

Spårbarhet av användbarhetskrav

Examensarbete i Teknisk Datavetenskap, 20 p.

Peter Haraldsson
pops@cs.umu.se

Abstract

Usability has become more and more important in software development today. Not only should the final product be able of solving specific problems, it should also be satisfactory to the user.

The problem at hand is to unite usability goals, given in qualitative terms, and functional requirements, given in quantitative terms, and make them traceable throughout the development process. This is more or less manageable depending on the scope of the development process.

Therefore the purpose of this thesis is to find a way to trace usability goals through systems developed with *Rational Unified Process*. The major issue is the conflict between usability's overall perspective and *RUP*'s detailed perspective.

Usability and *RUP* are explained in this thesis and different methods of traceability are examined to find some that are applicable to the problem. Finally this thesis describes the outline of a solution that creates traceable usability goals, and works just as well for any non-functional requirements, within *RUP* as a development process.

Innehållsförteckning

Inledning	7
Problembeskrivning	7
Syfte	7
Riksskatteverket IT	7
Användbarhet	11
Rational Unified Process - RUP	13
En användningsfallsdriven utveckling.....	14
Struktur	15
Roll, Aktivitet, Artefakt och Arbetsflöden	16
Roller – Ett exempel: User Interface Designer.....	17
Arbetsflöden – Ett exempel: Kravhantering.....	18
KUR.....	21
Spårbarhet	23
Definition	23
Pre-KS- & Post-KS-spårbarhet.....	24
Metod.....	27
Fallstudie.....	27
Intervjuer	27
Kända metoder.....	29
Korsreferenser	29
Kravspårbarhetsmatris.....	29
Beroende nyckelfraser.....	30
Quality Function Deployment.....	31
Matrissekvenser (Matrix Sequences).....	34
Integrationsdokument.....	36
Diskussion av kända metoder.....	36
Automatiserade hjälpmedel	39
Integrerade utvecklingsmiljöer	39
Resultat	45
Anpassa användbarhet efter RUP.....	45
Anpassa RUP efter användbarhet.....	46
Slutsatser.....	49
Tack.....	51
Referenser	53
Bilaga A – Intervjumateriel.....	A-1
Intervju med Fredrik Bergström 2001-09-10.....	A-1
Intervju med Malin Petterson, 2001-09-10.....	A-4
Intervju med Helén Ununger, 2001-09-21	A-6

Inledning

Problembeskrivning

Riksskatteverket (RSV) IT [15] har uppmärksammat ett problem med spårbarheten av krav vid mjukvaruutveckling. Vissa krav som ställs upp av kund eller designers, för t.ex. användbarhet, försvinner någonstans på vägen under resten av utvecklingsarbetet och det har varit svårigheter att avgöra var och varför detta inträffar.

RSV har bestämt sig för att inte förvalta någon egen metod för mjukvaruutveckling utan istället har man köpt in en metod som lanserats av Rational Software. Metoden kallas *Rational Unified Process* [9] (RUP) och är en variant av vad som kallas *Unified Software Development Process*. Detta är dock en ganska generell metod som därför har anpassats efter RSVs behov och struktur. Denna anpassade version av RUP, kallad KUR [16], har använts i c:a 20 projekt hittills men problemet med spårbarhet, framförallt vad det gäller användbarhetskrav, kvarstår.

Användbarhetskrav ingår i gruppen ickefunktionella krav som har den gemensamma egenskapen att de är svåra att dela upp i komponenter i ett system då de oftast omfattar hela systemet. Då RUP bygger på användningsfallsdriven utveckling (use-case driven development) uppstår det problem eftersom en sådan metod skapar systemet med utgångspunkt i dess olika komponenter. Problemet ligger alltså i att förena ickefunktionella krav och en användningsfallsdriven utveckling.

Syfte

Examensarbetets syfte är att söka, konstruera eller modifiera en metod för att öka spårbarheten av användbarhetskrav i projekt under rådande förhållanden hos RSV IT.

Riksskatteverket IT

Riksskatteverkets IT-avdelning [15] har cirka 700 anställda över hela landet. IT-avdelningen tillhandahåller IT-stöd för hela RSV-koncernens verksamhet. Dessutom utvecklas och förvaltas IT-stöd för skattehantering, folkbokföring, allmänna val och indrivning med hjälp av både ny och beprövad teknik. Genom att alla personer och företag i Sverige kan anses vara RSV:s ”kunder” är flera av applikationerna som

utvecklats klassade som samhällsviktiga. De huvudsakliga användarna av det IT-stöd IT-avdelningen levererar är skatte- och kronofogdemyndigheternas medarbetare.

IT-avdelningens vision är: ”IT-avdelningen ska med helhetssyn ständigt ge rätt IT-stöd till rätt kostnad och kvalitet.”

De tjänster som tillhandahålls av IT-avdelningen kan delas upp i interna och publika. De interna tjänsterna är:

- Utveckling och underhåll. Huvuddelen av de verksamhets-specifika systemen/applikationerna har utvecklats och utvecklas fortlöpande i organisationens egen regi. Nya applikationer utvecklas likaväl som de applikationer som ligger i förvaltningsfasen.
- Daglig drift och övervakning av koncernens IT-system.
- Leverera information till och inhämta information från externa organisationer, till exempel Statistiska centralbyrån, Riksförsäkringsverket, Lantmäteriverket, Bilregistret, Patent- och Registreringsverket.
- En central helpdesk placerad i Mariestad.
- IT-avdelningen driver och underhåller också en Internettjänst för allmänheten.

Arbetet på IT-avdelningen kan delas upp i tre huvudprocesser och stödprocesser till dessa. Huvudprocesserna är System, Produktion och Användarservice.

Huvudprocess System omfattar utveckling av nya applikationer och underhåll/utveckling av applikationer som är i produktion.

Huvudprocess Produktion innefattar drift och övervakning av applikationer i produktion och den tekniska infrastrukturen.

Huvudprocess Användarservice innebär att det finns en etablerad kanal för stöd och service. Koncernens alla medarbetare kan vända sig till ett ställe med alla ärenden som rör IT i deras vardag och få den service och det stöd de behöver.

Den tekniska miljön är i huvudsak HP-UX med Oracles databaser. Man strävar mot att arbetet ska bygga i allt högre utsträckning på ett

objektorienterat synsätt. IT-avdelningen arbetar med metoder, verktyg och programspråk som till exempel RUP, UML, XML, Rose, Java, JSP, HTML och C++.

Utmaningar finns det gott om för IT-avdelningen. Några saker är folkbokföring via Internet och webbaserad deklarationshantering. Dessutom har man ett regeringsuppdrag att utveckla SHS, spridnings- och hämtningssystem, för alla för all statlig verksamhet. Syftet är att skapa en säker kommunikation mellan myndigheter och näringsliv över Internet. Vidare förekommer samarbete med PRV och Nutek för att med hjälp av Internet förenkla för dem som vill start av nya företag.

En stor del av IT-avdelningens verksamhet är placerad i Solna. Därutöver finns mindre avdelningar på cirka 25 orter runt om i landet.

Förutom i Solna bedrivs systemutveckling på ett tiotal orter, där fyra av dessa ska utvecklas till bärkraftiga systemutvecklings- och förvaltningsorter. Dessa fyra är Visby, Göteborg, Västerås och Umeå.

Drift/övervakning bedrivs förutom i Solna också i Västerås, Härnösand, Vänersborg, Stockholm, Kristianstad och Växjö.

Användbarhet

Termen användbarhet är inte helt lätt att definiera. Många ser termen användbarhet synonymt med användarvänlighet, men det är att göra en något snäv definition och är svårt att mäta. Det som önskas är ett system som ger ett gott stöd för att lösa den uppgift det är designat för och vid utveckling av ett sådant system vill man också kunna mäta det stödet.

ISO (9241-11, 1998) definierar användbarhet på följande sätt:

”The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.”

Definitionen omnämner tre attribut som måste tillgodoses i sitt speciella sammanhang:

Effectiveness – Hur väl systemet stöder den uppgift som ska lösas och att det görs på ett bra sätt.

Efficiency – Att systemet ska göra detta utan att kräva för mycket av användaren.

Satisfaction – Resultatet ska vara till belåtenhet när uppgiften är löst.

Ett annat sätt att se på användbarhet i mätbara termer presenteras av Löwgren [12]. Han definierar användbarhet med REAL, som betecknar fyra attribut som måste tillmötesgås i ett användbart system:

- *Relevansen* för ett system är hur väl det motsvarar användarnas behov.
- *Effektiviteten* i ett system mäter hur effektivt en användare kan genomföra en uppgift genom att använda systemet.
- *Attityd* handlar om användarnas subjektiva uppfattning av systemet.
- *Lärbarhet* avser hur enkelt systemet är att lära sig från början, samt hur enkelt det är att komma ihåg över en längre tidsperiod.

Löwgrens definition omfattar ISOs definition och lite till, men utelämnar kontexten som tas upp i ISO-definitionen.

En annan närbesläktad definition har gjorts av dansken Nielsen [13]. Han menar användbarhet kan beskrivas med fem attribut:

- *Learnability*: Systemet ska vara enkelt att lära sig så att användaren snabbt kan börja arbeta med det.
- *Efficiency*: Systemet ska vara effektivt att använda, så att en hög produktivitet kan uppnås när användaren en gång lärt sig systemet.
- *Memorability*: Systemet ska vara enkelt att komma ihåg så att en tillfällig användare kan återvända till systemet efter en viss tid utan att behöva lära om allt igen.
- *Errors*: Systemet ska ha en god felhantering så att användaren gör få fel vid användandet av systemet och när fel görs ska de enkelt kunna åtgärdas. Vidare så får kritiska fel inte uppkomma under några omständigheter.
- *Satisfaction*: Systemet ska vara smidigt att använda så att varje användare tycker om det och känner sig nöjd efter användandet.

Det som i första hand skiljer Nielsens definition från Löwgrens är att han betonar att även felhantering är ett attribut att ta hänsyn till.

Rational Unified Process - RUP

Rational Unified Process [9] (RUP) är en mjukvaruutvecklingsprocess som bygger på ett iterativt arbetssätt. Med iterativt arbetssätt menas att det finns ett antal faser i utvecklingen som upprepas gång på gång, men med tillägg och utökningar.

Dagens mjukvaror är ofta så pass komplexa och omfattande att utvecklingen av dem kräver en metod med struktur och koordination. En bidragande orsak är också att mjukvaruutveckling vanligtvis omfattar en stor grupp människor som måste hitta ett sätt som underlättar samarbetet. RUP tillhandahåller för detta syfte arbetsflöden (*workflows*) och en mängd dokument mallar (*templates*). RUP är dock inte en helt färdig paketlösning utan ett ramverk som måste anpassas efter varje företag som vill använda sig utav det, och i vissa fall även anpassas efter ett aktuellt projekt.

RUP är en specificering av den mer generella *Unified Software Development Process*, som skapats av Booch, Jacobson och Rumbaugh, vilka också skapat *Unified Modeling Language* (UML). RUP marknadsförs av *Rational Software Corp.* och utnyttjar Internet för att erbjuda kontinuerliga uppdateringar till sina kunder. Gränssnittet är också helt baserat på HTML för att miljön ska vara bekant och enkel att använda sig av. Detta har snabbt blivit en uppskattad metod för mjukvaruutveckling.

Hela idén med RUP är att samla alla goda erfarenheter från tidigare projekt på ett och samma ställe. I processen har man försökt samla de metoder som visat sig vara enkla att använda och som givit goda och effektiva resultat. Framförallt är det sex goda erfarenheter som utgör grunden för RUP:

- Utveckla programvaran iterativt – detta arbetssätt gör att risker tidigt kan elimineras genom att man upprepar de olika faserna i projektet efter ett förbestämt iterationsmönster.
- Hantera föränderliga krav – ett krav är något som systemet måste uppfylla och det är viktigt att vara medveten om att de förändras under produktens utveckling. Det kan hända att krav inte kan uppfyllas och då måste en justering av kravet göras. Även prioritering av kraven kan vara nödvändigt och då är det de mest riskfyllda och kritiska kraven som måste tillgodoses först.

- Använda komponentbaserad utveckling – vilket betyder att man återanvänder redan befintligt material. Det kan t.ex. gälla att en databas eller att källkod återanvänds.
- Visualisera programvarumodellen – en modell är en enklare beskrivning av ett system. För att underlätta överblicken, specifikationen, konstruktionen och dokumentationen av produkten görs visualiseringar av systemet. Ett av de verktyg som används i RUP är modelleringsverktyget UML.
- Verifiera programvarans kvalitet – det är viktigt att testa produkten kontinuerligt och stämma av mot de krav som finns.
- Kontrollera förändringar i programvaruutvecklingsmodellen och se till att alla i projektet är införstådda med de nya förändringarna. Det kan även t.ex. gälla att det är god koordinering vid iterationer.

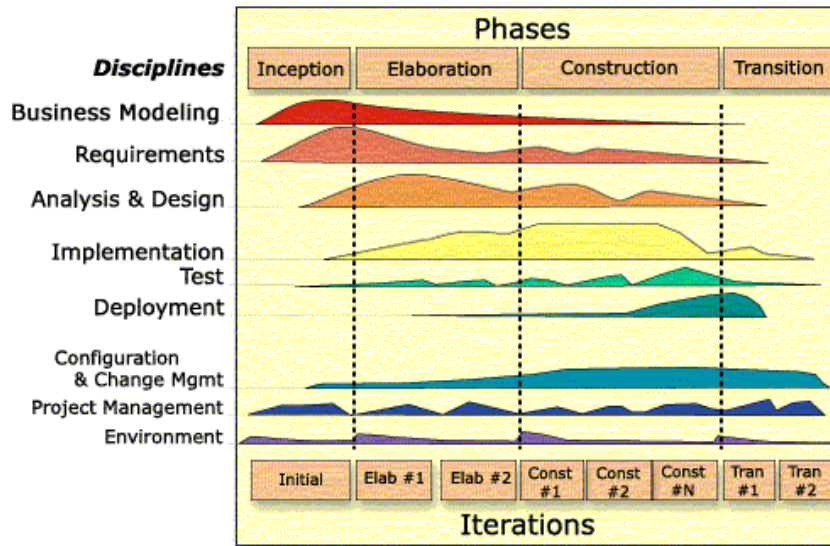
En användningsfallsdriven utveckling

RUP använder sig av användningsfall (*Use-cases*) för att täcka in krav istället för en traditionell kravspecifikation där alla krav skrivs i en lång lista. Samtliga användningsfall struktureras i en så kallad användningsfallsmodell (*Use-case model*). Idealt ska alla krav representeras i den sammanställda användningsfallsmodellen. Kraven kommer dels från kunden (*Stakeholder Request*), den vision som satts upp för systemet (*Vision*) samt ickefunktionella krav som samlats i en egen specifikation (*Supplementary specification*). Hela utvecklingsarbetet bygger sedan på dessa användningsfall och den kompletterande specifikationen.

Ett användningsfall är en sekvens av transaktioner som ett system utför som resultat av att en aktör ger någon form av indata till systemet. En aktör kan vara både ett annat delsystem eller en ”riktig” användare. Beskrivningen av ett användningsfall definierar vad som händer i systemet om just detta användningsfall körs. Användningsfall hjälper till att identifiera angränsade klasser i systemet och vad som skiljer systemet från yttervärlden. Ett användningsfall måste leda till något som är av värde för en aktör och systemet, det måste alltså vara något som leder till en ”nyttig” händelse. Detta för att hålla nere antalet användningsfall.

Struktur

Ett projekts livscykel i RUP går genom fyra olika faser, där varje fas i sig består av en eller flera iterationer av projektet. För projektet finns även en mängd olika aktivitetsflöden som i olika grad är representerade i de olika faserna. Förhållandet mellan dessa och livscykeln kan exemplifieras i figur 1.



Figur 1 - Faser och aktivitetsflöden [17]

Den första fasen, *Inception*, fokuserar på uppstarten och visionen för projektet. Under denna fas skapas en produktvision och en affärsmodell för att förstå helhetskrav och hitta de mest kritiska användningsfallen för projektet. Vanligtvis består fasen av planering för resten av projektet, men det kan även förekomma att slutpunkten blir en enkel prototyp att visa för kund.

Den andra fasen, *Elaboration*, fokuserar på aktiviteter och krav. Här krävs vanligtvis flera iterationer, framför allt om det är ett nytt projekt. Prototyparbete, riskuppskattningar och kravhantering är de huvudsakliga uppgifterna och en nära kontakt med kund eller slutanvändare är önskvärt. Denna fas är mycket kritisk eftersom den lägger grunden för projektet och den produkt som ska skapas.

Den tredje fasen, *Construction*, handlar om att bygga själva projektet. Arkitekturen för systemet ska utvecklas mot visionen som sattes upp i

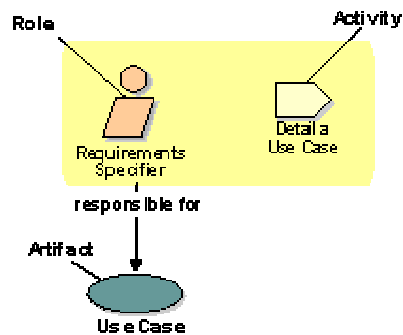
första fasen. Tyngdpunkten i denna fas ligger på design och implementering. Detta är den fas som vanligtvis innehåller flest iterationer under projektet.

Den fjärde och sista fasen, *Transition*, behandlar överlämnandet av produkten till kund och användare. I denna fas sker mycket utvärderingsarbete för att få en återkoppling från kund och slutanvändare.

De olika arbetsflödena kan delas upp i två huvudsakliga delar – utvecklingsflöden och stödfloeden. Utvecklingsflöden är de flöden som driver utvecklingen produkten: *business modeling, requirements, analysis & design, implementation* och *deployment*. Stödfloeden är de flöden som understödjer och styr projektet: *configuration & change management, project management* och *environment*.

Roll, Aktivitet, Artefakt och Arbetsflöden

För att veta vem som ska göra vad, när och hur har RUP fyra olika modelleringsverktyg för att illustrera detta.



Figur 2 - Roll, aktivitet och artefakt [17]

Roll, *Worker*, representerar vem i processen som ska utföra en viss aktivitet, det är alltså definierat en mängd arbetsuppgifter för varje roll. En roll är dock inte knuten till en viss person utan kan innehas av flera personer i ett projekt. En person kan också ha flera olika roller. Det är ofta beroende på projektets storlek.

Aktiviteter, *Activities*, som är kopplade till en roll talar om vad som ska göras. Till varje aktivitet i RUP finns även en beskrivning för utförande och vilka artefakter som ska bli resultatet av aktiviteten.

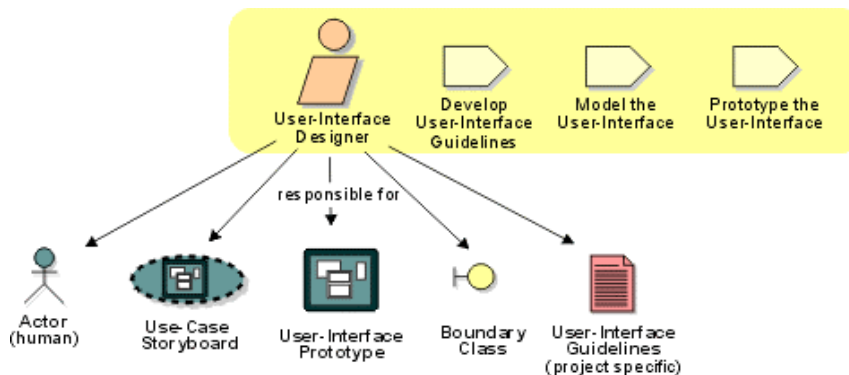
Artefakter, *Artifacts*, är saker som understödjer projektet eller är produkten av en aktivitet, t.ex. ett dokument eller källkod.

Arbetsflöden, *Workflows*, beskriver när i processen olika aktiviteter ska utföras. I figur 1 kan man se att olika arbetsflöden är olika mycket representerade under olika faser av utvecklingen.

Utöver dessa fyra modelleringsverktyg finns vanligtvis en mängd andra stödjande delar i processen, t.ex. dokumentmallar och riktlinjer, *guidelines*.

Roller – Ett exempel: User Interface Designer

User-Interface Designer (vidare benämnd som UID) är en *roll* som finns beskriven i RUP (figur 3). Dess främsta uppgift är att under kravflödet ansvara för den visuella utformningen av gränssnittet. Detta innebär att UID leder och koordinerar prototyparbetet och designen av användargränssnittet. Som indata för att börja sitt arbete erhålls användningsfallsmodellen och den kompletterande specifikationen. Rollens arbete resulterar i en detaljerad beskrivning av användaren och en realisering av användargränssnittets specifika delar. Baserat på detta material byggs en prototyp av användargränssnittet.



Figur 3 – User-Interface Designer [17]

UID utför följande aktiviteter:

- *User Interface Modeling* – som indata till denna aktivitet finns användningsfallsmodellen. Den är organiserad kring användarna av systemet och sammansatt av flera olika användningsfall. Här tillåts UID att bortse från de fall där inte aktören är en människa.

Aktiviteten syftar till att för varje användningsfall specificera ett s.k. storyboard som beskriver händelser i systemet som har med användargränssnittet att göra. Till denna aktivitet hör också att identifiera klasser och objekt som angränsar till varandra, det kan underlätta då strukturen av systemet ska byggas. I RUP kallas de för *Boundry classes*.

- *Develop User Interface Guidelines* – UID ansvarar för de generella riktlinjerna av användargränssnittet. Detta är riktlinjer som ligger till grund för att specificera användargränssnittet och det är riktlinjer som är specifika för just det projekt som drivs.
- *User Interface Prototyping* – den information som ligger till grund för prototyparbetet är det som kommit fram i aktiviteten User Interface Modeling. Initialt används enkla pappersskisser som övergår till enkla dataprototyper. UID låter andra utvecklare ta del av prototyperna så som t.ex. specificerare av användningsfallen.

I och med dessa aktiviteter är UID ansvarig för en rad artefakter. Ur den första aktiviteten, User Interface Modeling, specificeras aktörer till systemet, storyboards för varje användningsfall tillverkas och gränsklasser identifieras. Ur den andra aktiviteten skapas riktlinjer för användargränssnittet (User Interface Guidelines). Den sista aktiviteten resulterar i olika prototyper av användargränssnittet (i RUP: User Interface Prototyping).

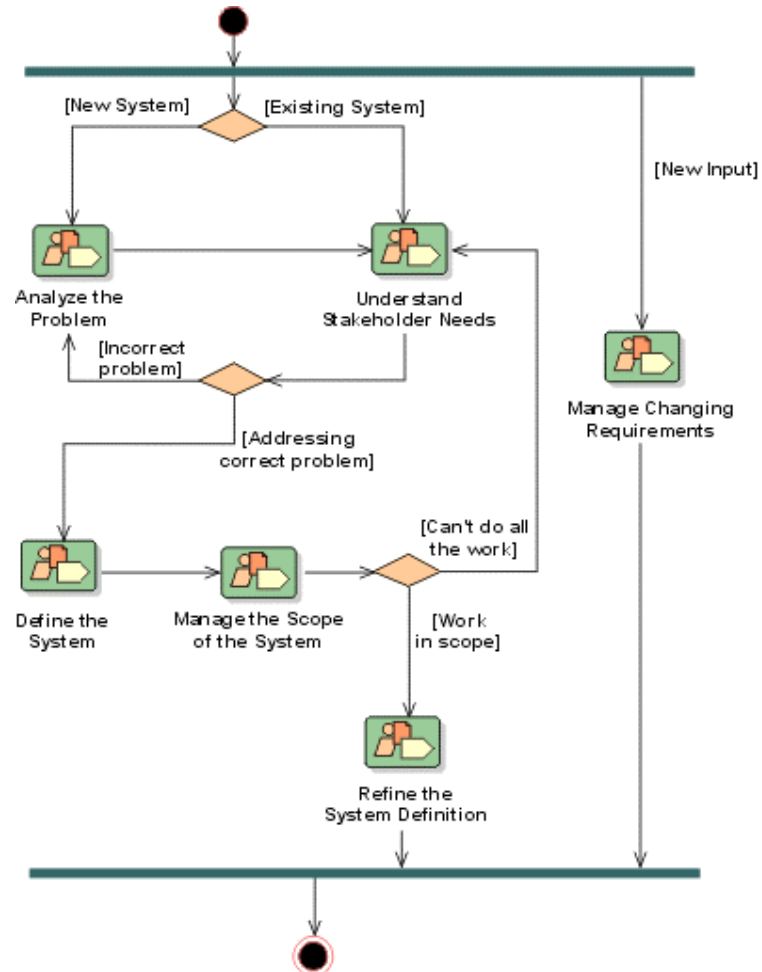
Arbetsflöden – Ett exempel: Kravhantering

Kravflödet i RUP tar hand om arbetet för att identifiera de krav systemet ska uppfylla. Med hjälp av ett aktivitetsdiagram (figur 4 på nästa sida) visas hur flödet ser ut idag och vilka övergripande arbetsflöden som är inblandade i det.

Figuren visar endast övergripande aktiviteter i kravflödet. I onlineversionen av RUP [17] är de övergripande aktiviteterna, t.ex. ”Define the system”, länkar till mer detaljerade beskrivningar av aktiviteterna.

Rollen UID är inblandad i kravflödet. Här har, som tidigare nämnts, denna roll ansvar för två aktiviteter; User-Interface Modeling och User

Interface Prototyping. Dessa återfinns i det övergripande arbetsflödet ”Refine the system Definition”.



Figur 4 - Aktivitetsdiagrammet kravhantering [17]

KUR

För att RUP ska vara användbart i ett visst sammanhang krävs en del anpassning och konfigurering för lokala egenskaper. RSVs Konfigurering av Utvecklingsmetoden RUP (KUR) [16] innehåller många delar där de största är Utvecklingsfall för RSV, vilket är en beskrivning av RSVs konfigurering av KUR, och en ordlista med RUPs vanligaste termer är översatta till och förklarade på svenska samt nya termer som införts i KUR. Utöver dessa ingår dokumentmallar och exempel i KUR.

Ett första steg man gjort är att dela upp de artefakter som ingår i RUP i obligatoriska respektive önskvärda. Nedanstående tabell visar de artefakter, indelade efter arbetsflöden, som är obligatoriska i KUR.

Project Management	Software Development Plan Iteration Plan Risk List Iteration Assessment QA Plan Review Record
Business Modeling	Business Use-Case Business Use-Case Model Business Object Model
Requirements	Supplementary Specification Glossary Use-Case Use-Case Model User-Interface Prototype
Analysis & Design	Software Architecture Document Analysis Model Design Model Data Model Use-Case Realization
Implementation	Implementation Model Software Architecture Document, Implementation View
Test	Test Plan Test Case Workload Analysis
Deployment	Deployment Plan Product End-User Support
Configuration & Change Management	Change Management Plan
Environment	Development Case Guidelines

Tabell 1 - Obligatoriska artefakter i KUR

Därefter innehåller KUR bearbetningar av de arbetsflöden som förekommer i RUP. Dessa bearbetningar är mer eller mindre omfattande. Typiska ändringar är tillägg av aktiviteter för att beskriva företagsspecifika arbetssätt.

För att underlätta planering av respektive arbetsflöde har man skapat tabeller med aktiviteterna i arbetsflödet och markerat dem som önskvärda respektive obligatoriska och också markerat i vilken fas aktiviteten förväntas genomföras (1 = huvudsakligen, 2 = tillägsarbete). Tabell 2 visar exempel på aktiviteter som ingår i arbetsflödet Kravanalys (Requirements).

Detaljerat arbetsflöde	Huvud-aktivitet	Prioritet	Utförs i fas				Artefaktnamn i RUP
			Ince	Elab	Cons	Tran	
Analyze the problem	Develop Vision	OBL	1	2			Glossary (D), Vision (D), Requirements Mangement Plan (D), Use-Case Model (M), Supplementary Specification (D)
Understanding Stakeholder Needs		Ingår i Analyze the problem					Glossary (D), Vision (D), Use-Case Model (M), Supplementary Specification (D)
Define the system	Find Actors and Use-Cases	OBL	1	1	2	2	Glossary (D), Use-Case Model (M), Supplementary Specification (D)
Manage the Scope of the System	Prioritize Use-Cases	Ingår i Develop Software Development Plan under Project Mangement					Iteration Plan (D), Software Architecture Document (D)
Refine System Definition	Detail a Use-Case, Model the User-interface	OBL	1	1	1	2	Use-Case Specification (D), Supplementary Specification (D), User-Interface prototyp (M)
Manage Changing Requirements	Review Requirements	Ingår i Refine System Definition					Use-Case Specification (D), Supplementary Specification (D)

Tabell 2 - Aktivitetsflödet kravanalys enligt KUR

Spårbarhet

Gotel och Finkelstein [6] har skrivit en bra, sammanfattande artikel om spårbarhet av krav vid mjukvaruutveckling. De skriver att spårbarhet är något som fått allt mer uppmärksamhet i standarder och riktlinjer för mjukvaruutveckling. Trots det växande intresset ses det dock som ett problemområde av industrin, bland annat på grund av att det inte är självklart vad som avses med spårbarhet. Betydelsen har betonats olika i olika litteratur, vilka alla påverkar tyngdpunkt och avgränsningar för spårbarheten. Detta gör att man kan dela in spårbarhet i fyra huvudgrupper:

- **Ändamålsdriven** (definierad i termer av vad den ska åstadkomma)
Spårbarhet är förmågan att fastställa affärsläge, omfattning och nyckelkrav för projektet och stå fast vid dem genom utvecklingen.
- **Lösningsdriven** (definierad i termer av hur det ska lösas)
Spårbarhet är förmågan att se sammanhangen mellan en enhet i projektet och en annan genom semantiska relationer.
- **Informationsdriven** (tyngdpunkt på spårbar information)
Spårbarhet är förmågan att skapa en länk mellan funktion, data och krav i den dokumentation som refererar till dem.
- **Riktningdriven** (tyngdpunkt på spårbar riktning)
Spårbarhet är förmågan att följa en specifik enhet från indata till en fas i mjukvaruutvecklingens livscykel till en specifik enhet i utdata från den fasen.

Problemet blir alltså hur man ska kunna prata om spårbarhet när man menar olika saker med begreppet. Ett behov för att komma tillrätta med problemet är en gemensam definition av begreppet spårbarhet.

Definition

Det finns en standardiserad definition av spårbarhet från ANSI/IEEE som lyder [7]:

En specifikation av mjukvarukrav är spårbar om

- (i) ursprunget för vart och ett av kraven är klarlagt och
- (ii) den möjliggör referenser till varje krav i framtida utvecklad eller förbättrad dokumentation.

Denna definition betonar spårbarhet *bakåt* till alla tidigare dokument och spårbarhet *framåt* till alla dokument som kommer att skapas. En

alternativ definition kan utvecklas från ordet ”trace” (spåra) i *Oxford English Dictionary* [2]:

Spårbarhet är förmågan att ”särskilja” och ”markera” ”tydliga tecken på vad som har existerat eller hänt” under livstiden för ett krav för att möjliggöra ”en väg att följa” genom dokumenteringen.

En sammanslagning av dessa två ger följande definition av spårbarhet:

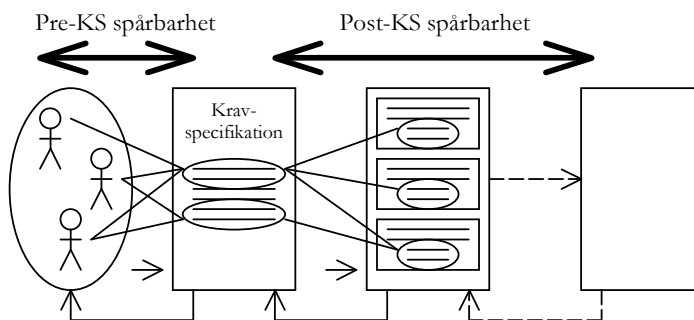
Spårbarhet betecknar förmågan att beskriva och följa ett kravs livscykel, både framåt och bakåt i tiden, det vill säga från ursprunget, genom utveckling och specificering, till efterföljande införing och användning och alla perioder av pågående förfining och iterering under någon av dessa faser.

Denna definition kommer runt problemet med ANSI/IEEE-definitionen, vilken begränsar spårbarheten till tiden efter tillkomsten av en kravspecifikation. Många gånger är vägen för ett krav till en kravspecifikation minst lika viktig, och denna inkluderas i den sammanslagna definitionen.

Pre-KS- & Post-KS-spårbarhet

Vidare tar Gotel och Finkelstein [6] upp att spårbarhet kan delas upp i två huvudsakliga grupper:

- **Pre-KS-spårbarhet**, vilken behandlar aspekterna av ett kravs livstid före dess inskrivning i en kravspecifikation (KS).
- **Post-KS-spårbarhet**, vilken behandlar aspekterna av ett kravs livstid som resulterar av inskrivningen i en kravspecifikation (KS).



Figur 5 - Pre- och Post-KS spårbarhet

Att kunna spåra krav framåt och bakåt är viktigt, men trots det måste man göra skillnad på pre-KS- och post-KS-spårbarhet. Båda behövs men man måste känna till bådars egenskaper då båda sätter upp vissa krav på hur ett stöd för spårbarheten kan se ut.

Post-KS-spårbarhet ger en möjlighet att spåra krav från, och tillbaka till, en bas (kravspecifikationen) genom hela den efterföljande processen. Förändringar i basen måste propageras genom hela kedjan av efterföljande aktiviteter. Post-KS-spårbarhet är relativt elementär och har gott stöd i många verktyg.

Pre-KS-spårbarhet ger en möjlighet att spåra krav från, och tillbaka till, deras ursprung genom processen av kravspecificering och förfining innan alla olika källor integreras till en enda kravspecifikation. Förändringar i kravspecifikationen måste spåras till sitt ursprung och propageras genom processen därifrån. Den ger en möjlighet till omarbeting och omvärdering av krav i kravspecifikationen och ger en förståelse för hur kravspecifikationen skapats och kan hållas uppdaterad.

Metod

Efter en bakgrundsstudie om RUP och spårbarhet beslutade jag mig för att göra en fallstudie och intervjuer på RSV-IT för att få en bild av hur arbetet fungerar idag. Syftet var att kunna studera befintliga metoder för spårbarhet och sedan på ett smidigt sätt modifiera och överföra en av dem, alternativt skapa en ny, till det aktuella sammanhanget.

Fallstudie

På grund av bristande intresse hos RSV kom aldrig fallstudien till stånd, så arbetet bygger endast på bakgrundsstudierna, intervjuer samt studier av befintliga metoder.

Intervjuer

För att få en övergripande bild av hur mjukvaruutvecklingen i allmänhet, och kravhanteringen i synnerhet, fungerar på RSV har jag gjort tre intervjuer med personer i olika instanser av mjukvaruutvecklingen. De personer jag intervjuat är Fredrik Bergström (system- och verksamhetsanalytiker), Malin Pettersson (användbarhetsdesigner) och Helén Ununger (provningsledare, systemutvecklare). Personerna valdes ut av min handledare på RSV, Malin Pettersson. En sammanställning av samtliga frågor och svar finns i bilaga 1 – Intervjumateriel.

Ur intervjuerna framkom att den enda gemensamma strategi för kravhantering som används är den som RUP erbjuder genom en summering av tre artefakter: *Stakeholder Request* dokumentet, *Vision* dokumentet och *Supplementary Specification*. Det råder delade meningar om huruvida dessa är tillräckliga. Både Fredrik och Helén menar att det beror helt på användbarhetsdesignern hur ickefunktionella krav, till exempel användbarhetskrav, bevakas.

Det visar sig också vara svårt att konstruera användningsfall så att alla krav täcks in, vilket leder till problem vid utvärderingen av användningsfall och testfall. Extra problematiskt är det med just krav på användbarhet då dessa ofta är svåra att formulera i kvantitativa termer och inte heller kan delas upp då de till största delen berör helheten, t.ex. konsekvent gränssnitt och enkel inläring.

Kända metoder

Efter mycket sökande på Internet, framförallt med utgångspunkt från de referenser som omnämns i Gotel och Finkelsteins artikel [6], bibliotek och god hjälp från min handledare fann jag referenser till en mängd kända metoder för att skapa spårbarhet vid mjukvaruutveckling. Nedan har jag i korthet beskrivit några av dem som tydligast visar olika varianter av spårbarhet, utöver dessa finns många, många varianter på samma tema, och i många fall påminner de jag valt ut också om varandra på ett eller annat sätt.

Korsreferenser

Evans [5] presenterar en metod för spårbarhet som bygger på att kraven för projektet fastlagts från början. Utifrån kravspecifikationen görs sedan en preliminär design, där designen blir en funktionell syntes av kraven i kravspecifikationen. Successivt delas kraven upp i lägre och lägre nivåer av funktionalitet tills man når *mjukvaruenheter*. En mjukvaruenhet är en enkel rutin eller samling rutiner, som utför en väldefinierad funktion, kan testas separat och uppfyller en del av kravspecifikationen. Parallellt med detta skapas flödesscheman över varje delsystem vilket blir en egen del av kravdokumentationen. Slutprodukten blir en designspecifikation, vilken är en detaljerad beskrivning av delsystemens funktion, gränssnitt och en sammanfattning av delsystemens krav och karaktäristik.

Genom att koppla ihop enheterna i delsystemen och kravspecifikationen för projektet skapas en matris som fungerar som korsreferens från kravspecifikationen till designspecifikationen.

Kravspårbarhetsmatris

En metod, som påminner mycket om korsreferenser och kanske den metod man själv kommer på först, är kravspårbarhetsmatrisen (KSM) [3]. Denna matris fungerar i flera olika delar av utvecklingsprocessen, men vanligast vid design och test.

Den fungerar så att varje krav i kravdokumentet numreras på ett eller annat sätt, t.ex. hierarkiskt med kapitel- och avsnittsnummer, 2.3.1. Om så krävs kan man även peka ut en mening i ett avsnitt genom att lägga till s och meningens nummer, 2.3.1s4. I sin enklaste form blir KSM då en tvådimensionell matris med en rad för varje krav och en kolumn för

varje komponent i projektet. Ett kryss i en ruta visar att komponenten i den kolumnen uppfyller kravet på den raden. På så vis går man igenom komponenterna från början till slut och markerar med kryss där de uppfyller olika krav. I slutändan kan man då se att en rad utan ett enda kryss är ett krav som inte tillgodoses i projektet, och att en kolumn utan ett enda kryss är en onödig komponent i projektet då den inte relaterar till ett enda krav. Vidare är matrisen en hjälp vid felsökning då man enkelt kan följa en rad med det krav som påverkats och se vilka komponenter av projektet som är möjliga felkällor.

Beroende nyckelfraser

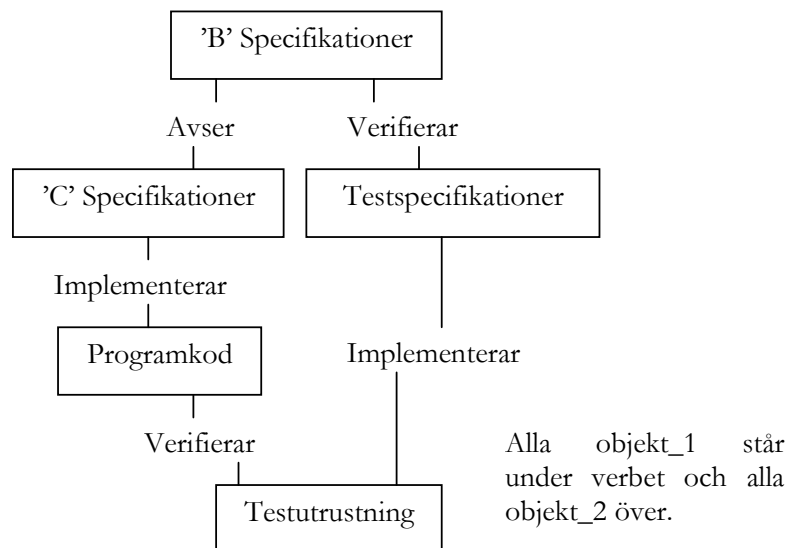
Genom att använda nyckelfraser [8] vid objekt i olika dokument kan man länka ihop dokument och på så vis åstadkomma spårbarhet. Förhållandet mellan objekten är uppåtgående, det vill säga man vet varför ett objekt finns där det finns, men inte vad som gjorts efteråt. Detta ger att det på högsta nivån i systemet finns objekt som inte är knutna till något annat. Själva spårningen kan utföras antingen manuellt eller med hjälp av en parser för dokumenten, detta tack vare en konsekvent och relativt enkel syntax. I sin enklaste form ser syntaxen ut på följande sätt:

```
>>>objekt_1:version[ verb dok_ref:objekt_2 [ ... ]]<<<
```

Tecknen ”>>>” och ”<<<”gör det enkelt att identifiera början och slutet på en nyckelfras. ”objekt_1” är det symboliska namnet på objektet som definieras i det aktuella dokumentet, och det följs av en versionssiffra. Verbet specificerar ett förhållande, som till exempel ”avser”, ”implementerar” eller ”testar”. Objektet eller objekten på andra sidan av relationen, ”objekt_2”, identifieras som del av ett specifikt dokument. Det kan finnas flera dokument och nyckelfrasen kan avse en okänd mängd text i dokumentet. Lägg märke till att endast objekt_1 har en versionssiffra, vilken måste ökas om ändringar i objektet görs, medan objekt_2 inte har någon här. Om det haft det hade en ändring i det objektet inneburit en väldig massa dokumentjusteringar där hänvisningar till objektet finns.

Om en ändring görs i objekt_1 måste förhållandet till objekt_2 kontrolleras för att se om nyckelfrasen fortfarande är giltig. Omvänt gäller att om en ändring görs i det som här kallas objekt_2 i ett visst dokument måste samtliga dokument gås igenom för att söka nyckelfraser som hänvisar till objekt_2 i just det dokumentet. I det senare fallet har man stor hjälp av ett verktyg med parser för dokumenten.

Om man extraherar nyckelfraserna ur samtliga dokument erhålls samtliga förhållanden mellan dokumenten, och det går enkelt att skapa en graf som illustrerar detta. Ett enkelt exempel på en sådan graf visas i figur 6.



Figur 6 – Beroende nyckelfraser

Genom att skapa ett verktyg som kontinuerligt uppdaterar relationerna mellan dokumenten är det enkelt att se vilka dokument som behöver kontrolleras då en ändring sker i en entitet.

Quality Function Deployment

Quality Function Deployment (QFD) [18] är en metod som matchar kundens önskemål och behov mot produktens krav i en matris. Metoden tar också hänsyn till hur viktiga de olika önskemålen är för kunden.

Indata till metoden är kundens egna önskemål och behov, deras betydelse för produkten samt hur väl dessa önskemål uppfylls i konkurrerande produkter. Kundens önskemål brukar skrivas med vanligt språk, och kallas därför "kundens röst". Att som önskemål skriva att ett flygplan ska kunna flyga fortare än 1600 mph kan därför vara fel om kundens önskemål var att det ska kunna flyga fortare än en MIG 25. Dessa önskemål och behov från kunden förs in längs matrisens vertikala axel. Hur viktigt önskemålet är för kunden beskrivs med en siffra från 1

till 9 i första kolumnen jämte önskemålen. Därefter formuleras krav på produkten för att möta dessa önskemål och behov, helst i kvantitativa termer, och förs in längs matrisens horisontella axel. Teamet som skapar dessa krav ska bestå av representanter för alla delar i organisationen så att alla krav och aspekter kommer med.

Därefter följer det svåraste steget i skapandet av matrisen, nämligen att bestämma hur betydande ett visst krav på lösningen är i förhållande till önskemålet från kunden. Betydelsen anges som stark, medium, svag eller ingen och skrivs in i matrisen som siffror i skalan 9, 3, 1 och 0 (9 = stark betydelse, 0 = ingen betydelse). Låt säga att kravet ”Tid för inläring” är starkt betydande för ”Snabb inläring” i tabell 3, då sätts siffran 9 in i knutpunkten för dessa två i matrisen.

	Kundens prioritering	Tid för inläring (# timmar)	Antal misstag (# per panel)	Inmätningstid (# s. per panel)	Responstid (# s.)	Antal användare (# användare)	Antal transaktioner (# per timme)	Datavolym (# bytes)	Backupfrekvens (# per timme)	Tillgänglighet (% av dagen)	Nätverkssupport (Ja/Nej)	Allvarliga fel (# fel)
<i>Enkelhet</i>												
Snabb inläring	8	9	9	3	3							
Bra hjälp	6	9	9	3								
Få knapptryckningar	7		3	9	3							
<i>Kapacitet</i>												
God svarstid	8	3		9	9	9	9					
Stöd för min avdelning	9					9	9	9				
<i>Pålitlighet & tillgänglighet</i>												
Förlorar aldrig data	9								9		3	9
Tillgängligt vid behov	6								3	9	3	
Tillgängligt överallt	4									3	9	
Förhindrar mina misstag	8		3						3			9
Teknisk betydelse		150	171	177	117	153	153	81	123	66	81	153

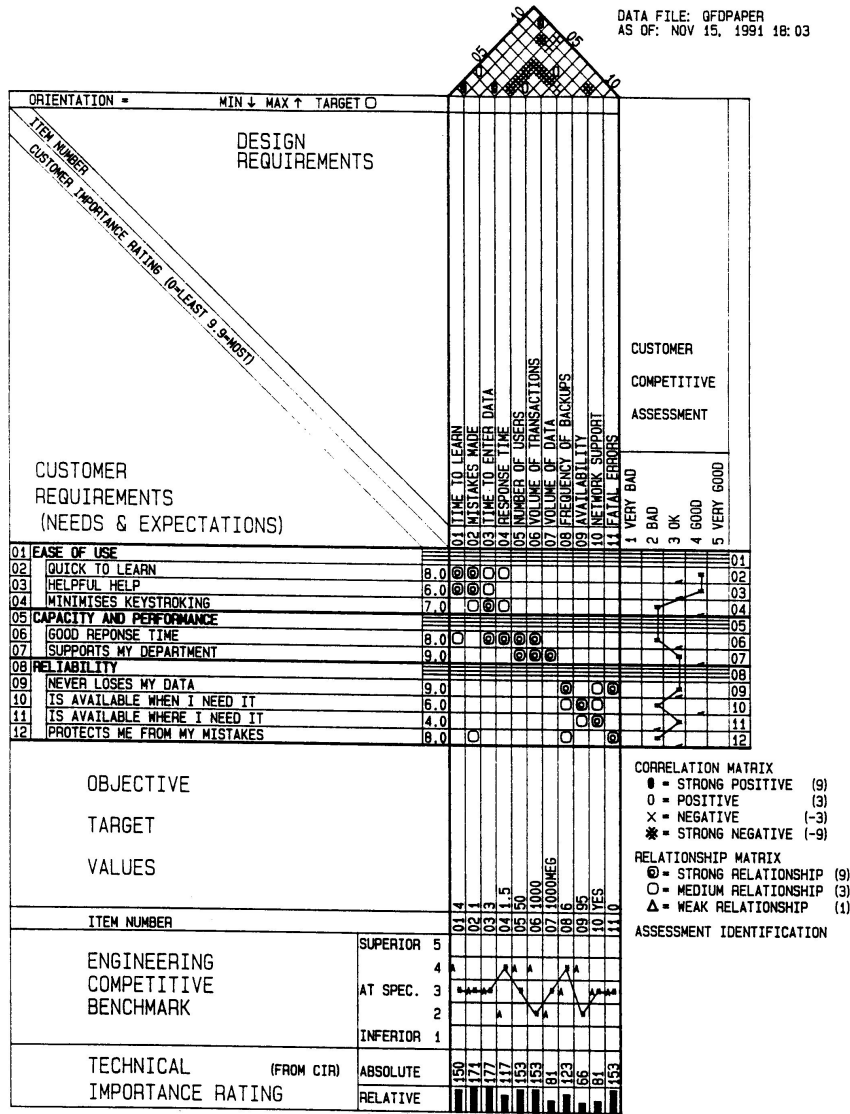
Tabell 3 - QFD-matris

Efter det att samtliga celler i matrisen bearbetats och fått ett värde (nollor kan utelämnas) beräknas den tekniska betydelsen för varje krav genom att för varje krav multiplicera de celler som fått ett värde med kundens prioritering för det önskemålet och sedan summera dessa produkter. Dessa summor betecknar hur viktiga kraven är i den slutliga lösningen i förhållande till kundens önskemål. Ur tabell 3 kan man utläsa att det är mycket viktigt att skapa produkten så att få misstag görs vid användandet, men tillgängligheten av systemet kan däremot vara begränsad.

När denna grundmatris väl är gjord byggs den ut med kundens bedömningar av konkurrenters produkter och hur väl de motsvarar önskemålen. Man kan även föra in jämförelsetal för konkurrenters lösningar på kraven.

Slutligen görs en korrelation mellan kraven på lösningen och en bedömning hur de konkurrerar/stöder varandra. I detta fall används en skala starkt negativ (-9), svagt negativ (-3), svagt positiv (3) och starkt positiv (9) för att beteckna korrelationen mellan kraven. Bedömningen görs så att om man gör en förbättring av krav 1 och det samtidigt leder till en förbättring av krav 2 så råder positiv korrelation, och om det skulle leda till en försämring av krav 2 så råder negativ korrelation.

Alla dessa data samt grundmatrisen skrivs in i vad som kallas *House of Quality*, och ger en bild av hela projektet i förhållande till kundens önskemål samt konkurrerande produkter. Ett exempel på ett sådant *HoQ* kan ses i figur 7.



Figur 7 - House of Quality [18]

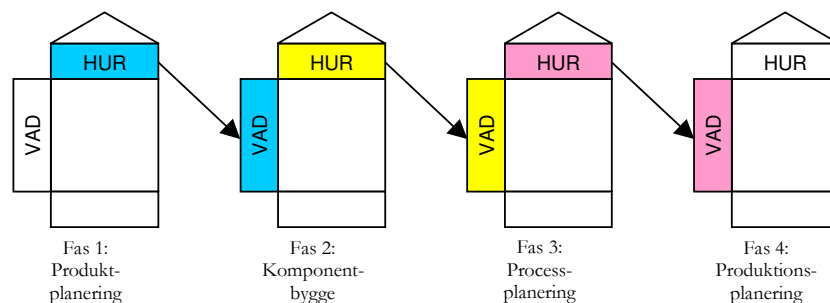
Matrissekvenser (Matrix Sequences)

En vidareutveckling av möjligheterna med QFD beskrivs av Brown [1]. Han menar att man kan använda QFD till mer än bara funktionella krav på en produkt. Han visar på hur man kan prioritera arbetsuppgifter i ett projekt, till exempel rapportskrivning, test och utvärdering, genom att

placera dessa uppgifter som krav (HUR) och i matrisen fylla i deras vikt i förhållande till önskemålen (VAD), till exempel krav från ledningen på veckovis rapportering. Värderna tilldelas enligt samma metod som i grundanvändningen av QFD och de kalkylerade värdena ger en bild av vad som har högst prioritet.

Vidare beskriver han att man genom att använda sig av fyra matriser i sekvens kan åstadkomma en mer heltäckande och finkornigare bild för ett helt projekt, men ändå bara använda sig av enkla matriser. Metoden bygger på att efterföljande matris VAD är föregående matris HUR. På så vis kan man få fyra faser, med en matris för varje, enligt följande:

1. *Produktplanering* – behandlar identifiering av tekniska krav (HUR) som satisfierar kundens önskemål (VAD).
2. *Komponentbygge* – skapar komponentkaraktistika (HUR) utifrån de tekniska krav som identifierats i fas 1 (VAD).
3. *Processplanering* – tar fram processens prioriteringar (HUR) som leder till komponentkaraktistika från fas 2 (VAD).
4. *Produktionsplanering* – definierar kraven för produktionen (HUR) för processens prioriteringar från fas 3 (VAD).



Figur 8 - Matrissekvenser

Figur 8 visar en schematisk bild över hur matrissekvensen ser ut. Denna metod ger en beskrivning för hela arbetsprocessen, inte bara för vad produkten ska motsvara.

Integrationsdokument

Ett integrationsdokument [11] är i grund och botten en samling länkar från ett dokument till ett annat. Inte bara 1:1 utan tillåter även n:m länknings. I integrationsdokumentet lagras även beroenden mellan länkarna. Detta åstadkommer spårbarhet på så vis att när något ändras i ett dokument kan man enkelt följa länkarna i integrationsdokumentet för att se vad som påverkas av förändringen.

Denna metod lämpar sig bra både då man använder en utvecklingsmetod, till exempel *Structured Analysis*, eller om man väljer att märka upp med taggar i dokumenten. Det hela kan sedan skötas antingen manuellt i pappersform där alla kopplingar skrivs upp i till exempel en matris, med det går också relativt enkelt att skapa ett verktyg som läser dokumenten och som skapar länkar automatiskt. Ett automatiserat verktyg kan dessutom programmeras till att ställa frågor till användaren då den stöter på en ofullständig länk eller då något ändrats så att länken inte längre är korrekt.

I detta fall, framförallt om en utvecklingsmetod används, behöver man aldrig göra ändringar i käll- och måldokument för att uppnå spårbarhet utan alla eventuella förändringar sker i integrationsdokumentet.

Versionshantering av grunddokumenten såväl som integrationsdokumentet säkerställer även historisk spårbarhet.

Diskussion av kända metoder

Som jag inledningsvis skrev påminner metoderna som jag presenterat i mångt och mycket om varandra. Skillnaderna beror mycket på projektets komplexitet. De tre först nämnda (korsreferenser, KSM och beroende nyckelfraser) tillsammans med Integrationsdokument bygger alla på ett koncept där man på ett eller annat sätt markerar varje krav och sedan kopplar de olika komponenterna i projektet mot dem på ett eller annat sätt. QFD och matrissekvenser bidrar till att kunna prioritera kraven och även se alternativa lösningar på dem innan man designar själva systemet då man måste färdigställa en matris innan man kan flytta över HUR från en matris till VAD i nästa.

I ett enkelt projekt lämpar sig en alltså matris, som då inte behöver bli allt för omfattande, där man snabbt och överskådligt ser hur väl projektet motsvarar kraven. I ett större projekt kan man ha god nytta av att kunna prioritera kraven, t.ex. med Matrissekvenser. Självklart skapas

problem vid versionshantering och eftersom mitt arbete syftar till att kombinera metoderna med RUP, som är en iterativ metod, är det nödvändigt att kunna bygga ut kraven och således även spårbarheten bit för bit i varje iteration. Därför har man god hjälp av att kunna automatisera underhållet av spårbarhetsmetoden. Däremot är det enkelt att överföra användningsfallstänkanet till samtliga nämnda metoder då man istället för att sätta en specifik funktion eller modul sätter ett användningsfall i relation till kraven.

Det kan också vara intressant att beakta på vilket sätt dessa metoder klarar av att hantera pre- respektive post-KS spårbarhet. Samtliga metoder utgår från en kravspecifikation och täcker processen därifrån till färdig produkt, alltså post-KS spårbarhet. När det gäller pre-KS spårbarhet kan vi dock se vissa skillnader. För att pre-KS spårbarhet ska fungera med korsreferenser eller KSM måste det till en ny matris som matchar ursprungskällorna mot kraven i den sammansatta specifikationen. När det gäller beroende nyckelfraser går det dock bra att helt enkelt lägga en rad källdokument som länkas till den sammanställda kravspecifikationen på samma sätt som efterföljande dokument, alternativt att källdokumentet utgör högsta nivån i hierarkin och den sammanställda kravspecifikationen blir näst högsta nivån (alla krav i kravspecifikationen *inkorporerar* kraven i källdokumentet). QFD hamnar i samma fälla som korsreferenser och KSM, men om man sätter samman dem och skapar en matrissekvens blir det helt enkelt en fas före alla de andra som ger de sammanställda kraven som resultat. Slutligen integrationsdokument: dessa kan hantera källdokumentet på samma sätt som alla andra dokument och skapa länkar till den sammanställda kravspecifikationen. Integrationsdokumentet kan alltså redan från början hantera både pre- och post-KS spårbarhet.

Vilken metod man än väljer kan man få en bra kontroll över att projektets funktionalitet uppfyller kraven från kunden. Ett problem kvarstår dock, och det är att på ett bra sätt få in ickefunktionella krav i sammanhanget, t.ex. lärlbarhet, då dessa är globala krav för systemet och inte kan isoleras till ett användningsfall. Det är också svårt, för att inte säga omöjligt, att i ett användningsfall formulera de ickefunktionella kraven. Daryl Kulak och Eamonn Guiney [10] har skrivit en bok om just kravhantering och användningsfall: *Use Cases – Requirements in context*. Idén i boken är att öka spårbarheten i ett projekt genom att ställa upp kontextmatriser, vilka kopplar ihop användningsfall som beror av varandra. När det gäller ickefunktionella krav försöker man göra på samma sätt. De samlar alla ickefunktionella krav i en särskild matris och

noterar för varje krav vilka delar av systemet alternativt vilka användningsfall de förekommer i. Problemet är att detta oftast slutar med att några enskilda användningskrav får specifika hänvisningar medan resten får ”alla” som referens eftersom ickefunktionella krav oftast är globala och är svåra att bedöma på användningsfallsnivå.

Automatiserade hjälpmedel

Ovan nämnda metoder har stora vinningar att göra om man kan automatisera dokumentationen på ett eller annat sätt, i vissa fall har möjligheter till detta redan nämnts. Rational Requisite PRO och DOORS är två exempel på sådana hjälpmedel som finns att köpa. Det är förstås fullt möjligt att skapa ett hjälpmedel själv. Principen för ett sådant beskrivs nedan.

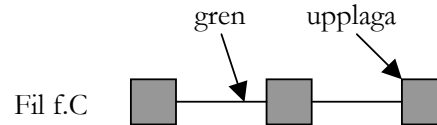
Integrerade utvecklingsmiljöer

Bell-Northern Research [14] har skapat en utvecklingsmiljö där spårbarhet integrerats som en del, de kallar den för *Bell-Northern Research's Integrated Development Environment (BNR IDE)*. Utvecklingsmiljön hanterar alla typer av filer, till exempel rapporter och källkodsfiler. All specifik information om en fil sparas i en metafil som lagras tillsammans med originalfilen.

När en fil sparas delas den dessutom upp i så kallade *designelement (design elements)*. Hur dessa ser ut beror på vad originalfilen är för typ av fil, men om det är ett dokument kan det vara varje stycke eller kapitel som bildar ett designelement, för en källkodsfil kan det vara varje funktion eller deklaration. Hur varje fil ska delas upp finns bestämt utifrån filtypen och det finns regler för hur olika filer ska skrivas för att parsern ska kunna tolka informationen korrekt. BNR IDE lagrar alltså inte bara information om associationer filer emellan utan även design element emellan.

Vidare tillåter BNR IDE skapandet av attribut för filer och även designelement, vilket innebär att man kan markera till exempel status för ett visst designelement och på så vis snabbt få en överblick över hur ett projekt framskrider.

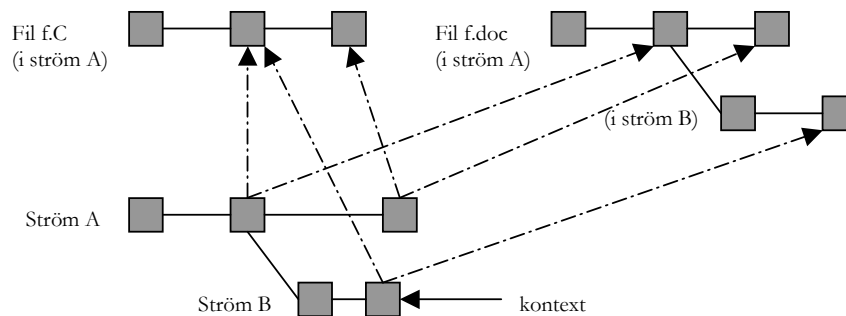
För att hålla reda på historiken har BNR IDE en integrerad versionshantering, konfigurationshantering och ändringshantering. Varje gång en fil förändras sparas en ny version av filen i systemet. På så vis kan man alltid gå tillbaka till äldre versioner av filen om problem skulle uppstå. I BNR IDE får varje fil en egen *gren*, där varje gren består av ett antal *upplagor*. En upplaga är filens utseende vid ett visst tillfälle, och nya upplagor adderas till slutet av grenen och när en ny upplaga läggs till klassificeras den som *öppen*. Endast öppna upplagor är åtkomliga för förändring. Figur 9 (nästa sida) illustrerar gren och upplaga för en fil.



Figur 9 - Gren & upplaga

En konfiguration kan ses som en stillbild av produktens utveckling vid ett givet tillfälle, den innehåller alltså varje ingående fils tillstånd vid just det tillfället. Även konfigurationer versionshanteras i likhet med filerna så att man kan gå tillbaka i tidigare konfigurationer vid behov.

Inom ett projekt skapas sedan *strömmar* där varje ström representerar ett antal relaterade filer. Strömmarna är också versionshanterade men deras upplagor kallas för *kontext*. Detta för att en fil kan vara representerad i flera strömmar och dessutom kan den skilja sig i utseende. Om en fil har förändrats i en ström skapas en förgrening och således existerar filen bara i ett visst utseende i varje kontext. En ström kan också förgrenas vid behov. Figur 10 är en illustration av två filer och en förgrenad ström.



Figur 10 - Förgreningar och strömmar

Ändringskontrollmekanismen i systemet bidrar med många saker. Den ger en anledning till förändringar som görs, grupperar filer som ändrats av samma anledning och identifierar vilken användare som gjort förändringar samt när det skett. På så vis kan ändringskontrollen användas för att spåra när och var problem kommit in i systemet.

I BNR IDE skapas ändringshanteringen genom *uppdateringar*. En uppdatering identifierar strömmen som förändras och användaren som gör förändringen. Den innehåller också en förklarande text till

förändringen, referenser till externa problembeskrivningar samt en lista över upplagor, en för varje fil som förändras.

För att skapa en förändring i BNR IDE skapar användaren först en uppdatering i strömmen där ändringen ska ske. Därefter lägger användaren till filerna som ska ändras. För varje fil skapas då en ny upplaga på den gren som är associerad med strömmen. När förändringarna är gjorda skickas uppdateringen in i systemet. Då sparas alla förändrade filer och läggs till de nya upplagorna i rätt kontext.

Att använda systemet med strömmar ger också fördelen att man kan arbeta med flera versioner av systemet samtidigt. Den redan släppta versionen finns i en ström som är separat från den där utvecklingen av nästa version pågår. BNR IDE har sedan stöd för att slå samman strömmar. Vid en sammanslagning rapporteras vilka filer i två skilda strömmar som är i konflikt med varandra och erbjuder också verktyg för att jämföra filerna. När inga konflikter finns sker sammanslagningen automatiskt.

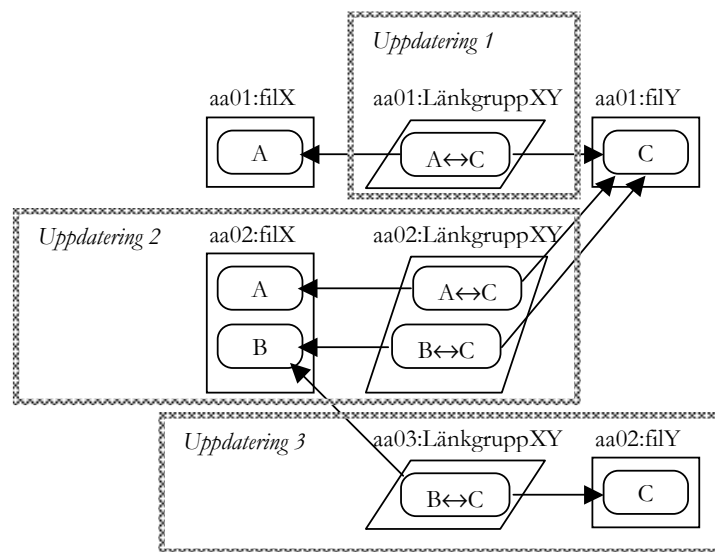
En åtkomstkontroll är också inbyggd i systemet så att skaparen av varje fil kan välja vilka användare eller grupper av användare som ska kunna läsa eller redigera en fil, detta för att begränsa felaktiga uppdateringar samt skydda känslig information. Åtkomstkontrollen skapar en separat lista för varje gren.

När det gäller spårbarhet erbjuder BNR IDE många möjligheter. Redan från början lagras information om hur filer är associerade till varandra, och genom att denna information versionshanteras genom systemets utveckling garanteras en god spårbarhet. Problemet när det gäller spårbarhet handlar om att en fil kan innehålla en stor mängd designelement som är associerade till en eller flera andra filer. Detta kallas för *filbarriären*, att endast hela filer men ej designelement hanteras.

Designelementen är lösningen för att komma runt filbarriären. En inbyggd parser kan genom de förutbestämda definitionerna ta ut designelementen ur en fil och lagra associationerna till övriga filer, även i fallet då designelementen i en fil är associerade till fler än en annan fil.

Ytterligare hjälp fås genom möjligheten till attribut på designelementen. Vid förändringar av filen kan alla inblandade parter sätta status på sina designelement, och först när alla designelement fått godkänt för till exempel uppdatering får filen godkänt för uppdatering.

Sist men inte minst ska alla dessa delar länkas samman på ett lämpligt sätt. I det fallet har BNR IDE skapat något som kallas för *länkgrupp*. En länkgrupp innehåller information om två filers associationer till varandra och versionshanteras helt internt inom BNR IDE. Systemet lägger självt märke till när förändringar görs och uppdaterar länkgrupperna. Ett exempel på hur länkgrupper fungerar beskrivs i figur 11.



Figur 11 - Uppdateringsexempel

Uppdatering 1 hanterar situationen där designelementen i två filer, aa01:filX och aa01:filY länkas samman. Då ingen förändring av designelementen sker behöver inte filerna uppdateras, endast associationerna skapas. Länkgruppen skapas (aa01:LänkgruppXY) och ett länkelement ($A \leftrightarrow C$) skapas för associationen mellan designobjekt A och C. När alla associationer omvandlats är uppdateringen klar.

I *uppdatering 2* läggs designelement B till i filX och ska också länkas till designelement C. En ny version av filX (aa02) måste skapas och så även en ny version av LänkgruppXY (aa02) så att associationen ($B \leftrightarrow C$) kan läggas till. När ändringarna gjorts är uppdateringen klar.

Uppdatering 3 är en förändring av designelement C så att det inte längre ska vara associerat till A. aa02:filY skapas för att spara förändringarna av

C samtidigt som aa03:LänkgruppXY skapas för att notera borttagningen av associationen mellan A och C.

Denna typ av utvecklingsmiljö lämpar sig bra för en mjukvaru-utvecklingsmetod baserad på användningsfall. Varje användningsfall beskrivs i en fil, där de olika delarna av användningsfallet blir var sitt designelement. Då kan källkod, kravspecifikation och övrig dokumentation kopplas samman med användningsfall, och ända ner på designelements nivå, och på så vis åstadkomma spårbarhet genom hela projektet. Systemets versionshantering ser också till att bevara länkarna mellan filer och på så vis även spårbarheten.

Resultat

Redan problembeskrivningen visar att arbetet i första hand bestod i att försöka kombinera två skilda perspektiv. Användbarhet kräver överblick och omfattar ett helt projekt (globala mål), medan RUP bygger på att dela upp krav på användningsfall (lokala mål). Detta ger två vägar att gå: att anpassa användbarhet efter RUP eller att anpassa RUP efter användbarhet.

Anpassa användbarhet efter RUP

För att göra så små ingrepp som möjligt på nuvarande arbetssätt ansåg jag att det första alternativet (anpassa användbarhet efter RUP) var enklaste vägen att gå.

Det största problemet med användbarhet är att sätta upp mål som går att utvärdera. Att t.ex. sätta upp ”produkten ska vara lätt att använda” som ett användbarhetsmål må vara riktigt, men det är mycket svårt att testa och utvärdera. Joseph S. Dumas och Janice C. Redish [4] skriver om detta i sin bok *A Practical Guide to Usability Testing*. De betonar att man måste se bakom det generella målet och sätta upp mer specifika användbarhetsmål. De tar upp ett exempel med en datorfirma som hade problem med instruktionerna för hur datorn skulle sättas ihop. Inför en ny version av dator och instruktionsbok kunde man ha sagt att: ”datorn ska vara lätt att sätta ihop och konfigurera”. Istället satte man upp följande användbarhetsmål:

”En helt okunnig användare ska klara av att avemballera, koppla ihop och konfigurera datorn på mindre än en halvtimme utan att behöva ringa hjälpcentralen.”

Bedömningen är alltså att en dator är lätt att sätta ihop och konfigurera om en okunnig användare klarar det på egen hand på mindre än en halvtimme. Detta är ett kvantitativt men tydligt användbarhetsmål, som går bra att testa och utvärdera.

Min idé är att man genom att arbeta på detta sätt ska försöka, med utgångspunkt i definitionen av användbarhet, kvantisera användbarhetsmålen och göra dem tydliga för projektet. Då kan man skapa en lista över konkreta användbarhetsmål, lämpligtvis i RUPs *supplementary specification*, och sätta upp varje mål i den spårbarhetsmetod man väljer som vilket annat krav som helst.

De krav som man inte lyckas kvantisera kan sättas upp i *supplementary specification* och för varje mål försöker man skriva vilka användningsfall som berörs, även om det ofta blir alla. När man sedan utvärderar dessa mål får man för varje användningsfall det berör försöka sätta ett värde på en glidande skala, t.ex. målet uppfylls helt, målet uppfylls till stor del, målet uppfylls i viss mån, målet uppfylls ej. I utvärderingen kan man sedan räkna ut ett värde på hur väl målen uppfylls som helhet och söka de punkter som drar ner värdena. Sammanställningen kan också jämföras mot kundens önskemål (likt QFD). På detta vis skapas spårbarhet för användbarhetsmålen till varje enskilt användningsfall och följer således helt RUPs struktur.

Låt mig ta konsekvens som ett exempel. En generellt användbarhetsmål är att systemet ska vara konsekvent, det vill säga att t.ex. en knapp med en viss funktion i ett gränssnitt helst ska ha samma placering och attribut genom hela systemet. För varje användningsfall får man då beakta de gränssnitt det berör och bedöma hur väl det svarar mot användbarhetsmålet.

Detta ger då att man vid en utvärdering av hela systemet kan summera de olika användningsfallen och se hur väl systemet som helhet motsvarar användbarhetsmålen. Vid det tillfället kan man också studera hurvida ett visst användningsfall kan tillåtas bryta mer eller mindre mot uppsatta användbarhetsmål, t.ex. kan ett användningsfall som väldigt sällan inträffar tillåtas bryta mot användbarhetsmålen i större utsträckning än de som inträffar ofta.

Anpassa RUP efter användbarhet

RUP bygger på att dela upp alla krav i användningsfall och sedan studera hur varje fall uppfyller kraven eller inte. I slutet på varje iteration sker sedan test på systemet, och i början av processen (Inception och Elaboration) innebär det i första hand utvärdering av systemet som helhet.

För att anpassa RUP till användbarhet och dess globala mål kan man då tänka sig att skapa ett ytterligare aktiviteter i requirements och test så att man i slutet av varje iteration utvärderar hur väl systemet motsvarar användbarhetsmålen. Alternativt skapas ett eget arbetsflöde för användbarhet som i stor grad samverkar med requirements och test.

En ny roll – användbarhetsövervakare – och en ny artefakt – användbarhetsmål – införs. Användbarhetsdesignern ansvarar för användbarhetsmålen och huvudsakliga aktiviteter är att sätta upp användbarhetsmål respektive utvärdera användbarhetsmål. På detta vis skapas en överblick över systemet och användbarheten kommer in redan från början, inte bara som *supplementary specification* utan som en egen artefakt.

Vidare dokumenteras i varje iteration hur väl användbarhetsmålen uppfylls (detta kan fortfarande göras på en glidande skala), vad som behöver ändras (vidarebefordras till *Configuration & Change Management*, ett arbetsflöde i RUP) och förändringar av målen. På så vis får man en spårbarhet av användbarhetsmålen till iterationsnivå, om än inte användningsfallsnivå. Den stora fördelen är att det genom hela processen finns dokumenterat när, var och hur användbarhetsmålen uppfylls och förändras.

Precis som med alla andra roller i RUP kan denne användbarhetsövervakare vara samma person som har hand om övrig kravspecifikation, men jag tror att det kan hjälpa att ha en särskild person med detta uppdrag som då helt kan koncentrera sig på användbarhetsfrågor i ett projekt.

Slutsatser

Det är svårt att kombinera två olika perspektiv, eller rättare sagt omöjligt. Man måste kompromissa på det ena eller andra sättet. Detta gäller alla icke-funktionella krav, inte bara användbarhet. Mina handledare på RSV förespråkade att anpassa RUP efter användbarhet då de tyckte det var för svårt att kvantisera användbarhet och att utvärderingen av användbarhet uppdelat på användningsfall blir allt för subjektiv.

Min handledare på Umeå Universitet, Jürgen Börstler, anser dock att det är att blunda för problemen och att den andra metoden, att anpassa användbarhet efter RUP, inte är någon större skillnad mot hur RUPs grundstruktur är utan att det istället blir tårta på tårta utan någon förbättring.

Själv förespråkar jag att i största möjliga mån anpassa användbarhet efter RUP, enligt den första metoden. Jag tror att man i större mån kan kvantisera många användbarhetsmål och göra dessa tydliga tidigt i ett projekt. Ett problem för RSVs användbarhetsgrupp idag är att de ofta får komma in för sent i projekten och då blir det mest en snabb utvärdering utifrån definitionen av användbarhet. Har man fastnat i det är det svårt att se hur man ska kunna kvantisera användbarhet och göra det till ett krav bland andra tidigt i projektet.

Jag tror vidare att man genom att börja arbeta på detta sätt kan höja medvetenheten kring användbarhet och på så vis skapa ett större utrymme i framtida projekt för användbarhet.

Tyvärr havererade mitt samarbete med RSV då jag upplevde ett bristande intresse från deras sida för mitt arbete. Därför har största delen av denna uppsats kommit till på eget initiativ med stor hjälp av min handledare Jürgen Börstler. Metoden blev aldrig testad och därför har jag inte själv kunnat utvärdera hur min anpassning av användbarhet (och andra icke-funktionella krav) fungerar i praktiken. Jag har inte heller haft möjlighet att försöka integrera metoden med befintliga kravhanteringsverktyg som Requisite Pro eller DOORS, så jag lämnar det åt den som önskar fortsätta detta arbete.

Tack

Ett stort tack till min handledare Jürgen Börstler som bidragit mycket till att jag tillslut blev färdig med denna uppsats.

Referenser

- [1]
Brown, Patrick G: *QFD: Echoing the Voice of the Customer*, AT & T Technical Journal, March/April 1991, pp. 21-31.
- [2]
Concise Oxford Dictionary of Current English, 8th ed., Edited by R. E. Allen, Clarendon Press, Oxford, 1990
- [3]
Davis, Alan M.: *Software Requirements: Objects, functions and states*, kap. 3, 1993, Prentice Hall Inc.
- [4]
Dumas, Joseph S. och Redish, Janice C.: *A Practical Guide to Usability Testing*, 1999, Intellect Ltd.
- [5]
Evans, Michael W.: *The Software Factory*, kap. 9, 1989, John Wiley and Sons.
- [6]
Gotel, Orlena och Finkelstein, Anthony: *An Analysis of the Requirements Traceability Problem*, IEEE International Conference on Requirements Engineering, Los Alamitos, California: IEEE Computer Society Press, April 1994, sid. 94-101.
- [7]
IEEE, *IEEE Guide to Software Requirements Specifications*, ANSI/IEEE Standard 830-1984.
- [8]
Jackson, Justin: *A Keyphrase Based Traceability Scheme*, in Tools and Techniques for Maintaining Traceability During Design, IEE Colloquium, Computing and Control Division, Professional Group C1, Digest no.: 1991/180, p. 2/1-2/4.
- [9]
Kruchten, Philippe: *The Rational Unified Process – An Introduction, Second Edition*, kap. 2-3, 2000, Addison Wesley, Massachusetts.

-
- [10]
Kulak, Daryl och Guiney, Eamonn: *Use Cases – Requirements in Context*, 2002, ACM Press, New York
- [11]
Lefering, Martin: *An Incremental Integration Tool Between Requirements Engineering and Programming in the Large*, Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego, California, 1993, Jan. 4-6, pp. 271-276.
- [12]
Löwgren, Jonas: *Human-computer interaction*, kap. 5, 1993, Studentlitteratur, Lund.
- [13]
Nielsen, Jakob: *Useability Engineering*, s. 26, 1993, Academic press, Inc. San Diego.
- [14]
Macfarlane, Ian a. och Reilly Ian: *Requirements Traceability in an Integrated Development Environment*, Proceedings of the Second IEEE International Symposium on Requirements Engineering, Los Alamitos, California, 1995, pp. 116-123.
- [15]
Riksskatteverkets IT-avdelning, <http://www.rsv.se/jobb/it/main.html> (besökt 2002-10-17)
- [16]
RSV-IT: *Development Case RSV - RSV:s Konfigurering av Utvecklingsmetoden RUP*. Version 2.0, 2002
- [17]
Onlineversionen av RUP. Tillgänglig för 30-dagars test på www.rational.com
- [18]
West, Martin: *Quality Function Deployment in Software Development*, in Tools and Techniques for Maintaining Traceability During Design, IEE Colloquium, Computing and Control Division, Professional Group C1, Digest no.: 1991/180, p. 5/1-5/7.

Bilaga A – Intervjumateriel

Intervju med Fredrik Bergström 2001-09-10

Fredrik Bergström är Systemanalytiker (Systemanalytist) och Verksamhetsanalytiker (Business process designer, Business analyst) vid RSV. Han har varit anställd i 7 år och har under den tiden varit inblandad i ett 10-tal projekt, varav 5 före införandet av RUP.

När man läser om RUP låter allt så fantastiskt bra, vad är din åsikt om RUP i praktiken?

Med rätt rollfördelning fungerar RUP bra. Det är dock avgörande hur systemanalytikern fokuserar på krav snarare än att försöka vara arkitekt i projektet.

Nackdelen med RUP är att det är väldigt mycket artefakter inblandade, vilket kräver erfarenhet för att flyta. Det är gott om artefakter redan i kravsteget vilket kan göra det svårt att komma igång.

För att RUP ska fungera bra är det extremt viktigt att ha gott om tid med verksamhetsfolk vid kravfångsten i början av projektet. Gärna i tvådagarspass så att man hinner sova på saken inför andra dagen. Detta måste följas upp med tät kontakt mellan utvecklare och verksamhetsexperten. Det är dock alltid en svårighet att få fram ickefunktionella krav.

RUP är bra i mindre projekt. När det gäller stora projekt finns mycket kvar att göra, t.ex. brister rollfördelning och ansvarsfördelning.

Problemen minskar förhoppningsvis med större erfarenhet av RUP.

Har användandet av RUP påverkat hur projekten flyter nu jämfört med före RUP, eller är det bara något som tar mer tid?

RUP tar tid men ger en röd tråd att följa under projektet.

Har RUP inneburit en större inblandning av slutanvändare i projektet?

Nej, ingen större förändring vad det gäller inblandningen av slutanvändare.

Hur dokumenteras krav i ett projekt i nuläget?

Allra först skapas en vision för systemet. Där brukar kraven i första hand vara de uppenbara features man ser i visionen. Därefter modelleras kraven fram allt eftersom.

Senare under processens gång byggs kraven in i use-cases och beskrivs i specifikationerna för dessa. Ibland används tilläggsdokument. Spårbarheten är svår, man kan vanligtvis bara nå tillbaka till föregående use-case. Verktygshjälp är nödvändigt, t.ex. Requisite Pro.

Hur hanteras ändringar av krav under projektets gång?

Vissa projekt har Change Control Board (CCB). Icke frysta krav som växer fram under projektet tillåts att förändras fritt. När inget CCB finns är det projektledaren som styr och har frihet att ändra fritt.

Alla artefakter versionshanteras. Godkännandet sker på användningsfallsnivå. Även här vore ett verktygstöd att önska.

Innan man skriver användningsfall försöker man frysa mängden krav, och i och med det införs också en ändringshantering i processen.

Vissa saker leder dock alltid till diskussion i skapandet av användningsfall:

- Hur dokumenteras och beskrivs verksamhetsregler?
- Hur ska användningsfallen beskrivas? Nivå och omfattning?

Var brister RUP?

- Prototyping täcks inte ordentligt av RUP.
- Vore mycket bra med kvalitetskriterier. RUP har dåligt med generella checklistor.
- Stöd för att själv skapa checklistor.

Fördelar?

RUP föreslår att man antar två perspektiv vid verksamhetsmodellering: kundnytta vs. intern

Görs det skillnad mellan användbarhetskrav och andra krav?

För det mesta bakas krav och användbarhetskrav ihop. Det är användbarhetsdesignern som gör skillnaden. I andra fall sker det i en utvärdering först efteråt.

Det varierar huruvida alla krav står tillsammans eller i skilda artefakter. (Användbarhetsdesignern måste finnas med.) Användaranalys görs utanför use-cases etc. Bör vara med i starten innan man skriver användningsfall.

Det vore bra att kunna plocka användarens syn redan i verksamhetsanalysen.

Hur utvärderas kraven för att se om de uppnåtts?

Vanligt är att man verifierar mot modelleringsgruppen. Ibland verifieras mot användaren.

Mellan iterationerna sker verifieringen mot modelleringsgruppen. Det förekommer dock att man verifierar mot användare.

Vad gör ni för att uppnå spårbarhet hos kraven idag?

De flesta nöjer sig med att hålla reda på aktuella krav. Ingen spårbarhet underhålls. Verktygsstöd skulle underlätta.

Om det fanns verktygsstöd skulle det, givet en systemvision, underlätta verifieringen av det som implementerats samt skapa konsistens i kravmassan. Det är ofta för jobbigt att manuellt underhålla en systemvision. Användningsfall baserade på systemvisionen skulle vara bra att ha, p.g.a. att resurser byts ut under projektets gång och de kan då ge validitet åt kraven i visionen. Styrgruppen som spikar visionen önskar också en indikering på att alla krav uppnås eller varför de ej gör det.

Är det någon skillnad i kravhanteringen nu jämfört med före RUP?

Idag kan krav växa fram. Alla krav måste inte frysas från början. Alla krav var tvunget att frysas före designen tidigare, så är inte fallet nu. Det har också lett till en bättre struktur på kraven. Framtagningen sker dock ungefär på samma sätt.

Kravändringar hamnar lite mer ”rätt” nu. Iterationstänkandet kan skapa problem då det gäller tekniska lösningar som inte finns i tidiga iterationer men som är nödvändiga för projektet och därför ej kan verifieras. I nuläget är det inte lättare att ändra krav än tidigare, men det görs lite mer ”rätt”.

Intervju med Malin Petterson, 2001-09-10

Malin Petterson är användbarhetsdesigner i bl.a. Kuling. Hon är också uppdragsledare för användbarhetsstöd på RSV. Bevakare av användbarhetsfrågor i SM-proj. Hon har jobbat 3 år på RSV och varit inblandad i c:a 10 projekt, varav 4-5 med RUP.

Hur dokumenteras krav i nuläget?

Användbarhetskrav ska dokumenteras i generella krav, det har det slarvats med.

Användningsfall dokumenteras noga. Hög koncentration på dessa, men de är inte heltäckande. Detta ger att användbarhetsfolk inte är helt nöjda. Nivån är ofta illa vald och beskrivningarna otillräckliga.

Görs det skillnad mellan användbarhetskrav och övriga krav?

Det delas på generella (här ingår användbarhet) och funktionella krav. Fokus ligger vanligtvis på funktionella krav, med säkerhet som undantag bland generella krav.

Hur utvärderas kraven för att se om de uppnåtts?

Kontinuerliga uppföljningar görs av prototyper, även konceptuella. Dels expertutvärderingar enligt en heuristisk metod (liknar Nielsen) och användartester med riktiga slutanvändare. Vissa tester görs även mot beställare.

Kravfångsten sker under modellering. Användbarhetskraven försöker man fånga upp från slutanvändarna. Ganska bra användarcentrering på RSV. Man blandar också in slutanvändarna vid modellering, så att användarna är inblandade hela processen. Det som styr är hur många % som användbarhetsdesignern är inblandad i projektet.

Intuition styr snarare än metoden när det gäller användbarhetsdelarna i KUR. Relationen till projektledaren är ofta avgörande för inflytandet i projektet. Många projekt vill ha med ANA (ANvändbarhetsAnalys & användbarhetsstöd), men ger sällan fria tyglar. Oftast sker endast en utvärdering när produkten är i stort sett klar för att kunna få ”ANA-stämpel” utan att föreslagna åtgärder genomförs.

Vad gör ni för att uppnå spårbarhet hos kraven idag?

Svårt! Ex: deklARATIONER – dessa ska genomföras av allmänheten. Krav: Enkelhet. Hur genomför man det? (Det bör vara enklare än papper

annars väljer man den vägen.) Hur realiseras det i en design? Idé: flytta pappersblanketten till internet. Då sker diskussion kring detaljer som t.ex. att knappar i windowsmiljö blir ”grå” när de är inaktiva, men i projektet vill man att de helt ska försvinna för att inte förvirra för användaren, alltså i stort sett en ”wizard”. Användbarhetsprincipen om att ha kontroll över datorn kommer då i skymundan. Vilket är viktigast? Är styrbarheten eller förvirringen det viktigaste? Vilka konsekvenser får det? Spårbarhet saknas.

Utvecklarna har funnit vad som kallas för boundry classes användbart i detta sammanhang.

Spårbarheten sker idag till största delen i huvudet. (Vilket är dåligt då folk tillkommer och försvinner i projekten.)

Hur hanteras krav som ändrats under projektets gång?

Här uppstår problemen! Ingen kontroll...

Är hanteringen av krav annorlunda idag jämfört med före RUP?

Användbarhet före RUP var ”systemet ska vara användbart”. Man studerade inte användbarhet för sig. Idag är det fokus på användbarhet på ett annat sätt. Fokus på fler krav. Dock är det IT-sidan som skriver ner krav, inte beställaren, vilket vore bättre.

När man läser om RUP låter allt så bra, vad är din åsikt om RUP som helhet?

Tycker inte allt i RUP låter bra, men tror mycket på RUP då den tillåter att man tar in egna metoder (och uppmuntrar till det). Det har givit en väg in för användbarhetsfrågor och har potential. RUP är mycket bristfälligt gällande användbarhet och fungerar definitivt ej för RSV. Inga steg för steg för att ta fram användbarhetskrav. Och det som finns är i fel workflow, nivå etc. Då skapas KUR för att anpassa RUP till RSV. Man vill baka in användbarheten i processen, ej som parallellspår som RUP förespråkar. Vill integrera prototyping och framtagandet av användningsfall.

Är det någon skillnad mellan stora och små projekt?

Jag tror att användbarhet lättare kommer in i stora projekt då det går något långsammare. Små projekt ska ofta gå så fort. Vi vill komma in tidigt för att kunna skapa en bra design från början. Inte bara ”läppstiftet på liket”.

Intervju med Helén Ununger, 2001-09-21

Helén är provningsledare i projektet Kuling, annars är hon systemutvecklare på RSV. Hon har arbetat på RSV i sju månader och varit inblandad i två projekt (Kuling och SM) vilka båda utvecklas med RUP.

Hur dokumenteras krav? (kravfångst)

RUP är ej bra på att fånga användbarhetskrav och hanteringen av GUI är klart eftersatt. Vad det gäller övriga krav är RUP bra.

Hur utvärderas kraven för att se om de uppnåts?

(Då projektet är så ungt så har ingen direkt testning gjorts, så denna fråga handlar snarare om huruvida kraven fångats upp eller inte.)

Tillsammans med SSK (Särskilda SkatteKontoret) går man igenom kraven för att se att allt tagits med och man noterar differenser. Det dokumenteras sedan i användningsfall och dokumentet generella krav. Även mer formella granskningar genomförs, men hittills ingen.

Görs det någon skillnad vid utvärdering av anv.krav och andra krav?

Det är på grund av användbarhetsdesigners som användbarhetskraven bevakas.

Hur ska krav ställas upp för att vara lämpliga att testa?

Helén har dragit upp riktlinjer för hur användningsfallen ska skrivas där hon betonar att kraven ska skrivas i mätbara och tydliga termer. Vad det gäller användbarheten är detta ett problem då det ofta handlar om subjektiva åsikter från slutanvändarna. Detta saknar stöd i RUP. Själv har hon ingen erfarenhet av detta utan förlitar sig på stöd från ANA-gruppen.

Vad gör ni för att uppnå spårbarhet hos kraven idag?

Helén har en metod där hon mappar kraven (då de frysts för aktuell iteration, det gäller även generella) mot provfallen så att hon får korsmatriser. På detta sätt ser hon sedan till att alla krav behandlas i något testfall. Versionshanteringen sker m.h.a. Clear Case.

Hur hanteras krav som ändras under projektets gång?

Ändringshanteringen, som är tänkt att skötas m.h.a. Clear Quest, ger nya/ändrade krav. Vissa saker måste då testas om och provfall skrivas om. Här är spårbarheten mycket väsentlig. Metoden har fungerat bra

enligt tidigare erfarenheter. RUP har dock dåligt stöd för denna process, det saknas t.ex. exempeldokument för att ge spårbarhet i processen.

De fel eller korrigeringar som upptäcks skickas vidare till ändringshanteringen (CCB).

Allmän åsikt om RUP?

RUP är bra för stora projekt som kräver mycket för att kunna få en överblick, däremot riskerar mindre projekt att kvävas av RUP på grund av den stora mängden artefakter som ska framställas. SM-projektet har stor roll att spela här och bör ta fram 2 anpassningar av RUP: en för stora projekt och en för små.

RUP har mycket teoretisk bakgrund men brister i exempel, malldokument etc. Är bättre anpassat för utveckling av hyllprodukter än handläggarstöd.

RUP är bra som metod men bör inte följas blint.

I nuläget är processen för Kuling klar. Vad det gäller ändringshantering har Alf Gustavsson varit med och tagit fram generella riktlinjer för ändringshantering på RSV. Kort sagt gäller att kraven fryses vid en given tidpunkt och därefter ska allt ske via CCB. Helén har goda erfarenheter av detta handlingsätt sedan tidigare.