varje nytt steg i processen, får en ny version. Då design ofta har fler än en ”korrekt” lösning på hur en komponent kan utformas, så kan det vara bra att ibland ha mer än en lösning på hur designen av komponenten ska se ut. I det fallet bör det finnas en versionshantering som beskriver att det är samma komponent som är utformad på olika sätt. De komponenter som ingår i varje komponent, på en lägre nivå, ska också få varsin version. Det ska vara enkelt att återgå

till en tidigare version vilket kräver en dokumentation som beskriver följande:

- Varför behovet fanns att skapa en ny version
- Dess kännetecken
- Skillnaden från tidigare versioner
- Vilken komponentversion som användes på en lägre nivå

Denna kunskap är nödvändig i versionshanteringen för att ge ett fullständigt stöd till designprocessen och för att det ska vara någon mening med att ha flera versioner av en komponent.

3.3.7.6. Ärendehantering vid testning (IM - TM)

Noor (2001) menar att hantering av ärenden och brister är en problematisk del av testprocessen. Det mest grundläggande sättet att rapportera ärenden samt eventuella lösningar är genom en bearbetningsplan av bristerna som även beskriver aktuell status på processen. Detta möjliggör att testaren kan definiera hur bristen har blivit behandlad under dess utredningstid och även vem som stod för förändringen. Slutligen kan bristen rapporteras som avklarad då man även tar hänsyn till de omständigheter som krävs för att bristen officiellt ska kunna räknas som avslutad. Även om testaren har dokumenterat väl så krävs ofta ett visst underhåll för att uppnå en fullständig historik av hur ärendet behandlades. Det kan vara bra att lägga en del tid på att dokumentera bristens historik i en logg eftersom det underlättar analysen och kan ge upphov till bättre kvalitet på applikationen.

3.3.7.7. Bygga och testa (Bygg - TM)

Pavlo et. al (2006) menar att en process där det som utvecklas eller byggs blir testat med jämna mellanrum kan ge många fördelar. Utvecklarna har genom processen möjlighet att identifiera och analysera applikationens problem redan vid införandet, istället för att mjukvarufelen dyker upp när det dagas för en ny produktionsutgåva. Genom att testa ofta så minskas avståndet mellan den senaste lyckade utformningen av en applikation och senaste misslyckade versionen, vilket gör det enklare att urskilja vilka modifieringar i källkoden som är orsaken till att applikationens kompilering eller testning blir felaktig.

4. Resultat

I detta avsnitt utvärderas alla komponent. Verktyg kopplas till vart och ett av de områden som den praktiska delen innefattar, nämligen, kravhantering, ärendehantering, design/bygg och testhantering. I avsnittet kommer även en utvärderingsmodell presenteras där kommersiella verktyg jämförs med Open Source utifrån ett antal kriterier. En enkätundersökning som utformades för att få fram alla resultat i utvärderingsmodellen kommer också presenteras.

4.1. Utvärdering

Detta avsnitt ger en beskrivning av de praktiska delarna vid implementering av ett eller flera ALM-verktyg, med avseende på allt mellan införskaffandet av verktyget till installation, funktionalitet, fördelar och nackdelar samt integration med andra verktyg i processen och även en del annat som har med utvärdering av verktygen att göra.

4.1.1. Open Source verktygens funktion

Här beskrivs verktygens funktion i en helhetslösning. Det ska ge en sammanfattning av vad de olika delarna är till för, och hur de kan användas tillsammans.

Eclipse (Bygg)

Eclipse, bredvid webbrowsern, är klientplattformen som levererar ALM-tjänsten till användarna. ALM verktyg bygger alla på klient-server lösningar. Förut hade alla verktyg sina egna klienter, t.ex. CaliberRM server och CaliberRM klient, StarTeam server och StarTeam klient. Eclipse är en plattform som gör det möjligt att leverera alla tjänster som utvecklare (kravställare, testare) behöver. I Eclipse kan man utveckla, testa, läsa krav, kolla på tester, följa traces, skapa nya ärenden osv.

Trac/Bugzilla (Ärendehantering)

Programmen fungerar som en databas för alla ärenden. Databasen kan nås via ett webbgränssnitt. Om det upptäcks fel vid testningen kan man rapportera felet via ärendehanteringen, och på så sätt få en bra historik på felaktiga eller buggiga funktioner. Databasen kan hjälpa till att förbättra mjukvaran i senare versioner och kan även vara till stöd för slutanvändare då de kan se hur felet kan repareras.

Testlink (Testhantering)

Testlink är en databas för alla tester. Databasen kan nås via ett webbgränssnitt. I databasen kan man mata in användarfall och testfall som är intressanta att testa. Det finns även möjlighet att utföra en kravbaserad testning. När testerna sedan utförs kan man rapportera huruvida testet gick igenom eller inte. Vidare är då tanken att Trac/Bugzilla eller något annat ärendehanteringssystem ska beskriva vad som gick fel i testerna. Har man vidare redigerat testet så att det går igenom, eller om man väljer att ta bort funktionen, så går det att ändra tillstånd på testet till ”Passed”, ”Blocked” eller ”Failed”.

Subversion (Konfigurationshantering)

Subversion är ett versionhanteringssystem som används när flera personer vill arbeta med en fil samtidigt. En användare har möjlighet att låsa filen så att ingen annan har möjlighet att ändra i den på samma gång. När användaren är färdig kan han vidare låsa upp filen igen åt de övriga användarna som då har möjlighet att se vilka förändringar som gjorts i den nya versionen. Varje förändring skapar nämligen en ny version av filen så att man har möjlighet att se alla förändringar

som har skett i en viss fil, eller en mapp med en uppsättning filer, som ingår i ett projekt. Det finns även möjlighet för en administratör att godkänna förändringen och skapa en baslinje av alla filer som ska gälla i kravspecifikationen.

OSRMT (Kravhantering)

Verktyget har använts för att specificera upp krav på systemet. Krav beskriver den funktionalitet som systemet ska klara av. All funktionalitet kan vidare kopplas ihop med varsitt krav, för att på så sätt få en överblick till varför funktionen finns till. Detta är ett sätt att ta bort onödig funktionalitet och även för att inte missa någonting som systemet ska klara av att utföra.

4.1.2. Utvärderingsmodellen

I utvärderingsmodellen finns några begrepp avseende verktygens möjligheter och begränsningar. Detta avsnitt har för avsikt att beskriva hur termerna har använts. Till att börja med kommer en översikt av icke-funktionella krav beskrivas, där jag ger en bild av hur komplicerade de olika verktygen upplevs vara i avseende på installation, pris, operativsystem, integrationsmöjligheter, riskmedvetenhet och funktionalitet. Alla dessa krav bedöms i en skala mellan 1-5, där 1 är det sämsta betyget man kan få och 5 är det bästa. 1-5 skalan utgår ifrån uppsatta kriterier för vad de olika poängen ska innebära. Betyget 5 innebär att kravet uppnås på bästa möjliga sätt (exempelvis om licenskostnad är lika med noll eller om alla testade funktioner är uppnådda). De lägre betygen innebär att kravet inte riktigt har blivit uppnått, och ju lägre poäng verktygen har fått, desto sämre upplevs kravet uppfyllas. Kraven kommer även bedömas utifrån hur viktiga de anses vara i jämförelse med de andra. Respektive krav kommer alltså tilldelas ett procenttal som sedan adderas med 1-5 skalan. Alla källor till modellen finns att tillgå i avsnitt ”7.1 Källor till utvärderingsmodellen”.

	Översikt	Installation	Pris	Operativsystem	Integrationsmöjligheter	Riskmedvetenhet	Funktionalitet	TOTAL SUMMA
	Vikt:	5,00%	30,00%	7,00%	25,00%	13,00%	20,00%	
Program								
RM								
Borland CaliberRM		4	4	1	5	3	4	**3,91**
IBM Rational RequisitePro		3	2	3	5	4	5	**3,73**
Telelogic DOORS		3	3	1	5	2	5	**3,63**
OSRMT		5	5	5	1	2	4	**3,41**
CM								
Borland StarTeam		4	3	1	5	4	5	**3,94**
IBM Rational ClearCase		3	3	3	5	3	5	**3,9**
Telelogic SYNERGY		3	2	1	5	4	5	**3,59**
Subversion		2	5	5	4	3	5	**4,34**
IM								
Trac		1	5	5	4	2	4	**3,96**
Bugzilla		4	5	5	4	3	4	**4,24**
Borland SilkCentral IssueManager		3	3	1	5	4	4	**3,69**
TM								
Testlink		3	5	3	2	3	5	**3,75**
Borland SilkCentral TestManager		3	4	1	5	4	5	**4,19**
Design/Bygg								
Eclipse		5	5	5	4	3	4	**4,29**
Borland Together Arcitect		3	1	1	5	4	4	**3,09**
IBM Rational Software Architect		3	1	3	5	3	4	**3,1**
Telelogic TAU Architect		3	1	1	5	3	3	**2,76**
RM, Design, CM								
Microsoft Visual Studio Team System & Team Foundation Server (TFS)		3	4	5	5	4	3	**4,07**
Summa per uppsättning								
Borland		3,4	3	1	5	3,8	4,4	**3,76**
IBM		3	2	3	5	3,33	4,67	**3,58**
Telelogic		3	2	1	5	3	4,33	**3,33**
Open Source		3,8	5	4,6	3	2,8	4,4	**4,01**
Microsoft		3	4	5	5	4	3	**4,07**

***Figur 17**: Utvärderingsmodellen: översikt*

Vikt

De icke funktionella kraven som beskrivs i utvärderingsmodellens översikt anses ha olika mycket betydelse i förhållande till varandra. De krav som anses vara viktigast har därför tilldelats ett högre procenttal än de som är mindre viktiga. Bedöms modellen både utifrån 1-5 skalan och även dessa vikter går det att få ut vilket av verktygen som passar bäst utifrån en viss bedömning.
Tanken är att vem som helst ska kunna använda modellen och att det vara möjligt att ändra värdena på de olika vikterna och på så sätt få ut ett värde som passar den egna organisationen. Värdena i tabellen är ett exempel för att visa hur uppdelningen kan se ut. Uppdelningen har utgått ifrån att pris och integrationsmöjligheter är de viktigaste faktorerna eftersom det är just dessa två faktorer som det finns problem inom de kommersiella verktygen. Se även i avsnitt ”1.2. Problemområde”.

Installation

Begreppet installation ska ge en överblick hur svårt jag upplevde att installera de olika verktygen. Det skiljer sig ofta hur svår eller enkel installationen upplevs vara, vilket beror på att miljön som verktyget installeras på kan vara av lite olika karaktär från gång till gång. Det kan exempelvis vara ett annat operativsystem eller möjligen processer och andra verktyg som försvårar eller förenklar installationens utförande. Rutin är också en faktor som spelar in; har man installerat ett visst program ett antal gånger så vet man vilka misstag man ska undvika och liknande.

Det är också stor skillnad gällande att installera ett enskilt program utan förbindelser och integrationer med övriga verktyg. Begreppet innefattar även aspekten att verktyget ska kopplas ihop med övriga verktyg som tillsammans ska kunna utföra en end-to-end process, vilket beskrivs i avsnitt ”4.1.4. Ett praktiskt exempel: Adressboken”.

Gällande de kommersiella verktygen, så har jag laddat hem testversioner med en begränsad användningstid och funktionalitet. Själva installationsdelen skiljer sig dock inte så mycket på en demoversion i jämförelse med en licensierad version. Jag har gjort en bedömning i skalan 1-5 där ”1” betyder komplicerad och ”5” betyder enkel.

Pris

Pris ska ge en bild av hur mycket en licens av produkten kan kosta. Priset kan skilja väldigt mycket beroende på hur många klienter man är intresserad av att installera verktyget på och även i vilket projekt man vill använda verktyget i. Ett projekt mot en större leverantör kan exempelvis få priset att stiga betydligt. Försöker man istället begära produkten utan att nämna något speciellt ändamål så får man väldigt många gånger ett diffust svar eller också inget alls.

Skalan för pris avser följande:
5: 0kr
4: 1-5000kr
3: 5001-10000kr
2: 10001-15000kr
1: 15001 -->

Operativsystem

Operativsystem ska vidare ge en bild av vilka system verktyget kan implementeras på. De flesta ger stöd för alla typer av system, men andra har valt att begränsa verktygen till vissa plattformar.

Skalan på operativsystem är följande:
1: Microsoft Windows
3: Microsoft Windows och Unixbaserade system
5: Microsoft Windows, Unixbaserade system och Mac OS X

Integrationsmöjligheter

I utvärderingsmodellen beskrivs integrationsmöjligheterna i en skala mellan 1-5, där 1 syftar till ingen integration, 2 syftar till integration med komponenter inom den egna uppsättningen och 3 syftar till att det går att integrera med alla komponenter inom den egna uppsättningen . 4 syftar till att den även klarar av integration med komponenter utanför den egna uppsättning och 5 betyder att de klarar av att hantera mer än två utomstående komponenter. Alla undersökta företag verkar ge bra möjlighet till integration till mjukvara runt omkring, men då många av dessa inte kan relateras till ämnet så kommer endast relevanta relationer beskrivas. Gällande Open Source-produkterna så finns inte de övergripande produktsammansättningarna på samma sätt. Skalan får därför en annorlunda betydelse där. En 2:a beskriver möjlighet att integrera med ett annat Open Source verktyg, medan en 4:a betyder att verktyget kan integrera med flera andra verktyg. Betyget 3 och 5 kommer inte att nyttjas i detta sammanhang.

Riskmedvetenhet

Här beskrivs riskmedvetenheten i att implementera och använda verktygen i en skala mellan 1-5. Inom projekt, beskriver Borland (2005), att en för låg riskmedvetenhet bland annat kan innebära

bristande planering, budget och kvalitet. Angående implementeringen är det viktigt att ta hänsyn till den existerande miljön genom att undersöka dess brister och utvärdera hur eventuella säkerhetsluckor kan täppas igen. Det är även viktigt att ha en bra förståelse av kundens behov, så att de får ut vad de förväntar sig av verktygen.

Avsnittet utgår ifrån informationen från de undersökta företagen, och hur jag bedömer deras riskmedvetenhet. Gällande den information som samladess in från de undersökta företaget, så var det främst hemsidor och sammanfattande dokument om de olika verktygen som användes. För att även få fram en uppfattning från slutanvändaren så utformades en enkät där ett urval IT-chefer inom området fick besvara ett antal frågor. Enkätundersökningen handlade främst om att hitta personer som arbetat i verktygen och som har en viss erfarenhet av dem. Till stöd av denna erfarenhet var hoppet att de skulle kunna ge mig verktygens starka sidor och brister gällande just risk. Utifrån de utformade frågorna, som presenteras i avsnittet ”Resultat och diskussion”, har jag sedan gjort en bedömning på hur hög riskmedvetenhet de olika verktygen anses ha. Mer om hur undersökningen gick till behandlas i avsnitt ”3.2.2: Frågor till utvärderingsmodellen”.

Då ett antal verktyg blev obehandlade i enkätundersökningen så har jag i dessa fall endast utgått ifrån den information som hämtades in från företagen själva. De berörda verktyg har markerats med en asterisk.

De verktyg som blev bedömda av slutanvändare i undersökningen var dock väldigt få, vilket gör att det svårt att dra några generella slutsatser.

Funktionalitet

All termer beträffande verktygens funktionalitet beskrivs på ett övergripande sätt här nedan, där även figurer visar vilka funktioner de olika verktygen klarar av. Kan inte verktyget nå upp till precis det som står i förklaringen, så har jag valt att presentera funktionen som obefintlig i detta program. Det kan alltså i vissa fall vara så att verktyget kan uppfylla delar av kravet, eller att de har löst problemet på något annat sätt. Då funktionaliteten utgår ifrån en viss standard, som tagits fram tillsammans med min handledare på företaget, så är min uppfattning att verktygens alternativa lösningar inte behöver beskrivas mer ingående. En övergripande bedömning av hur bra funktionalitet verktyget har kommer presenteras i en skala mellan 1-5 där fem är bäst.

Konfigurationshantering (CM)

	Program	Borland StarTeam	IBM ClearCase	Microsoft Visual Studio	Telelogic SYNERGY	Subversion
Funktioner CM						
Pakethantering		x	x	x	x	x
Förena/merge		x	x	x	x	x
API åtkomst		x	x	x	x	x
Access kontroll		x	x	x	x	x
Configuration items		x	x	x	x	x
Sökfunktion		x	x	x	x	x
Sand box		x	x	x	x	x

***Figur 28**: Funktioner konfigurationshantering (CM)*

Pakethantering

Möjlighet att kunna hantera paket eller uppdateringar, exempelvis komponenter med versionskontroll.

Förena/merge

Det skapas flera kopior av samma fil, vilket ger möjlighet att låta flera stycken ändra i samma fil samtidigt utan att påverka varandra.

API

Syftar till ett gränssnitt som ger en överblick av vilka skript och kommandon som finns tillgängliga

Access control

Syftar till att man ska kunna begränsa en användares åtkomst till mjukvaran och även att kunna dela in dem i olika grupper, såsom testare, utvecklare, gäst osv.

Configuration items

Mjukvaran ska kunna hantera alla artefakter som skapats under mjukvaruutvecklingen.

Sökfunktion

Möjlighet att söka på all typ av data i CM-komponentens databaser, med undantag från sådan data som inte är menad för allmänheten

Sand box

Användaren ska ha möjlighet att kontrollera förändringar i en filuppsättning genom att bestämma hur och när andras förändringsförslag ska accepteras.

Kravhantering (RM)

	Program	Borland CaliberRM	IBM Rational RequisitePro	Microsoft Visual Studio	Telelogic DOORS	OSRMT
Funktioner RM						
Unikt ID			x	x	x	x
Länkar		x	x	x	x	x
Begärd länk		x	x	x	x	x
Multipla länkar		x	x	x	x	x
Länköversikt			x		x	x
Kravklassificering		x	x	x	x	x
Versionshantering		x	x	x	x	x
Exportera data			x	x	x	x
Säkerhetsnivåer		x	x		x	x
Web interface		x	x		x	x
Rapportgenerering		x	x	x	x	x
Ordlista		x	x		x	
Spec av testcase		x	x	x	x	x
Sökfunktion		x	x	x	x	x
Bilder i krav		x	x	x	x	
Analys		x	x		x	

Figur 29: *Funktion kravhantering (RM)*

Unikt ID

Varje krav ska kunna identifieras med ett specifikt nummer eller annan typ av identifiering.

Länkar

Verktyget ska ha användarvänliga och lätthanterliga länkar. Det ska vara enkelt att se kopplingen mellan olika krav och det ska vara enkelt att skapa en länk.

Begärd länk

Komponenten ska upplysa om det har uppstått en förändring som påverkar en länk. Det kan handla om att länkar på en lägre eller högre nivå har förändrats eller att någon förbindelse har förändrats.

Multipla länkar

Komponenten ska utan problem kunna hantera ett stort antal länkar samtidigt, den ska både klara av granskning, skapande och bevakning och även multipla länkar till eller från ett krav.

Länköversikt

Möjlighet att få en översikt av alla förhållanden samt en trädstruktur.

Kravklassificering

Ett krav måste kunna tilldelas attribut angående dess ”status”, exempelvis godkänt, implementerad, testad, den ska även kunna presentera en ”tidsplan för en ny utgåva”, ”prioritet på krav”, ”svårighetsgrad på krav”.

Versionshantering

Verktyget ska kunna ge kraven en ny version vid förändring, då ett krav många gånger kan förändras under utvecklingsfasen. Versionen ska även kunna kopplas till en baslinje där exempelvis alla godkända krav, baserat på den senaste versionen, kopplas samman.

Exportera data

Man ska kunna skicka kraven till en annan typ av filformat, såsom XML, ordbehandlare eller PDF format.

Säkerhetsnivåer

Möjlighet att ställa in säkerhetsnivåer på olika användare/grupper då de ska få skilda rättigheter till vissa kravuppsättningar.

Web interface

Man ska kunna använda verktyget i ett webbgränssnitt. Det ska exempelvis gå att ändra och skriva krav via gränssnittet.

Framställning av rapport

Man ska kunna skapa en egen rapport baserad på önskad information om kraven och även ha möjlighet att skriva ut rapporten som en mall.

Ordlista

Verktyget bör ha stöd för en ordlista där det ska gå att lägga till egna attribut.

Spec av testcase

Verktyget ska kunna specificera testfallen, istället för att behöva använda ett speciellt testhanteringsverktyg.

Analys

Möjlighet till att skapa en analys gällande förändringar i huvudkrav eller krav på en lägre nivå, och även hur förändringen påverkar övriga krav.

Design/bygg

Program	Eclipse	Borland Together Arcitect	IBM Rational Software Architect	Microsoft Visual Studio	Telelogic TAU Architect
Funktioner Design					
Användarfall	x	x	x	x	x
Förena/Merge	x		x	x	x
Spec på ägare	x	x	x	x	
API	x	x	x	x	x
Språkstöd	x			x	
Metodaspekter	x	x	x	x	x
Versionsförändringar	x		x		x
Webbpublisering	x	x	x		x
UML 2.0 stöd	x	x	x		x
Design patterns	x	x	x	x	
Analys av komplexitet	x	x	x	x	x
Kravhantering		x	x		x

***Figur 30**: Funktioner design/bygg*

Användarfall

Verktyget ska på ett effektivt sätt kunna hantera användarfall och sekvensdiagram för att underlätta skapandet av en utvärderingsmodell.

Förena/Merge

Verktyget bör kunna förena olika versioner av designmodeller. Man ska exempelvis kunna förena versionernas klassdiagram, sekvensdiagram och klassdefinitioner.

Specifikation på ägare

Det ska vara enkelt att se vem som är upphovsman till vad i en modell. Upphovsmannen ska även kunna ställa in vad man behöver för rättigheter för att få tillgång till det.

API

En API som beskriver modellen genom att hämta information från den. APIn bör kunna hantera C# eller Java skript.

Språkstöd

Verktyget ska kunna hantera Java, C++ och C#

Modeller

Verktyget ska kunna göra modeller på en lägre nivå genom att utgå ifrån en huvudmodell, och även ge möjlighet för en grupp designers att jobba med den förenklade modellen.

Semantisk kontroll

När en modell ska exekveras så ska verktyget se till så att modellen är giltig rent språkmässigt, och den ska visa vilka element i en modell som fungerar samt vilka som inte fungerar.

Metodaspekter

Verktyget ska kunna informera att en viss operation kan leda till att fel uppstår, och den bör även ge möjlighet att ställa in vilka rättigheter som krävs för att utföra dessa operationer.

Versionsförändringar

Det ska finnas möjlighet att jämföra olika modellversioner, för att se vilka förändringar som uppstått. Detta underlättar när en ny utgåva ska lanseras, eftersom man enkelt kan se vilken funktionalitet som tillkommit.

Webbpublicering

Modellen ska kunna publiceras i ett webbgränssnitt.

UML 2.0 stöd

Stöd för UML 2.0

Design patterns

Verktyget ska kunna lokalisera och återanvända designmönster.

Analys av komplexitet

Verktyget ska kunna ge en analys av designens komplexitet.

Kravhantering

Verktyget ska kunna ge analys av vilka krav de olika användarfallen ger upphov till

Ärendehantering (IM)

	Program	Trac	Bugzilla	Borland SilkCentral Issue Manager
Funktioner IM				
API		x		x
Meddelande		x	x	x
Relation mellan ärenden			x	
Status på ärende		x	x	x
Grafisk översikt			x	x
tilldelningsförutsättning		x	x	x
Rapport		x	x	x
Exportera till ordbehandlare				x
Exportera till kalkylprogram				x
Projektplanering		x	x	x
Hantering av dubbletter		x	x	x
Skriva ut			x	x

***Figur 31**: Funktioner ärendehantering (IM)*

API

Syftar till ett gränssnitt som ger en överblick av vilka skript och kommandon som finns tillgängliga. Är även nödvändigt vid integrering med andra verktyg, såsom Subversion.

Meddelande

Ett meddelande skickas ut till berörda personer vid skapande eller förändring i ärenden. Meddelandet skickas till personens/personernas epost om de är kopplade till ärendet. Det kan även vara önskvärt att bevaka ett visst ärende, och få ett meddelande när en förändring sker.

Relation mellan ärenden

Ärenden bör kunna placeras i en hierarki, med huvudkriterier och underkriterier.

Status på ärende

Ett ärende ska kunna tilldelas status beroende på i vilket stadium den befinner sig i. Statusen kan exempelvis vara ”öppen”, ”stängd”, ”accepterad”, ”felaktig” och liknande.

Grafisk översikt

En grafisk översikt av alla ärenden och hur de är relaterade till varann rent hierarkiskt.

Tilldelningförutsättning

Man ska kunna tilldela ett ärende till en speciell person. En smart lösning är att ställa in så att en viss person blir kopplad till en viss typ av ärende.

Rapport

Man ska kunna skapa en egen rapport baserad på önskad information.

Projektplanering

Det ska finnas stöd för en planering av projekt. Det ska då finnas möjlighet att sätta upp delmål, deadline, en tidsplan och liknande.

Hantering av dubbletter

Om det finns två ärenden som anses syfta till samma bug, så ska detta kunna rapporteras. En administratör kan då välja att stryka ärendet och hänvisa till det existerande lösningsförslaget.

Skriva ut

Man ska kunna infoga buggen i en tabell som sammanfattar dess tillstånd och ger ett bra utskriftsunderlag.

Testhantering (TM)

	Program	Testlink	Borland SilkCentral Testmanager
Funktioner TM			
Exekvering av test		x	x
Testresultat		x	x
Bugresultat		x	?
Användarfallsrelationer		x	?
Testplan		x	x
Kravbaserad testning		x	x
Skriva ut		x	?
Rapport		x	x

***Figur 32**: Funktion testhantering (TM)*

Exekvering av test

Ett test ska kunna ”köras” och man ska kunna rapportera huruvida användarfallen går igenom eller inte.

Testresultat

Man ska kunna få ut ett resultat av de ”körda” testen, för att få en överblick av vilka test som misslyckades och vilka som lyckades.

Bugresultat

En lista där man kan se buggarnas tillstånd och till vilket test de tillhör.

Användarfallsrelationer

Användarfallen ska kunna bli sammankopplade i testgrupper och man ska även kunna skapa undergrupper till testerna.

Testplan

Man ska kunna specificera upp sina tester i en testplan. Testplanen kan då användas för att skilja på olika former av tester och även för att dela in olika typer av funktionalitet.

Kravbaserad testning

Det ska vara möjligt att sätta upp krav på användarfallen. Detta kan vara till användning om ett användarfall bygger på ett annat. Exempelvis att man måste vara inloggad för att kunna logga ut.

Skriva ut

Man ska kunna skriva ut testbeskrivningarna och även testplanen. Man ska kunna infoga datan i en tabell som ger ett bra utskriftsunderlag.

Rapport

Man ska kunna skapa en egen rapport baserad på önskad information. Informationen bör beskriva exekveringsresultat från de olika testen, för att på så sätt stödja kvalitetskontroll och viktiga beslut om hur projektet kan fortskrida.

Summa

”TOTAL SUMMA” och ”Summa per uppsättning” beskriver båda en poängsummering. Skillnaden är att ”TOTAL SUMMA” beskriver ett enskilt verktygs totala poäng utifrån de uppsatta icke funktionella kraven och att ”Summa per uppsättning” ger en bild av vad en hel uppsättning av verktyg från samma leverantör får för poäng. Open Source har i detta fall räknats till en enstaka leverantör just för att kunna jämföra hela uppsättningar mot varandra. Modellen visar då att Subversion har den högsta poängen för ett enskilt verktyg med 4.34 poäng av 5 möjliga. Den bästa uppsättningen verktyg är Microsoft med 4.07 poäng, tätt följt av Open Source som fick 4.01 poäng.

4.1.2.1. Frågor till utvärderingsmodellen

Anledningen till att skicka ut ett frågeformulär till ungefär 600 personer var att få slutanvändarens uppfattning om verktygens riskmedvetenhet i utvärderingsmodellen. Genom att läsa de olika leverantörernas egna verktygspresentationer så fick jag en uppfattning att de lovar väldigt mycket. Dessa uppfattades ofta som väl valda ord för att marknadsföra produkten. Många gånger framgick det inte vad de jämförde sig med, när de exempelvis påstod att deras produkt var användarvänlig och enkel att använda. De beskriver inte heller för vem det skulle vara användarvänligt och enkelt, gäller det även för en person som har väldigt låg datorvana, eller måste man ha en viss kunskap för att uppleva denna användarvänlighet?

I mitt utskick lovade jag att svaren skulle behandlas anonymt. I avsnittet ”Resultat och diskussion”

har jag bifogat det meddelande jag skickade ut till alla, och även själva frågeformuläret samt några utvalda svar utan att ange källan. I nästkommande avsnitt ”4.1.2.2.: Upplevda problem gällande verktygens funktionalitet och riskmedvetenhet” beskrivs hur bedömningen av enkätundersökningssvaren behandlades.

Den e-postlista som användes för att komma i kontakt med personerna som användes i undersökningen kommer inte att bifogas. Det som får sägas om listan är att den innehöll e-postadresser till branschens IT-chefer.

4.1.2.2. Upplevda problem gällande verktygens funktionalitet och riskmedvetenhet

Syftet med detta avsnitt är att ge en bild över hur personerna i enkätundersökningen besvarade de frågor som ställdes till dem. Avsnittet ska också ge en uppfattning hur svaren utnyttjades i den utvärderingsmodell som var anledningen till hela undersökningen. Då det även fanns möjlighet att beskriva sådana verktyg som jag inte har analyserat så kommer ett och annat namn dyka upp som inte kan kopplas till utvärderingsmodellen. Svaren kunde ändå vara till nytta på så sätt att jag kunde se vissa mönster hos en viss typ av verktyg. Sammanfattningen kommer därför även inkludera vilken typ av verktyg (exempelvis kravverktyg eller testhanteringsverktyg) det handlar om, så att det ska bli enklare att förstå dessa samband. I avsnitt ”4.2 Enkätundersökning” finns även en sammanfattning av alla svar. Sammanfattningen innehåller namn på verktyg samt vilka problem och fördelar som slutanvändaren har kopplat till dem.

4.1.2.3. Bedömning av verktygens integrationer och riskmedvetenhet

Borland

Borlands har mycket bra integrationsmöjligheter mellan sina egna produkter. CaliberRM som är deras kravhanteringsverktyg har exempelvis potential att integrera med konfigurationshanteringsverktyget StarTeam och även byggverktyget Together Arcitect. Genom Microsoft Source Code Control Interface (MSSCCI) kan Borlands samtliga produkter integrera med Microsoft Visual Studio och det finns som sagt även möjlighet att exportera data till Word och Excel. StarTeam ger även möjlighet att integrera med Eclipse. Det är inte heller något krav att investera i alla produkter i uppsättningen eftersom alla verktygen har möjlighet att integrera med varandra.

Angående Borlands syn på risk (Borland.com.tr) så får man intrycket av att de har lagt ner mycket arbete på att få det så pålitligt och förutsägbart som möjligt. De har bakat in olika tester för att få reda på slutanvändarens förkunskaper och även tydliga kopplingar mellan olika roller och uppgifter för att hitta fel så tidigt som möjligt. Mycket talar för att de tänker både på användaren och på det system som deras verktyg ska implementeras på.

IBM

Även IBMs produktuppsättning ger möjlighet till integration mellan sina egna verktyg. Det finns inga krav gällande vilka verktyg man har eller inte har då det är enkelt att få de kopplingar som önskas av de olika verktygen. Rational RequisitePro har även en integration med Microsoft Word samt Microsoft Project. Även IBMs konfigurationshanteringsverktyg kan kopplas ihop med Eclipse och det finns även här kopplingar till MSSCCI.

ComponentSource som bland annat ägnar sig åt att sälja och recensera mjukvara har i en artikel som främst behandlar IBMs kravhanteringsverktyg RequisitePro, men som även är relativt generell vilket betyder att artikeln på många sätt även avser hela uppsättningen. De attribut som IBM vill

lyfta fram och som kan kopplas till risk och användarvänlighet är förbättring av kommunikationen gällande ett projekts mål, minska risken i ett projekt samt öka kvaliteten på applikationer innan de offentliggörs.

Telelogic

Telelogics produkter har även de en bra integration i sin egen produktuppsättning. De gör inte heller något undantag från de andra stora företagen inom området, utan stödjer även dem Visual Studio samt Eclipse.

Spectrum-systems vars främsta uppgift är att hjälpa företag med bland annat utveckling, ledning av nätverk och annan informationsegendom menar att Telelogics verktyg är kraftfulla, har en positiv verkan gällande kostnader samt innebär en låg risk. Man kan även få en tydlig bild av projektets nuvarande tillstånd, samt risker och trender vid implementeringen i ett existerande system.

Microsoft

Visual Studio Team System och Team Foundation Server ger möjlighet till integration med samtliga nämnda företags verktyg. Det har heller inga begränsningar till sitt eget Officepaket, vilket gör det möjligt att importera filer till exempelvis Microsoft Word och Excel.

Microsoft menar att Team System ger möjlighet att förbättra ett projekts design och validitet, det ger även möjlighet att minska risken som kan uppstå i samband med en ny produktutgåva.

Codeplex, som kan beskrivas som ett Open Source projektssamfund, beskriver att Team Foundation Server ger möjlighet att skapa en riskbedömningsrapport under en viss tidsperiod, som exempelvis kan innefatta ett visst projekt eller en viss utvecklingsperiod.

Open Source

De verktyg som har undersökts inom Open Source är alltså Open Source Requirement Management Tool (OSRMT), Testlink, Trac, Bugzilla Subversion, Eclipse. I asnitt ”3.4 Ett praktiskt exempel - adressboken” visas hur de olika verktygen kan kopplas samman.
Nedan beskrivs vilka integrationer som jag tycker har fungerat tillfredsställande:

Testlink: Trac, Bugzilla
Bugzilla: Testlink, Eclipse
Trac: Testlink Subversion, Eclipse
Subversion: Trac, Eclipse
Eclipse: Subversion, Trac, Bugzilla

4.1.3. Sammanfattning av installation och integration

En viktig del av examensarbetet bestod i att undersöka om det fanns Open Source-verktyg som kunde kopplas ihop för att stödja en ALM utvecklingsprocess. En förutsättning blev att själv installera en uppsättning verktyg och sedan försöka koppla ihop dem. En process som var mycket svår och tidskrävande, då jag inte hade någon kunskap om verktygen sedan tidigare.

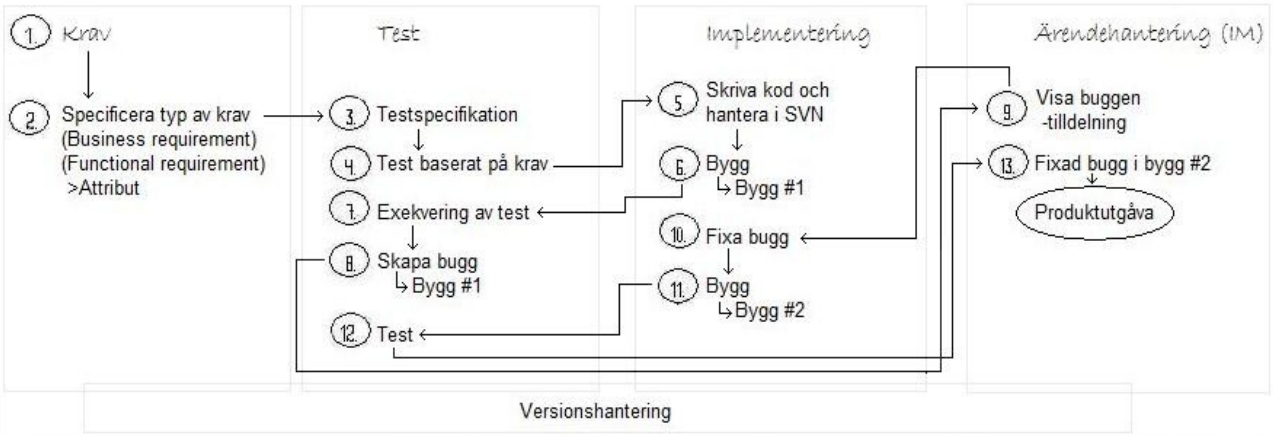
Under installations- och integrationsarbetet påträffades ett antal problem, som ibland kunde ta någon timme att lista ut, medan andra problem tog flera dagar och till och med veckor att lista ut. Några av problemen ledde till att jag fick byta verktyg då jag aldrig kunde hitta vad problemet berodde på.
I ”Appendix 1: Installation och integration” beskrivs i vissa fall grundläggande funktionalitet och

även sådant som upplevdes krångligt eller svårt.

Har man en annan utvecklingsmiljö än den som jag använde så är det inte uteslutet att andra problem påträffas, eller också att vissa av dessa problem inte uppkommer alls.

4.1.4. Ett praktiskt exempel – adressboken

För att ge en förståelse av hur verktygen fungerar tillsammans och hur de får nytta av varandra, så följer här nedan ett exempel hur ett nytt affärskrav blir till och sedan får man följa dess väg till ett exempel på ett slut. Det är ett exempel på hur en utvecklingscykel kan gå till, och naturligtvis kan den se annorlunda ut för ett annat projekt. Exemplet är hämtat från företaget Borland, som har tagit fram en webb-baserad adressbok över alla företagets kunder och intressenter. Syftet att ge en förståelse av hur utvecklingsprocessen fungerar och alla dess ingående delar.



***Figur 18**: End-to-end exempel*

Förberedelse: Konfigurationshantering av projekt

Akt: Versionshantering
Roll: Implementerare
Verktyg: Subversion

Versionshantering kan sägas pågå under hela utvecklingsperioden och ska täcka alla områden. Egentligen kan man säga att detta steg är det första steget, men eftersom steget är en del av alla andra steg så förklarar ordet ”förberedelse” dess innebörd bättre. I exemplet hanteras främst versionshantering vilket innebär att det finns ett antal bevakade mappar och/eller filer som uppdateras varje gång en förändring uppstår. Verktyget möjliggör en historik av alla förändringar, vilket gör det enkelt att återgå till en tidigare version om något blir fel.

Subversion

Subversion är ett konfigurationshanteringsverktyg som bland annat kan användas vid versionhantering. Det finns möjlighet att ha ett gemensamt utrymme för alla aktuella filer och liknande i ett projekt. Detta gemensamma utrymme kan exempelvis nås via ett webbgränssnitt där det även finns möjlighet att skydda informationen via inloggning och olika rättigheter.

Subversion nås via:
http://localhost/svn

Infoga projekt i gemensam förvaringsplats (repository)

- Skapa en egen mapp där alla projektfiler ligger (c:\tobias\material om alm\end-to-end

exempel\testprojekt).

- Infoga sedan ett kravdokument från openoffice ”krav.odt” (mer om krav, och hur man kan skapa en kravspecifikation finns förklarat i ”Steg 1: Sätta upp krav” nedan.
- Vidare använde jag högerklick-menyn och tog **TortoiseSVN > import**
- Vidare matade jag in URLen till min trunk-mapp i repositoryn (c:\myproj\trunk)
- Sen kryssade jag i de filer som jag ville skulle infogas i trunk-mappen

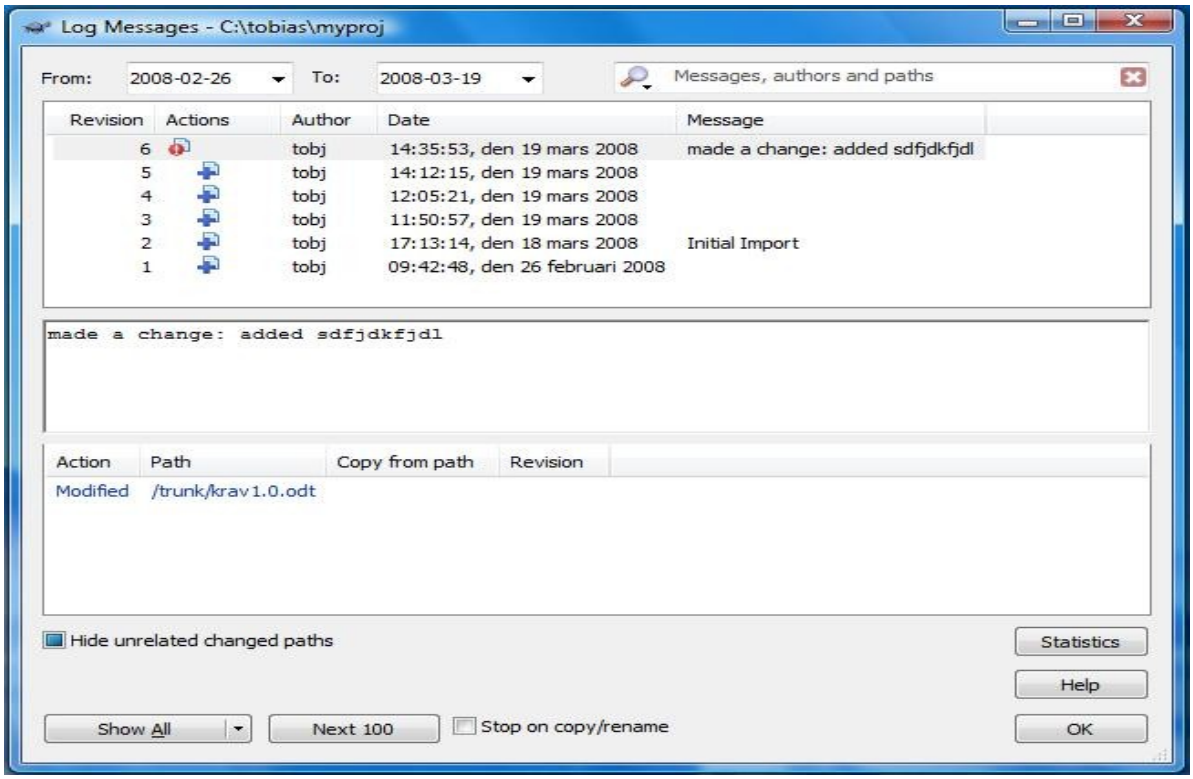
Ny version

- Jag gjorde en ändring i min ”krav.odt” i projektmappen och sparade filen
- Väljer man sedan **SVN update** så infogas ändringen av ”krav.odt” från repositoryn

Ny fil till repositoryn

- Stå i projektmappen och välj alternativet **SVN commit...**
- Kryssa i de filer som du vill ska placeras i repositoryn
- Tryck **OK**
- Gör sedan en **SVN Update** på repositoryn så kan du se ändringen

Figuren nedan visar en historik på förändringar av mappen c:\tobias\myproj



***Figur 19**: Logg över förändringar i en förvarings- (repository-)mapp (TortoiseSVN)*

Steg 1: Sätta upp krav

Akt: Sätta upp krav
Roll: Affärsanalytiker (business analyst)
Verktyg: OSRMT

Krav används som en utgångspunkt för hur ett system ska fungera, och ska främst specificera vad systemet ska kunna göra och möjligen även vad det inte ska göra (vilket ofta blir underförstått). Krav bör specificeras innan man börjar med något annat, och de används senare för att hjälpa

utvecklaren/na att ta fram ett system som tillfredsställer intressentens behov. Till att börja med sätter en affärsanalytiker (business analytiker) upp affärskrav som ofta är väldigt övergripande.

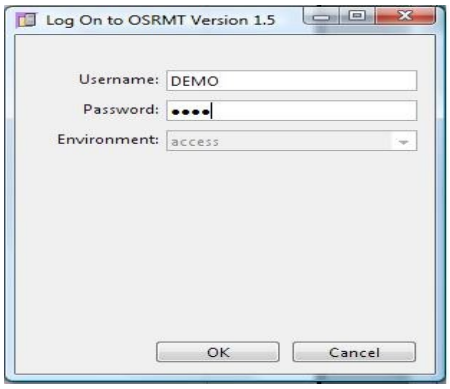
I adressboken ska det exempelvis vara möjligt att söka på ett antal attribut som kan kopplas ihop med personerna i adressboken.

Ett affärskrav kan innebära följande:
”Anställda ska ha möjlighet att söka på kontakterna i adressboken genom att mata in namn, stad, land eller telefonnummer.”

OSRMT

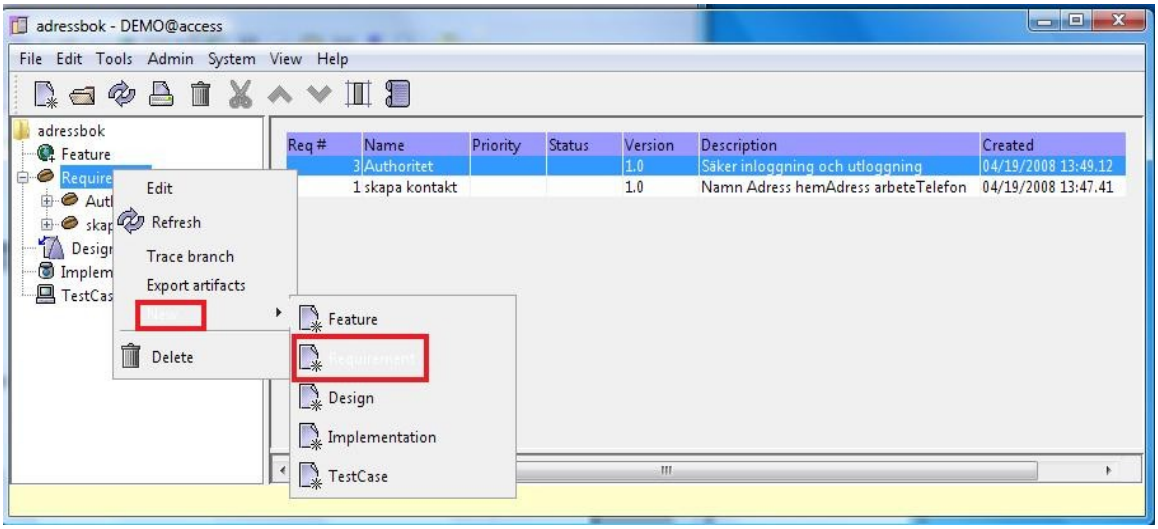
OSRMT är ett kravverktyg som ger möjlighet att specificera upp projektets alla krav. Det går att dela in kraven i olika komponenter som då kan baseras på en speciell funktionalitet i systemet. Denna indelning gör det enklare att få en översikt varför kraven finns och vad de ska användas till.
Logga in genom:

- Användarnamn: DEMO
- Lösenord: demo



***Figur 20**: Inloggning OSRMT*

- Högerklicka på fliken som heter Requirement så får du upp en meny: *New-> Requirement*



***Figur 21**: Skapa krav i OSRMT*

Steg 2: Specificera krav

Akt: Göra om affärskrav till funktionellt krav
Roll: Kravanalytiker (Requirement analyst)
Verktyg: OSRMT

Alla affärskrav skickas sedan vidare till en kravanalytiker som beskriver exakt hur kravet ska kunna genomföras i systemet.

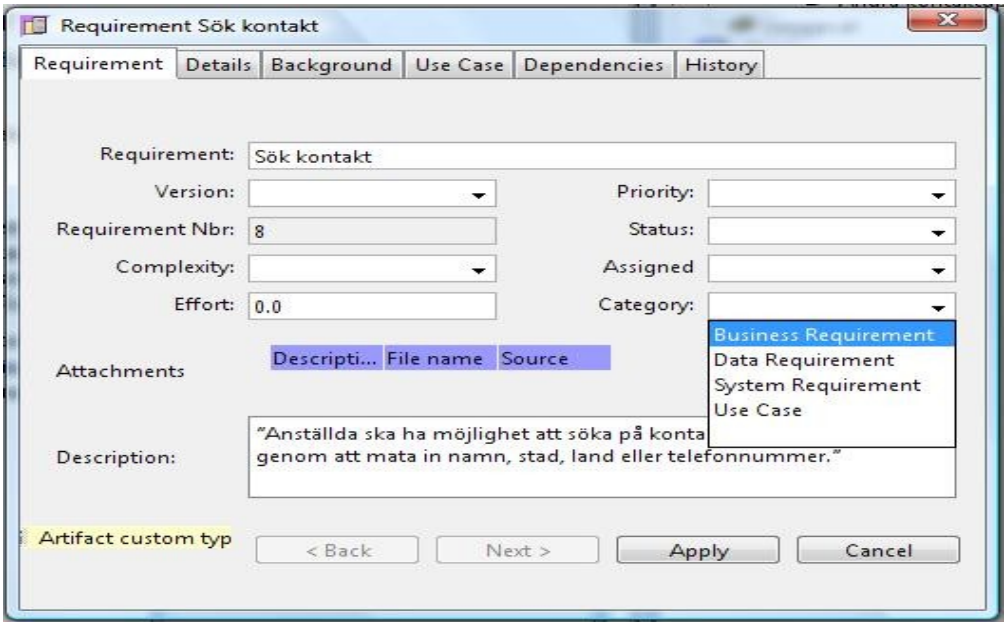
Ett funktionellt krav kan se ut så här:
Den anställda kan genom ett webbgränssnitt dra nytta av en sökfunktion där alla attribut

representeras tillsammans med ett sökningsfält:
”Sök på namn”
”Sök på stad”
”Sök på land”
”Sök på telefonnummer”

Funktionen aktiveras genom en knapp ”Sök”

OSRMT

I inställningarna när man skapar ett nytt krav finns det möjlighet att ställa in vilken typ av krav det handlar om, bland annat business requirement (affärskrav), system requirement (systemkrav) och även use case (användarfall). Det finns även möjlighet att inte välja något av dessa alternativ, vilket då kan syfta till en annan typ av krav, som exempelvis funktionella krav. Se även figur nedan.



***Figur 22**: Att specificera typ av krav i OSRMT*

Steg 3: Testspecifikation

Akt: Specificera test
Roll: Testare
Verktyg: Testlink

Test skapas för att kontrollera så att kraven är implementeringsbara i systemet. De utförs även för att kontrollera så att ingen funktion är överflödig, eller att någon saknas. Test specificeras upp av utvecklaren och varje test bör täcka någon kravfunktion som sattes upp tidigare. Ett lämpligt sätt är att specificera upp exakt vad kravet innebär först och sedan beskriva vilka olika steg eller åtgärder som krävs samt det förväntade resultatet av att genomföra varje steg. Får man ej ut det förväntade resultatet av varje åtgärd så går inte testet igenom, vilket betyder att testet är ”misslyckat”.

Om man går vidare på samma exempel som togs fram i kravdelen så skulle ett test av detta krav kunna se ut såhär:

Sammanfattning
Du har möjlighet att söka i databasen baserat på följande variabler:

Namn
Stad
Land
Telefonnummer

Åtgärder	Förväntade resultat
1. Mata in variblen ”Jay” i namnfältet	1. Namnet ”Jay” kommer synas i fältet där namn ska fyllas i
2. Tryck på knappen "Start search"	2. Personer som heter ”Jay”-någonting dyker upp i en lista

Testlink

I Testlink finns det möjlighet att skapa tester för skilda projekt, och även för olika komponenter i projekt. Det finns möjlighet att exekvera testerna och se testernas resultat. Vid nedladdning via programmets hemsida medföljer en mycket övergripande manual över hur man utför de olika funktionerna.

Testlink kan nås via:
http://localhost/testlink/index.php

Steg 4: Test baserat på krav

Akt: Funktionstest
Roll: Testare
Verktyg: Testlink

Testerna bör alltså först kopplas ihop med funktionalitet, men det är dock inte alltid så att funktionen är grundläggande, utan den kan även bygga på ett annat krav. Syftet med detta steg är att testaren ska veta vad som krävs för att funktionen ska uppfyllas, och även för att utvecklaren ska veta när en viss funktion ska vara möjlig att genomföra.

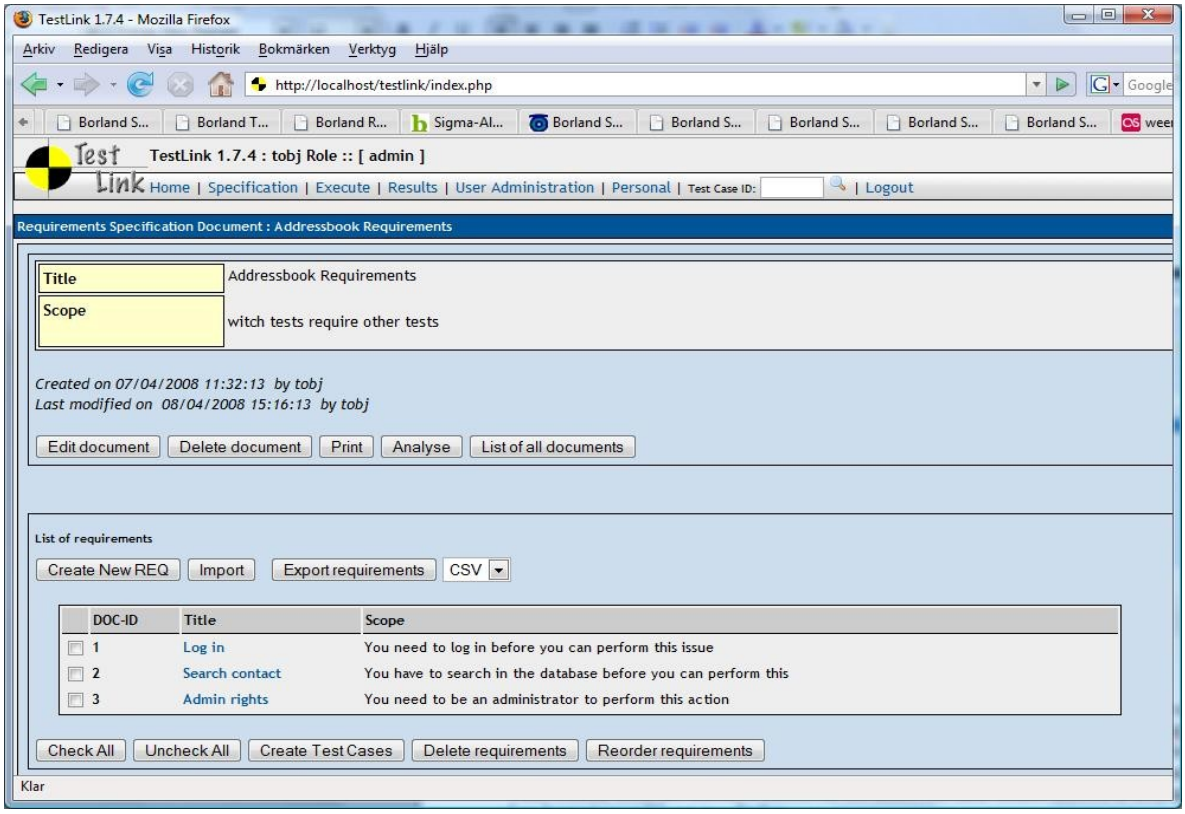
Utgår jag från samma exempel så var det alltså så att företagets anställda skulle kunna söka i databasen, men av någon anledning vill man inte att någon utomstående ska få tillgång till dessa uppgifter. Kravet för att kunna söka i databasen är med andra ord att man är inloggad i systemet. Illustrationen beskriver att man måste vara inloggad för att kunna utföra de specificerade testfallet:

Krav: Inloggad	Testfall
	Sök kontakt

Testlink

Det finns ett antal olika möjligheter att koppla testerna till uppsatta krav. Nedan följer två exempel på hur detta kan genomföras. De ena sättet är att använda Testlinks egna relateringsverktyg. För att aktivera denna funktion måste man välja alternativet ”Yes” vid förfrågan om ”Enable Requirements functionality” när man skapar ett nytt testprojekt. Vidare är det också ett krav att det finns minst två olika testfall, så att det ena kan relateras till det andra. Används denna funktion är möjligen kravhanteringsverktyget överflödigt (se figur ”Figur 23: Kravbaserad testning i Testlink”). Det andra sättet är att använda sig av ”Custom fields” där man har möjlighet att skapa en variabel som kan relateras till exempelvis OSRMT. Ett enkelt sätt är att använda ett unikt attribut för kraven, såsom kravets ID. På så sätt går det att ange vilket ID kravet har i OSRMT när ett testfall skapas, vilket ger en överblick hur testerna är kopplades till kraven.

Fördelen med att ha ett separat kravdokument är att det ger en bättre överblick, och det är även enklare att se förändringar i kraven om dokumentet kopplas ihop med versionhantering.



***Figur 23**: Kravbaserad testning i Testlink*

Steg 5, 6: Skriva kod och hantera i SVN – bygg ”Bygg #1”

Akt: Programmering/Implementering
Roll: Programmeringsutvecklare/Implementerare
Verktyg: Eclipse/Subversion

Koden tas fram av en programmeringsutvecklare som vid utformningen utgår ifrån de uppsatta kraven. Programmering eller utformning av kod är den fas då all funktionalitet implementeras i systemet, vilket gör att kraven kan testas rent praktiskt.

Eclipse

Eclipse ger möjlighet att bygga upp ett system. Det är enkelt att se om man exempelvis har använt en variabel felaktigt eftersom alla programmeringstermer får en färg. Verktygets främsta syfte är alltså hantering av kod och även agera som en klientplattform. I ”Figur 24: Hitta fel i Eclipse” finns det möjlighet att se hur det kan se ut vid hantering av kod i Eclipse.

Infoga koden i Subversion – ”bygg #1” skapas

Kod från Eclipse kan vidare infogas i Subversion för att ge en överblick av när förändringar skett i koden och vem som utförde förändringen, och även vad förändringen innebar. En exakt beskrivning om hur man infogar ett projekt i Subversion finns att tillgå i avsnittet ”Ny fil till repositoryn” som förklarades tidigare. Detta steg är grunden för första utformningen eller ”bygget”, det skapas alltså en version.

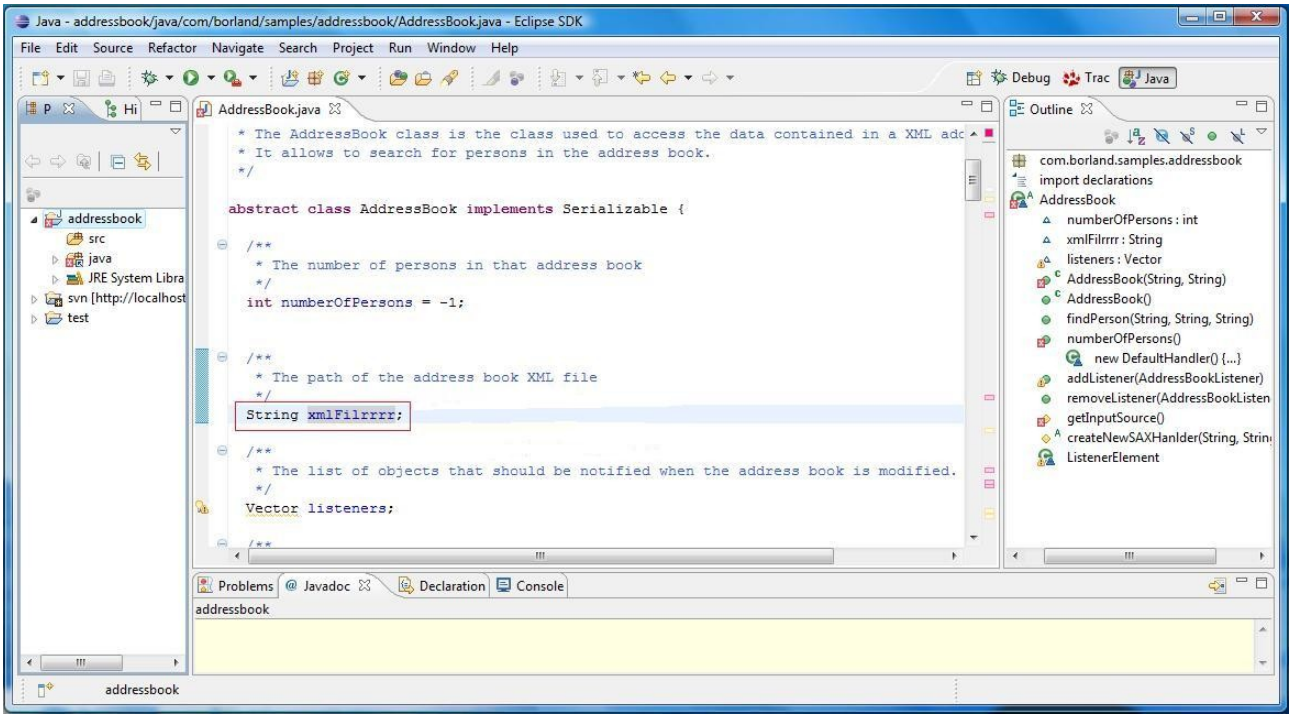
Steg 7: Exekvera test och hitta fel

Akt: Exekvera test
Roll: Testare
Verktyg: Testlink

När utvecklaren tagit fram all önskad funktionalitet, så skickas den vidare till en testare vars uppdrag är att se till så att alla funktioner fungerar på ett korrekt sätt. Varför det kan vara bra att testa funktionerna beror på att utvecklaren kan ha missat vissa aspekter under framtagandet eller möjligen glömt någonting. Det allra enklaste fallet är naturligtvis att bara testa funktionen utan något speciellt tillvägagångssätt.

Samma exempel kan då innebära:
Får jag upp alla personer som heter ”Jay”-någonting om jag matar in detta i namnfältet?
Svar: Ja!

För att ge en förståelse av att ett test kan misslyckas, så har jag skapat ett fel i koden. Felet innebär att en variabel i koden av någon anledningen heter något annat än vad den ska. Oftast är felet mycket mer komplext än så, men syftet är som sagt att visa hur alla delar i utvecklingscykeln fungerar. I figuren nedan visas hur det kan se ut:



Figur 24: *Hitta fel i Eclipse*

Testlink

I Testlink kan vidare varje specifik funktion testas. Verktyget är uppbyggt på att åtgärder samt vad det förväntade resultatet ska bli av varje åtgärd. Följer man åtgärderna och får det förväntade resultatet så är testet lyckad, medan om något inte fungerar enligt förväntan så är det med största sannolikhet något som är fel. Exemplet utgår ifrån att något gick fel.

Steg 8 Skapa bugg – slutfas ”bygg #1”

Akt: Bugg skapas

Roll: Testare
Verktyg: Bugzilla/TRAC

Nästa steg för testaren blir att rapportera in den bugg eller det fel som har påträffats. Anledningen till att rapportera in felet är att utvecklaren som har hand om denna typ av ärenden kan se över felet och rätta till det så att systemet fungerar ännu bättre. Nedan följer tre olika sätt att rapportera in en bugg.

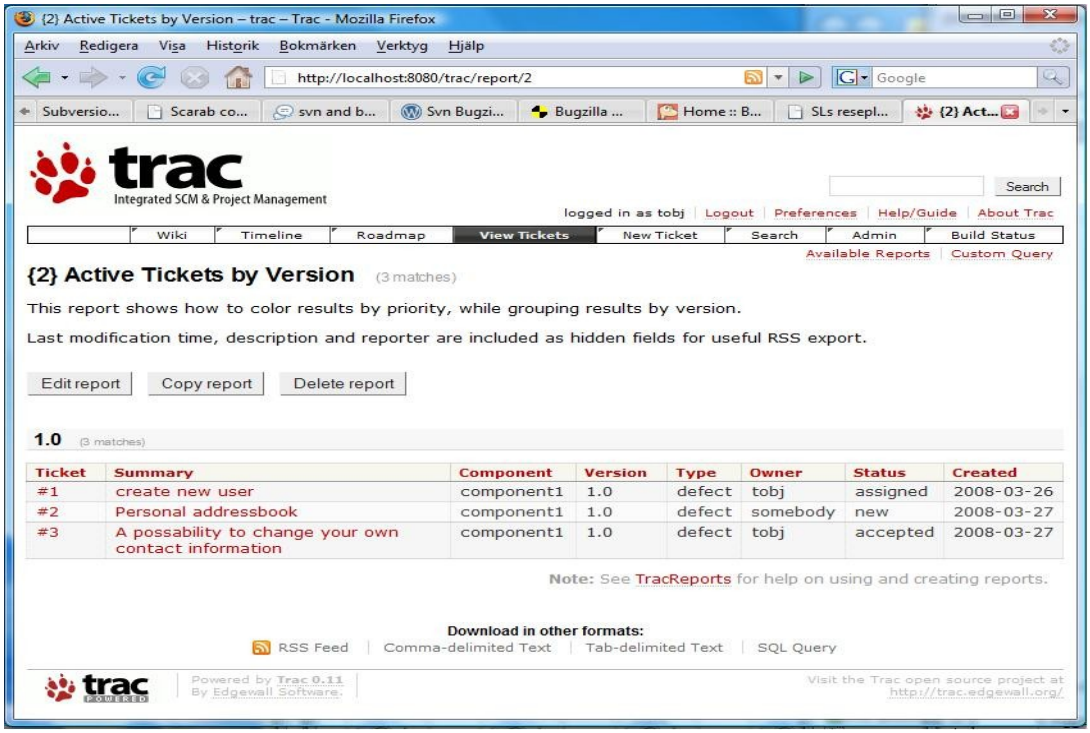
Bugzilla i Eclipse

Vid exekvering av kod uppstår inte sällan något oväntat fel, vilket exempelvis kan innebära att man pekar på ett felaktigt objekt. Det kan därför vara lämpligt att ge möjlighet att rapportera dessa fel direkt när de uppstår, då man i annat fall kanske glömmer av exakt vad felet handlade om eller hur det uppstod. Funktionen kopplas ihop med Bugzillaservern och ger exakt samma gränssnitt som en webbläsare gör. I ”Figur 25: Lista över 'Tickets' i TRAC” visas hur buggen har rapporterats i en bugglista i verktyget TRAC. Funktionen ser ut och fungerar på liknande sätt i Bugzilla.

Eftersom en bugg har hittats kommer det krävas någon form av förändring i systemet, detta leder till att det skapas en ny version efter denna fas

TRAC i Testlink

- Starta Trac-servern
- Gå in på servern via en webbläsare: http://localhost:8080/trac
- Logga in
- Skapa tickets i Trac som du kan relatera till i Testlink (Flik ”New Ticket”)



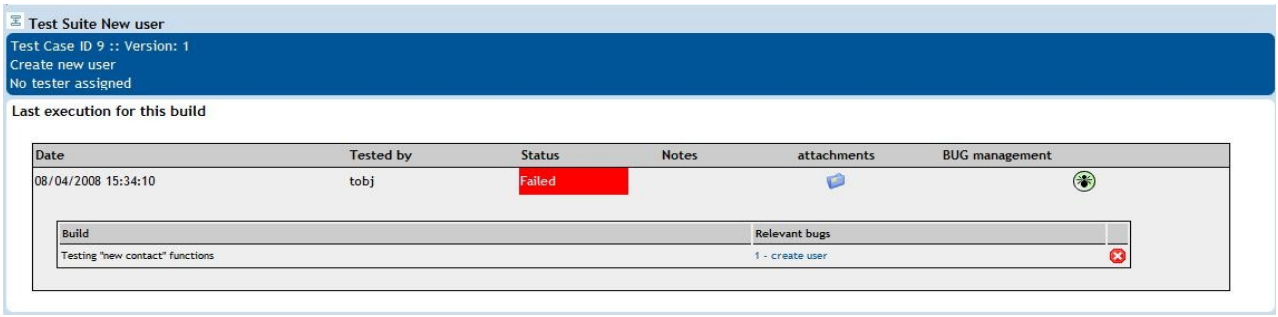
Figur 25: *Lista över 'Tickets' i TRAC*

- Rapportera sedan via Testlinks ”Add bug” funktion som ska dyka upp när man har exekverat ett test.
- När buggen är rapporterad ska ett ”pop-up”-fönstret tala om att buggen är tillagd. Har något

fel uppstått vid inmatning, exempelvis ett felaktigt ”bugg-ID”, så får man ett meddelande att buggen inte finns. I ”Figur 26: Buggrapport i Testlink med Bugzilla eller TRAC” finns en figur hur det kan se ut om funktionen hittar buggen i ärendehanteringssystemet och därmed kan relatera till densamma.

Bugzilla i Testlink

- Gå in på servern via en webbläsare: http://localhost/bugzilla
- Logga in
- Skapa tickets i Bugzilla som du kan relatera till i Testlink (Flik ”New”)
- Rapportera sedan via Testlinks ”Add bug” funktion som ska dyka upp när man har exekverat ett test
- När buggen är rapporterad ska ett ”pop-up”-fönstret tala om att buggen är tillagd. Har det dykt upp något fel i samband med inmatningen så får man ett meddelande att buggen inte finns. Figuren nedan visar hur funktionen ser ut i Bugzilla och i TRAC .



***Figur 26**: Buggrapport i Testlink med Bugzilla eller TRAC*

Steg 9: Visa buggen och tilldelning

Akt: Buggen visas och skickas till en lämplig utvecklare
Roll: Utvecklare
Verktyg: TRAC/Bugzilla

När buggen är rapporterad kan en utvecklare som har hand om denna typ av ärenden se över problemet. I de ärendehanteringssystem jag har analyserat går det att koppla ihop en viss typ av ärenden med en viss typ av utvecklare; handlar exempelvis buggen om något i webbgränssnittet så skickas det till en utvecklare som sysslar med det. Nedan beskrivs två olika ärendehanteringssystem.

TRAC

TRAC är ett ärendehanteringsverktyg, där man kan specificera upp vad som upplevdes när ett fel eller en bugg uppstod. Det finns möjlighet att använda inloggning för att få mer specificerade rättigheter, och det går även enkelt att se vilka ärenden som är kopplade till vem.

Bugzilla

Bugzilla är det andra ärendehanteringssystemet, och fungerar på liknande sätt som TRAC som beskrivs under det första alternativet i ”Steg 9: Visa buggen och tilldelning”.

Steg 10, 11: Fixa buggen – ny bygg ”Bygg #2”

Akt: Buggen behandlas
Roll: Utvecklare

Verktyg: Bugzilla/TRAC

För att återgå till exemplet ovan, med den felaktiga variabeln, så går detta steg ut på fixa till felet. Utvecklarna bör för slutanvändarnas skull se över buggrapporterna ofta, så att felen rättas till så snart som möjligt. Utvecklaren kan även rapportera in om det hänt något nytt med ärendet; rapporteringen i ärendehanteringssystemet behöver inte bara betyda att utvecklaren löst felet, utan kan även visa att den är under behandling och ett antal andra alternativ. Det är även möjligt att kommunicera med slutanvändaren genom att lägga in kommentarer till utvecklingsarbetet och även en eventuell lösning om felet blev löst. Förändringen skapar en ny byggkomponent.

Eclipse med TRAC/Bugzilla

I Eclipse kan man sedan försöka ta fram en lösning på buggen. Lyckas man få en lösning rapporteras detta i TRAC/Bugzilla, vilket i sin tur skapar en ny byggkomponent.

Steg 12: Test

Akt: Exekvera test
Roll: Testare
Verktyg: Testlink

En testare bör sedan se över förändringen för att se så att lösningen är korrekt och klarar av alla tänkbara påfrestningar. Detta steg är en upprepning av ”Steg 7: Exekvera test och hitta fel”

Testlink

Steget går ut på att testa och exekvera den nya byggkomponenten. Går inte testet igenom så får man börja om från ”Steg 8 Skapa bugg – slutfas ”bygg #1”. Går testet däremot igenom så går det bra att fortsätta till nästa steg.

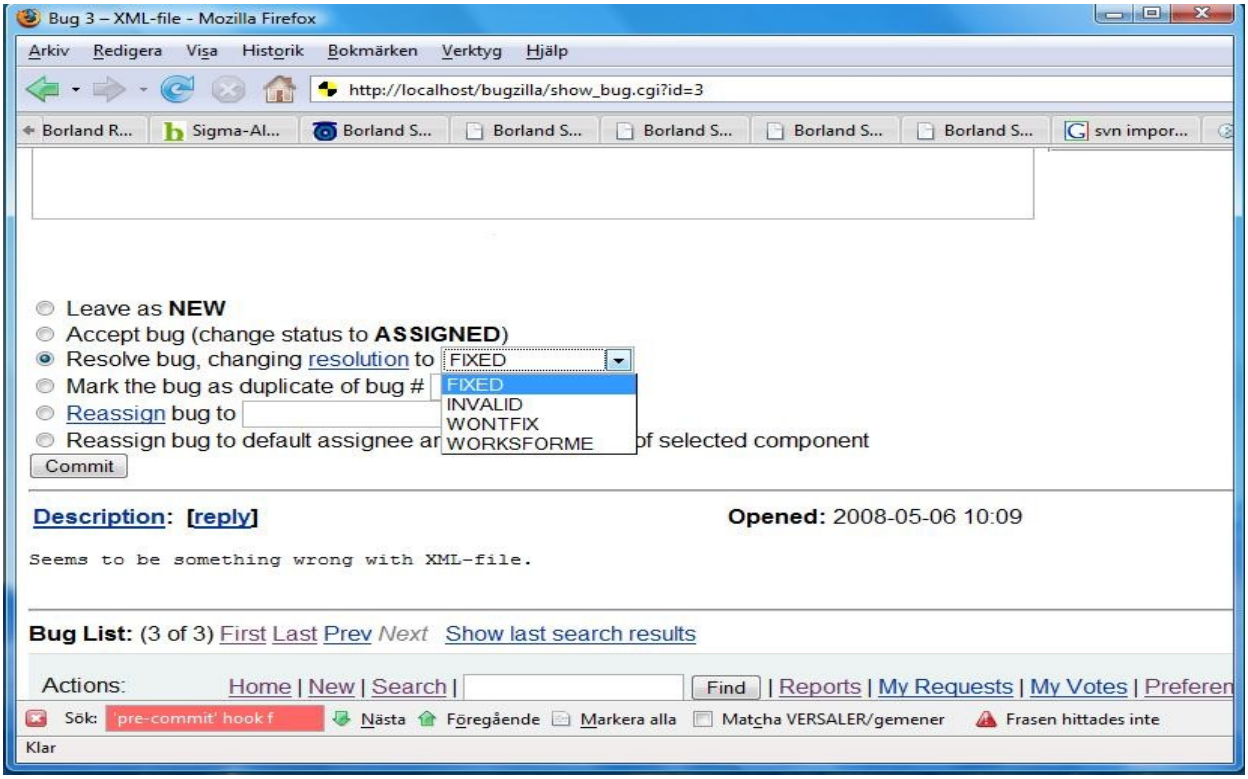
Steg 13: Bugg fixad i ”bygg #2” - produktutgåva

Akt: Rapportera att bugg är nehandlad
Roll: Utvecklare
Verktyg: Bugzilla/TRAC

Går testerna igenom kan sedan testaren eller utvecklaren rapportera in att buggen är löst. För det mesta går det då att uppgradera sin version med hjälp av en patch, om det släppts någon version tidigare, i annat fall blir det en ny produktutgåva. Är det ett Open Source-verktyg finns det även möjlighet att visa slutanvändaren vilka förändringar som gjorts i koden vilket öppnar för att justera buggen på egen hand. Nedan följer två exempel på hur det kan se ut i ett ärendesystem.

Bugzilla

Finns det en tillfredsställande lösning på den inrapporterade buggen så beskrivs alla åtgärder som togs vid i ärendehanteringssystemet och buggen kan ”strykas”. Att den stryks innebär att buggen anses löst och att inblandade personer kan ta del av alla förändringar. Steget leder även till en ny produktutgåva. I figuren nedan visas hur man kan rapoortera in att buggen är löst:



Figur 27: Bugzilla buggrapportering: bugg löst

TRAC

TRAC är det andra ärendehanteringssystem som fungerar på liknande sätt som Bugzilla gör. Buggrapporteringen ser ut på liknande sätt som i Bugzilla, som illustrerades i ”Figur 27: Bugzilla buggrapportering: bugg löst”

4.2 Enkätundersökning

I detta avsnitt bifogas frågor och svar till min enkätundersökning.
Först bifogas det meddelande som jag skrev för att ge all information om anledning till mitt utskick. Syftet var ett ge en bild av bakgrunden samt också lite information om beräknad tidsåtgång och vad de skulle vinna på att besvara enkäten. Nedan bifogas meddelandet:

Hej.
Mitt namn är Tobias Björk och jag är inne på mitt fjärde och sista år av min utbildning inom systemvetenskap på Göteborgs Universitet. Jag undrar om du har 15 till 20 minuter att hjälpa mig med en undersökning om risker kopplat till införandet av ALM-verktyg (Application Lifecycle Management). Självklart kommer ni att ha möjlighet att få del av resultatet av min undersökning.

Jag håller på med ett examensarbete på Signifikant i Stockholm som består i att ta fram en utvärderingsmall för verktyg för systemutveckling. Det som ska undersökas är införandet av ALM-verktyg i ett existerande system. Utvärderingsmallen ska innehålla verktygens brister och fördelar. Mallen kommer bland annat beskriva verktygens supporterade operativsystem, pris, integrationsmöjligheter, upplevd svårighetsgrad vid installation, funktionalitet samt risk. Vidare vill jag även jämföra kommersiell mjukvara och Open Source-mjukvara inom området, med hjälp av utvärderingsmallen. Verktygen kommer väljas ut baserat på deras funktionalitet, marknadsposition och förmåga att bli del av en helhetslösning. De processområden jag har koncentrerat mig på är kravhantering (RM), konfigurationshantering (CM), ärendehantering (IM) testhantering (TM), design och bygg (build). Det här frågeformuläret behandlar risker med införandet, och ska bli en del av helheten.

Svaren kommer hanteras anonymt då syftet med min enkät är att få en helhetsbild av vad slutanvändaren har för uppfattning av de olika verktygen. Frågorna riktar sig främst till personer som har erfarenhet av implementering av utvecklingsverktyg. I enkäten kan du även se vilka verktyg jag är intresserad av och där du följaktligen besvarar frågorna för de verktyg eller de uppsättningar av verktyg som du anser ha erfarenhet av. Om du exempelvis anser att Borlands (som är en av företagen jag har undersökt) uppsättning av verktyg ger ungefär samma svar så ge gärna ett sammanfattande svar där du skriver företagets namn som ”produktens namn”. Gällande Open Source-verktygen vill jag dock ha skilda svar för varje verktyg.

Jag bifogar ett frågeformulär (frågeformulär.doc) där du får all information om verktygen samt lite bakgrund. Jag bifogar även ett formulär som gör det enklare att svara på frågorna (svarformulär.doc). I det dokumentet ser du bara frågorna och det finns även mer plats för ett svar. Jag rekommenderar att du tittar på frågeformuläret först, för att få all information som är nödvändig för att besvara frågorna.

Jag önskar svar på frågorna senast om 14 dagar (torsdagen den 1/5), men skulle även uppskatta snabbare svar.

Hälsningar,
Tobias Björk

(Mailadresserna har hämtats från en lista över branschens IT-chefer.)

4.2.1. Frågeformulär ”risker med införandet av ALM-verktyg”

Nedan finns det frågeformulär som bifogades i det epost-meddelande jag skickade ut. Tillsammans med meddelandet som finns ovan, så ansåg jag att de fått all nödvändig information som krävdes för att besvara dessa frågor på ett tillfredsställande sätt:

Frågor om:

- **Borland** CaliberRM, StarTeam, SilkCentral IssueManager, SilkCentral TestManager, Together Arcitect
- **IBM** Rational RequisitePro, ClearCase, Software Architect
- **Telelogic** DOORS, SYNERGY, TAU Architect
- **Microsoft** Visual Studio Team Systems och Team Foundation Server
- **Open Source** Eclipse, Subversion, Trac, Bugzilla, Testlink, Open Source Requirement Management Tool (OSRMT)

Bakgrund

Anledningen till att jag utformat följande frågeformulär beror på att jag vill få en allmän uppfattning om programmens riskmedvetenhet. Det är svårt att få ett helhetsperpektiv genom att läsa företagens information om de olika produkterna eftersom de ofta undviker att skriva sina nackdelar. Jag har även fått en känsla av självgodhet i beskrivningarna, där de gärna lovar mycket för att det ska låta bra och för att sälja.

Innan frågorna besvaras vill jag ge en beskrivning av hur statskontoret (1997) tolkar ordet risk. De beskriver ett hot som en handling eller händelse som kan komma att skada en IT-resurs, exempelvis information eller en applikation. Hotet kan uppstå om systemet har en svag punkt som kan utnyttjas. Vidare beskrivs risk som sannolikheten att hotet ska realiseras.

Besvara frågorna för varje verktyg som du anser ha en viss erfarenhet av:

Del 1 av 3. Verktyg

Var god och ange vilka verktyg ni har erfarenhet av att implementera i er nuvarande eller tidigare organisation:

Requirement mgt
- o Borland CaliberRM
- o IBM RequisitePro
- o Telelogic DOORS
- o Open Source Requirement Management Tool (OSRMT)

- o Annat: _ _ _ _ _ _ _ _

Design/Build
- o Borland Together Architect
- o IBM Software Architect
- o Telelogic TAU Arcitect
- o Eclipse

- o Annat: _ _ _ _ _ _ _ _

Issue management
- o Borland SilkCentral IssueManager
- o Microsoft TFS
- o Trac
- o Bugzilla

- o Annat: _ _ _ _ _ _ _

Test management
- o Borland SilkCentral TestManager
- o Microsoft TFS
- o Testlink

- o Annat: _ _ _ _ _ _ _

Configuration mgt
- o Borland Starteam
- o IBM ClearCase
- o Telelogic SYNERGY
- o Microsoft TFS
- o Subversion

- o Annat: _ _ _ _ _ _ _

Uppsättningar (mer än en produkt ur listan)

- o Borland: CaliberRM, StarTeam, SilkCentral IssueManager, SilkCentral TestManager, Together Arcitect
- o IBM: RequisitePro, ClearCase, Software Architect
- o Telelogic: DOORS, SYNERGY, TAU Architect
- o Microsoft: Visual Studio Team Systems och Team Foundation Server
- o Open source: Eclipse, Subversion, Trac, Bugzilla, Testlink, Open Source Requirement Management Tool (OSRMT)

- o Annat: _ _ _ _ _ _ _ _ _ _ _ _

Del 2 av 3. Risker i samband med implementering av verktyg

Besvara frågorna för varje verktyg som du anser ha en viss erfarenhet av. Om du har angett flera produkter innan, ange gärna för vilken produkt(er) svaret gäller.

- Upplevdes någon risk vid implementering i ett existerande system?
- Kan det innebära stor risk att börja använda verktyget utan att besitta rätt kunskap?
- Finns det tester som bedömer en användares förkunskaper?
- Upplevs verktyget pålitligt? Om inte, kom med exempel. Försök svara på nedanstående frågor:
  - Har du upplevt att information försvunnit?
  - Hänger sig verktyget ofta?
  - Upplever du alla funktioner säkra och förutsägbara?

  Om verktyget känns pålitligt, kan du då beskriva vad du tycker känns bra? Nedan följer ett par exempel på vad som kan upplevas pålitligt:
  - Tillgänglighet: Verktyget finns tillgängligt under de tider som det ska och för behöriga användare i beslutad omfattning. Verktyget hänger sig sällan eller aldrig.
  - Hög säkerhet gällande inloggning och rättigheter.
  - Förutsägbara och enkla funktioner
  - Spårbarhet: Funktioner och rutiner som gör det möjligt att se vilka operationer som utförts av vem.

Del 3 av 3. Risker efter implementering av verktyg

Det finns även en annan aspekt gällande risker och mjukvara som handlar om värdering av IT-investeringen. Denna aspekt behandlar de beslut som ska fattas om den IT som ska köpas in eller som redan finns i organisationen. Angående de kommersiella produkterna så kan det handla om att förnya licenser eller att köpa in både produkt och licens till ett stort antal klienter. Gör man en stor investering så gäller det att vara väl medveten om vad man vill ha, så att man fattar rätt beslut.

Tänk dig följande scenarion, och besvara sedan de punkter som följer:

5. Företaget man har investerat i går i konkurs eller också dyker det upp en mycket bättre produkt som gör att din investering blir föråldrad.
   (Gällande open source så kanske projektet läggs ner, eller byts ut mot ett annat som är mycket bättre)

   - Försök ge exempel på hur du tror att denna händelse skulle påverka din organisation
   - Vad skulle det innebära för kostnader för den egna organisationen?
   - Hur sannolikt känns detta scenario?

6. Din organisation köper in den allra senaste versionen i någon av programuppsättningarna. Det uppstår dock ett visst motstånd mot att använda detta nya system, speciellt bland de lite äldre och mer erfarna inom organisationen. Detta leder till att de flesta använder det gamla systemet medan en liten del använder det nya

   - Försök ge exempel på hur du tror att denna händelse skulle påverka din organisation
   - Vad skulle det innebära för kostnader för den egna organisationen?
   - Hur sannolikt känns detta scenario?

4.2.2. Svar på frågeformulär

I detta avsnitt har jag sammanfattat de svar som samlades in via enkätundersökningen. Avsnittets syfte är att ge en överblick av hur en slutanvändare kunde besvara frågorna, för att på så sätt ge en bättre förståelse hur de behandlades i utvärderingsmodellen. Jag har därför för varje verktyg delat upp svaret i för- respektive nackdelar och även delat in varje enskilt svar i en kategori. De kategorier jag har utgått ifrån är användarvänlighet, utbildning, tillförlitlighet, åtkomst, säkerhet, informationsförlust, spårbarhet och tillgänglighet. Då kategorierna ibland går in i varandra och är ganska lika så kan ett svar ibland innefatta mer än en kategori. För varje kategori tilldelas poäng, som då kan bli negativt om det finns antydningar till att det är ett problem i verktyget, eller positivt om det anses vara något som verktyget är bra på att hantera. Bedömningen utgår ifrån en skala på 1-5 där 5 är bästa möjliga.

Verktygets namn:

IBM RequisitePro (kravhantering)
IBM ClearCase (konfigurationshantering)
Microsoft Visual Studio TFS (design)

Problem:

- *Användarvänlighet:* Svårt att få utbrett användande hos alla
- *Användarvänlighet:* Svårt att ta fram ramverk för verktyget

- *Utbildning:* Svårt att få fram interna resurser (för support och administration av verktyget)
- *Utbildning:* Vid för liten kunskap är risken att bygga in ett svårförvaltat arbetssätt, vilket leder till höga kostnader

Verktygets namn:

Serena Dimension (konfigurationshantering)

Problem:

- *Utbildning:* Hög tröskel för ovana användare – risk för felaktiga auktioner, frustration och felaktiga beslut.
- *Användarvänlighet:* Motsättningar av typen ”det gamla fungerade så här – varför har man gjort allt mycket krångligare?”

Fördelar:

- *Tillgänglighet:* Verktyget hänger sig sällan eller aldrig
- *Åtkomst och säkerhet:* Hög säkerhet gällande inloggning och rättigheter.
- *Spårbarhet:* Funktioner och rutiner som gör det möjligt att se vilka operationer som utförts av vem.

Verktygets namn:

CVS (konfigurationshantering)
Subversion (konfigurationshantering)

Problem:

- *Tillförlitlighet:* Att konvertera repository mellan exempelvis CVS och Subversion är en risk
- *Informationsförlust:* Förlorad information vid merge i Subversion

Fördelar:

- *Användarvänlighet:* Det är enkelt att använda och förstå
- *Tillgänglighet:* Verktyget hänger sig sällan eller aldrig

Verktygets namn:

IBM ClearCase (konfigurationshantering)

Problem:

- *Utbildning:* CM-verktyg bör man ha viss kunskap innan man börjar använda
- *Användarvänlighet:* Kan upplevas mycket komplext.
- *Användarvänlighet:* När merge har gjorts i clearcase har det uppstått problem (pga okunskap?)

Fördelar:

- *Informationsförlust:* Ingen informationsförlust
- *Tillgänglighet:* Verktyget hänger sig sällan eller aldrig

Verktygets namn:

Telelogic Doors (kravhantering)

Problem:

- *Åtkomst:* Dåligt stöd för tunna klienter
- *Säkerhet:* Svårt att överblicka rättigheter som sätts i systemet på ett effektivt sätt
- *Informationsförlust:* Information har försvunnit i samband med mismatch mellan history och aktuellt läge i ett objekt.

Fördelar:

- *Tillförlitlighet:* Det finns ett skript som letar rätt på mismatch
- *Användarvänlighet:* Man får en bra överblick av projektet (dokument baserat i FramMaker).
- *Spårbarhet:* Man kan effektivt spåra och följa information i systemet.
- *Tillgänglighet:* Verktyget hänger sig sällan eller aldrig

Verktygets namn:

SAP solution manager (kravhantering)

Problem:

- *Användarvänlighet:* Vissa funktioner Upplevs osäkra och ologiska

Verktygets namn:

Microsoft Visual Studio TFS (design)

Problem:

- *Utbildning:* Vid för liten kunskap löper stor risk att det blir en del ogjort arbete och risk för merkostnader, framförallt teknikära områden

Fördelar:

- *Utbildning:* Certifikat som kan bedöma viss förkunskap
- *Tillförlitlighet:* Enhetligt, robust.
- *Tillgänglighet:* Verktyget hänger sig sällan eller aldrig

Verktygets namn:

Eclipse (kod)

Problem:

- *Tillgänglighet:* Kan stängas ner helt plötsligt.
- *Tillförlitlighet:* Refactoring funkar inte alltid helt igenom.

Fördelar:

- *Användarvänlighet:* Det är smidigt och enkelt att använda med en mångfald integrationer med övriga verktyg

Verktygets namn:

Borland CaliberRM (kravhantering)

Problem:

- *Informationsförlust:* Det har hänt att användare fått backa tillbaka till senaste backup. (Ej under normala omständigheter)

Fördelar:

- *Tillgänglighet:* Hänger sig sällan eller aldrig
- *Tillförlitlighet:* Funktionerna upplevs säkra och förutsägbara

Verktygets namn:

Borland Starteam (konfigurationshantering)
Fördelar:

- *Informationsförlust:* Inga informationsförluster
- *Tillgänglighet:* Hänger sig sällan eller aldrig
- *Tillförlitlighet:* Funktionerna upplevs säkra och förutsägbara
- *Tillgänglighet:* Den snurrar och snurrar och gör sitt jobb.

Verktygets namn:

Borland (en hel produktuppsättning; kravhantering, ärendehantering, testhantering, konfigurationshantering)

Problem:

- *Utbildning:* Utan rätt kunskap kan inte verktygen utnyttjas effektivt
- *Användarvänlighet:* Har man som administratör för lite kunskap finns det risk att tappa mycket information och många arbetsdagar när man t ex ej har kontroll över kopiorna.
- *Åtkomst och säkerhet:* Svårt för en oerfaren att få en översikt över säkerhets- och åtkomstsrättighetersinställningar.

4.3 Övriga resultat

Vad som har framgått väldigt tydligt när jag har försökt koppla ihop de olika komponenterna med varandra är att programmens version har en väldigt stor betydelse. Många gånger kan man tro att den senaste versionen är den bästa, just eftersom det var den som kom ut sist. Det är dock inte all mjukvara som är bakåtkompatibel (alltså att all funktion som fanns i äldre versioner även finns i den nyaste), vilket har skapat en hel del oreda. Ett program kan då rekommendera att man använder en viss version vid integration, medan ett annat program fungerar bäst med en helt annan version. Exempelvis fungerar Python 2.3 bättre tillsammans med Apache 2.0.x, än vad Python 2.5 gör. Eftersom jag till att börja med lade in Python 2.5 så var jag tvungen att göra om allting som var bundet till den versionen. Detta problem har upprepats ett antal gånger under arbetets gång. Detta är ett stort problem både inom Open Source och även inom den kommersiella marknaden.

Ett annat vanligt problem som påträffades när jag experimenterade med de olika verktygen var att olika inställningar, attribut och liknande ställde till det en hel del. När någonting inte fungerade berodde det ofta på verktygens konfigurationsfiler, som pekade på fel mapp eller fil. Den information man kunde få genom programmens loggar var ofta till liten nytta, eftersom den är otydlig och inte ger någon exakt information av vad problemet är. Några av de fel som uppstod kunde redas ut genom att söka på olika forum och även genom att göra en google-sökning på det fel som jag fick presenterat för mig. I övrigt fick jag många gånger försöka lista ut problematiken själv.

Vad jag har fått lära mig är att det är viktigt att göra en kopia på sådana filer eller mappar man vill ändra i. Många gånger har jag försökt rätta till ett fel genom att testa mig fram på olika sätt. Vad som kan hända är naturligtvis att det blir ännu sämre än vad det var innan man började ändra. Många gånger minns man inte exakt vilka ändringar som har gjorts och det är bara till att börja om från början igen.

5. Diskussion

ALM är synsätt som bygger på ett antal andra tidigare modeller och synsätt. Många av dessa modeller bygger just på att framställa system eller projekt på ett smart sätt. Detta är genomförbart först om man funderar igenom alla innehållande komponenter innan själva utvecklingsarbetet börjar. Meningen med att förbereda och planera projekt är att minimera antalet fel samt att minska risken för att spräcka budget och att dra ut på den beräknade tidsåtgången. En allmän uppfattning idag är att det inte lönar sig att utföra en så omfattande och tidskrävande process som ALM är, men faktum är att allt eftersom marknadens behov och krav på informationssystem ökar så blir även projekten och verktygen allt mer avancerade och krävande. Detta får till följd att utvecklarna måste ta hänsyn till allt fler aspekter, vilket i slutändan kan bli mycket problematiskt om man inte planerat utvecklingsarbetet tillräckligt.

För att ge en bättre förståelse av hur verktygen fungerar både enskilt och hur de kan få nytta av varandra diskuterades först en utvecklingsprocess där varje steg i processen beskrivs. Till att börja med får man då reda på vad varje del har möjlighet att stödja vid systemutveckling och varför det kan vara bra att inkludera i en utvecklingsprocess. Med tanke på att synsättet innehåller så många olika komponenter kan det dock vara svårt att begripa hur en viss del kan vara till nytta för de övriga delarna. Det finns därför även ett avsnitt som beskriver hur de olika komponenterna möts. Dessa två avsnitt ger en överblick av hur processen kan gå till rent teoretiskt.

Min uppfattning var därför att det även krävdes ett praktiskt exempel som visar varje steg i utvecklingsprocessen och hur det ena steget hela tiden är en förutsättning för att kunna utföra nästa steg. Detta betyder dock inte att det är en ”allt-eller-inget”-metod; utan det finns även möjlighet att ta nytta av en eller ett par av de olika delarna på ett effektivt sätt som kan vara till stor fördel vid framtagande av ett system.

Vid utförandet av utvärderingsmodellen av alla verktyg så läste jag mycket om företagens egna löften om vad verktygen ska klara av att leverera. Många gånger upplevde jag det svårt att veta vad de jämför med och vad deras ord egentligen betyder. Det är enkelt att skriva att systemet ger en positiv kostnadsbild och att risken är mycket låg vid implementering med ett existerande system. Frågan är dock vad de jämför med och hur de mäter sina attribut som de lovar att de kan leverera. Många gånger kan man som kund uppleva att leverantören lovar för mycket och att man därför blir besviken på sin produkt. Min uppfattning är därför att det är viktigt att ha en balansgång av hur mycket man ska förespråka sin egen lösning.

Med hjälp av enkätundersökningen, egen erfarenhet samt information från de olika leverantörerna så har en utvärderingsmodell tagits fram. I utvärderingsmodellen har kommersiella produkter jämförts med Open source med avseende på funktionella och icke funktionella krav. De icke funktionella kraven inkluderar installation, pris, operativsystem, integrationsmöjligheter, riskmedvetenhet. I en översikt av alla verktyg har jag även bedömt hur bra funktionalitet varje enskilt verktyg har. Alla dessa attribut har fått en vikt som beskriver hur viktig den anses vara i förhållande till de övriga attributen. Då tanken är att vem som helst ska kunna använda modellen så är det också möjligt att ändra värdena på de olika vikterna och på så sätt få ut ett värde som passar den egna organisationen; tycker man exempelvis att risk är av större vikt så ökar man värdet på det kravet och minskar med samma värde på de andra kraven.

Metoden som användas för att skicka ut frågor och samla in svar till enkätundersökningen anses i efterhand inte ha uppnått det resultat som förväntades. Det ultimata resultatet hade varit att få in svar från ett trettiotal personer och även att svaren skulle ge mig en bild av hur alla de verktyg som

har analyserats i utvärderingsmodellen upplevs i avseende på riskmedvetenhet. För att uppnå detta resultat hade det krävts att jag utförde undersökningen i ett tidigare stadium samt att frågorna hade behandlats i ett elektroniskt formulär där det skulle vara mindre tidskrävande att svara. Då undersökningen inte kunde utföras på detta sätt av den enkla anledningen att jag började för sent så blev resultatet istället att jag fick svar från åtta olika personer som gav mig en bild av riskmedvetenhet på åtta av de arton verktyg som analyserats. Med hjälp av egen erfarenhet och insamlad information från företagens hemsidor och artiklar kunde jag själv bedöma riskmedvetenheten på de resterande verktygen på ett mindre övergripande sätt. De svar som samlades in kunde användas för att jämföra verktygen utifrån ytterligare en aspekt, men eftersom de var ett så litet urval som svarade var det svårt att dra några slutsatser utifrån dem.

6. Slutsats

Trots alla problem som uppstått på vägen så har jag lyckats få fram en uppsättning Open Source-verktyg för systemutveckling inom ALM-synsättet. Uppsättningen begränsades till kravhantering, testhantering, ärendehantering och design/bygg. Efter en omfattande installation och integration mellan verktygen kunde alla steg i processen utföras. Målet att sätta ihop en fullständig uppsättning Open Source verktyg för utveckling av informationssystem och mjukvara har därmed uppnåtts. Gällande dess funktionalitet och integrationsmöjligheter så visade jag genom utvärderingsmodellen att kommersiella verktyg inte skiljer sig så mycket ifrån de Open Source verktyg som analyserades. Det finns därför anledning för organisationer att byta till dessa verktyg för att dra ner på kostnaderna, speciellt gällande licenser. För en del kommer det krävas mer support vid byte från kommersiella verktyg till Open Source, men dessa kostnader kommer med största sannolikhet inte vara större än nuvarande licens-, underhålls- och supportkostnader. Modellens resultat visar exempelvis att det bästa enskilda verktyget är Subversion och att den bästa uppsättningen är Microsoft, där Open Source verktygens uppsättning inte var speciellt långt efter med sin andra plats.

I utvärderingsmodellen ville jag även jämföra verktygen utifrån ett antal icke-funktionella krav. Det mest invecklade kravet att bedöma utan att besitta tillräcklig kunskap om verktygen var dess riskmedvetenhet. Till att börja med funderade jag över vilka attribut som kan kopplas ihop med risk och som är möjliga att bedöma huruvida de fungerar tillfredsställande eller inte. De attribut som användes var användarvänlighet, tillförlitlighet, åtkomst, säkerhet, informationsförlust, spårbarhet och tillgänglighet. Informationen hämtades vidare ifrån verktygens leverantörers hemsidor och artiklar och även ifrån en enkätundersökning. Enkätundersökningen skulle komplettera materialet från leverantörerna genom att ge slutanvändarens syn på verktygen. Undersökningen gav mig dock såpass lite information tillbaka att jag anser att det inte går att dra några egentliga slutsatser utifrån aspekten risk. Dess innebörd blev dock inte helt irrelevant då jag genom det material som samlades in kunde visa att många av de verktyg som har analyserats är mycket komplexa och att många slutanvändare inte kan utnyttja dem fullt ut. De flesta verkar även börja nyttja verktygen utan att besitta den rätta kunskapen om dem. Det finns därför en stark antydan till att det krävs mer utbildning innan verktygen börjar användas för utveckling. Utbildning är en aspekt som jag inser att ha tagit för liten hänsyn till. En uppfattning är att detta är en aspekt som många inte är medvetna om, de utgår snarare ifrån att slutanvändaren lär sig verktygen efter hand vilket i många fall inte stämmer överens med verkligheten. De flesta är mer eller mindre medvetna om vilka risker det kan innebära att inte besitta den rätta kunskapen, men ser ändå inte ut att ta någon större hänsyn till detta. Det finns exempelvis personer som beskriver att bristande kunskap kan leda till stora säkerhetshål gällande åtkomsträttigheter vilket i sin tur bland annat kan leda till intrång och sabotage. Detta är naturligtvis en aspekt som företag bör ta på större allvar än vad den allmänna uppfattningen ser ut att vara idag.

7. Källor

Aharonovitz, M. (2006). *Three tips to improbe your requirements-based testing (RBT).* Article for software test and performance magazine 9.8.06

Alinconstantin.net – MSSCII. Hämtet från MSSCII: http://alinconstantin.members.winisp.net/webdocs/scc/MSSCCI.htm den 04 08 2008

Altcom AB. (2005). *Ärendehantering eFix*

Applicationsoftwaredeveloper.com. (2007). Hämtat från Applicationsoftwaredeveloper.com http://www.applicationsoftwaredeveloper.com/borland.html *den 04 08 2008*

Asklund, U. (1999). *Configuration Management for distributed development – practice and needs.* Lund: Lunds universitet

Asklund, U. & Bendix, L. (2002). *A study of configuration management in Open Source software projects*, Lund: Lunds universitet

Bach, J. (1999). *Risk and requirements-based testing.* IEEE computer society

Backman, J. (1998). *Rapporter och uppsatser.* Studentlitteratur

Bergvall, O. & Demblad, A. (2003). *Kravhantering: Viktiga faktorer att ta hänsyn till vid utveckling och utvärdering av kravspecifikationen. Luleå:* Luleå tekniska universitet

Borland.com - Open ALM for application development http://www.borland.com den 04 08 2008

Borland.com.tr - Borland Solutions for the Application Lifecycle Hämtat från Borland.com.tr: http://www.borland.com.tr/tr/products/alm/index.html den 02 05 2008

Borland. (2005). *Mitigating risk with effective requirements engineering.* Borland White paper

Borland. (2007b). *Open application lifecycle management (ALM): Unlocking the full value of managed software delivery.* Borland White paper

Borland. *(2007c). Project and portfolio management: Optimizing IT portfolio management, project selection, execution, and visibility.* Borland White paper

Björk, T. (2007). *Skillnaden mellan öppen och stängd källkod utifrån ett användarperspektiv.* Göteborgs Universitet

Björk, T. (2008). *– Application lifecycle management: metod och struktur för ett examensarbete.* Göteborgs Universitet

Brown, David William (2002). *An Introduction to Object-Oriented Analysis.* John Wiley & Sons

Callahan, J. R. & Khatsuriya, R. R. & Hefner, R. (1998). *Web-based issue tracking for large software projects.* IEEE Internet computing

Cardin, L. (2007). *The forrester wave: project portfolio management tools.* Forrester

Codeplex.com. Hämtat från Risk Visual Studio Team Foundation Server: http://www.codeplex.com/TFSGuide/Wiki/View.aspx?title=How%20To%20-%20Create%20a%20Risk%20Over%20Time%20Report%20for%20Visual%20Studio%20Team%20Foundation%20Server&referringTitle=Home den 04 08 2008

ComputerworldUK. (2007). Hämtat från The open ALM state. http://www.computerworlduk.com/technology/development/opinion/index.cfm?articleid=1028 den 08 02 2008

CMMI Product Team. (2002). *Capability maturity model integration (CMMI), version 1.1.* Carnegie Mellon Software engineering Institute

Componentsource.com. Hämtat från IBM Rational Requisite Pro: http://www.componentsource.com/products/ibm-rational-requisitepro/summary.html den 22 05 2008

Crnkovic, I. (1999). *Processing requirements by software configuration management.* Eskilstuna, Västerås: Mälardalen University

Crowther, M. (2007) *Test configuration management: ITIL based appoach to test asset configuration management*, Discussion document

Dahmström, K. (1991). *Från datainsamling till rapport: att göra en statistisk undersökning.* Studentlitteratur

Dietze, S. (2005)- *Agile Requirements Definition for Software Improvement and Maintenance in Open Source Software Development.* Tyskland, Bonn: Fraunhofer Institute

El-Najjar, D. (2006). *Ärendehanteringssystem – Brister och anpassningar i ett befintligt system.* Sundsvall: Mittuniversitetet

Fernandez, E. B. & Wu, J. & Qian, H. (1994). *A combined functional and object-oriented approach to software design.* USA, Florida: Atlantic University

Finkelstein, A. & Emmerich, W. (2000). *The future of requirements management tools.* England, London: University College

Fitzgerald, D. (2005). *Everything you ever wanted to know about issue management,* Knowth-consulting

Fogel, K. (2005). *Producing Open Source software: How to run a successful free software project.* CreativeCommons Attribution-ShareAlike license

Freshmeat.net. (2000). Hämtat från Bug testing: http://freshmeat.net/articles/view/184/ den 11 02 2008

Gareis, R. (2002). *Professional project portfolio management.* Roland Gareis Consulting, presented at the IPMA World Congress

Guo, M. (1997). *Automatic transformation from data flow diagram to structure chart.* Software Engineering notes vol 22

Göhlman, D. & Alexander, I. & Taborda, J. (2002). *Ta kontroll över kraven.* Telelogic AB

House of Representatives. (1999). *System Development Life-Cycle Policy*

Hughes, K. (2007). *Successful Portfolio Management.* Collegiate project services

Hägglund, A. (okänt årtal). *Introduktion till Capability Maturity Model (CMM).* Systemvaruhuset

Johnsson, M. (2002). *Bra kravhantering viktigast.* Elektronik i Norden

Korpiaho, K. (2007). *Project portfolio management in a R&D organization.* Finland, Helsingfors: Helsinki University

Kumar, S. (okänt årtal). *Expanding Dimensions of Test Management for Object Oriented Software.* Zenar technologies

Freshmeat.net. (2000). Hämtat från software testing: http://freshmeat.net/articles/view/147/ den 08 02 2008

Mayfield, K. (2005). *Design methodogy.* eSolutions

Microsoft.com. Hämtat från Visual Studio Team System: http://msdn.microsoft.com/en-us/vsts2008/products/bb933749.aspx den 22 05 2008

Noor I. (2001). *Issue and Risk Management- Principles and Practice.* PMI Southern Caribbean Chapter

Patel, R. & Davidson, B. (1991). *Forskningsmetodikens grunder: Att planera, genomföra och rapportera en undersökning*, andra upplagan, Studentlitteratur

Pavlo, A. & Couvares, P- & Gietzel, R. & Karp, A. & Alderman, I. D. & Livny, M. & Bacon, C. (2006). *The NMI build & test laboratory: continuous integration framework for distributed computing software.* 20th large installation system administration conference (LISA '06)

Pinto, S. (2006). *Requirement and risk based testing: A new approach to ensure user's satisfaction.* AtYourSide Consulting

Ramakrishnan, R. & Ram, D. J. (1996). *Modeling design versions.* Indien: Indian Instisute of Technology

Raymond, E. S. (2001). *Katedralen och basaren: En oavsiktlig revolutionärs tankar kring Linux och öppen källkod.* Nya Doxa

Reed, P. (2002). *Transitioning from requirements to design.* Jackson-Reed Inc.

Scacchi, W. (2001). *Understanding the requirements for developing Open Source software systems.* USA, Kalifornien: University of California

Scacchi, W. & Jensen, C. & Noll, J. & Elliot, M. (2005). *Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes.* USA, Kalifornien: Santa Clara University

Schwaber, C. & Rymer, J. R. & Stone, J. (2006). *The changing feace of application lifecycle management*, Forrester

Signifikant.se. Hämtat från ALM:
http://www.signifikant.se/en/alm_benefits.php den 22 05 08

Smith, L. W. (2003). *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems*, Department of the Air Force – Software Technology Support Center

Sommerville, I. (2004). *Software engineering 7th edition,* Addison Wesley

Spectrum-systems.com. – Hämtat från Telelogic:
http://www.spectrum-systems.com/vendors/telelogic/dashboard_0606.pdf den 22 05 08

Starrin, B. & Svensson, P.-G. (red.) (1994). *Kvalitativ metod och vetenskapsteori,* Studentlitteratur

Statskontoret. (1997). *Handbok i IT-säkerhet: Del 1*

TechExcel.com. – Hämtat från ALM:
http://www.techexcel.com/solutions/alm/ den 31 01 2008

van Veenendaal, E. & Pol, M. (1997). *A test management approach for structured testing.* Den Bosch

Freshmeat.net. (2004) – Hämtat från Open Source testing tools:
http://freshmeat.net/articles/view/1369 den 08 02 2008

Waldo, J. (2006). *On system design.* Sun labs

Weber, S. (2000). *The political Economy of Open Source Software.* USA: University of Berkeley

Weber, S. (2004). *The success of Open Source.* USA: Harvard university

Whittaker, J. A. (2000). *What is software testing? And why is it so hard?*. USA: Florida Institute of Technology

Wiegers, K.E. (1999). *Automating Requirements Management*, Process impact

Zwartjes, G. & van Geffen, J. (2005). *An agile approach supported by a tool environment for the development of softtware components.* Holland: Technische Universiteit Eindhoven

7.1 Källor till utvärderingsmodellen

Amazon.com – Hämtat från Visual studio team system: http://www.amazon.com/Microsoft-Visual-Foundation-Additional-License/dp/system-requirements/B000WM04HA den 22 05 2008

Bitpipe.com – Hämtat från Telelogic Synergy: http://www.bitpipe.com/detail/RES/1113929179_847.html den 22 05 2008

Borland.com – Hämtat från Borland CaliberRM: http://www.borland.com/resources/en/pdf/products/caliber/caliber_family_faq.pdf den 22 05 2008

Borland.com – Hämtat från Borland CaliberRM: http://www.borland.com/resources/en/pdf/products/caliber/caliber_family_datasheet.pdf den 22 05 2008

Borland.com – Hämtat från Borland Starteam: http://www.borland.com/us/company/news/press_releases/2005/07_25_05_new_version_of_borland_starteam.html den 22 05 2008

Borland.com – Hämtat från Borland Silkcentral test: http://www.borland.com/resources/en/pdf/products/silk/silkcentral_test_datasheet.pdf den 22 05 2008

Borland.com – Hämtat från Borland Silkcentral test: http://www.borland.com/resources/en/pdf/products/silk/silkcentral_test_mgr_faq.pdf den 22 05 2008

Borland.com – Hämtat från Borland Silkcentral issue: http://www.borland.com/resources/en/pdf/products/silk/silkcentral_issue_datasheet.pdf den 22 05 2008

Borland.com – Hämtat från Borland issue: http://www.borland.cz/products/silk/silkcentral_issue/index.html den 22 05 2008

Borland.com – Hämtat från Borland CaliberRM: http://www.borland.com/resources/en/pdf/services/consulting/caliberrm_quickstart.pdf den 22 05 2008

Borland.com – Hämtat från Borland ALM: http://www.borland.com.tr/tr/products/alm/index.html den 22 05 2008

Borland.com – Hämtat från Borland Risk: http://www.borland.com/resources/en/pdf/white_papers/mitigating_risk_with_effective_requirements_engineering.pdf den 22 05 2008

Borland.com – Hämtat från Borland Silkcentral test: http://www.borland.com/resources/en/pdf/products/silk/silkcentral_test_mgr_faq.pdf den 22 05 2008

CDW.com – Hämtat från IBM clearcase:

http://www.cdwg.com/shop/products/default.aspx?EDC=767902 den 22 05 2008

CDW.com – Hämtat från IBM Software Architect:
http://www.cdw.com/shop/products/default.aspx?EDC=758384 den 22 05 2008

Eclipse.org – Hämtat från Eclipse:
http://www.eclipse.org/downloads/ den 22 05 2008

Eclipse.org – Hämtat från Eclipse help:
http://help.eclipse.org/help33/index.jsp den 22 05 2008

IBM.com – Hämtat från IBM Requisite Pro:
http://download.boulder.ibm.com/ibmdl/pub/software/rational/web/datasheets/version6/reqpro.pdf
den 22 05 2008

IBM.com – Hämtat från IBM Clearcase:
http://www-1.ibm.com/support/docview.wss?rs=984&uid=swg21155708 den 22 05 2008

IBM.com – Hämtat från IBM Requisire Pro:
http://www-306.ibm.com/software/awdtools/reqpro/features/ den 22 05 2008

J solutions.se – Hämtat från Clearcase plugin Eclipse:
http://jsolutions.se/?p=147 den 22 05 2008

Microsoft.com – Hämtat från Microsoft Visual studio:
http://support.microsoft.com/kb/321434 den 22 05 2008

Microsoft.com – Hämtat från Visual studio:
http://support.microsoft.com/kb/323302 den 22 05 2008

Microsoft.com – Hämtat från Visual studio:
http://msdn.microsoft.com/en-us/vsts2008/products/bb933749.aspx den 22 05 2008

Microsoft.com – Hämtat från Visual studio:
http://www.microsoft.com/downloads/details.aspx?FamilyID=F5539A90-DC41-4792-8EF8-F4DE62FF1E81&displaylang=en den 22 05 2008

Moonsoft.net – Hämtat från Borland CaliberRM:
http://www.moonsoft.net/products/000054.aspx den 22 05 2008

Moonsoft.net – Hämtat från Borland Starteam:
http://www.moonsoft.net/products/000085.aspx den 22 05 2008

Moonsoft.net – Hämtat från Borland Together:
http://www.moonsoft.net/products/000089.aspx den 22 05 2008

Mozilla.org – Hämtat från Bugzilla:
http://wiki.mozilla.org/Bugzilla:FAQ:Managerial_Questions den 22 05 2008

Mozilla.org – Hämtat från Bugzilla:

http://www.bugzilla.org/docs/tip/html/security.html den 22 05 2008

Newsdesk.se – Hämtat från Telelogic plugin för Microsoft Visual studio:
http://www.newsdesk.se/view/pressrelease/16809 den 22 05 2008

Ostatic.com – Hämtat från Open Source Requirement Management Tool (OSRMT):
http://ostatic.com/50589-software-opensource/osrmt den 22 05 2008

Paper-review.com – Hämtat från IBM Requisite Pro:
http://www.paper-review.com/tools/rms/response.php?vendor=IBM%20Rational%20RequisitePro%20(updated%2010%20Oct%2006) den 22 05 2008

Pricegrabber.com – Hämtat från Microsoft Viual studio:
http://software.pricegrabber.com/misc-programming/m/13954430/ den 22 05 2008

Regdeveloper.co.uk – Hämtat från Subversion:
http://www.regdeveloper.co.uk/2006/04/21/risk_opensource_subversion/ den 22 05 2008

Sourceforge.net – Hämtat från Testlink:
http://sourceforge.net/projects/testlink/ den 22 05 2008

Spectrum-systems.com – Hämtat från Telelogic:
http://www.spectrum-systems.com/vendors/telelogic/dashboard_0606.pdf den 22 05 2008

Subversion.tigris.org – Hämtat från Subversion:
http://subversion.tigris.org/faq.html#portability den 22 05 2008

Subversion.tigris.org – Hämtat från Subversion:
http://svnbook.red-bean.com/en/1.4/svn-book.pdf den 22 05 2008

Telelogic.com – Hämtat från Telelogic:
http://www.telelogic.com/Company/events/index.cfm?showdetail=y&ID=2370&EventType=Seminar den 22 05 2008

Telelogic.com – Hämtat från Telelogic:
http://www.telelogic.com/download/get_file.cfm?id=3726 den 22 05 2008

Trac.edgewall.org – Hämtat från Trac:
http://trac.edgewall.org/wiki/TracFaq den 22 05 2008

8. Appendix

Appendix 1: Installation och integration

Innan du börjar

Det finns alltid risk att det uppstår fel när de olika delarna läggs in för att senare kopplas ihop. En rekommendation är därför att alltid kopiera alla beroende mappar och placera dem på ett oberoende ställe så att du kan återgå till den senaste förändringen, om något oväntat händer när man experimenterar.

Praktiska delen

ECLIPSE

Hemsida: http://www.eclipse.org

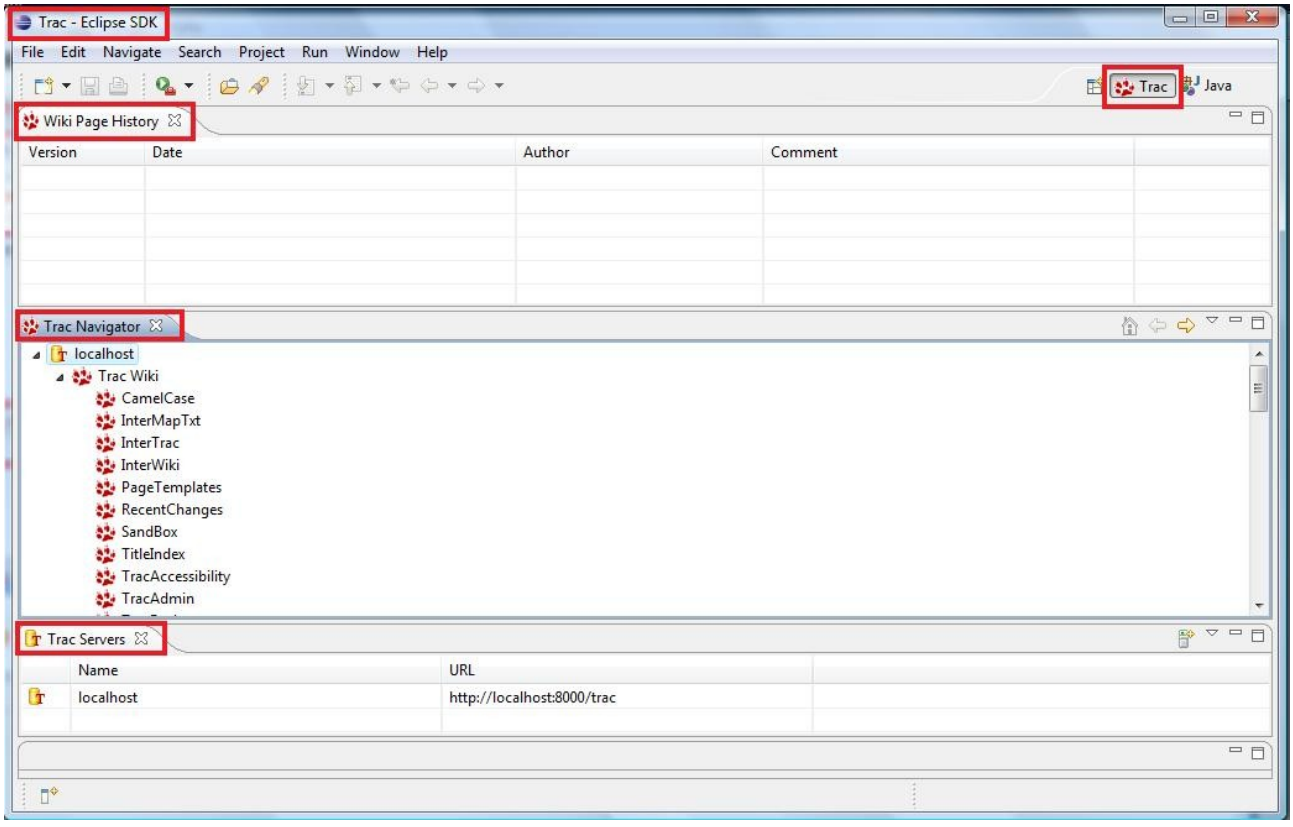
Lägga in nya plugins i eclipse

help > software updates > find and install > search for new features to install >

New Remote site

Eclipse och trac

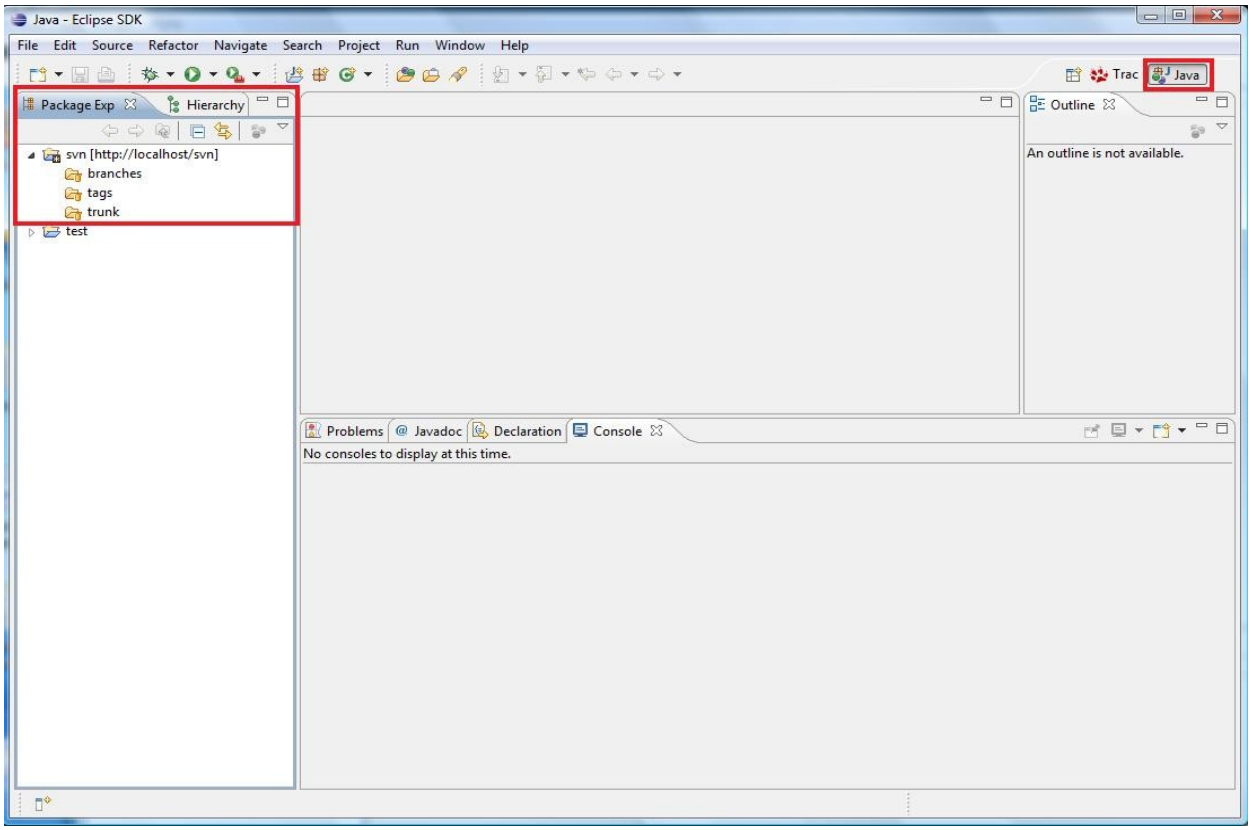
namn: eclipsetrac
Hemsida: http://trac-hacks.org/svn/eclipsetracplugin/eclipse/update/



***Figur 33**: TRAC i Eclipse*

Subversion och eclipse

namn: subclipse 1.2
Hemsida: http://subclipse.tigris.org/update_1.0.x



***Figur 34**: Subversion i Eclipse*

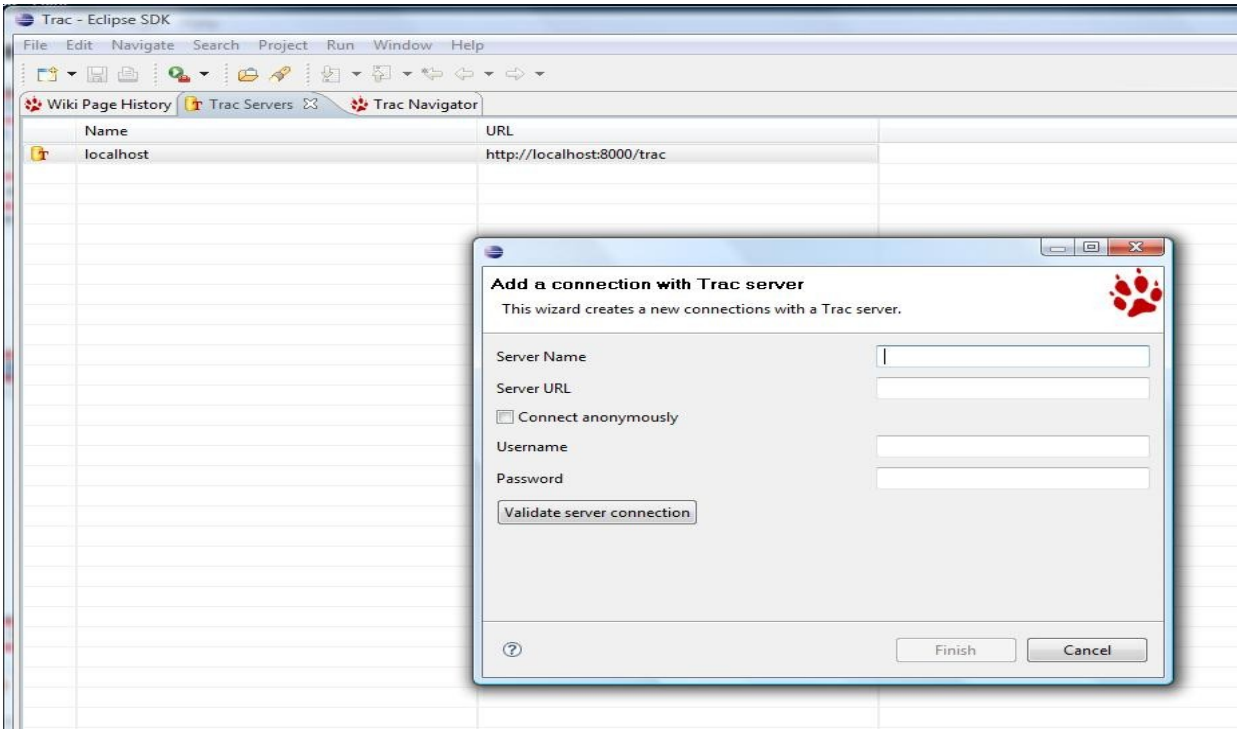
Ett plugin till eclipse som bland annat stödjer Bugzilla och TRAC

MYLYN
namn:Mylyn
http://download.eclipse.org/tools/mylyn/update/e3.3

Skapa tracserver i Eclipse

servername: localhost
server URL: http://localhost:8000/trac
username: tobj
password: help

validate server connection



***Figur 35**: Skapa TRAC-server i Eclipse*

TRAC

Hemsida: http://trac.edgewall.org
Guide: http://trac.edgewall.org/wiki/TracOnWindows

För att installera Trac, så behöver du följande paket:

- Python, version 2.3.5. (http://www.python.org/)
- Subversion, version 1.5 (http://subversion.tigris.org/)
    - Python bindings for Subversion, version 1.4.6 (måste stämma överens med både Python och Subversions version) (http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=91&filter=py)
- Apache, version 2.0.63 (http://www.apache.org/)
- SilverCity>=0.9.4 (http://silvercity.sourceforge.net/)
- Docutils>=0.3 (http://docutils.sourceforge.net/docs/dev/repository.html)
- Clearsilver, version 0.9.14 (http://www.clearsilver.net/downloads/win32/clearsilver-0.9.14.win32-py2.3.exe)
- PySQLite, version 1.1.6 (http://initd.org/pub/software/pysqlite/releases/1.1/1.1.6/pysqlite-1.1.6.win32-py2.3.exe)
- Trac, version 0.11b1 (http://trac.edgewall.org/wiki/TracDownload)
- Testlink 1.7.4 (http://sourceforge.net/project/showfiles.php?group_id=90976)
- genshi version 0.4.4 (http://ftp.edgewall.com/pub/genshi/Genshi-0.4.4.zip)
- http://pypi.python.org/pypi/setuptools/#id4 setuptools 0.6c8 (lägger in scriptet easy_install i c:\python\scripts)
- xmlrpcplugin (http://trac-hacks.swapoff.org/wiki/XmlRpcPlugin/) (tanka först hem .zip-filen och zippa upp, gör sedan: easy_install -Z C:\program files\python\scripts\xmlplugin\0.10)
- eclipsetracplugin.tracrpvcext-0.10 (http://trac-hacks.org/attachment/wiki/EclipseTracPlugin/EclipseTrac.zip) (tanka först hem .zip-filen och zippa upp, gör sedan: easy install)
- mod python-3.3.1.win32-py2.3-Apache2.0.exe (http://ftp.solace.miun.se/pub/apache/httpd/modpython/win/3.3.1/)

Xmlrpc

XmlRpc är ett plugin som TRAC måste ha för att kunna hantera buggar tillsammans med Testlink. Det uppstod en del problem med detta plugin och jag är inte helt säker på att min installation blev korrekt. Nedan finns det kommando som jag använde för att pluginet skulle dyka upp i TRAC. Innan detta kommando kan utföras så måste ett skript som är kopplat till Python aktiveras. TRAC bygger mycket på paket och integrationer.

easy_install -Z C:\xmlplugin\0.10

C:\trac\conf\trac.ini

infoga följande:

[components]
tracrpc.* = enabled
tracrpcext.* = enabled

Auth till tracservern

Om inte trac-digest.py finns så använd apache/bin/htdigest
I en kommandoprompt:

C:\mapp för apache\bin\
htdigest -c digest.txt realm användarnamn, exempelvis:
htdigest -c digest.txt trac tobj
Sedan får man mata in önskat lösenord.

Skapa en admin i TRAC

Trac-admin c:\trac permission add tobj TRAC_ADMIN

Kommando för att starta auth på servern

Gå in i c:/program files/python/scripts
skapa en fil som heter tracserver.bat
redigera den med följande kommando:
tracd -p 8080 --auth=trac,"c:\program Files\Apache Group\Apache2\bin\digest.txt",trac c:\trac

Det är noga att du använder dig av –auth-argumentet, för om du inte gör det så kommer du få problem med att logga in i TRAC.

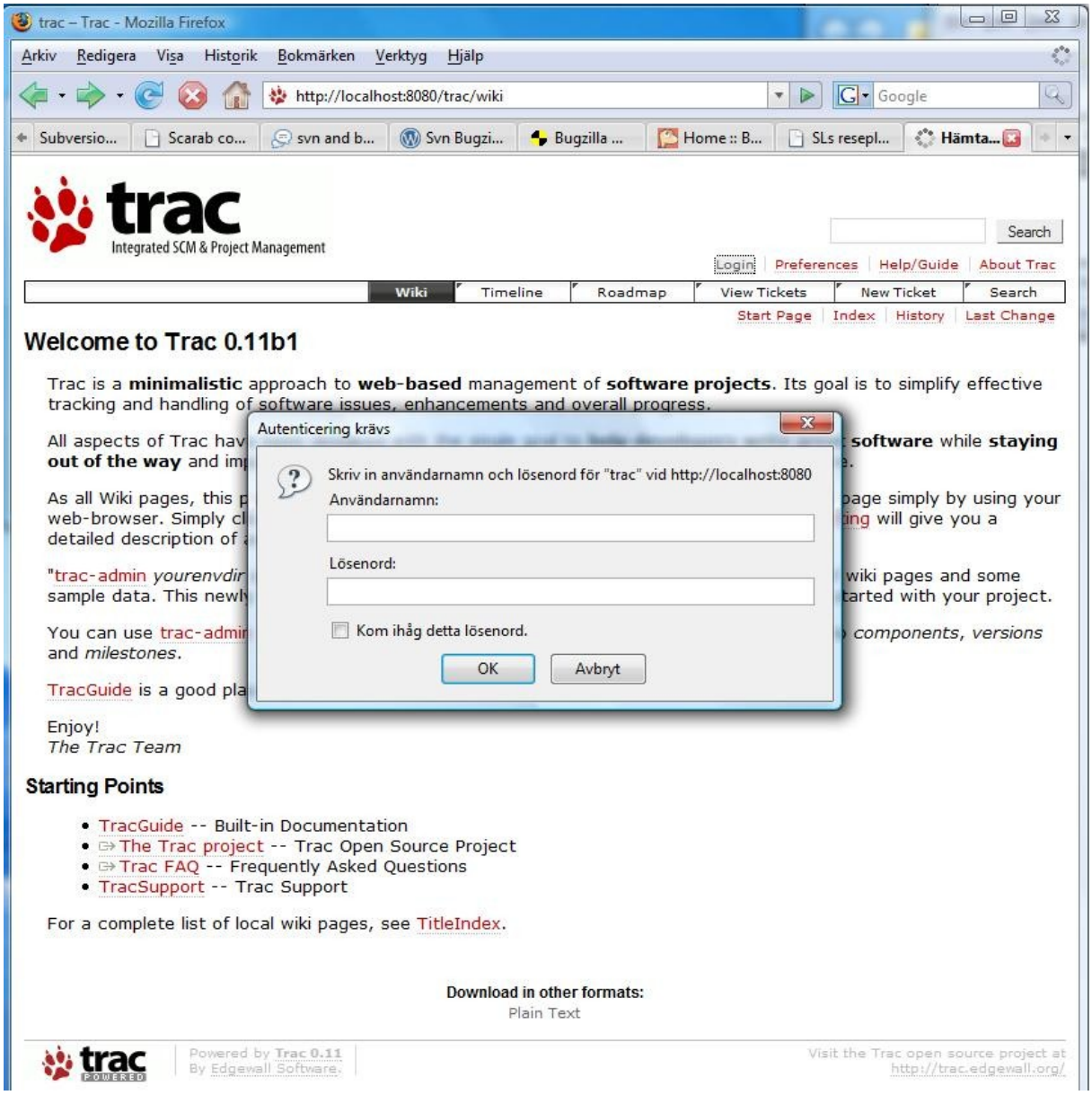
Webbgränssnitt

Precis som de flesta andra programmen som har undersökts, så bygger TRAC på ett webbgränssnitt. I detta gränssnitt finns det möjlighet att ställa in vilka rättigheter olika användare ska ha. Även en person som inte är inloggad har viss möjlighet att navigera i systemet. Syftet med att ge rättigheter till en ”utomstående” är att alla exempelvis ska kunna ta del av inrapporterade ärenden och läsa projektets manual (wiki).

Trac kan nås via
http://localhost:8080/trac

Logga in via webbgränssnittet med följande uppgifter:

användarnamn: tobj
lösenord: help

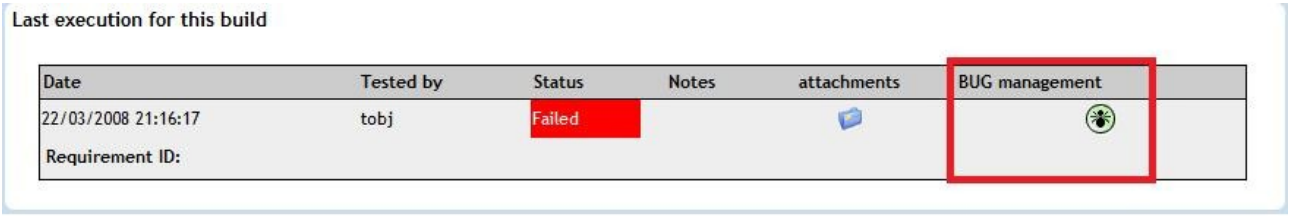


***Figur 36**: Inloggning i TRAC*

TRAC i Testlink

Lägg till följande i **custom_config.inc.php** (som ligger i roten av din testlink-katalog):
$g_interface_bugs='TRAC';
define ('TL_INTERFACE_BUGS', 'TRAC');
define('BUG_TRACK_DB_HOST', 'http://localhost:8080/trac');

om du använder custom_config.inc.php är det viktigt att du kommenterar bort define('BUG_TRACK_DB_HOST') och //define('BUG_TRACK_HREF') i /cfg/trac.cfg.php



Figur 37: *Lägga till en bugg i Testlink via TRAC*

Subversion

Hemsida: http://subversion.tigris.org

Infoga i httpd.conf:

Httpd.conf är Apaches konfigurationsfil som finns tillgänglig i /conf i din Apache-mapp. Då apache är det program som styr localhost (den personliga servern), så är det i denna fil som förutsättningarna för Subversion ska infogas. Kopiera texten och infoga den längst ner i filen, så att förändringen enkelt kan lokaliseras om något fel uppstår. Efter att texten har infogats måste du spara förändringarna och starta om Apache.

<Location /svn>
DAV svn
SVNPath C:\myproj

#SVNListParentPath on
#SVNParentPath c:\myproj
#SVNListParentPath on #visa index

AuthType Basic
AuthName "myproj"
AuthUserFile passwd
#AuthzSVNAccessFile svnaccessfile
Require valid-user
</Location>

Apache, PHP och MYSQL

Apache, PHP och MYSQL har tidigare kopplats samman per automatik, men i takt med att brandväggar skapade problem kring detta så har Apache valt att funktionaliteten måste infogas manuellt. Precis som för Subversion så behöver du infoga en del text Apaches konfigurationsfil httpd.conf. Även här krävs omstart av Apache för att ändringarna ska bli aktiva.

Infoga i httpd.conf:

LoadModule php4_module php/sapi/php4apache2.dll
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps

Infoga följande i PHP.ini:

PHP.ini är PHPs konfigurationsfil, den finns i roten av PHP-mappen. Infoga nedanstående och starta sedan om Apache

[PHP_MYSQL]
extension= php_mysql.dll
[PHP_MYSQLI]
extension= php_mysqli.dll

- Se till så att även PHP.ini pekar på rätt mapp, exempelvis:
  extension_dir = "c:/php/ext/"
- Se till så att dessa filer finns på följande ställen:
  php_mysql.dll ska finnas i din phpmapp, exempelvis c:/php/ext/
  libmySQL.dll ska finnas i system32 mappen, vilken brukar finnas här:
  c:/windows/system32/
- Skapa en fil som heter test.php
  Mata in koden nedan för att se om PHP fungerar
  <?
  echo '
 php running';
  $dbcnx = mysql_connect('localhost');
  echo '
 connection ran OK';
  ?>
- Lägg testfilen i apache/htdocs-mappen
- Skriv localhost/test.php i en browser för att köra filen

Det kan även vara nödvändtigt att kopiera php.ini till roten av din Apache-mapp, exempelvis C:\Program Files\Apache Group\Apache2

Skapa en förvaringsplats (repository)

För att enkelt navigera sig i Subversions databas så kan man skapa en förvaringsplats på servern eller din egna dator där alla filer och ändringar finns tillgängliga:

- svnadmin create c:/myproj

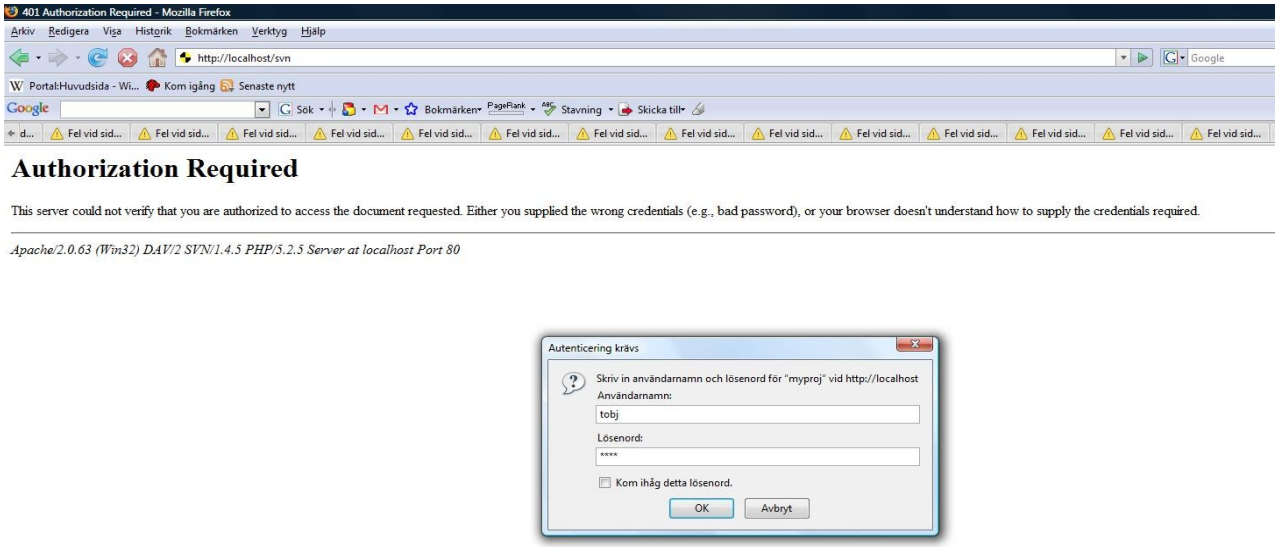
Webbgränssnitt

Det finns även möjlighet att nå förvaringsplatsen (repository) via ett webbgränssnitt. Gränssnittets syfte är att samtliga projektmedlemmar ska kunna ta del av det material som finns på förvaringsplatsen, och på så sätt alltid vaar uppdaterade med de senaste förändringarna.

Subversion webbgränssnitt nås via

http://localhost/svn

Logga in via webbgränssnittet med följande uppgifter:
användarnamn: tobj
lösenord: help



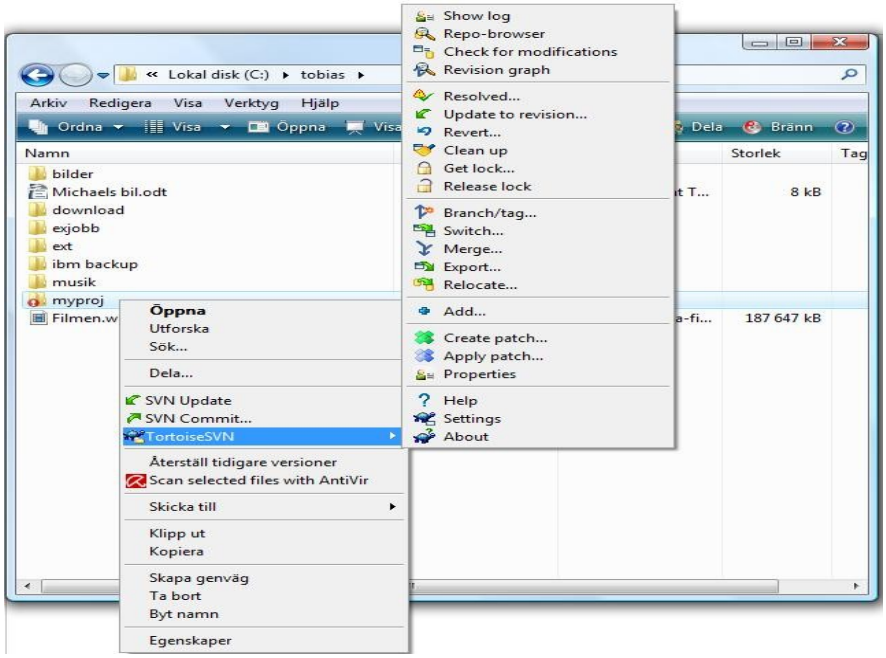
***Figur 38**: Inloggning Subversion*

TortoiseSVN

Hemsida: http://tortoisesvn.tigris.org/

TortoiseSVN är ett gränssnitt som förenklar användandet av Subversion betydligt eftersom man inte behöver skriva in några kommandon utan kan utföra dem genom musens högerklick-meny. Nedan följer ett exempel på hur man koppla ihop en mapp med förvaringsplatsen (repository):

- Gå till en annan mapp (exempelvis c:\tobias\myproj)
- Skapa mapparna branches, trunk och tags
- Högerklicka SVNtortoise > import och placera dem sedan i c:\myproj



***Figur 39**: TortoiseSVN*

Testlink

Hemsida: http://www.teamst.org

Tillvägagångssätt för att få tillgång till Testlink på localhost

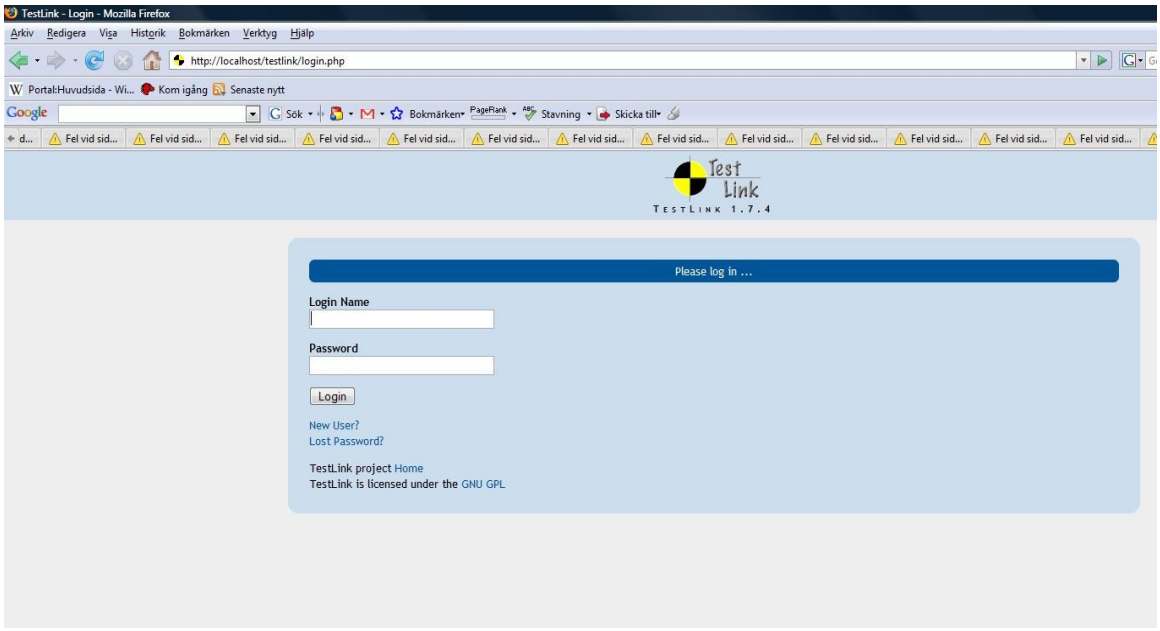
Lägg hela testlink-mappen i apache/htdocs
då hamnar servern på /localhost/testlink
skriv sedan in /localhost/testlink/installation/index.php

Database type: mysql
database host: localhost
database name: testlink
database login: root
database password:
testlink db login: tobj
testlink db password: help

sedan fyller man i
name: admin
password: admin
Vidare är det för säkerheten smart att ändra adminlösenordet, eller skapa en ny användare och ta bort admin-kontot.

Testlink rekommenderar även att du tar bort mappen ”install” som ligger i roten av testlink, exempelvis C:\Program Files\Apache Group\Apache2\htdocs\testlink\

Inloggning sker via följande adress
http://localhost/testlink/index.php



***Figur 40**: Inloggning Testlink*

Bugzilla

Hemsida: http://www.bugzilla.org/
Guide: http://www.bugzilla.org/docs/win32install.html

Tillvägagångssätt för att få tillgång till Bugzilla på localhost

- Lägg hela bugzilla-mappen i apache/htdocs
- då hamnar servern på /localhost/bugzilla

Kräver följande

Perl 5.8.1 or Higher MSI (http://www.activestate.com/store/activeperl/download/)
(tänk på att du inte får lägga programmet i en mapp som innehåller mellanslag, exempelvis ”program files”)
För att installera alla paket som krävs i bugzilla så måste man installera Perl. I Perl finns det en pakethanterare som kallas PPM. Nedan beskrivs hur den används.

PPM GUI

Guide:http://aspn.activestate.com/ASPN/docs/ActivePerl/5.8/faq/ActivePerl-faq2.html

Gå in i perl/bin/ppm.bat för att öppna programmet

Gå in i Edit->Preferences, välj sedan Repositoties, mata in valfritt namn i fältet ”Name” och skriv i adressen i fältet ”Location”:
http://ppm4.activestate.com/MSWin32-x86/5.10/1002/package.xml
http://landfill.bugzilla.org/ppm/
http://www.apache.org/dist/perl/win32-bin/ppms/
http://www.bribes.org/perl/ppm/
http://ppm.tcool.org/archives/package.xml
http://trouchelle.com/ppm/
http://trouchelle.com/ppm10/package.xml
http://openinteract.sourceforge.net/ppmpackages/
http://theoryx5.uwinnipeg.ca/ppms/
http://cpan.uwinnipeg.ca/PPMPackages/10xx/

starta sedan om PPM för att få alla uppdateringar

För att se alla paket så tryck på View->All packages
Vilka paket som ska vara installerade är enkelt att följa i guiden.

PPM-Shell

Skulle du vilja göra dina paket-installationer manuellt, som de gör i guiden så gör så här:
För att starta ppm gå in i c:\plats\perl\bin
för att öppna ppm i kommandotolken så skriver du ppm-shell

För att skapa en repository (ett ställe som du hämtar paket ifrån):
skriv följande ppm rep add trouchelle.com

Istället för att ändra documentroot till c:\bugzilla så placera hela bugzillamappen i apache/htdocs (precis som vi gjorde med Testlink)

Efter paketinstallationen

När alla paket är inlagda ska man skriva in **checksetup.pl** för att se så att alla paket verkligen finns med. Finns inte alla med så får du en specifikation över vilka som saknas. Finns alla med så börjar den skapa alla tabeller i databasen. Vidare skapas en fil som heter localconfig i roten av bugzilla. Mata in följande uppgifter:

```
$db_host = "localhost";      # where is the database?
$db_port = 3306;             # which port to use
$db_name = "bugs";           # name of the MySQL database
$db_user = "root";           # user to attach to the MySQL database
#
# Enter your database password here. It's normally advisable to specify
# a password for your bugzilla database user.
# If you use apostrophe (') or a backslash (\) in your password, you'll
# need to escape it by preceding it with a \ character. (\') or (\\)
#
$db_pass = '';
```

Manuell installation via hemsida

Skulle det vara något paket som saknas även fast man matat in alla repositoryn ovan, så brukar det gå ganska enkelt att hitta den senaste versionen av just denna fil genom att göra en Google-sökning.

Här ett exempel på hur det skulle kunna gå till:

- Skriv in GD.ppd i Googles sökfunktion
- Gå in på en sökträff (http://www.bribes.org/perl/ppm/) och installera de filer som saknas genom att mata in ppm install sökväg/filnamn.ppd
- exempelvis *ppd install http://www.bribes.org/perl/ppm/GD.ppd*

Det kan också vara viktigt i vilken ordning man installerar paketen eftersom de är bundna till varandra.

Får man följande fel när man kör perl checksetup.pl

Undefined subroutine &Bugzilla::Util::ThrowTemplateError called at
Bugzilla/Util.pm line 498.

Så beror det på att bugzilla har skapat två mappar som heter bugzilla
i mitt fall (c:\program files\bugzilla) samt (c:\program files\bugzilla\Bugzilla)
kopiera isåfall allt innehåll i (c:\program files\bugzilla\template\) till (C:\Program Files\bugzilla\Bugzilla\template)

Får du följande fel när du går in på localhost/bugzilla

Internal Server Error
The server encountered an internal error or misconfiguration and was
unable to complete your request.

Så beror det på att du inte har redigerat i Start -> kör -> regedit
HKEY_CLASSES_ROOT\.cgi\Shell\ExecCGI\Command
key :- C:\Perl\bin\perl.exe -T
Finns inte HKEY_CLASSES_ROOT\.cgi så får du ska den genom att ha

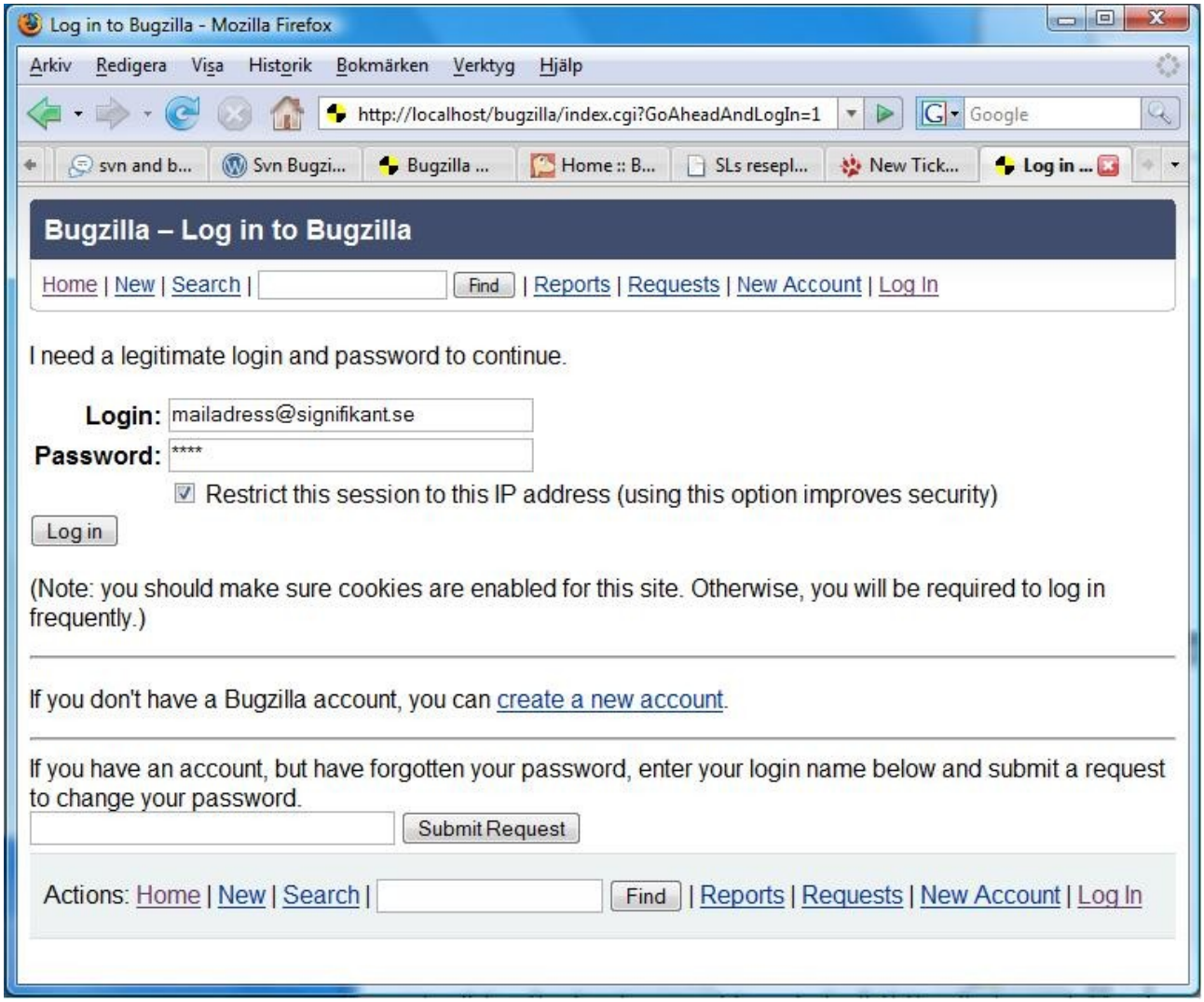
HKEY_CLASSES_ROOT markerad och högerklicka->Nytt->Nyckel och sedan skapa alla undermappar (\Shell\ExecCGI\Command). Vidare klickar du på filen standard och fyller i *key :- C:\Perl\bin\perl.exe -T* eller där du lagt din Perlkatalog

Inloggning och inställningar

väl inne på /localhost/bugzilla så loggar man in med den emailadress och det lösenord man angav vid installationen

exempel:
användarnamn: mailadress@signifikant.se
lösenord: help

Scrolla ner på sidan och gå in på parameters.
Där måste maintainer justeras till en emailadress: mailadress@signifikant.se
URLbase måste ändras till projekts huvudsida (i mitt fall http://localhost/bugzilla). Använd dock inte localhost om det finns en annan adress.



Figur 41*: Inloggning Bugzilla*

Skapa ett projekt

I menyn längst ner finns en flik som heter Products, klicka på den
Klicka sedan på ”Add a product”
skriv i önskat produktnamn och en beskrivning
Vidare måste du skapa en komponent genom att klicka på ”add at least one component”
döp komponenten och gör en beskrivning om så önskas.
Default Assignee syftar på produktens ägare och där ska en mailadressen som används vid inloggning användas.
Defaul CC List Syftar till de personer som också är relaterade till projektet, vilka ska presenteras via email och listas genom att sätta ett kommatecken emellan

Bugzilla Eclipse

Guide: http://wiki.eclipse.org/index.php/Mylar_User_Guide#Bugzilla_Connector

Ett plugin som möjliggör buggrapportering direkt i Eclipse, och kan infogas på följande sett

- Help --> Report Bug or Enhancement...
- En lista dyker upp där bland annat Bugzilla finns med, om Mylyn installerats på rätt sätt Beskrivning av hur Mylyn infogas finns beskrivet i avsnittet om Eclipse
- Vidare fyller man i de uppgifter som är relevanta om felet eller buggen
- Utförs detta på korrekt sätt kommer en sammanfattning av buggen att visas

Bugzilla Testlink

- För att få Bugzilla att fungera med testlink så måste man ha en databas, vilket du bör ha skapat redan för att kunna skapa några fel/buggar.
- I guiden som du hittar på http://www.bugzilla.org/docs/win32install.html finns detta beskrivet.
- Vidare rekommenderas att använda custom_config.inc.php för att ställa in de variabler som krävs för att integrera med testlink:

```
$g_interface_bugs='BUGZILLA';
define ('TL_INTERFACE_BUGS', 'BUGZILLA');
define('BUG_TRACK_DB_HOST', '127.0.0.1');
define('BUG_TRACK_DB_NAME', 'bugs');
define('BUG_TRACK_DB_USER', 'root');
define('BUG_TRACK_DB_PASS', '');
define('BUG_TRACK_HREF', "http://localhost/bugzilla/show_bug.cgi?id=");
define('BUG_TRACK_ENTER_BUG_HREF',"http://127.0.0.1:80/bugzilla/");
```

Tänk på att användarnamn och lösenord inte är de som du ställt in i din bugzilla, utan är root-lösenordet till MYSQL. Jag hade stora problem med att ändra dessa uppgifter (med tanke säkerhetsaspekter), så testa alltid med root-kontot först och ändra detta när du ser att det fungerar.