Förebildsanalys som grund i agil utveckling: en studie i prioritering av bruksegenskaper

Magisteruppsats i kognitionsvetenskap
Institutionen för datavetenskap (IDA)
Linköpings universitet
Författare: Petter Bergqvist
Handledare: Arne Jönsson
Examinator: Arne Jönsson
ISRN: LIU-IDA/KOGVET-A--10/004--SE

Sammanfattning

Att utveckla ett program som lyckas uppfylla alla förväntningar som en användare kan ha är svårt. Att utveckla ett program som lyckas uppfylla vissa av alla de förväntningar en användare kan ha är betydligt mycket lättare. Men att kunna hitta just de egenskaperna som gör att en användare inte blir besviken utan nöjd med ett program är en svår process. Ett sätt att hitta de egenskaperna kan vara genom att nyttja en användarcentrerad metod som extrem programmering. Agil utveckling och extrem programmering har som område fokuserat relativt lite på användarnöjdhet som mått för att avgöra om ett program är lyckat eller ej.

Med detta som bakgrund har ett utvecklingsprojekt tagit form med syfte att utveckla ett modelleringsverktyg för analysmetoden FRAM. I detta projekt har en metod bestående av dels förebildsanalys och agil utveckling legat som grund. För att gruppera och prioritera programegenskaper har Kanos modell för kundnöjdhet använts.

Uppsatsen visar att en metod som grundar sig i förebildsanalys kan användas för att ta fram de mest basala egenskaperna hos en artefakt. Detta i kontrast till viss agil utveckling ska det visa sig. Uppsatsen visar även på att den tillgängliga tiden för ett utvecklingsprojekt kan vara avgörande för vilka sorts egenskaper man utvecklar

Innehållsförteckning

1 Inledning

Den här uppsatsen tar sitt avstamp i analysverktyget Framvisualizer och de problem som uppstod efter utvecklingen då programmet togs i bruk. Problemen som uppstod var att användare förväntade sig funktionalitet som inte fanns där och som aldrig ens diskuterats att den skulle finnas. Det var de mest basala funktionerna som verkade saknas som förbluffade oss som utvecklare att ingen hade tänkt på. Det var någonstans här som grunden för detta arbete började ta form. Hur kunde den agila utvecklingsprocessen vi hade använt oss av missat sådana basala funktioner? Det visade sig att allt som sagts, som var explicit var med i Framvisualizer, men allt implicit saknades.

Uppsatsen och utvecklingsarbetet kommer att kretsa kring en metod vid namn FRAM. FRAM står för Functional Resonance Analysis Model och är en modell avsedd att användas för analyser av komplexa kognitiva system. Dessa analyser har ofta som mål att se vart i ett system som det finns en eller flera samverkande brister som kan göra att det uppstå ett tillbud eller en olycka. Detta görs ofta vid just ett tillbud eller olycka för att se vad som orsakade den samma.

Syftet med utvecklingsarbetet är att skapa ett modelleringsverktyg för FRAM. Detta för att man på ett enklare sätt än idag kunna göra analyser och se vart i ett komplext system som brister kan uppstå. Syftet med uppsatsen är att undersöka hur förebilder och förebildsanalys kan användas som grund för brukskrav. Att se hur detta kan användas för att använda tidigare artefakter och användares kunskaper kring dessa för att bättre identifiera, kategorisera och prioritera dessa krav. Som grund för detta kommer Kanos modell om kundnöjdhet att användas och det kommer att ske med en agil utvecklingsmetod för att realisera dessa krav.

Frågeställningen för uppsatsen lyder; hur kan förebildsanalys och en agil utvecklingsmetod stödja framtagandet av de mest basala bruksegenskaper hos en artefakt?

2 Teoretisk bakgrund

I detta kapitel presenteras de teoretiska ramverk som används som referens för arbetet, såväl de teorier som rör FRAM men även teorier som rör agila utvecklingsmetoder och interaktionsdesign.

2.1 FRAM

Här följer en kort introduktion till vad FRAM är och vad det används till. De delarna som kommer att presenteras är endast de delar som kommer att beröras i det kommande utvecklingsarbetet (se kapitel 7).

FRAM är en förkortning för functional resonance analysis model. FRAM är således en funktionell analysmetod och är avsedd för kvalitativa analyser av sammansatta, komplexa kognitiva system. FRAM är en systemmodell som inte avser att visa på ett enskilt fel utan på att flera normalt fungerande funktioner i ett visst givet tillstånd kan orsaka en olycka. Med FRAM så kan samtliga M(änniska)T(eknik)O(rganisation)-faktorer modelleras för att visa dess påverkan på systemet de ingår i. (Hollnagel, 2004)

FRAM används idag av en växande skara forskare och riskanalytiker och har bland annat använts för att visa på systemiska fel i hanterandet av en kärnbränsle”modul” (ref till Jos + Karin), hur ett tänjande av intervall för service av styrstag kan leda till flygolyckor (Woltjer, 2006).

2.1.1 Modell

En FRAM är en modell av ett system. Denna FRAM konstitueras av sina interna funktioner och dess egenskaper.

2.1.2 Funktion

Grundstrukturen i FRAM är att de olika funktionerna i ett system identifieras och beskrivs var för sig. Funktionerna förhåller sig till andra funktioner genom sex möjliga kopplingspunkter. Input och output är de två huvudkopplingarna där input förvandlas till output, till exempel patienter kommer till akuten i ett virrvarr, men kommer ut ”sorterade” i kategorier, om inte helt behandlade.

En funktion i ett FRAM definieras av att den är funktionellt sett en enskild enhet som är separat från andra funktioner.

En FRAM-funktion definieras av sex olika aspekter:

- Input (I): Nödvändiga villkor för att utföra funktionen och utgör också länken till tidigare funktioner. Input kan antingen förändras eller användas i uträttandet av funktionen för att producera ett output.

- Time (T): Tid kan fungera som en begränsning, men kan också ses som en resurs.
- Control (C): Kontroll eller restriktioner som grundas hos naturlagar, arbetsorganisation, kontroll och skyddssystem, riktlinjer som existerar för att övervaka eller begränsa funktionen.
- Output (O): Resultatet av vad som produceras av funktionen och även en länk till efterföljande funktioner.
- Resource (R): Representerar det som behövs för att funktionen ska kunna hantera input – t.ex. personal, teknik eller material.
- Precondition (P): Systemvillkor som måste uppfyllas eller funktioner som måste köras före funktionen kan utföras.

Notationen för en funktion kan se ut som följer:

Aspekt	*Värde*
Function	Baka bröd
Input	N/A
Output	Bröd
Preconditions	
Resources	Mjöl
Time	

Denna funktion bakar bröd, kräver inget specifikt input, däremot mjöl som resource, och ut ur funktionen kommer bröd.

Den grafiska presentationen av en funktion är i form av en hexagon. Varje aspekt är förkortad med så input blir ”I” output blir ”O” i hexagonen osv.

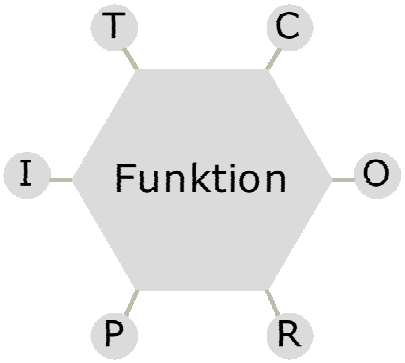
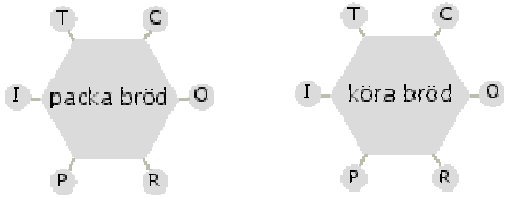


Bild 1:

Om man har fler funktioner i sitt system så kan det se ut som i bild 2.



2.1.3 State

States är näst intill oprövad del eller tillägg i FRAM som till stor del ännu är outvecklat

Ett state i ett FRAM är ett tillstånd med ett eller flera värden. Tillståndet kan ej ha flera värden vid samma tidpunkt. En aspekt kan ha ett eller flera states tilldelade sig, men en aspekt kan även bara ha ett värde.

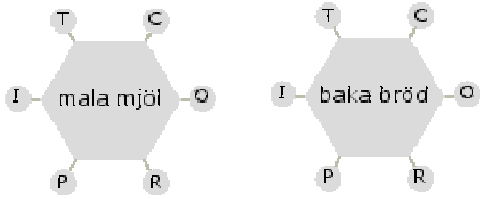


Bild 2:

Exempelvis kan ett state se ut som:

Variabel: värde

Sätter man ihop detta med aspekterna för funktionen så blir det;

Aspekt: variabel: värde

Och det kan visas genom

Function	Baka bröd
Input	N/A
Output	Bröd: bakat
Preconditions	
Resources	Pengar: > 300
Time	

Men en aspekt behöver inte ha ett state tilldelat sig, utan kan även endast ha ett värde, detta kan visas som:

Function	Baka bröd
Input	N/A
Output	Bröd
Preconditions	
Resources	Pengar
Time	

Ett state till skillnad från ett värde ger möjlighet att följa ett visst state genom ett system. Ex:

Function	Baka bröd

Input	
Output	Bröd: bakat
Preconditions	
Resources	
Time	

Function	Köpa bröd
Input	
Output	Bröd: köpt
Preconditions	Bröd: bakat
Resources	Pengar: > 300
Time	

2.1.4 Instans

Om nu en FRAM är en modell av ett system så är en instans en ögonblicksbild av ett system. I denna instans kan ett state endast ha ett värde.

Detta kan exemplifieras genom följande instans. I denna instans så har vi gett bröd värdet bakat, detta kommer att göra att funktionerna baka bröd och köpa bröds argument kommer att bli sanna och de kan då bli aktiva. Då ett state endast kan ha ett värde i en enskild instans så kan ej funktionen ”slänga gammalt bröd” bli aktiv då dess input ej är sant.

Function	Baka bröd
Input	
Output	Bröd: bakat
Preconditions	
Resources	
Time	

Function	Köpa bröd
Input	
Output	Bröd: köpt
Preconditions	Bröd: bakat
Resources	Pengar: > 300
Time	

Function	Slänga gammalt bröd
Input	Bröd: mögligt
Output	Bröd: slängt
Preconditions	
Resources	
Time	

2.2 Agila utvecklingsmetoder

En agil utvecklingsmetod är lättrörlig och flexibel metod avsedd att göra det lättare att programmera i en snabbt föränderlig värld där krav och mål kan skifta snabbt. XP (eXtreme Programming) är en av många agila utvecklingsmetoder Några av grundstenarna med XP är:

- XP är lätt, i XP gör du endast det som behövs för att skapa värde för kunden.
- XP är en metod baserad på att lösa brister i mjukvaruutveckling.
- XP kan fungera i grupper av alla storlekar.
- XP anpassar sig till vaga eller ständigt ändrade krav.

XP har en filosofisk grund att stå på där värden som god kommunikation, enkelhet, återkoppling, mod och respekt är de övergripande målen. Sedan under detta så kommer ett antal principer och praktiker avsedda att konkretisera värdena. Allt ovan (Beck & Andres, 2004)

XP kan vid en första anblick se ut som en användardriven utvecklingsmetod men flera problem finns. Constantine (2001) har i sin forskning funnit att XP fungerar mycket bra som metod när det ej rör GUI-applikationer. När det kommer till gränssnitt så verkar XP som metod luta sig mot att producera ett minimalistiskt gränssnitt som sedan under iterationerna ska utvecklas till det bästa tänkbara. För att lösa detta problem föreslås en metod inkorporerar användardriven design och en agil utvecklingsmetod som består av tio steg där värderande, skissande, uppskattande avlöser varandra. (Constantine, 2001)

Constantine säger att en helt agil och ickeprediktiv process ibland kan vara en nackdel och att det kan vara bra att ha vissa prediktiva element med i processen för att bättre kunna styra utvecklingen och förutse eventuella problem. (Turk, France & Rumpe, 2002)

En av praktikerna i XP för att strukturera och prioritera bland kraven är att skriva ner kraven på en liten lapp (story). Kraven ska vara funktionalitet som kunden kan se nyttan i. På lappen skrivs en titel och en kort beskrivning samt en uppskattning av tidsåtgång som behövs för att implementera en passande lösning.(Beck & Andres, 2004)

För att kunna prioritera bland alla stories som man arbetar fram så föreslår Cohn att man

skapar teman av stories med närliggande innehåll. Detta för att det är lättare att bedöma tidsåtgången för ett större sjok med stories än en enskild story i sig. Teman ska byggas så de blir diskreta enheter med stories som rör funktionalitet som ökar användar- eller kundvärden. På detta sätt blir det möjligt att beräkna hur mycket ett visst tema kan bidra i ökande försäljningar genom att underlätta för användarna och man kan då lättare ta ställning om det är värt att implementera temat. (Cohn, 2005)

I en diskussion mellan Kent Beck och Alan Cooper så säger Cooper att tidig programmering endast producerar kod (t ex kod för gränssnittet) som i senare stadier inte går att ändra på. Kod dör inte utan fortsätter att leva vidare, oavsett hur temporär den var menad från början. Det är detta som Cooper vill ändra på med att låta en interaktiondesigner utföra sitt arbete först så en plan eller ritning utarbetas. För att se till att det verkligen är rätt problem löses. Användaren eller kunden vet ofta inte själva vad som är problemet utan de kan endast uttrycka vad de upplever som det mest närvarande problemet. (Nelson, 2002)

Jeff Patton och hans grupp har implementerat det Cooper och Beck diskuterade och finner att det är plausibel lösning på problemet. (Patton, 2002)

2.3 Interaktionsdesign

I detta kapitel kommer en kort introduktion till de övergripande teorierna som används som grund för det kommande utvecklingsarbetet. Metoden jag ämnar använda mig av är en blandning av interaktionsdesign och en agil utvecklingsmetod. Detta då jag själv erfarit i ett flertal utvecklingsprojekt att de agila utvecklingsmetoderna inte lyckas fånga det mest grundläggande i användandet.

2.3.1 Processen

Processen i interaktionsdesign som jag tänker använda mig av hämtar jag mycket från Löwgren och Stolterman och denna del är hämtad därifrån (Löwgren & Stolterman, 2004). Design är en process som växer fram under processens egen gång. Under processen så måste man reflektera över de resultat som framkommer och kunna värdera dem enligt lämpligt mått (se brukskvaliteter nedan). I det reflekterande synsättet ingår även att man måste kunna omvärdera de resultat man tidigare fått om kontexten och förutsättningarna ändrats och kunna arbeta mot nya mål. I en designsituation så växer problemet och lösningen fram tillsammans. Ett problem är inte färdigformulerat utan omformuleras eftersom man arbetar sig framåt i processen.

Som designer är det viktigt att man kritiskt granskar det problemet som uppdragsgivare framställer åt en. Är det verkligen det egentliga problemet eller finns det något mer bakomliggande? Att fråga varför är att ifrågasätta gamla idéer och tankebanor och öppna för nya idéer och tankar. (Löwgren & Stolterman, 2004)

Detta ger designern en möjlighet att se om uppdragsgivaren har tänkt på problemet så långt som det är möjligt. Detta för att se om det finns möjliga lösningar som ligger närmre till hands än vad man kan se, för lever man med problemet så kan det vara svårt att se problemet ur en annan vinkel.

2.3.2 Brukskvaliteter

Med begreppet brukskvaliteter, eller bruksvärden, så vill man fånga alla de kvaliteter en artefakt har i en viss brukssituation, dvs när den används. Brukskvaliteter är således ett subjektivt begrepp som skapas av designern men som värderas av användaren i stunden. (Löwgren & Stolterman, 2004)

Detta gör man för att få en bild av vilka kvaliteter som användarna värdesätter och hur de står i relation till varandra.

2.3.3 Förebildsanalys

För att kunna skapa en bra design så är det bra att ha en överblick över vilka artefakter som finns idag och vilka styrkor och svagheter dessa har. Detta kallas för förebildsanalys och är något som Löwgren och Stolterman pratar om när det nämner att man ska ha en repertoar av exempel i sitt designarbete. (Löwgren & Stolterman, 2004)

En förebildsanalays utgår alltid ifrån det givna designproblemet då varje problem är unikt. En förebildsanalys går ut på att hitta lösningar som kan uppfylla de brukskvaliteter man eftersträvar med den tänkta lösningen. Med en lösning så kan man mena både en rent funktionell lösning som löser en specifik uppgift, men även på ett högre plan, hur lösningen bidrar till att ge förebilden en viss känsla eller liknande. En förebildsanalys kan vara av mer explorativ art för att sondera bredden av möjliga lösningar eller av mer riktad karaktär för att fokusera på ett visst område eller lösning.

2.3.4 Affinitetsdiagram

Affinitetsdiagram är ett sätt att strukturera och gruppera idéer i större enheter. Processen går i kort till som följer:

1. Skriv ner varje idé på ett kort
2. Leta efter idéer som verkar vara närbesläktade
3. Placera kort som hör ihop, gör detta tills alla kort är sorterade

(Beyer & Holtzblatt, 1999)

2.3.5 Tänka högt

Tänka högt handlar om att få användaren att externalisera tankar och processer så att man

kan få reda på hur de tänker. Detta går vanligtvis till som så att användaren blir ombedd att prata samtidigt som den utför en viss uppgift. Man får insikt i vilken strategi som användaren nyttjar för uppgiften.(Sharp, Rogers & Preece, 2007)

2.3.6 Hierarkisk målanalys

En hierarkisk målanalys är en analysmetod där man omvandlar abstrakta mål till mer konkreta mål. Detta används bland annat för att konkretisera övergripande designmål. Analysen slutar i en graf som går från vänster till höger. I den vänstra kanten är de övergripande målen. I den högra de konkreta målen. Dessa noder binds ihop med noder och bågar för att visa vilka abstrakta och konkreta mål som sitter ihop, går man från vänster till höger i grafen får man reda på hur något ska genomföras, går man från höger till vänster hur det ska genomföras. (Jones, 1992)

2.4 Kanos modell om kundnöjdhet

För att kunna ranka de olika temana så har jag valt att använda mig en modell skapad av Kano.

Kano har skapat en modell för kundnöjdhet. Denna modell används för att gruppera olika egenskaper hos en artefakt eller tjänst. Detta i sin tur för att lättare kunna prioritera vad man ska utveckla av en produkt. I modellen så delas en artefakts kvaliteter upp i tre olika grupper av egenskaper, måste-ha, mer-är-bättre och attraktiv-användbarhet (översättning av Johan Blomkvist (Blomkvist, 2007)). (Cohn, 2005):111

Jokela finner i sin artikel att Kanos modell kan vara applicerbar på relationen mellan användbarhet och kundnöjdhet (jag väljer här nedan att prata om användarnöjdhet). Jokela använder Kanos modell som ett sätt för att hantera och förstå relationen mellan artefakters kvaliteter och användarnöjdhet. Måste-ha-egenskaper är något som en artefakt i princip är tvunget att ha för att en användare ska acceptera artefakten. Om måste-ha-egenskaperna ej finns så blir användaren snabbt och mycket besviken på artefakten och det kan även inverka på användarens fortsatta upplevelse av

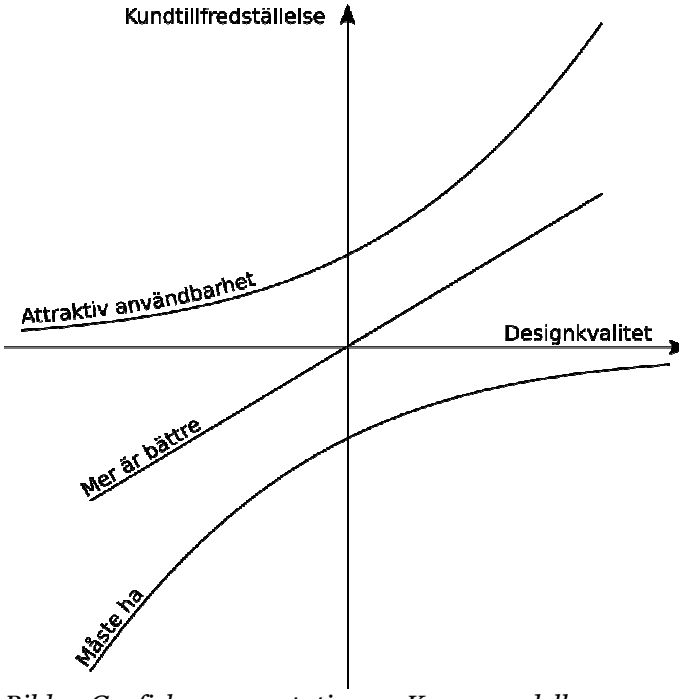


Bild 3: Grafisk representation av Kanos modell

artefakten. Mer-är-bättre-egenskaper har en linjär påverkan på användarnöjdheten (se bild 3). Attraktiv-användbar är de egenskaper som särskiljer en artefakt som gör att den står ut från de övriga och kan om de används på rätt sätt ge en hög användarnöjdhet. Det verkar som att dessa egenskaper ej kan vägas samman till ett slutligt mått utan att de tre egenskaperna är ortogonala i sin natur. Måste-ha-egenskaper måste finnas i artefakten, attraktiv-användbarhetsegenskaper kan ej kompensera för brister i detta. Att endast uppfylla måste-ha-egenskaperna leder inte automatiskt till användarnöjdhet utan endast till en icke missnöjdhet, vill man uppnå nöjdhet så är mer-är-bättre-egenskaperna de egenskaper man ska främja.(Jokela, 2004)

Kano föreslår själv en metod för hur man ska applicera hans/hennes modell. Man ställer två frågor till de presumtiva användarna, en fråga om hur användaren ställer sig till att egenskapen finns med, och en om hur de ställer sig till om egenskapen inte finns där. Användaren får gardera sitt svar på en femgradig skala.

1. Jag gillar det
2. Jag förväntar mig att det är på det sättet
3. Jag är neutral
4. Jag kan leva med det så
5. Jag ogillar det

Svaren sammanställs och korserefereras genom en tabell.

Sedan sammanställer man svaren och får fram en fördelning och ser hur många procent av användarna tycker att en viss egenskap är av den ena eller andra sorten. (Cohn, 2005)

Som exempel på hur olika egenskaper kan värderas så kan följande exempel ges.

Om man ska gå och köpa en bil så förväntar man sig att bilen i fråga har ratt, dörrar, hjul och motor. Har den inte det så blir besvikelsen stor och det osäkert om man ens kan kalla det för en bil längre. Men finns sakerna där så blir man inte överförtjust, men man blir inte besviken heller. Om man upptäcker att bilen har elhissar, mp3-spelare och färddator kanske man blir glatt överraskad. Ens förtjusning över bilen växer. Men finns även automatisk fickparkeringsfunktion, gps och magnesiumfälgar så kanske blir överförtjust och imponerad över alla finesser. Attraktiv-användbarhetsegenskaperna kan användas för att locka till köp men saknas måste-ha-egenskaperna så blir kunden snabbt besviken på produkten.

3 Metod

Att samtidigt utveckla ett program och presentera resultatet kring denna utveckling kan vara svårt, speciellt som det i detta fallet rör sig om ett arbete över flera iterationer. I detta kapitel presenteras arbetet som utförts och de resultat som uppnåtts efter varje iteration.

3.1 Användare

Användare i utvecklingsarbetet har varit studenter och doktorander som är intresserade av FRAM och som känner till metoden. Användarna var datorvana och bekanta med program som Excel, Framvis, Visio etc. Totalt sett har sex olika personer deltagit i försök och workshops.

3.2 Utvecklingsarbete

Utvecklingsarbetet har löpt i treveckorsiterationer, detta då arbetet utförts ensam och det tar naturligt längre tid än om man är en grupp. Varje steg bestod som grund av:

1. En workshop eller användarstudie där relevanta frågor ställdes.
2. Resultatet av det tidigare steget analyserades och utvärderades enligt lämplig metod
3. Ett affinitetsdiagram och en gruppering av användarkraven skapades, eller modifierades från föregående steg. Dessa prioriterades i enlighet med Kanos modell.

Dessa steg kommer att beskrivas utförligt för iteration 1 och endast avsteg från den kommer att redovisas i de följande iterationerna.

Vissa avsteg från den traditionella XP-metoden har gjorts, t ex så har parprogrammering varit omöjligt. Även testdriven programmering har tagits bort ur processen, detta då interaktion med ett GUI inte behöver starta ett funktionsanrop och det blir då svårt att testa de saker som rör ren interaktion. Men praktiker från XP har använts så som t ex stories för att skriva ner krav och kodrevisionssystem för att snabbt kunna backa om fel uppstår i koden.

För att göra utvecklingsprocessen snabb och smidig valdes att arbeta med Python som programmeringsspråk och QT som grafiskt ramverk. Detta då Python är ett dynamiskt starkt typat språk och erbjuder ett språk som lämpar sig för prototypande och snabb utveckling. QT erbjuder en stor bredd av olika klasser om allt från grafik till nätverk och databaser. Det är även plattformsoberoende och kan användas på såväl Windows som Mac OS/X som X11.

3.2.1 Initial workshop

I ett tidigt skede i studien hölls en workshop med användare av Framvisualizer (hädanefter Framvis). Denna workshop hölls på en konferens om FRAM och deltagarna var yrkesarbetande, studenter eller doktorander som samtliga arbetade med risk- och olycksanalys. Antalet deltagare var ca 25 personer. Syftet med workshopen var att ta reda på

hur tid kunde modelleras i FRAM. Detta gjordes genom att ett antal skisser visades och det diskuterades runt dessa skisser, hur de stödde FRAM som metod, vad som var möjligt, vad som var bra eller dåligt.

Ett annat syfte med workshopen var att utröna hur Framvis användes, vad användarna fann bra och vad som de fann dåligt. Synpunkter som lyftes fram var att Framvis till vissa delar var styrande i sitt sätt att fungera och att det vore bra med ett friare verktyg. Det nämndes även att verktyget inte ska driva eller hindra FRAM som ramverk utan tvärtom.

3.2.1.1 Resultat

Ett krav som framkom var att det vore bra att ha ett öppet dataformat som gjorde det möjligt att flytta data till andra applikationer. Det skulle även vara önskvärt att själv kunna skriva script eller program för att komma åt viss specifik data. Utifrån detta krav så skapades en sql-databas och resultatet av detta kan ses i form av ett databasschema, se kapitel 4

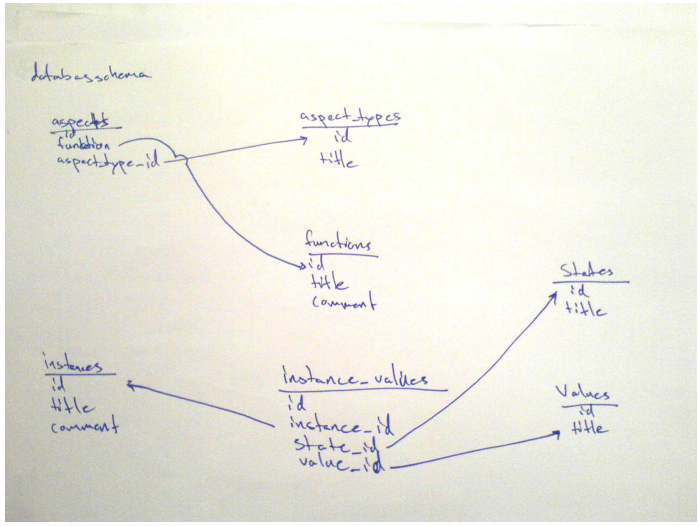


Bild 4 Databasschema

3.2.2 Iteration ett

Då en iterativ metod föder sig själv med material så kan det vara svårt att dra klara gränser mellan vad som är utvärdering och utveckling. I detta fall så ändrades användarstudien löpande då intressantare frågor väcktes under de tidiga användarstudierna i denna iteration.

3.2.2.1 Förebildsanalys och användarstudie

Detta steg gjordes tillsammans med användarna för att se hur de använde programmen. Detta genomfördes på en bärbar dator med GNU/Linux Kubuntu. Användartestet utfördes enligt tänka-högt-metoden, detta för möjligheten att ställa löpande frågor under testet. Testerna spelades även in med programmet recordmydesktop (http://recordmydesktop.iovar.org/,).

Det var tre användare som fick testa Excel, Framvis (svn-revision 218) och Kexi. Kexi är ett databasprogram för skrivbordet likt MS Access. Anledningen till att det var just dessa program som valdes var att de representerar program som är likvärdiga det tänkta verktyget. Användarna var alla vana datoranvändare och gick samtliga på det kognitionsvetenskapliga programmet i Linköping och hade vana av risk- och olycksanalyser. Förebildsanalysen gick

till som så att användarna fick lösa en lätt uppgift för varje tänkt program. För kalkylbladsprogrammen så var det att skapa en tabell med lite olika värden i, summera vissa kolumner, skapa en graf och spara en fil. För Framvis så var det två av deltagarna som hade använt sig av programmet tidigare och en var även med i den ursprungliga utvecklingen. Testet gick ut på att deltagarna fick programmet demonstrerat för sig med en enkel uppgift av ett sekventiellt system för att baka, transportera, sälja och äta bröd [Bild 1]. I Kexi fick användarna skapa en enkel databas, fylla i data, försöka söka efter data och sedan spara databasen i en fil. Resultatet av förebildsanalysen blev en lista över funktioner som användarna behövde för att kunna utföra uppgifterna, denna lista arbetades fram under själva användartestet.

Efter testet fick användarna gradera de nyss nämnda funktionerna enligt samma skala som nämns i kapitel 2.4 Dessa funktioner bröts sedan ner i user stories för vidare utveckling. Prioriteringen gjordes efter vilken grupp en viss story tillhörde. Måste-ha prioriterades högst och sedan måste-ha och sist attraktiv-användbarhet.

Egenskaper som användarna uppskattade var inmatning av data i Kspread/Excel då det var följsamt och gick att använda både mus och tangentbord, kortkommandona var logiska och gjorde att musinteraktionen blev onödig enligt vissa användare. I Kexi fann en användare att man faktiskt kunde ställa SQL-frågor direkt mot den inmatade datan, vilket möjliggör att man kan skapa sina egna frågor och vyer och inte behöver vara beroende av det gränssnitt som den som skapat databasen tillhandahållit.

Användartesterna analyserades och brukskvaliteter togs fram. Bruksvärdena som var mest framträdande var att man skulle ha ett *flöde* och kunna arbeta *otvunget*. Det som efterfrågades var ett program som inte hindrade en i ens naturliga arbete utan mer fanns där som ett stöd i den processen. Detta var även samstämmigt med vad som framkommit under den inledande workshopen. Utifrån brukskvaliteterna så gjordes en hierarkisk målanalys för att tydliggöra vad som krävdes för att uppnå de givna brukskvaliteterna. Detta gjorde även att brukskvaliteterna fylldes med mer innehåll och gick från att vara abstrakta till mer konkreta och praktiska.

3.2.2.2 Utveckling

Skisser för hur inmatning av data i tabellformat producerades där hela flödet, från start av program till färdig tabell var med. Interaktionsscheman för tangentbordskombinationer ritades för att se till att de basala bitarna kom med. När en del var färdigskissad så testades olika implementationsval för att se om det skissade var praktiskt genomförbart och hur det fungerade i en brukssituation. Parallellt med detta gjordes även UML-diagram för pythonklasserna och databasscheman för samtliga tabeller för att se vilken representation som skulle kunna fungera.

Fokus för utvecklingen lades på tabellinmatningen då det rent logiskt sett måste finnas en tabell innan en instans och en graf kan skapas. Detta var även vad som framkommit i den initiala workshopen och användarstudien.

3.2.2.3 Resultat av iteration 1

Resultatet kan delas upp i tre delar, de användarkrav som arbetades fram, den implementation som gjordes och de metodresultat som uppkom. I detta avsnitt presenteras användarkraven och implementationsresultatet. Metodresultatet presenteras i resultat/diskussion.

Följande egenskaper hittades och kategoriserades i denna iteration

Krav	**Typ av krav (enligt Kano)**
Filoperationer	Måste-ha
Interopabilitet	Måste-ha
Klipp och klistra	Måste-ha
Ångra	Måste-ha
Autokomplettering	Måste-ha
Flytta funktioner	Mer-är-bättre
Scriptande	Attraktiv-användbarhet

Tabell 1Funna funktioner och deras kategorisering

Mappat till programmen i förebildsanalysen blir fördelningen följande

	Excel	**Kexi**	**Framvis**
Filoperationer	X	X	-
Navigering	X		-
Klipp och klistra	X		-
Ångra	X		-
Autokomplettering	X	X	X
Scriptande		X	-
Flytta funktioner	-		X

Tabell 2 Funna funktioner i förebildsanalysen mappat till varje enskilt program

Här ovan ser vi en tabell som visar i vilket förebild som användarna hittade i vilken funktion. Ett kryss indikerar att här fanns funktionen, ett minus att det inte fanns och upplevdes som en nackdel.

Slutresultatet rent implementationsmässigt blev ett väldigt basalt verktyg som hade stöd för inmatning, autokomplettering av inmatningen. Man kunde även skapa väldigt enkla

instanser, se *Bild 5*

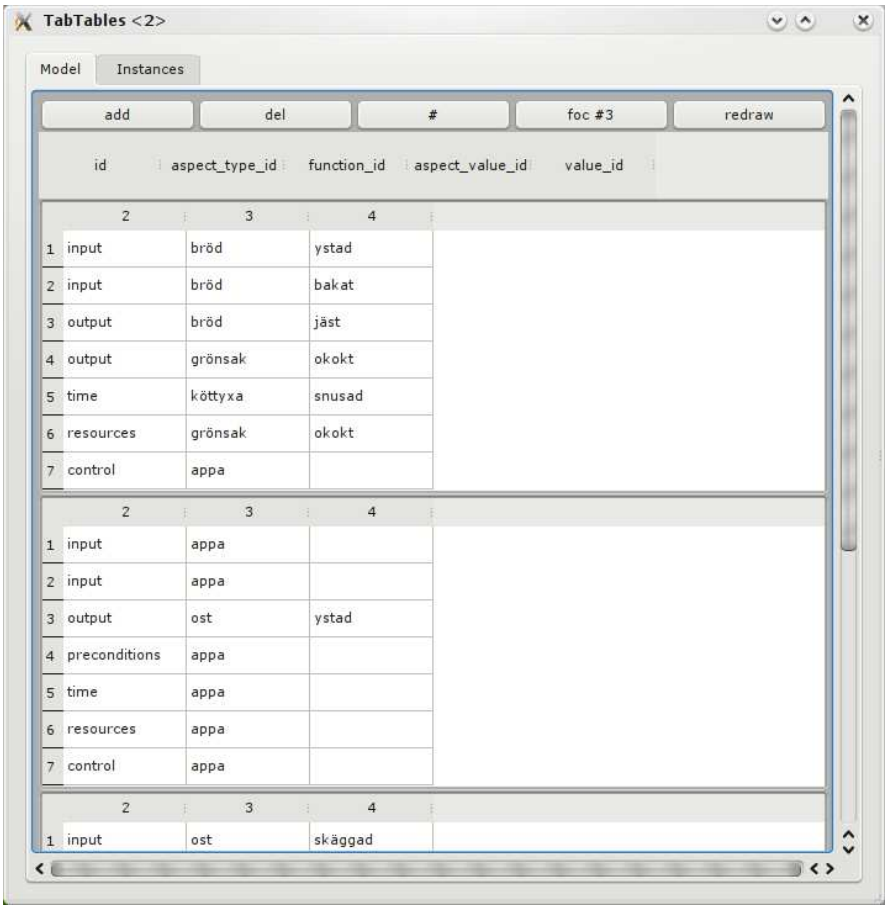


Bild 5 Ett enkelt verktyg för att skapa FRAM- funktioner

3.2.3 Iteration två

3.2.3.1 Utvärdering av föregående prototyp

Prototypen från den föregående iterationen utvärderades genom att jämföra de implementerade funktionerna med de i listan över basfunktioner och hur det var tänkt att de skulle fungera. Vissa implementationsval visade sig nu vara felaktiga, t ex att dela av varje funktion i en egen tabell. Detta val gjordes ursprungligen för att underlätta separeringen av datan mellan de olika funktionerna och underlätta t ex flytt av funktionerna som var ett krav som framkom. Det fanns fördelar med denna implementation men även nackdelar, som att en helt egen klass för att hantera fokusändring med piltangenterna, sedan kunde inte mer än en funktion åt gången markeras och kopieras, vilket var måste-ha-funktioner identifierade i iteration 1. En fördel med detta upplägg var att man i QT kan koppla en hel databastabell eller del av en databastabell till en tabell i gränssnittet (QsqlTableModel kombinerat med QTableView), detta gör att man helt slipper ta hand om uppdatering etc av databasen utan delegerar detta åt QT. Detta gjordes dels som en expertutvärdering men även som en användarstudie med två användare från den tidigare iterationen.

3.2.3.2 Förebildsanalys

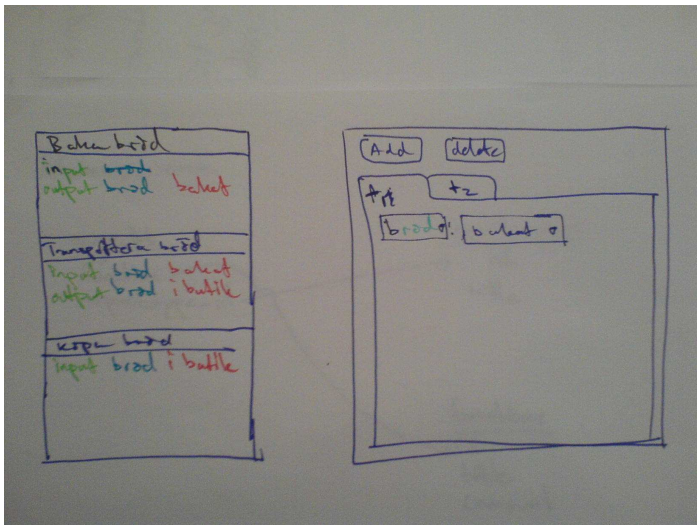


Bild 6 Skiss på FRAM-tabell (t v) och en instanstabell (t h)

I denna iteration utfördes förebildsanalysen som en expertutvärdering. Andra artefakter utvärderades för att se hur de funktionellt sett matchade de tidigare testade artefakterna och om det fanns en annan lösning på krav användarna uttryckt. Detta för att få en bättre och djupare bild av möjliga lösningar. I denna del så hämtades inspiration om hur datainmatning sker och hur tangentsbordskommandon fungerar. Detta för att försöka få dessa att likna den standard som ändå kunde anas bland förebilderna. Här hämtades även lämpliga och bra lösningar på problem och stoppades in i den hierarkiska målanalysen. Detta för att visa vilka lämpliga lösningar det fanns och hur de kunde hjälpa att uppnå brukskvaliteteterna. Program som utvärderades var återigen Excel och Kexi, men även andra program för mindmapping etc.

3.2.3.3 Utveckling

Utifrån utvärderingen fortsatte utveckling, med fokus på att få en mer följsam datainmatning. Målet var även att få ett fungerande om än basalt sätt att skapa instanser och att generera en graf av detta.

3.2.3.4 Resultat av iteration två

Slutresultatet kan delas upp i tre delar, de användarkrav som arbetades fram (som vi ser i *Tabell 3*), den implementation som gjordes och de metodresultat som uppkom. I detta avsnitt presenteras användarkraven och implementationsresultatet. Metodresultatet presenteras i resultat och diskussion.

Krav	**Typ av krav (enligt Kano)**
Interopabilitet	Måste-ha
Exportfunktioner	Måste-ha
Filtrering	Mer-är-bättre
Nod-i-nod	Attraktiv-användbarhet
Stöd för fler databaser	Attraktiv-användbarhet
Zooma i funktionstabellen	Attraktiv-användbarhet

Tabell 3 Användarkrav från iteration två

Implementationsmässigt så finns nu en tabell (se bild 7) som i mångt och mycket påminner om ett ordinärt kalkylblad, men med autokomplettering. I vänstra kolumnen (Aspect) så är kompletteringen låst till de sex olika aspekterna som finns (se avsnitt 2.1 om FRAM), i den i mitten är det autokomplettering av samtliga variabler som finns i tabellen och i den högra (Value) av alla värden för det variabelvärdet. De två sista kolumnerna kan man även skriva fritext i och då utökas värderymden.

En enkel vy för att skapa instanser och instansvärden kom till.

3.2.4 Iteration tre

Iteration tre gick till på liknande sätt som de föregående iterationerna. Två användare fick testa de implementerade valen och under en tänka högt-session säga vad de tyckte. I denna iteration så fokuserades det på skapandet av instanserna och grafen och att få in de krav som arbetats fram.

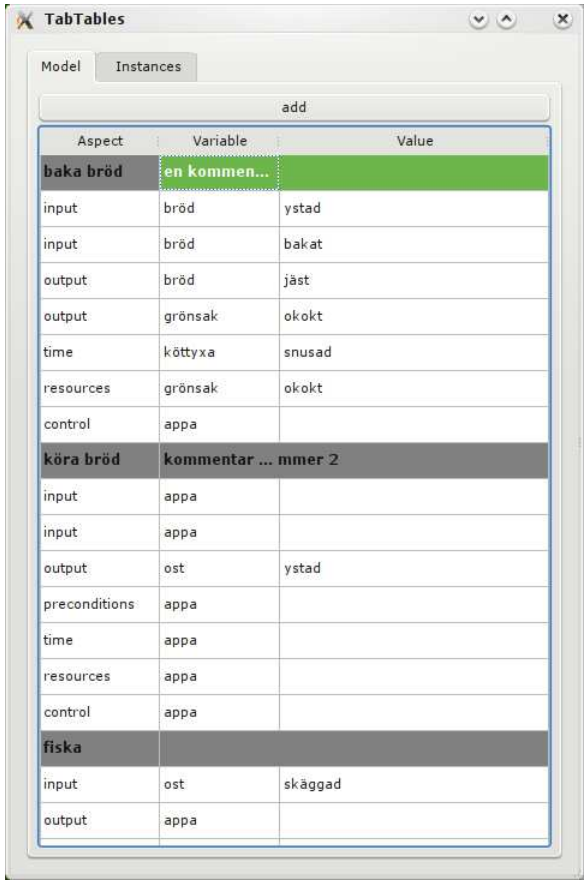


Bild 7: Resultatet av iteration tre, en tabell med stöd för tangentbordsnavigering etc.

Här gjordes även förebildsanalyser kring programmen Framvis, Freemind och CmapTools (för skärmdumpar se kapitel 8). Freemind är ett traditionellt mindmap-verktyg och CmapTools är ett conceptmap-verktyg. En concept map är en friare form av mindmap där användaren själv kan bestämma relationerna mellan olika noder. Två användare deltog i förebildsanalysen som gick till som i iteration ett. De fick i uppgift att skapa en grafisk representation av en skriven relation.

Både i CmapTools som i Freemind så uppskattade användarna att det var enkelt att formatera bågar och noder. En användare tyckte dock att Freemind var begränsande och inte tillät relationer fullt ut utan att det endast gick att skapa i CmapTools.

CmapTools har ett väl inbyggt stöd för att kunna dela kartor med andra och kunna bjuda in andra användare att kunna redigera ens egen karta. Detta såg användarna som en bra funktion men att den var en del av den attraktiva användbarheten.

Med dessa program som grund så skapades kravlistor för vad som

3.2.4.1 Resultat av iteration tre

De krav som arbetades fram i denna iteration kan vi se i tabell *Tabell 4*

Krav	**Typ av krav (enligt Kano)**
Skicka fil	Mer-är-bättre
Statistik	Mer-är-bättre
Zooma	Mer-är-bättre
Gömma bågar	Mer-är-bättre
Formatera noder och bågar	Mer-är-bättre
Nätverksstöd	Mer-är-bättre
Stöd för kollabrativt editerande	Attraktiv-användbarhet
Grammatik för länkprioritet	Attraktiv-användbarhet

Tabell 4 Användarkrav från iteration tre

Resultatet av iteration är slutresultatet på det faktiska utvecklingsarbetet är i och med denna iterations avslutande. Programmet har nu en tabellvy för att mata in data, och en grafvy som ritar ut funktionerna och kopplingarna mellan dem. Resultaten illustreras här av den del av programmet där den grafiska representationen av FRAM-funktionerna visas (se bild 8 på nästa sida).

I tabellen nedan ser vi från vilken förebild de olika kraven eller funktioner härstammar. Ett kryss markerar att funktionen finns i förebilden och ett kryss att någon användare uttryckligen sagt att funktionen saknas. ”Grammatik för länkar” för Framvis har både ett x och ett minus, detta då det finns en inbyggd grammatik som inte kan ändras av användaren.

	Framviz	Freemind	Cmaptools
Zooma	X	X	X
Gömma bågar	X		
Formatera noder och bågar	X	X	X
Nätverksstöd			X
Grammatik för länkar	X[1]		X
Exportfunktioner	-[1]	X	X
Nod i nod			X
Kollabrativt editerande			x

Tabell 5

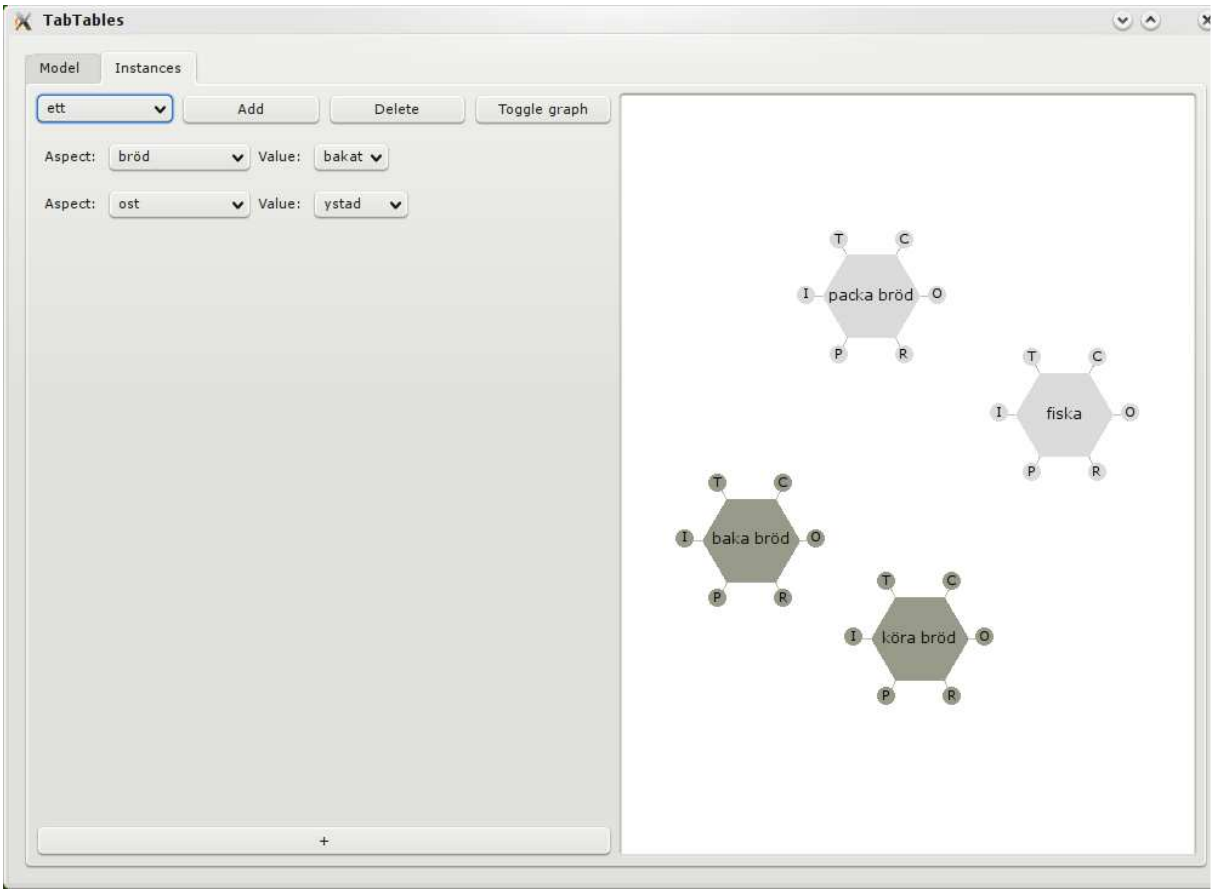


Bild 8: Slutresultat av iteration tre

4 Resultat

Kraven från utvecklingsarbetet kan delas upp i tre olika delar; krav för tabellvyn, krav för grafvyn och övergripande krav som är på ett mer allmänt plan. Här följer tre tabeller där dessa krav presenteras grupperade enligt Kanos modell. Man kan även utveckla dessa krav för att få fram mer detaljerade beskrivningar men detta presenteras ej här.

Måste-ha	**Mer-är-bättre**	**Attraktiv-användbarhet**
Filoperationer	Skicka fil	Nätverksstöd
Interopabilitet		Stöd för fler databaser
		Stöd för kollabrativt editerande
		Nod i nod

Tabell 6 Övergripande egenskaper

Måste-ha	**Mer-är-bättre**	**Attraktiv-användbarhet**
Navigering med mus och tangentbord	Flytta funktioner	Zooma i funktionstabellen
Klipp och klistra	Filtrering	
Ångra	Statistik	
Autokomplettering		
Exportfunktioner		

Tabell 7 Egenskaper för tabellvyn

Måste-ha	**Mer-är-bättre**	**Attraktiv-användbarhet**
Exportfunktioner	Filtrering	Nod i nod
	Zooma	Grammatik för länkprioritet
	Gömma bågar	
	Formatera bågar och noder	

Tabell 8 Krav för grafvyn

Sammanställer vi kraven grupperat på iteration och kategori så blir resultat följande:

Iteration #	**Måste-ha**	**Mer-är-bättre**	**Attraktiv-användbarhet**
Ett	5	1	1
Två	2	1	3
Tre		6	2

Tabell 9 Krav grupperat efter iteration

Som vi ser så är det som man vid första anblicken skulle kallas för ”grundläggande funktionalitet” klassat som måste-ha. Vi ser även att det i iteration två blir mer vikt på mer-är-bättre och attraktiv-användbarhet och att detta förskjuts än mer i iteration tre. Detta är intressant ur flera aspekter då det visar på att användarna ej har identifierat de mer avancerade kraven i de tidigare faserna av utvecklingen utan att det är något som växer fram fall eftersom. Kanos modell har varit till hjälp för att hitta en bra klassificering kring detta och jag tror att resultatet visar på i vilken prioritet funktioner ska implementeras. Denna prioritet tror jag ej hade kunnat uppnås utan Kanos modell.

Som ett steg i att få en portabilitet mellan Finit och andra program så utvecklades en SQL-databas, här nedan presenteras databasschemat för den databasen.

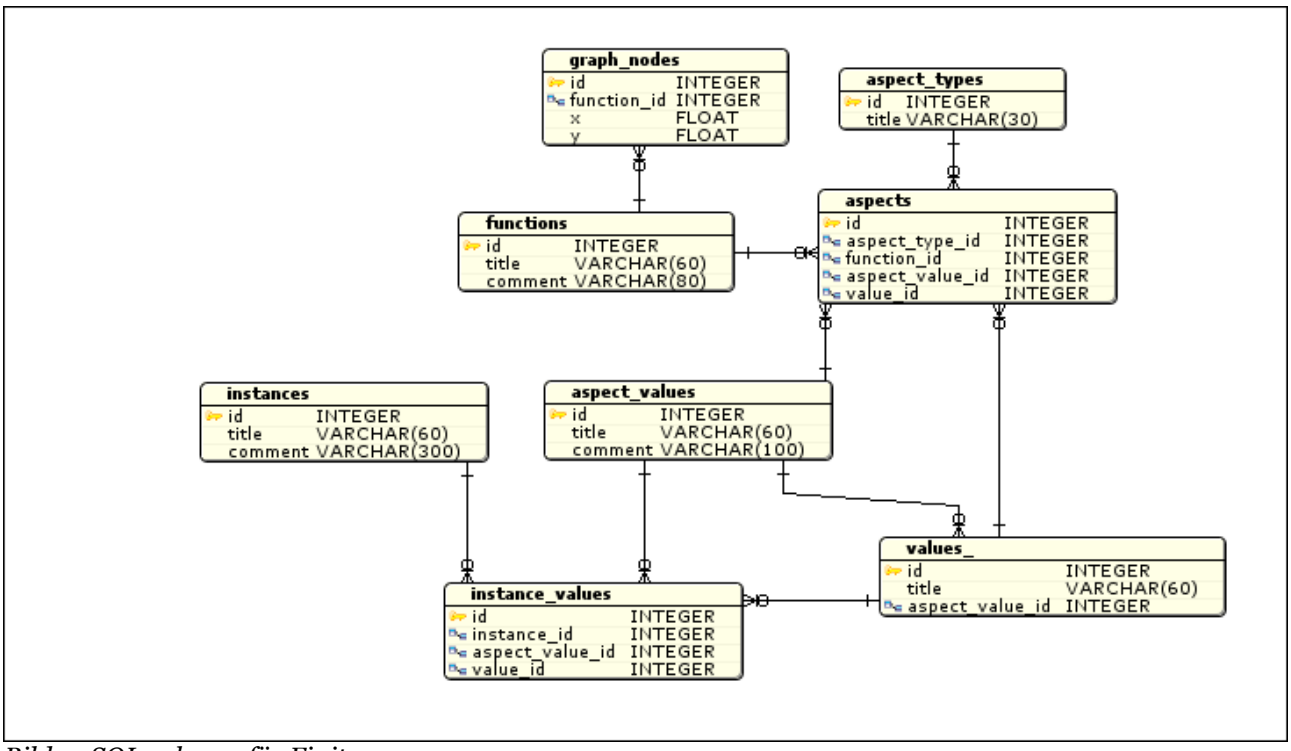


Bild 9: SQL-schema för Finit

5 Diskussion

I detta kapitel ges två diskussioner, dels en rörande själva metoden och en rörande resultatet.

5.1 Metoddiskussion

Kano ger i sin metod ingen vägledning till hur frågorna som ska ställas till användarna ska tas fram. Frågorna i sig kan vara helt subjektiva och enbart belysa de områden som utvecklarna själva finner intressanta. Det finns i metoden inget självklart sätt för användarna att interagera för att själva kunna bidra. Detta finner jag vara en nackdel för just ändamålet med denna uppsats men det är även något som jag tycker att jag har arbetat mig igenom på ett bra sätt i och med min metod med förebildsanalyser och på det sättet de genomfördes.

I mitt arbete skulle jag kunna ha använt mig av Kanos korsreferensmetod för att ta märka ut egenskaperna och därmed få en klarare bild av hur egenskaperna spelar mot varandra. Kanos metod förutsätter dock att man redan vet vilka eventuella egenskaper som kan finnas med i en produkt, vilket inte alltid är fallet vid en nyutveckling. Dock hade detta varit möjligt om jag hade använt mig av metod i fler steg, att först med en förebildsanalys ta fram egenskaper och sedan efter detta låtit användare värdera dem efter Kanos skala.

Jag anser efter att ha arbetat med Kanos modell att den kanske inte helt är tillämpbar på lågnivå på användarinteraktion utan att den kanske gör sig bättre på ett högre plan för att beskriva övergripande mål med interaktionen. Detta tar sig uttryck i att det kan vara svårt att gradera små delar av ett program separat då varje del i sig bidrar till en helhet.

5.2 Resultatdiskussion

Som vi kan se i resultattabellerna så har ganska lite av de slutliga kraven berört FRAM som metod. Väldigt lite i de framarbetade kraven är krav på stöd i själva utvecklandet av en FRAM-tabell utan det berör mer det rent interaktionsmässiga och grundläggande. Detta kan mycket väl att göra med att det var relativt få iterationer i arbetet och att det troligtvis skulle ha sett annorlunda ut längre fram i processen. Så om man har kort om tid att utveckla ett program kanske denna metod är att föredra, har man gott om tid och resurser så kommer man troligtvis att finna de flesta mer-är-bättre-krav då användarna fått använda programmet tillräckligt länge och därmed upptäckt att de saknas.

I resultatet så kan vi se att Framvis har många fler egenskaper som är kategoriserade som mer-är-bättre än i måste-ha. Detta kan ha sin grund i XP som metod, att inte koda förrän användaren säger så och att inte göra mer än att det uppfyller enhetstesterna. Om man även har en begränsad budget i form av tid eller pengar kan det kanske bli som så att de basala funktionerna måste komma efteråt då det kanske ses om helt självklara som inte behöver specificeras. Nu kanske du ställer frågan, är det inte bara att skriva ett test och koda om? Det

finns väl två svar på den frågan. Ja, teoretiskt är det, det skulle gå att skriva nya test och helt skriva om programmet för att uppfylla dessa. Jag håller med Cooper när han säger att skriven kod kommer att leva vidare, man är inte alltid beredd att offra skriven och implementerade funktioner, resurserna sätter ofta stopp för detta. Så för att komma tillrätta med detta så man måste ha en någorlunda riktning i åt vilket håll man ska. Nej, att skriva enhetstester som gäller interaktion blir sällan bra, det är inte säkert att en mus- eller tangentbordsinteraktion utlöser en händelse av något slag och kan därmed inte kontrolleras av ett enhetstest om det var lyckat eller ej.

I resultatet så ser vi även att Finit har betydligt fler måste-ha uppfyllda än Framvis. Detta kan ha sin förklaring i att dessa krav togs fram för det här arbetet och var av måste-ha-karaktär vilket gjorde att de blev implementerade först. Många av kraven skulle säkerligen kunna implementeras i Framvis men med stora bekymmer. Kravet på interopabilitet som var ett högst ställt krav från användarna blir extremt svårt att implementera, detta då vissa val i Framvis gör detta svårt. T ex så sparas all data i ren serialiserad form (binärform) i en textfil. Detta gör det svårt då intern datastruktur och filformat är samma, vilket gör svårare att ändra något av dem. Detta är ett exempel på att det är viktigt att planera sin utveckling anser jag. För detta problem gör att Framvis nu i princip en återvändsgränd för de som idag använder programmet. Datan är inlåst i ett binärt format och alla modeller måste göras om i ett nytt verktyg.

Kan man då hitta de där attraktiva-användbarhetsegenskaperna med hjälp av förebildsanalyser? Kanske inte direkt skulle mitt svar vara. Man kan hitta inspiration på lösningar med metoden men metoden begränsas av de förebilder (och då även metafor) man väljer. För som vi ser så kanske det är lättare att se de gemensamma dragen som program än att se hur en viss funktion kan användas för att lösa ett visst krav.

Om jag skulle ta och jämföra de implementerade kraven i Finit med de som finns tillgängliga i Framvis skulle vi få en tabell som följer. Detta är inte på något sätt en komparativ studie utan jag vill endast visa på skillnaden mellan den metod jag använt mig av och en vanlig agil metod.

Måste-ha	**Finit**	**Framvis**
Filoperationer	X	X
Interopabilitet	X	
Navigering med mus och tangenter	X	
Klipp och klistra	X	
Ångra		
Autokomplettering	X	X

Exportfunktioner	X	
Mer-är-bättre		
Skicka fil		
Flytta funktioner		X
Filtrering		X
Statistik		
Zooma	X	X
Gömma bågar		X
Formatera noder och bågar		X
Attraktiv-användbarhet		
Nätverksstöd		
Stöd för fler databastyper	X	
Nod i nod		
Zooma i funktionstabellen		
Grammatik för länkprioritet		x

Tabell 10 X markerar implementerad funktion

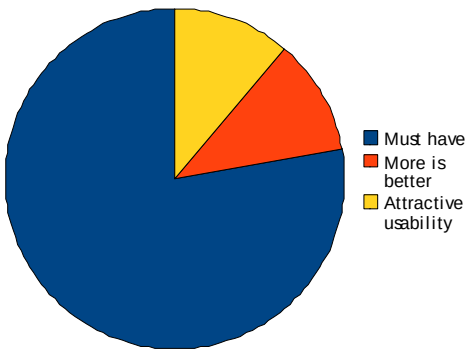
En sammanställning av denna tabell ger följande resultat

	Finit	Framvis
Måste-ha	7	2
Mer-är-bättre	1	5
Attraktiv-användbarhet	1	1

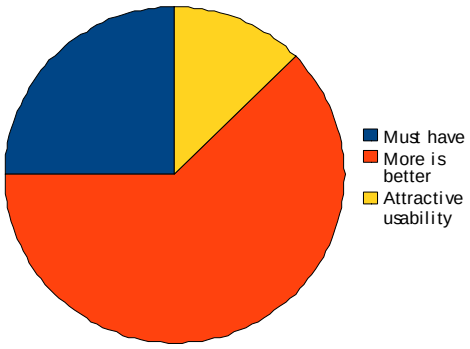
Tabell 11

I en graf kan vi tydligt se skillnaderna mellan implementationen i Finit och Framvis. Finit har betydligt fler måste-ha uppfyllda än Framvis, med Framvis har fler mer-är-bättre.

Figur 1: Graf för Finit



Figur 2: Graf för Framvis



Två grafer för att än tydligare visa på skillnaden mellan de två programmen.

Denna skevhet i vad som är implementerat behöver inte betyder mer än att i utvecklingen i denna uppsats så började det med basala verktyg som Excel och sedan utvecklades till mer avancerade förebilder. Detta stöds även att resultatet i ***Tabell 9 Krav grupperat efter iteration***, där man tydligt ser att kraven växt fram mot mer avancerade krav under processens gång.

6 Framtida forskning

Ett framtida område att utforska skulle vara vilka egenskaper i FRAM-metoden som är av mer-är-bättre etc karaktär. Dvs, utforska själva metoden och försöka identifiera de delar som är mer fundamentala och vilka som man senare kan arbeta med.

Ett annat område som skulle vara intressant skulle vara att se om kraven från användarna skiftar över tid, dvs om de efterfrågar en viss sak i början och om detta ändras längre fram i arbetet. En möjlig tanke är att användare i början vill ha måste-ha-egenskaper men att det sedan efter en tid växlar över till mer-är-bättre för att sedan gå över till attraktiv-användbarhet. Detta skulle kunna vara en möjlig utveckling och det vore intressant att se detta i ett längre perspektiv.

Som vi ser i resultatet så är de eftersökta egenskaperna väldigt få som faktiskt berör FRAM, även här vore det intressant om kravbilden ändras över tid. Om de tidiga kraven är mer generella och senare växlar över till att vara mer specifika, för just den metoden som programmet ska stödja.

För att underlätta för framtida forskning så finns all källkod lätt tillgänglig på http://bitbucket.org/peppelorum/finit/

7 Referenser

Beck, K. & Andres, C. (2004). Extreme Programming Explained : Embrace Change (2nd Edition). {Addison-Wesley Professional}.

Beyer, H. & Holtzblatt, K. (1999). Contextual design. *interactions*, *6*, 32-42.

Blomkvist, J. (2007). Genreanalys som en del av designprocessen, Linköpings universitet

Cohn, M. (2005). Agile Estimating and Planning (Robert C. Martin Series). {Prentice Hall PTR}.

Constantine, L. L. (2001). Process Agility and Software Usability: Toward Lightweight Usage-Centered Design.

Hollnagel, E. (2004). Barriers and Accident Prevention. Aldershot, UK: Ashgate.

Jokela, T. (2004). When good things happen to bad products: where are the benefits of usability in the consumer appliance market?. interactions, *11*, 28-35.

Jones, J. C. (1992). Design Methods (Architecture). Wiley.

Löwgren, J. & Stolterman, E. (2004). design av informationsteknik - materialet utan egenskaper. Studentlitteratur.

Nelson:
http://web.archive.org/web/20030621112434/http://www.fawcette.com/interviews/beck_cooper/default.asp

Patton, J. (2002). Hitting the target: adding interaction design to agile software development. , , 1-ff.

Sharp, H., Rogers, Y. & Preece, J. (2007). Interaction Design: Beyond Human Computer Interaction. Wiley.

Turk, D., France, R. & Rumpe, B. (2002). Limitations of Agile Software Processes.

URL 1: http://recordmydesktop.iovar.org/

Woltjer, R. (2006). A Systemic Functional Resonance Analysis of the Alaska Airlines Flight 261 Accident.

Finit, http://bitbucket.org/peppelorum/finit/

8 Bilagor

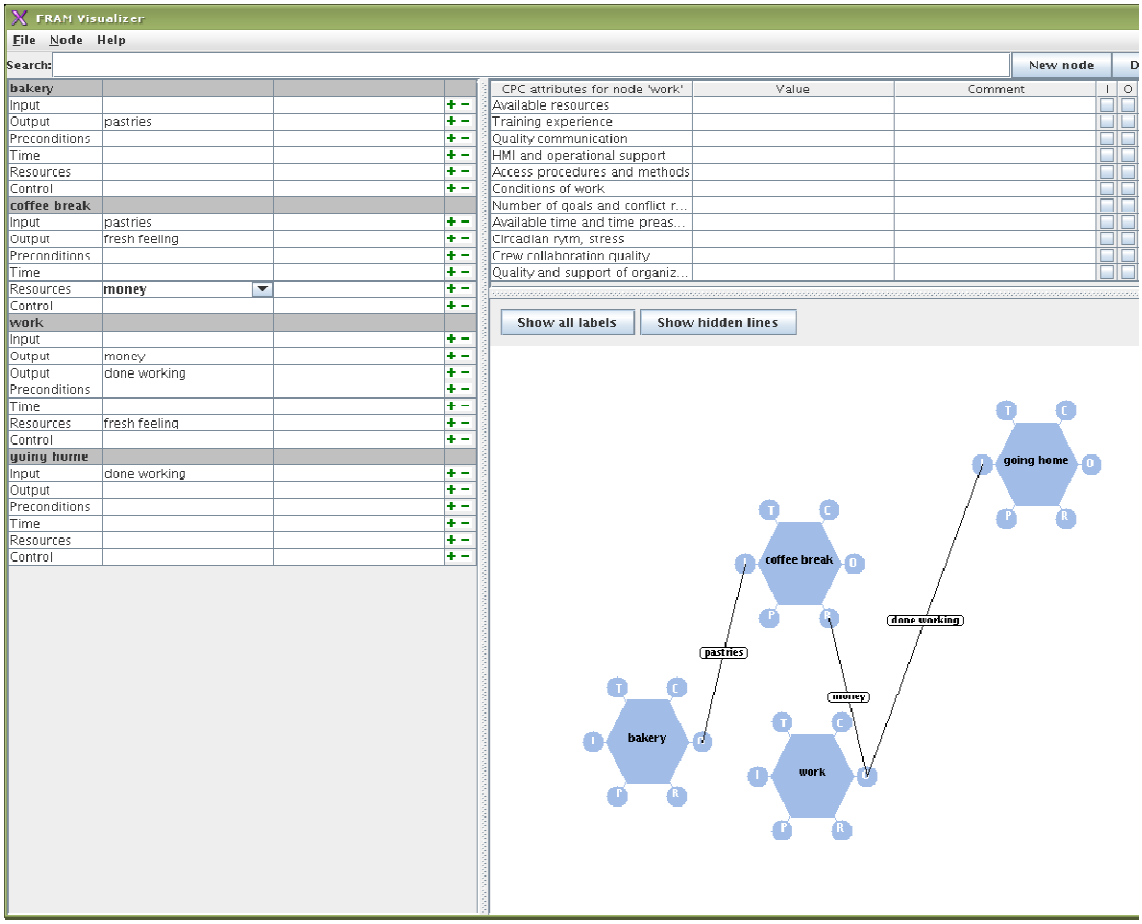


Bild 10: Framvisualizer

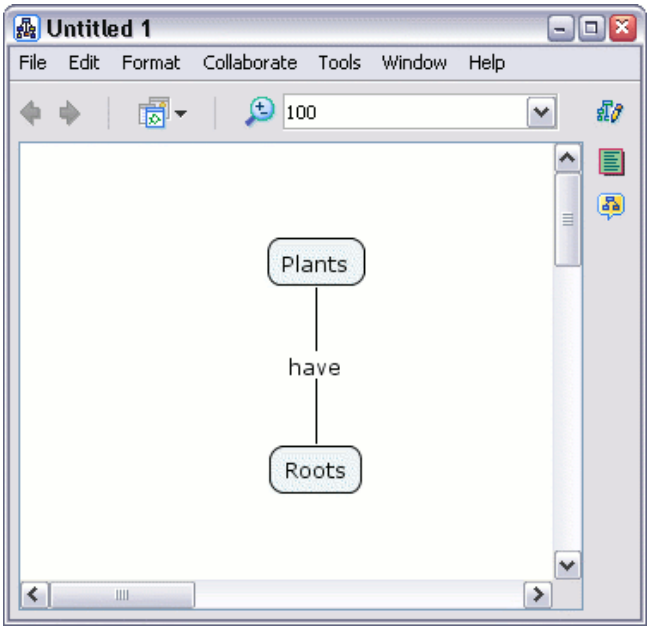
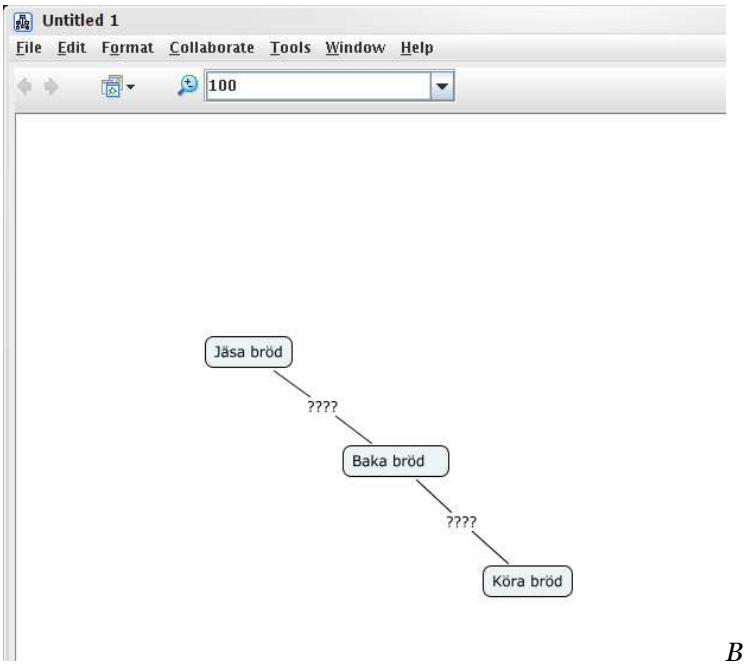


Bild 11: CmapTools - relation



B

Bild 12: CmapTools, huvudfönstret

Bild 13: CmapTools – kartor delade av andra användare