RISKfree

– ett system för riskhantering inom programvaruprojekt

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Programvaruteknik

Examensarbete:
Mikael Gullstrand

© Copyright Mikael Gullstrand

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds Universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Media-Tryck
Biblioteksdirektionen
Lunds Universitet
Lund 2006

Sammanfattning

RISKfree – ett system för riskhantering inom programvaruprojekt

I dagens IT-centrerade värld sker utvecklingen av hårdvara och programvara i en rasande takt. Om ett företag skall ha möjlighet att vara konkurrenskraftigt i denna utvecklingstakt, krävs det snabba och effektiva metoder i utvecklingsprocessen. Dessa krav från omvärlden på nya och mer komplexa produkter och tjänster innebär ofta nya och större risker för ett utvecklingsprojekt; både ekonomiska som tidsmässiga.

Analyser av programvaruprojekt som har misslyckats visar att många av deras problem och risker kunde ha undvikits eller åtminstone drastiskt reducerats med hjälp av riskhantering. Oftast är det en optimistisk entusiasm över projektet som överskuggar viktiga varningssignaler om risker och problem enligt B.W. Boehms artikel ”Software Risk Management: Principles and Practices”. Vidare säger Boehm att identifiering och hantering av risker tidigt i utvecklingsfasen sänker de långsiktiga projektkostnaderna samt hjälper till att undvika projektkatastrofer.

För att ett företag skall ha möjlighet att öka kvalitén och leveranssäkerheten av sina produkter, bör riskhanteringsprocessen bli en väl integrerad del i projektmodellen som genomsyrar ett projekts alla olika faser. För att undvika obehagliga överraskningar bör dessa risker identifieras, analyseras, prioriteras och hanteras på ett strukturerat sätt.

Med detta examensarbete har jag undersökt möjligheten att utveckla ett verktyg där riskhanteringsgruppen i ett programvaruprojekt kan strukturera och organisera detaljerad information om projektet och dess risker. Resultatet är en databasdriven applikation där projektinformationen sparas och kategoriseras för att sedan möjliggöra analys och återanvändning.

Enligt Roy och Woodings i artikeln ”A Framework for Risk Analysis in Software Engineering” är ett stort problem med riskhantering i utvecklingsprojekt att projekt och risker oftast skiljer sig markant från projekt till projekt. Men om informationen från tidigare projekt kategoriseras på ett strukturerat sätt bör tidigare erfarenheter vara möjlig att används för senare projekt.

Nyckelord: Riskhantering, programvaruprojekt, riskexponering, Monte Carlo-simulering

Abstract

RISKfree – a Risk Management System for Software Development Projects

In today's IT oriented world, the development speed of hardware and software is rapidly increasing. If a corporation shall be able to compete with other companies in this development pace, it needs fast and effective methods in the development process. The demand of more complex products and services can involve new and bigger risks to a development project, both financial and time wise.

Many analysis of software-project disasters have indicated that their problems would have been avoided or strongly reduced with the help of risk management. Frequently the projects were overshadowed with optimistic enthusiasm during their early phases that caused them to miss some clear signals of high-risk issues, according to B.W. Boehm in the article "Software Risk Management: Principles and Practices". Further more Boehm says that identifying and dealing with risks early in the development phase lessens long-term costs and helps prevent software disasters.

But if a company shall be able to increase quality and secure its delivery time of their products, then the risk management process should be a natural part of the life cycle of the product. To avoid unpleasant surprises, the risks should be identified, analysed, prioritized and managed in an effective way.

This final thesis is a research of the possibility to develop a risk management tool, where project groups in software development projects could arrange and organize detailed information about projects and the risks involved. The result is a database driven application where information is stored and categorized, so it will be possible to recycle and use it in a later project.

According to Roy and Woodings in the article "A Framework for Risk Analysis in Software Engineering"; a big problem with risk management in development projects is that the projects and risks often differ a great deal from project to project. But if information about earlier project is categorized in a structural way the experiences from these projects would be possible to use in later project.

Keywords: Risk management, software project, risk exposure, Monte Carlo-simulation

Förord

Detta examensarbete beskriver arbetet med att implementera ett system för riskhantering inom programvaruprojekt. Inspirationen för ett examensarbete inom detta område fick jag av Christin Lindholm under en föreläsning om kvalitetssäkring av programvara. Under denna föreläsning framgick det att det fortfarande till dags datum är väldigt tunt med system på marknaden som hantera risker på ett effektivt och strukturerat sätt inom programvaruprojekt.

Då många av dagens programvaruprojekt misslyckas eller har stora problem med leveranstider och budget, och samtidigt många av dem inte använder sig av riskhantering i utvecklingsprocessen, föreföll detta vara ett intressant och spännande område. En intressant fråga är då varför inte alla utvecklingsprojekt använder sig av riskhantering i sin projektmodell, om det nu vore lösningen för alla misslyckade projekt.

Avslutningsvis vill jag tacka min familj och vänner som ställt upp och hjälpt mig på olika sätt under dessa tre år av studier – utan ert stöd hade inte detta varit möjligt. Vidare vill jag tacka Christin Lindholm och Mats Lilja för all hjälp med granskningar och nyttiga synpunkter under resans gång. Slutligen vill jag tacka min kurskamrat Christian Jivenius för alla fina tips och trix jag har fått inom GNU/Linux-området.

Innehållsförteckning

Figurförteckning

1 Inledning

1.1 Bakgrund

Utveckling av programvara är oftast ett komplext och svårt hantverk. Många samverkande komponenter skall interagera på ett effektivt och korrekt sätt, som exempelvis operativsystem, drivrutiner och tredjeparts programvara.

Programvaruutveckling är en aktivitet där oftast projektgrupper och/eller enskilda personer skall samverka, och resultatet av projektets framgång beror till stor del av hur väl projektgruppen samarbetar [8]. Dessa ovanstående faktorer medverkar till en ökad komplexitet och osäkerhet för projektet och därav ökar riskerna [8] för projektet.

Två uppenbara problem som alltid är förknippade med programvaruutvecklingsprojekt är leveransförseningar och budgetöverskridningar [4, 7, 8, 10]. Då risker existerar även i små projekt blir konsekvenserna av dessa risker oftast mer moderata i jämförelse med stora projekt, där konsekvenserna kan bli ödestigna för projektets intressenter [3].

Leveransförseningar och budgetöverskridningar bör inte kategoriseras som risker i ett projekt, utan skall snarare ses som konsekvenser av underliggande risker i projektetaktiviteter som har gått snett under tidigare faser i utvecklingsprocessen [3].

1.2 Syfte och målsättning

Huvudmålet med detta examensarbete är att skapa ett enkelt och effektivt riskhanteringsystem, där det finns möjlighet att registrera projekt- och riskinformation under projektets olika faser, samt utnyttja erfarenheter från tidigare projekt i det aktuella projektet.

1.3 Problemformulering

Då närmare 70 % av alla utvecklingsprojekt inom programvaruindustrin misslyckas på ett eller annat sätt [10] bör man ställa sig frågan – varför?

En metod på vägen mot förbättrad styrning av dessa projekt är en väl integrerad riskhantering i projektmodellen. Med hjälp av en effektiv och strukturerad riskhantering blir det därför möjligt att höja kvalitén samt öka leveranssäkerheten av projektet.

Detta examensarbete kan indelas i två faser, där första fasen består i att undersöka teorin bakom riskhantering och hur den bör fungera i ett programvaruprojekt. Vilket innebär en närmare analys av vilka processer, metoder och attribut som används inom riskhantering. Genom denna analys

undersöks också hur kategorisering av projekt och risker skall genomföras på bästa sätt, så att dessa kan återanvändas i senare projekt.

I den andra fasen av examensarbetet kommer en databasdriven applikation med ett grafiskt gränssnitt att utvecklas, som baseras på den information som framkom under första fasen av examensarbetet.

1.4 Avgränsningar

Framförallt har mycket tid spenderats på design och dokumentation av systemet för att underlätta vidareutveckling och testning. Men även mycket tid har ägnats åt studier av ramverken för Mono och GTK#. Då dessa ramverk är under utveckling så saknas en hel del av denna dokumentation, vilket har minskat utvecklingstakten markant. Därav har bara grundläggande funktionalitet implementerats av applikationen, på grund av tidsramarna inte har riktigt räckt till. Av de funktioner som från början var tänkta att ingå i systemet men som senare valdes bort på grund av tidsbrist är serverns användarhantering och Monte Carlo-simulering. Implementationen av användarhantering har påbörjats men har inte slutförts på grund av tveksamhet kring realtidshantering av användarna. Monte Carlo-metoden avskrevs på ett tidigt stadium då denna innefattar komplexa beräkningar och samplingar av riskdiagram, vilket skulle ta för mycket tid i anspråk för att implementera.

Detta innebär att den nuvarande implementationen av riskhanteringssystemet skall ses som grundläggande och som en utgångspunkt för vidareutveckling av mer avancerade och djupgående funktioner.

Detta examensarbete har inte heller fokuserat på någon speciell utvecklingsmodell eller del i utvecklingsprocessen, utan betoningen vilar på vilken projekt- och riskinformation som skall lagras.

2 Riskhantering

Definitionen av ordet risk i Svenska Akademiens Ordbok är ”möjlighet att något icke önskligt (alt. olyckligt, obehagligt) skall inträffa eller skada”. Men inom programvarulitteratur använder man sig av en cirkulär definition av ordet risk [3]:

> *En risk är ett problem som har ännu inte inträffat och ett problem är en risk som har inträffat.*

Utifrån denna definition är det möjligt att skapa sig en bild av vad processen riskhantering innebär i ett programvaruprojekt. Det primära och grundläggande målet med riskhantering är att skapa undvikande åtgärder innan problemen har

uppkommit [3]. Bland de förebyggande åtgärderna ingår exempelvis identifikation, analys, prioritering och planering.

Motsatsen till riskhantering innebär krishantering, då problemet redan har uppstått och projektet får koncentrera sig på att hantera problemet.

2.1 Bakgrund

Projekt med utveckling av programvara har ofta stora problem med att hålla sig inom ramen för kostnader, leveranstider och användarbehov. Detta trots den enorma tekniska revolutionen som skett de senaste årtionden kvarstår dessa problem. Undersökningar visar att drygt 70 % av alla projekt misslyckas på ett eller annat sätt [10].

Risker existerar i alla typer av projekt men riskerna är oftast mer hanterbara i ett litet projekt än i ett stort projekt där konsekvenserna kan bli enorma ifall risken materialiseras. Dock är det viktigt att man inte inleds i falska förespeglingar om att små projekt är riskfria.

Sedan tidigt 80-tal har man försökt att begränsa problem och risker i utvecklingsprojekt genom riskhantering, bland annat har olika typer av listor och ramverk konstruerats för att underlätta projektledarens vardag [1]. Syftet med att använda listor och modeller är att identifiera, åtgärda och eliminera projektets risker innan de skapade problem för projektet.

2.2 Boehms riskhanteringsmodell

En av de mer kända modellerna för riskhantering är Boehms riskhanteringsmodell [1]. Denna modell specificerar ett antal aktiviteter som ingår i riskhantering under en produkts livscykel, samt ger exempel på hur dessa aktiviteter skall genomföras. Därför har detta examensarbete haft denna modell som utgångspunkt i utvecklingsarbetet av riskhanteringssystemet, för att på så sätt stödja hela riskhanteringsprocessen.

Modellen innefattar två huvudaktiviteter, riskuppskattning och riskkontroll, vilka är uppdelade i en rad underaktiviteter. I riskuppskattning ingår aktiviteterna riskidentifiering, riskanalys och riskprioritering. Samt i riskkontroll ingår de tre övriga aktiviteterna riskplanering, riskbeslut och riskövervakning.

2.2.1 Riskidentifiering

Riskidentifiering är grunden för all riskhantering. I denna process är det viktigt att upptäcka alla risker och problem som kan påverka projektet. Detta kan man åstadkomma genom en rad olika metoder som exempelvis använda en checklista eller använda tidigare erfarenheter från liknande projekt. Boehm har skapat en lista över de tio mest vanligaste risker i ett programvaruprojekt [Figur 1]. Denna

tabell i figur 1 är baserad på projekterfarenheter från ett antal projektledare inom programvarubranschen. Tabellen skall ses som en utgångspunkt för att finna mer projektspecifika risker för det aktuella projektet.

Risk	Riskhantering
Personalbortfall	Använd rätt personal, jobbmatchning, team-building, avtal med nyckelpersoner, personalutbildning.
Orealistiska tidsram och budget	Utveckla efter kostnad, inkrementell utveckling, återanvändning av programvara, vara restriktiv med krav.
Utveckling av felaktiga funktioner	Organisationsanalys, användarutfrågningar, prototyping, tidiga användarmanualer.
Utveckling av felaktigt användargränssnitt	Prototyping, scenario, medverkan av användare.
Utveckling av onödiga funktioner	Vara restriktiv med krav, prototyping, cost-benifit analys.
Kontinuerliga ändringar av krav	Hög kvalitet på krav, inkrementell utveckling.
Felaktiga tredjeparts komponenter	Kompabilitetsanalys, referenskontroll, benchmarking.
Felaktigt utförda externa uppgifter	Referenskontroll, team-building, resultatbaseradkontrakt
Realtidsbegränsningar	Simulering, benchmarking, prototyping
Tekniska begränsningar	Prototyping, referenskontroll, cost-benefit analys

Figur 1 B.W. Boehm topp tio-lista på risker i programvaruprojekt

2.2.2 Riskanalys

En riskanalys innebär att man gör en uppskattning av hur stor sannolikhet en risk har för att inträffa samt hur stora konsekvenser detta skulle få för projektet. Problemet med riskanalyser är att det är en subjektiv uppskattning av de personer som ingår i riskhanteringsgruppen, vilket innebär en osäkerhet ifall denna uppskattning är korrekt eller ej. Därför kan riskanalyser variera kraftigt mellan person till person.

Det finns en rad olika metoder för att minimera denna osäkerhet som exempelvis Monte Carlo-simulering, men även deltagarnas erfarenhet i

riskhanteringsgruppen och data från tidigare projekt kan vara till stor hjälp i riskanalysprocessen.

2.2.3 Riskprioritering

Under riskprioriteringsprocessen prioriteras de risker som har identifierats och analyserats. En populär metod för att prioritera risker är att använda sig av riskexponering (RE). Denna metod innebär att man multiplicerar den uppskattade sannolikheten med den uppskattade konsekvensen/kostnaden för risken.

Som exempel kan man använda sig av risker inom olika sportgrenar. Det kan tyckas att risken med att spela fotboll är betydligt lägre än att utöva fallskärmshoppning, men detta beror till stor del på vilken utgångspunkt man har.

De flesta personer som skall göra sitt första fallskärmshopp, funderar förmodligen över vad som kan gå fel, hur stor sannolikhet det är att det går fel och hur stor konsekvensen blir ifall något går fel. Detta är en kalkyl som de flesta utför automatiskt när vi ställs inför nya val och situationer som vi inte har utsatts för tidigare. Denna osäkerhet inför okända saker och situationer kan lätt överföras på riskhantering inom programvaruprojekt.

Enligt statistik från Socialstyrelsen [19] och doktoranden Anton Westman [20] visar följande sannolikhet av skador inom fallskärmshopp och fotboll.

Fallskärmshopp
Sannolikhet: 0.8 per 100 000 fallskärmshopp = 0.0008%
Kostnad: 10
*Riskexponering: 10 * 0.0008 = 0.008*

Fotboll
Sannolikhet: 0.35%
Kostnad: 5
*Riskexponering: 5 * 0.35 = 1.75*

Vilket ger skillnaden 219 gånger, med fördel fallskärmshoppning.

Enligt denna uträkning av riskexponenten, borde därför fallskärmshoppning ses som en ganska ofarlig aktivitet jämfört med att spela fotboll.

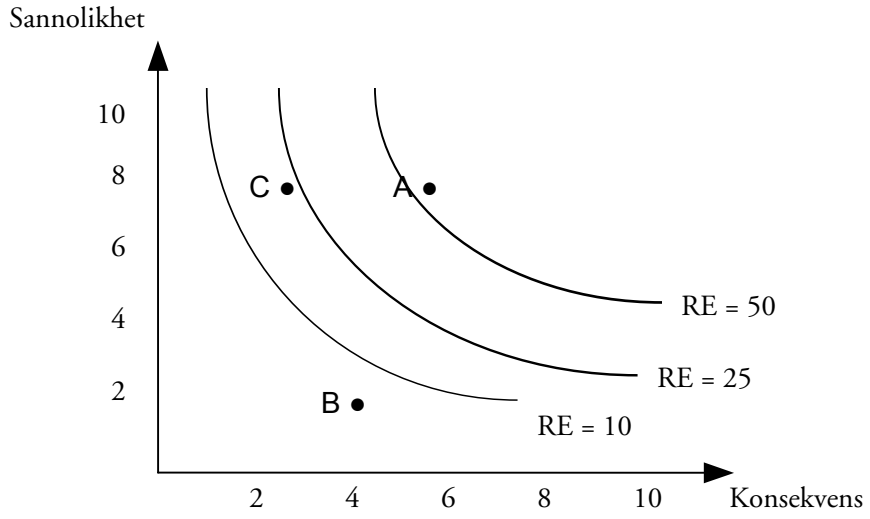
Tyvärr finns det flera problem med denna metod, den första och svåraste är att kontrollera osäkerheten i riskanalysprocessen. Uppskattning av sannolikhet och konsekvens för risken är inte helt trivialt, därför kan en felbedömning i riskanalysprocessen ge en helt felaktig prioritering av risken och som följd av detta väljer riskhanteringsgruppen att fokusera på fel risker. Detta kan i sin tur leda till förseningar, ökade kostnader eller rent av projektnedläggning.

Ett annat stort problem är hur man skall förhålla sig till risker med samma RE-värde, där exempelvis risker med stor sannolikhet och liten konsekvens får samma RE-värde som risker med liten sannolikhet och stor konsekvens.

Ett RE-diagram [Figur 3] kan vara till stor hjälp att grafiskt åskådliggöra hur RE-värde för riskerna i projektet ligger i förhållande till varandra. Detta underlättar för riskhanteringsgruppen att avgöra hur riskerna skall prioriteras och om det finns vissa områden i RE-diagrammet med risker som kan uteslutas.

Risk	S	K	Riskexponering RE
A. Kort frånvaro av gruppmedlem.	8	6	48
B. Problem med hårdvara.	2	4	8
C. Viktiga filer raderas.	8	3	24

Figur 2 Risktabell för RE-diagram



Figur 3 RE-diagram

2.2.4 Riskplanering
Nyckeln till en bra riskhantering är undvika att riskerna inträffar. Detta kan man åstadkomma genom att skapa förebyggande åtgärder som skall vidtas när en risk har identifierats och analyserats.

Varje risk tilldelas en detaljerad plan som innehåller ett antal åtgärdspunkter som skall ge förklaring på frågorna var, när, hur, varför, vem och hur mycket. Med denna framförhållning blir det enklare att hantera de risker som verkligen materialiseras, samt det ger projektgruppen trygghet med vetskapen om vad som skall göras preventivt för att undvika risken men även att det finns en åtgärdsplan ifall risken skulle inträffa.

2.2.5 Risklösning
Då syftet med riskplanering är att planera för vilka åtgärder som skall tas före och efter en risk har inträffat, så skapar en risklösning åtgärder för att risken kan elimineras från projektet. Exempelvis kan tester av hårdvara göras för att kontrollera så svarstider uppfylls eller prototyper av systemet kan visa att designen är korrekt.

2.2.6 Riskövervakning
Riskövervakning är den sista aktiviteten som knyter ihop begreppet riskhantering och gör det till en iterativ process.

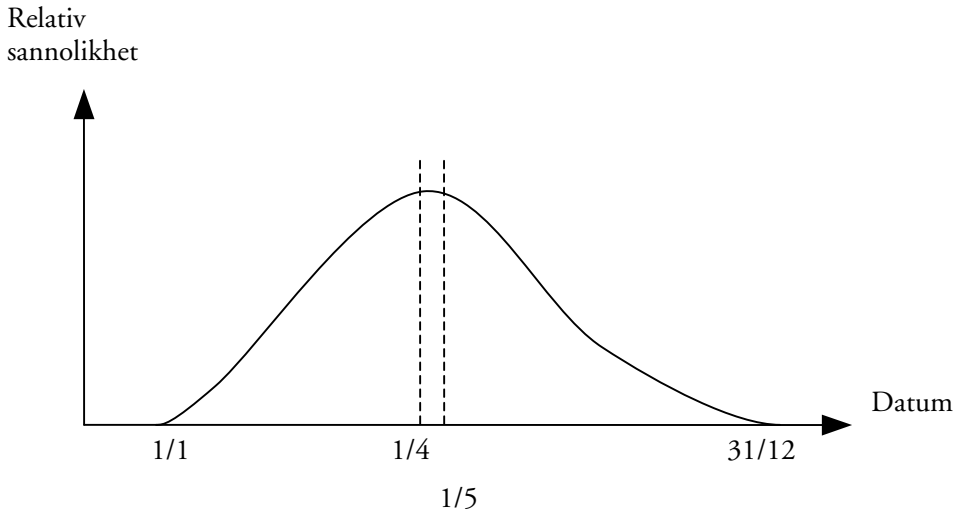
För att riskhanteringen skall vara effektiv måste riskövervakningen ske kontinuerligt under hela projektets livscykel. Det finns flera orsaken till detta; exempelvis kan risker uppstå under projektets olika faser, risker kan försvinna för att sedan dyka upp i ett senare skede eller risker kan avskrivas ifall de inte är aktuella.

2.3 Osäkerhet i riskhantering
Osäkerhet är ett nyckelbegrepp i riskhantering och detta beror främst på att stora delar av ett programvaruprojekt är omgärdat av osäkerhet. Exempelvis finns det osäkerhet kring vilka risker som existerar i ett projekt, osäkerhet om hur dessa risker kommer att påverka projektet, osäkerhet ifall om dessa risker kommer att materialiseras.

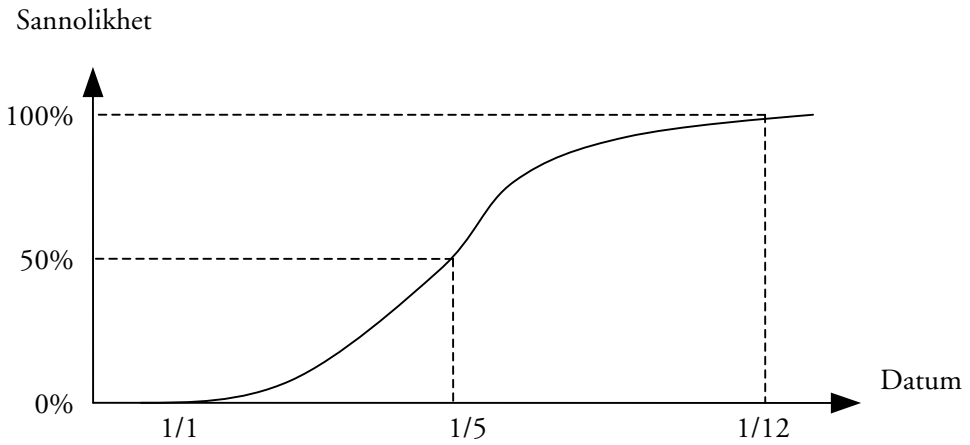
2.3.1 Riskdiagram

För att få ett närmare värde på hur osäker osäkerheten verkligen är, kan man använda sig av ett riskdiagram [3]. Nedan visas ett fiktivt svar från en projektledare på frågan; när han tror att hans projekt kan vara avslutat [3]. Detta svar kan sedan användas för att bygga upp ett riskdiagram [Figur 4].

> *Chansen att få projektet klart till den förste januari är lika med noll. Om vi skall leverera en någorlunda acceptabel produkt behöver vi åtminstone fram till den förste april på oss. Men detta datum är inte heller speciellt troligt utan vi bör nog vänta till tidigast den förste maj innan vi meddelar ett leveransdatum. Med ett leveransdatum i maj har vi i alla fall 50 % sannolikhet att klara det. Men om jag skall ange ett leveransdatum som vi definitivt inte kommer att missa så är det den siste december.*



Figur 4 Inkrementellt riskdiagram över leveransdatum



Figur 5 Växande riskdiagram över leveransdatum

Riskdiagrammet [Figur 4] visar den relativa sannolikheten när projektet kan förväntas att vara avslutat enligt projektledarens svar. Diagrammet skall därför tolkas på följande sätt: Innan den 1 januari är det 0 % sannolikhet att projektet är klart. Den streckade linjen som markerar den 1 april uppskattar att det är troligast här man kan visa en acceptabel produkt, men man bör vänta till den 1 maj då sannolikheten är 50 % att en acceptabel produkt kan uppvisas. Detta innebär att kurvans area är till hälften på vänster sida om den 1 maj och den andra hälften av arean är på höger sida om den 1 maj. Det inkrementella riskdiagrammet [Figur 4] kan även illustreras på ett mer lättförståeligt sätt genom att använda sig av ett växande riskdiagram [Figur 5]. I detta riskdiagram kan man enkelt avläsa att sannolikheten för att det fiktiva projektet skall vara klart den 1 maj är 50 % och att det definitivt kommer att vara klart den 1 december. Det växande riskdiagrammet är också användbart när man vill använda sig av Monte Carlo-simulering [1, 15, 18].

En viktig notering är att riskdiagram inte bara behöver användas i samband med tidsuppskattning, utan kan även användas på andra osäkra variabler såsom exempelvis kostnader och storlek.

2.3.2 Monte Carlo-simulering

Monte Carlo-simulering är en metod för att uppskatta ett specifikt resultat genom multipla samplingar som innefattar slumptalsvariabler.

Monte Carlo-metoden uppfanns av den polske matematikern Stanislaw Ulam 1946, när han funderade över hur stor sannolikhet det är att vinna ett parti patiens. Metoden har fått sitt namn genom staden Monte Carlo i Monaco som är mest känt för sina casinon med slumpartade spel. Teorin bakom metoden stödjer sig på att en aktivitet med ett slumpartat utfall som inte är beroende av ett tidigare utfall, som exempelvis ett tärningskast, är likformigt sannolikhetsfördelad. Dock gäller detta bara då aktiviteten upprepas ett oändligt antal gånger.

Om en person skulle kasta en tärning 100 gånger borde varje sida på tärningen komma upp 1/6 gånger av de 100 tärningskasten. Resultatet kommer dock att avvika något då antalet tärningskast är ganska få och slumpen är inblandad. Men om personen skulle genomföra 1000 tärningskast kommer resultatet närmare 1/6 än i fallet med de 100 tärningskasten, och en miljon tärningskast skulle ytterligare föra resultatet närmare 1/6.

Monte Carlo-simulering används i samband med riskdiagram för att skapa många slumpvisa prognoser i jämförelse med den ursprungliga uppskattade prognosen, som i detta fall är det ursprungliga riskdiagrammet. För att Monte Carlo-metoden skall vara effektiv krävs det ett stort antal samplingar. Desto fler samplingar som utförs på riskdiagrammet, desto större noggrannhet blir det

slutgiltiga resultatet. Metoden går ut på att generera ett slumptal mellan noll och ett där detta slumptalsvärde markeras på y-axeln i ett växande riskdiagram, exempelvis Figur 5, och det resulterande värdet på x-axeln avläses där det genererade slumptalsvärdet skär kurvan. Det avlästa värdet sparas sedan i ett histogram för att bygga upp ett nytt inkrementellt riskdiagram.

2.4 Riskhanteringens för- och nackdelar

Givetvis finns det både för- och nackdelar [3] med riskhantering inom programvaruprojekt. Dessa för- och nackdelar behöver vägas mot varandra innan projektledningen bestämmer sig för att integrera riskhantering i sitt utvecklingsprojekt.

2.4.1 Fördelar

En av de största fördelarna med riskhantering är att projektdeltagarna är förbereda på vilka delar inom projektet som är extra känsliga för projektets framgång. Detta medför att projektledaren kan lägga sina resurser där de kommer till bäst nytta.

Riskhantering inom utvecklingsprojekt synliggör även risker och problem för företagsledningen som oftast inte är direkt inblandade i det dagliga projektarbetet, vilket kan skapa en mer realistisk bild av projektet i förhållande till tidsåtgång och budget.

En majoritet av människor som deltar i utvecklingsprojekt tillhör oftast vi-kan-kategorin. Denna kategori motsätter sig, då oftast genom sitt numerära övertag gentemot projektdeltagare inom vi-kan-inte-kategorin, analyser av projektet som tillhör vi-kan-inte-kategorin. Men genom införande av riskhantering i utvecklingsprocessen kan man göra det legitimt med vi-kan-inte tänkande inom projekten.

2.4.2 Nackdelar

Det är viktigt att både inse och erkänna att riskhantering inte passar och/eller fungerar särskilt bra inom vissa typer av projekt.

Ett exempel på detta är projekt där endast en eller ett fåtal av deltagarna i projektet ägnar sig åt riskhantering. Risken finns att de andra deltagarna inom projektet anser att riskhantering är slöseri med tid och resurser, och därför ser ner på de personer som ägnar sig åt den aktiviteten och tycker att tiden kan användas till bättre saker.

En annan orsak till att undvika riskhantering kan vara att riskerna inom projektet är så osäkra att värdet på dem blir orealistiskt stora. Det är dock inget att rekommendera då projektet med största sannolikhet kommer att misslyckas.

3 Metodik och arbetsmetoder

3.1 Processmodeller

Under detta examensarbete har tre olika processmodeller använts, varav två har utförts parallellt med varandra under implementationsfasen. Den processmodell som har använts har varit beroende av vilken fas som examensarbetet har befunnit sig i.

I den inledande fasen med framställning av projektplan, domänanalys och kravspecifikation så användes den välkända vattenfallsmodellen [2]. Orsaken till valet av denna processmodell var att framställningen av ovanstående dokument har en naturlig ordningsföljd och när första versionen av ett dokument är avslutat kunde nästföljande dokument påbörjas. När de tre dokumenten hade granskats och satts i baseline övergick arbetet till nästa fas. Den nästföljande fasen av examensarbetet innebar att utvecklingsarbetet av systemet påbörjades. Denna fas följde en något modifierad testdriven processmodell där systemets funktionalitet utvecklades och testades kontinuerligt.

Vidare har svagheter och brister med de framtagna dokumenten upptäckts under utvecklingsarbetet, därför har revideringar och uppdateringar skett. På grund av dessa återkommande ändringar i dokument och implementation, så har en iterativ processmodell använts parallellt med den testdrivna utvecklingsmodellen.

3.2 Projektplan

Examensarbetet började med framställning av en projektplan [Appendix A]. Denna projektplan utgör grunden för examensarbetet och innehåller specifikationer för hur examensarbetet skall utföras.

Projektplanen specificerar bland annat vilka dokument som skall produceras, en tidsplan, vilka standarder som skall användas, dokumentmallar samt hur riskhantering skall utföras.

3.2.1 Dokument

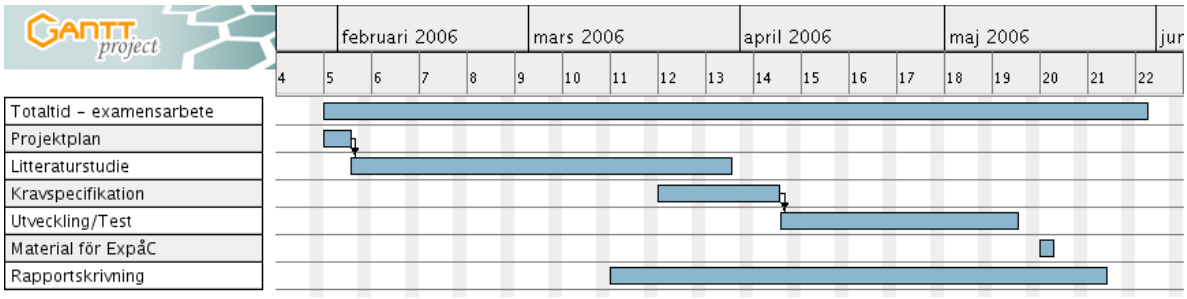
Under examensarbetets olika faser skall vissa givna dokument produceras. I den första fasen av examensarbetet där analys av domänen och framtagning av krav, så skapas domänanalysdokument och kravspecifikation. Därefter i fas nummer två produceras ett dokument som beskriver högnivådesign.

Ovanstående dokument har krävt kontinuerliga uppdateringar under projektets gång då fel och brister har upptäckts. Framförallt är det

kravspecifikationen och högnivådesign-dokumentet som har uppdaterats, men detta är naturligt då dessa är starkt knutna till implementationen av systemet.

3.2.2 Tidsplan

En grovt uppskattad tidsplan [Figur 6] upprättades för att visualisera de olika aktiviteternas tidsåtgång under examensarbetet. Tidsplanen har uppdaterats ett flertal gånger för att motsvara den verkliga tidsåtgången. Tidsplanen för skapandet av projektplan, litteraturstudie och kravspecifikation hölls väl men tiden för utveckling, test och rapportskrivning har krävt mer tid än planerat [se avsnitt 5.1].



Figur 6 Ursprunglig tidsplan

3.2.3 Standarder och hjälpmedel

I projektplanen finns standarder definierade för att projektarbetet skall utföras på ett enhetligt sätt och undvika oreda i dokument och design. Dessa standarder omfattar bland annat:

- Dokumentation av källkod.
- Filnamn och modulnamn.
- Notation av källkod.

Förutom standarder så specificeras de hjälpmedel som skall användas vid dokumentation och utvecklingsarbete.

3.3 Domänanalys

När första versionen av projektplanen var avslutad påbörjades framställning av domänanalysdokumentet [Appendix B]. En domänanalys är både ett dokument men framförallt är det en process där man inhämtar information om ämnesområdet som projektet omfattar. Dokumentet som beskriver

domänanalysen bör innehålla vilka aktörer som ingår i den inre respektive den yttre domänen, intressenter av projektet, kraveliciteringstekniker och risker.

Processen som domänanalysen utgör är en viktig grundpelare i alla utvecklingsprojekt, då utan en grundlig undersökning och förståelse för domänen finns en stor risk att projektet misslyckas. Domänkunskapen kan inhämtas på flera olika sätt, exempelvis genom litteraturstudie eller intervjuer av personer med domänkunskap [se avsnitt 3.3.2].

3.3.1 Utförande

Då riskhantering kan ses som ett känsligt område för flera företag, så vill de inte gärna offentligt diskutera problem och brister med utomstående. Därför var det ganska problematiskt att hitta några företag som ville ställa upp på en intervju. Dock lyckades en intervju genomföras med en projektledare på ett telekomföretag i Malmö.

Denna intervju gav en ganska bra inblick i hur en projektledare ser på riskhantering och hur denna hantering kan fungerar i en verklig miljö.

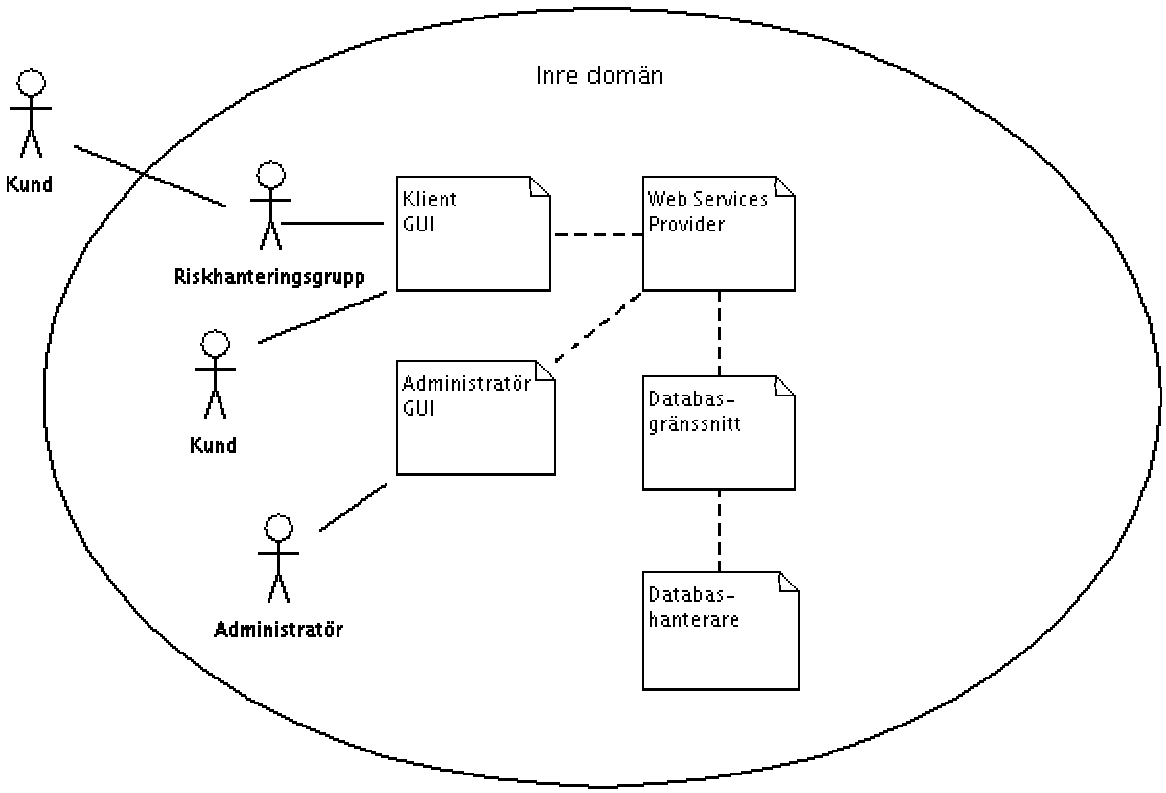
Tyvärr överensstämde intervjun ganska väl med den förmedlade bilden i artiklar och litteratur om riskhantering, som en aktivitet man utför i inledningsskedet av projektet för att sedan läggas åt sidan. Projektledarens motivering till detta resonemang var att projekten som företaget ifråga utför ofta är likvärdiga och därför har ungefär samma problem och risker. Därför görs en riskstudie i projektets inledningsfas, detta för att projektdeltagarna skall få en grundläggande kunskap om projektets problem och risker som de sedan kan bära med sig under projektets livscykel.

Då det var problematiskt att hitta företag som ville medverka i examensarbetet var det desto lättare att hitta litteratur som behandlar ämnet riskhantering. Litteratur som behandlar riskhantering sträcker sig ända tillbaka till tidigt 80-tal då W. McFarlan publicerade sin artikel inom området [6].

Då tillgången av riskhanteringslitteratur var stor användes denna som den främsta källan till domänkunskap.

3.3.2 Domändefinition

Genom definition av domänen och dess aktörer fås en bättre överblick av den miljö som systemet skall interagera med [Figur 7]. Definitionen av aktörer inbegriper både fysiska personer såväl som andra system och gränssnitt. Diagrammet ger även utomstående personer en bättre visuell överblick och förståelse av domänen.



Figur 7 Aktörer i den inre- och yttredomänen

3.3.3 Kraveliciteringsmetoder

För att specificera relevanta krav för systemet användes litteraturstudier som ensam kraveliciteringsmetod. Denna metod användes på grund av att den ansågs vara den mest lämpade, då den inledande domänanalysen bestod av litteraturstudier och på så sätt skapades en förståelse för hur riskhantering fungerar i både teori och praktik. En annan bidragande orsak till att valet föll på denna metod är att examensarbetet endast har en deltagare och de flesta andra kraveliciteringsmetoder kräver medverkande av flera deltagare.

Kunskapen som inhämtades under litteraturstudien användes sedan för att plocka fram alla möjliga och omöjliga krav. Kraven som framkom under denna process dokumenterades för att sedan reducerades till en lista med mer realistiska och relevanta krav för systemet.

3.3.4 Kravstilar

I kravspecifikationen [Appendix C] finns alla de krav som rör systemet definierade. Då dessa krav beskriver olika delar av systemet krävs det olika kravstilar för att beskriva dessa på bästa sätt.

- E/R-diagram har använts för att beskriva de krav som behandlar datamodellen. Ett E/R-diagram gör det enkelt att verifiera att de data som hanteras av systemet verkligen överensstämmer med kraven.
- Virtual windows som skall ses som förenklade skärmdumpar, gör det enkelt att validera den datamodellen som finns beskriven i E/R-diagrammet.
- Task deskriptions används på en mer övergripande nivå för att beskriva systemets krav. Denna kravstil är enkel att förstå både för utvecklare och användare.
- Feature requirements beskriver de krav som ovanstående kravstilar inte behandlar. De krav som specificeras med Feature requirements kan oftast översättas till en funktion i systemet.

3.3.5 Risker

I domänanalysen [Appendix B] ingår tabeller för riskhantering [Figur 8]. Denna riskhantering är något enklare än det riskhanteringssystem som detta examensarbete är tänkt att utveckla, då den endast innehåller grundläggande information om risker samt åtgärder för att undvika att riskerna materialiseras. Men här finns inga åtgärder för vad som skall göras ifall riskerna inträffar och blir reella problem, samt det saknas även prioritering och riskexponering av riskerna.

Men detta visar att även små utvecklingsprojekt kan dra nytta av riskanalyser. Dessa analyser behöver inte vara speciellt avancerad för att ge indikationer om vilka område som kan vara problematiska för projektet.

Riskerna i detta examensarbete har specificerats med ett unikt risknummer, en beskrivning av risken samt en åtgärd för att förhindra att risken inträffar. Riskerna har även värderats med hur stor sannolikhet risken inträffar och hur stor konsekvens risken har för examensarbetet.

ID	S	K	Beskrivning av risk	Åtgärd
6.1.1.1	4	3	Kort frånvaro av gruppmedlem.	Tidsplanera med en tidsbuffert.
6.1.1.2	1	5	Lång frånvaro av gruppmedlem.	Finns ingen. Projektet kommer att bli försenat.
6.1.1.3	3	3	Gruppen har inte tillräcklig kunskap.	Vidarutbilda.
6.1.1.4	1	4	Gruppmedlem är oengagerad.	Samtal med handledare.

Figur 8 Exempel på risktabell från domänanalys

3.4 Kravspecifikation

Då domänanalysen skall ses som en viktig grundpelare för att lyckas med ett utvecklingsprojekt, bör kravspecifikationen ses som en viktig grundpelare för att lyckas med implementationen av systemet. Därför är det viktigt att kravspecifikationen håller en hög kvalité och sena ändringar bör undvikas då dessa ofta blir väldigt dyra att åtgärda i implementationsfasen.

3.4.1 Utförande

Kravspecifikationen [Appendix C] skapades med hjälp av de kraveliciteringstekniker som är specificerade i domänanalysdokumentet [Appendix B]. I detta examensarbete användes endast litteraturstudier som kraveliciteringstekniker då denna ansågs vara mest lämpade för examensarbetet [se avsnitt 3.3.3].

Kraven som specificeras i kravspecifikationen är grupperade i projektkrav, domänkrav, funktionella krav och datakrav. Dessa krav är sedan uppdelade i mindre grupper med underkrav. Alla krav har ett unikt kravnummer för att kravet skall vara enkelt att identifiera och på så sätt inte vara möjligt att förväxla med ett annat krav.

Det finns flera olika attribut på ett krav för att kravspecifikationen skall hålla en hög kvalité:

- Korrekt, kraven skall överensstämma med kundens önskemål.
- Fullständig, alla krav för systemet skall vara med.
- Verifierbar, kravet skall vara möjligt att verifiera.
- Entydig, kravet skall bara vara möjligt att tolka på ett sätt.
- Motsägelsefull, kraven skall inte vara motsägelsefulla.
- Spårbar, kravet skall vara enkelt att referera till.
- Organiserad, kravspecifikationen skall vara enkel att hitta i.

3.4.2 Projekt-, mål- och funktionskrav

Dessa krav specificerar systemets funktionalitet på en detaljerad nivå och varje krav behandlar endast en funktion i systemet. Detta medför att kraven är enkla att testa och verifiera. Varje krav har ett löpnummer som inleds med SRSx, där x är kravets unika nummer. Varje krav innehåller även en ”skall”-sats. Exempel på ett projektkrav [Appendix C]:

> *SRS41401. Utvecklingen av systemet skall överensstämma med Projektplanen.*

3.4.3 Domänkrav

Task descriptions [Figur 9] har använts för att specificera domänkrav som beskriver systemets krav på en övergripande nivå och involverar inte någon systemdesign. Task descriptions ökar förståelsen för domänen utan att dessa behöver omgärdas av onödiga detaljer. Kraven är definierade med en övergripande uppgift som sedan delas upp i mindre underuppgifter. Kraven innehåller information om målet med uppgiften, hur ofta uppgiften utförs, vad som aktiverar uppgiften.

Uppgift:	3. Uppdatera riskuppgift
Mål:	Uppdatera en riskuppgift i ett projekt. Spara riskdata.
Trigger:	Användaren vill uppdatera informationen för en riskuppgift.
Frekvens:	En gång per vecka.
Kritisk:	Nej
Underuppgifter:	
1.	Välj projekt.
2.	Välj riskuppgift.
3.	Spara riskdata i logg.
4.	Uppdatera riskdata.
5.	Spara ny riskdata.
Varianter:	Inga varianter förekommer.

Figur 9 Exempel på domänkrav

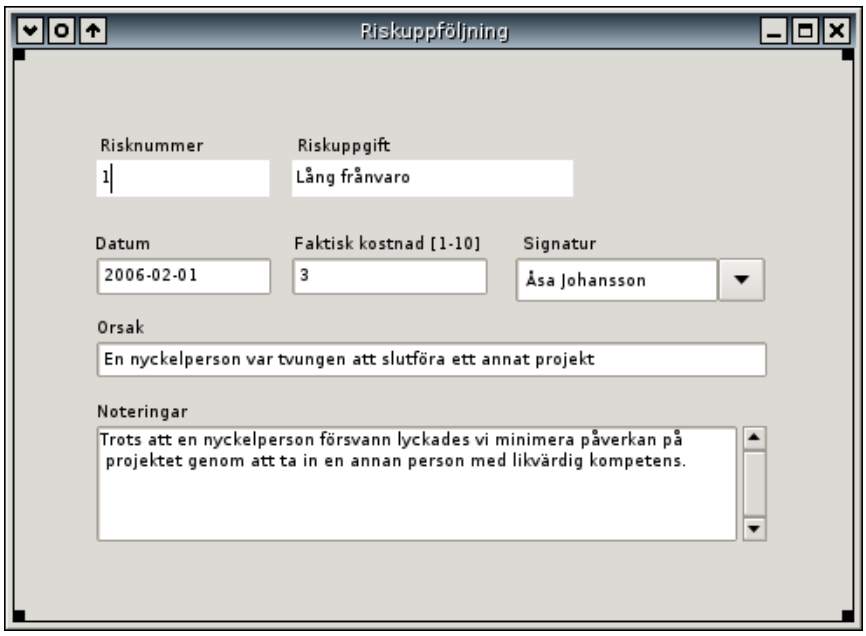
3.4.4 Datakrav

Datakraven är specificerade med hjälp av ett E/R-diagram [Figur 11] och Virtual windows [Figur 10]. Dessa krav innebär i princip högnivådesign av databasen då de definierar entiteter och relationer på en detaljerad nivå.

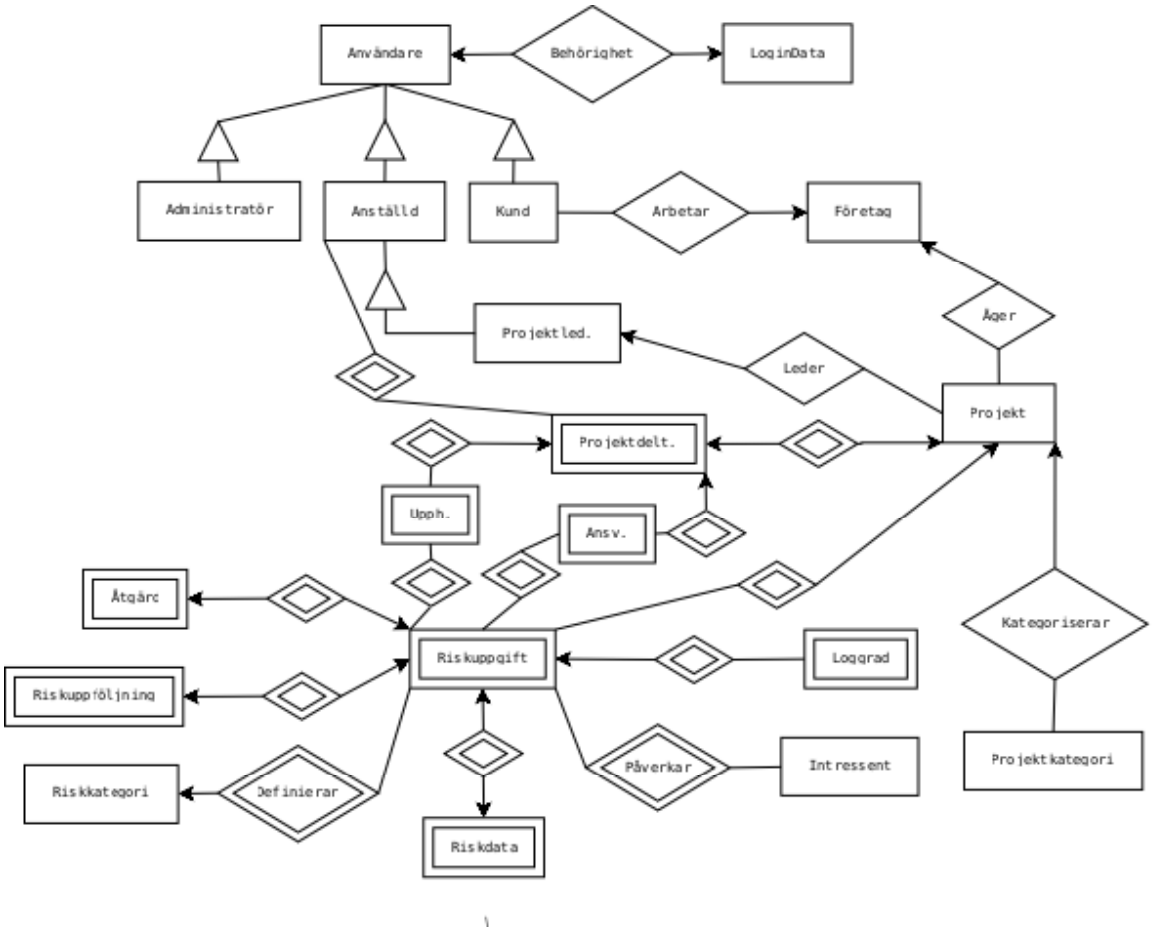
E/R-diagrammet visar entiteter och relationer som skall utgöra databasen. För att markera typ av relation såsom en-till-en, en-till-många eller många-till-många används pilar i relationerna.

Genom att använda sig av Virtual windows så kan man till stor del verifiera de datakrav som E/R-diagrammet behandlar, då Virtual Windows måste innehålla realistisk data från databasen.

För att skapa E/R-diagrammet omvandlades substantiv från funktionella krav, domänkrav samt domänanalysen till entiteter och relationer. Dessa entiteter och relationer knöts sedan samman så att E/R-diagrammet överensstämmer med domänen.



Figur 10 Exempel på Virtual window



Figur 11 E/R-diagram för RISKfree

3.5 Högnivådesign

Den första fasen av implementation av systemet är högnivådesign. Denna ger en övergripande bild av systemet samt databasens uppbyggnad dokumenteras.

Högnivådesigndokumentet [Appendix D] innehåller E/R-diagram, specifikationer för entiteter och UML-diagram för att beskriva riskhanteringssystemets uppbyggnad.

3.5.1 Utförande

Utvecklingen av högnivådesignen påbörjades egentligen i samband med framtagningen av E/R-diagram och Virutal window i kravspecifikationen.

Därför var det naturligt att utvecklingsarbetet fortsatte med att definiera attributen för entiteterna i databasen.

Då systemet är begränsat i storlek har utvecklingen av UML-diagram skett i ganska hög grad parallellt med själva implementationen av källkoden.

4 Implementation av RISKfree

Riskhanteringssystemet som har implementerats utifrån domänanalys och kravspecifikation, har som främsta uppgift att lagra detaljerad information om projektspecifika risker på ett strukturerat och effektivt sätt.

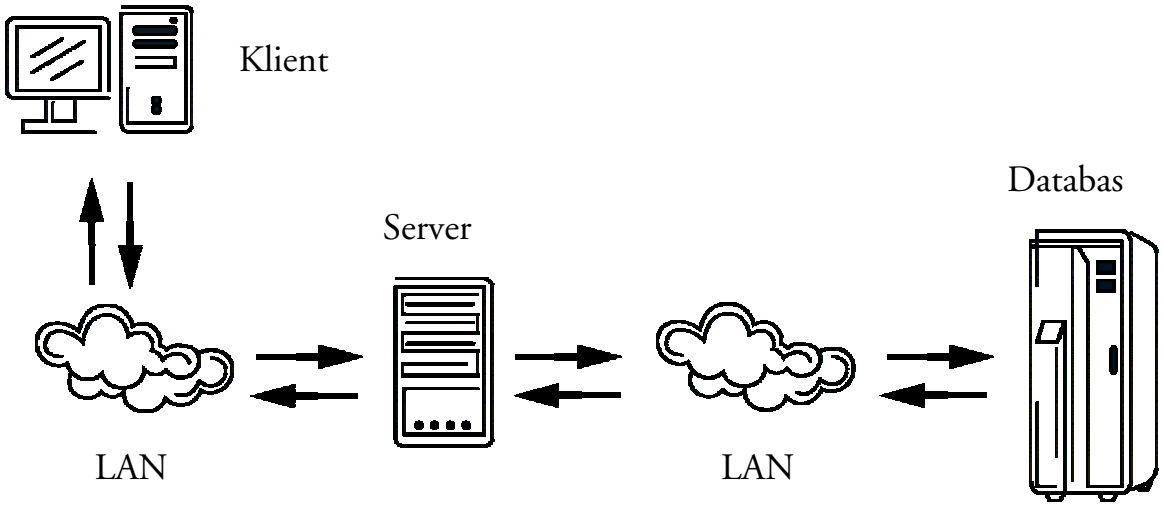
4.1 Översikt

Systemet består egentligen av tre delsystem; klientapplikation, server och relationsdatabas [Figur 12].

Datalagring sker i en relationsdatabas som består av ett antal entiteter och relationer. Databasen har normaliserats genom BCNF-normalisering [9] för att undvika problem med redundans vid uppdateringar och raderingar i databasen.

Systemet skall klara av att hantera flera samtidiga användare vilket innebär att anrop från flera klientapplikationer kan ske samtidigt. Detta realiseras genom att systemet använder transaktioner vid skrivning och läsning av dataposter i databasen.

Systemet är utvecklat i .NET-miljö och som utvecklingsspråk har C# använts. Detta möjliggör att systemet kan exekveras både på Windows- och GNU/Linux-plattformar. Som DBMS har Firebird [13] använts, men denna kan enkelt bytas ut mot någon annan databashanterare.



Figur 12 Översikt Riskhanteringssystem

4.2 Implementationsprocess

Tanken är att det skall vara möjligt att köra riskhanteringssystemet på åtskilda datorer och därför har systemet delats upp i tre skilda delsystem. Detta ger en rad fördelar som exempelvis; att funktionalitet kan tillföras i något av de tre delsystemen utan att något av de andra systemen påverkas, klientapplikationen kan implementeras med hjälp av andra utvecklingsspråk samt utvecklingsarbetet blir enklare med starkt åtskilda moduler.

Något förenklat kan man säga att definition av entiteter, relationer och Stored Procedures i databasen var det första som utfördes i utvecklingsarbetet. För utan att dessa är definierade finns det ingen möjlighet att skriva och läsa till databasen, och implementationen utfördes i enlighet med entitet- och relationsspecifikationen i högnivådesigndokumentet.

När alla centrala entiteter och relationer var definierade påbörjades utvecklingsarbetet med server-sidan av systemet. Den naturliga utgångspunkten var att skapa funktionalitet för att koppla databasen till servern samt möjlighet att läsa och skriva till databasen. Därefter skapades ett bibliotek av SQL-satser som servern och databasen kommunicerar med. Dessa SQL-satser används i sin tur av klasser i servern som har till uppgift att läsa eller skriva information till databasen.

Då funktionaliteten av server-delen hade nått så långt att läs- och skrivfunktioner till databasen hade implementerats påbörjades utvecklingsarbetet med den grafiska klientapplikationen.

Utvecklingsarbetet fortsatte därefter med en iterativ vattenfallsmodell där funktioner infördes i databasen för att sedan implementeras i severn och därefter slutligen infördes funktionen i klientapplikationen.

4.3 .NET och Mono

.NET är ursprungligen ett antal produkter och teknologier från Microsoft [12], som alla är beroende av .NET-ramverket som i sin tur består av att stort antal klassbibliotek. Dessa klassbibliotek gör det möjligt att sammankoppla information, enheter och system på ett snabbt och effektivt sätt.

Det finns andra implementationer av .NET än bara Microsofts, bland annat är Common Language Infrastructure (CLI) definierad som en ECMA/ISO-standard.

Mono-projektet är en implementation av CLI, men har även vissa delar av .NET Base Class Library (BCL) implementerat. BCL är ingen öppen standard och därav finns det vissa problem med att komplett implementera dessa på grund av upphovsrättsliga skäl. Mono-projektet är ett öppet källkodsprojekt som Novell står bakom, men man har även en stor frivillig utvecklarskara som driver utvecklingen framåt [16].

4.4 .NET Remoting och Web Services

Web Services har en central roll inom .NET som ett verktyg att låta applikationer och system utbyta data och information över olika hårdvaruplattformar och operativsystem. Denna kommunikation sker genom standardiserade protokoll såsom XML och SOAP, via Internet eller ett LAN.

Ett annat sätt att dela data och information över ett nätverk är .NET Remoting [11], denna teknologi är också en del av .NET-ramverket och benämns ofta som distribuerad programmering. Valet föll på att använda .NET Remoting vilket har en högre kommunikationshastighet, på grund av ett mer effektivt protokoll jämfört med Web Services. Däremot använder Web Services sig av http-protokollet som har fördelen att brandväggar och andra nätverksenheter oftast är öppna utan att extra konfiguration behövs.

Kortfattat kan distribuerad programmering beskrivas som fristående datorenheter arbeta tillsammans som de vore en ensam datorenhet. Detta innebär att en datorenhet kan skapa och/eller exekvera objekt i minnesarean på en helt annan datorenhet.
.NET Remoting stödjer tre typer av fjärrobjekt:

- Single Call – kan enbart hantera en förfrågan från en fjärrdator åt gången. Single Call-objekt behåller ej sin status mellan metodanrop och kan därför ej lagra information.
- Singleton – kan däremot betjäna flera klientanrop och behåller sin status emellan klientanrop. Denna typ

av objekt är användbara när flera klientdatorer behöver explicit dela information.

- Client-Activated – serverobjekt som skapas vid anrop från en klientdator på servern. Denna typ av objekt behåller sin status mellan metodanrop från den klientdator som skapade instansen av objektet.

Distribuerad programmering underlättar utvecklingsarbetet av ett Klient-Server system avsevärt då källkoden kan skrivas som om systemet endast består av en självständig applikation.

4.5 Relationsdatabas

Grunden för riskhanteringssystemet är en relationsdatabas där all information lagras. Det finns ett stort antal relationsdatabaser på marknaden, både fria och proprietära.

I detta examensarbete har Firebird [13] används som DBMS på grund av att dess snabbhet, stabilitet och har låga resurskrav. Denna är en fri ANSI SQL databas som kan köras på GNU/Linux, Windows och ett antal UNIX-plattformar. Firebird har sitt ursprung i Borland Interbase som släpptes fri under InterBase Public License v.1.0 den 25 juli 2000.

Firebird kräver ingen registrering, licens eller royalty-avgift för att användas och kan inkluderas med vilken tredjepart programvara som helst, både kommersiella och icke kommersiella.

4.6 Server

Servern fungerar som mellanhand mellan klientapplikation och databas, därför exekveras den som en process som väntar på anrop från klientapplikationer för att sedan vidarebefordra dessa till databasen. Severn är även tänkt att hantera alla användare som är inloggade i systemet, detta för att undvika exempelvis att lösenord skickas vid varje anrop till databasen.

Ett anrop från klientapplikationen innebär att klienten skapar ett Client-Activated-objekt på servern för det aktuella arbetet som användaren vill utföra. Servern exekverar sedan dessa Client-Activated-objekt för skrivning eller läsning i databasen. I de fall där läsning från databasen är aktuell, hämtas information från databasen för att sedan skapa ett objekt på servern innehållande informationen från den aktuella databasfrågan. Därefter vidarebefordras detta objekt till klientapplikationen där resultatet slutligen presenteras för användaren.

4.7 Klientapplikation

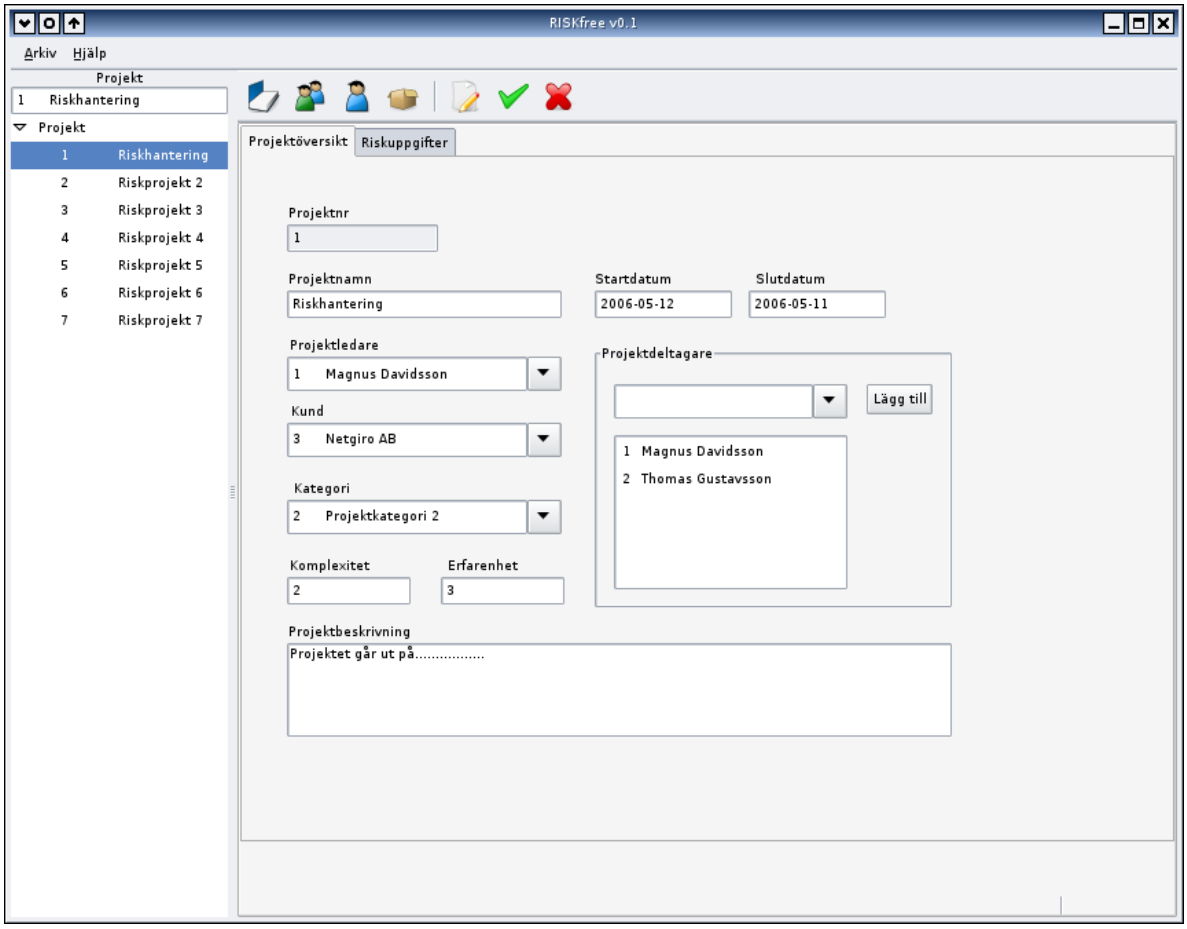
Den grafiska klientapplikationen har till uppgift att låta användaren interagera med databasen. För att klientapplikationen skall vara så modulär och portabel som möjligt kommunicerar den inte direkt med databasen, utan kommunikation sker uteslutande mot servern via .NET Remoting. Detta sker genom att klientapplikationen anropar severn med att skapa ett remote-objekt på servern, alternativt så skickas klient-objekt till servern som behandlas lokalt på denna.

För att det skall vara möjligt att köra den grafiska klientapplikationen under både GNU/Linux och Windows används ramverket GTK#. I Mono-projektet ingår även ett Windows.Forms-ramverk, men GTK# är det GUI-ramverk som har kommit längst i sin implementation och har därav störst funktionalitet under båda plattformarna.

4.7.1 Projektvy

Den konceptuella idén med systemet är att användaren först skapar ett nytt projekt [Figur 13] i projektvyn. Grundläggande information om projektet såsom projektnamn, kund, projektledare och projektdeltagare anges i de olika textfälten.

Därefter anges projektkategori, erfarenhet som organisationen har av denna typ av projekt samt projektets storlek. Denna information är viktig om det skall vara möjligt att skapa data utifrån andra projekt och på så sätt återanvända tidigare kunskaper.

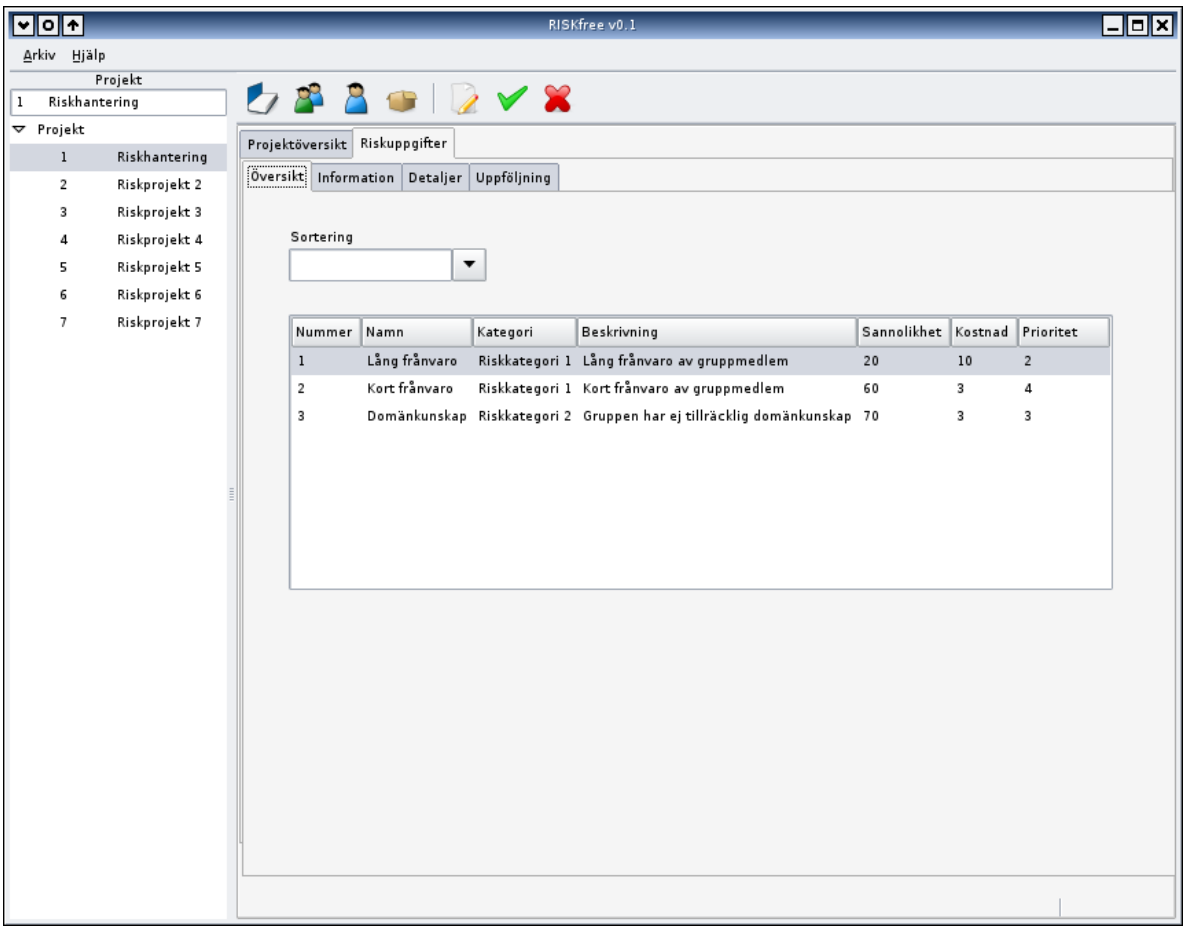


Figur 13 Projektvy

4.7.2 Riskvy

Till varje projekt finns en riskvy som presenterar riskerna som har identifierats för projektet. Det är också i riskvyn som riskerna dokumenteras för projektet samt analyseras och prioriteras.

Åtkomst till riskvyn sker genom att användaren väljer fliken *Riskuppgifter* i projektvyn. Detta innebär att en översiktsvyn [Figur 14] visas med en lista över alla risker som är lagrade för det specifika projektet. Denna lista skapar en bra överblick av vilka risker som tillhör ett visst projekt, och för att ytterligare öka överblicken så har listan möjlighet att sorteras med exempelvis risknummer, riskkategori eller prioritering.

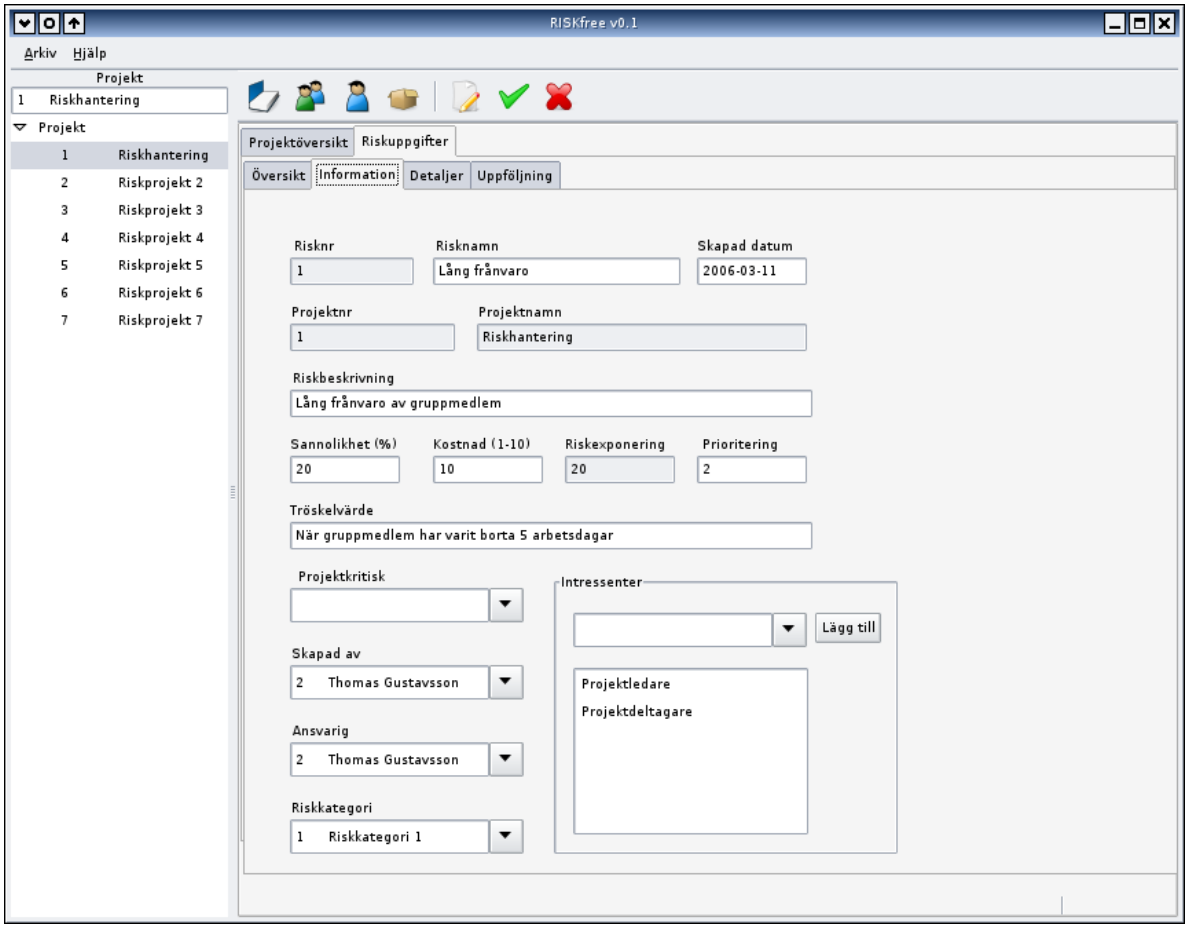


Figur 14 Riskvy - Översikt

Genom att dubbelklicka på en riskuppgift i listan kan användaren få upp en mer detaljerad information om den specifika riskuppgiften.

Denna information presenteras under fliken *Information* [Figur 15] som exempelvis risknamn, beskrivning, datum och riskens intressenter. Även här klassificeras risken på liknande sätt som projektet, där värde för sannolikhet och konsekvens anges. Utifrån dessa sannolikhet- och konsekvensvärde beräknas riskexponeringen och slutligen prioriteras risken.

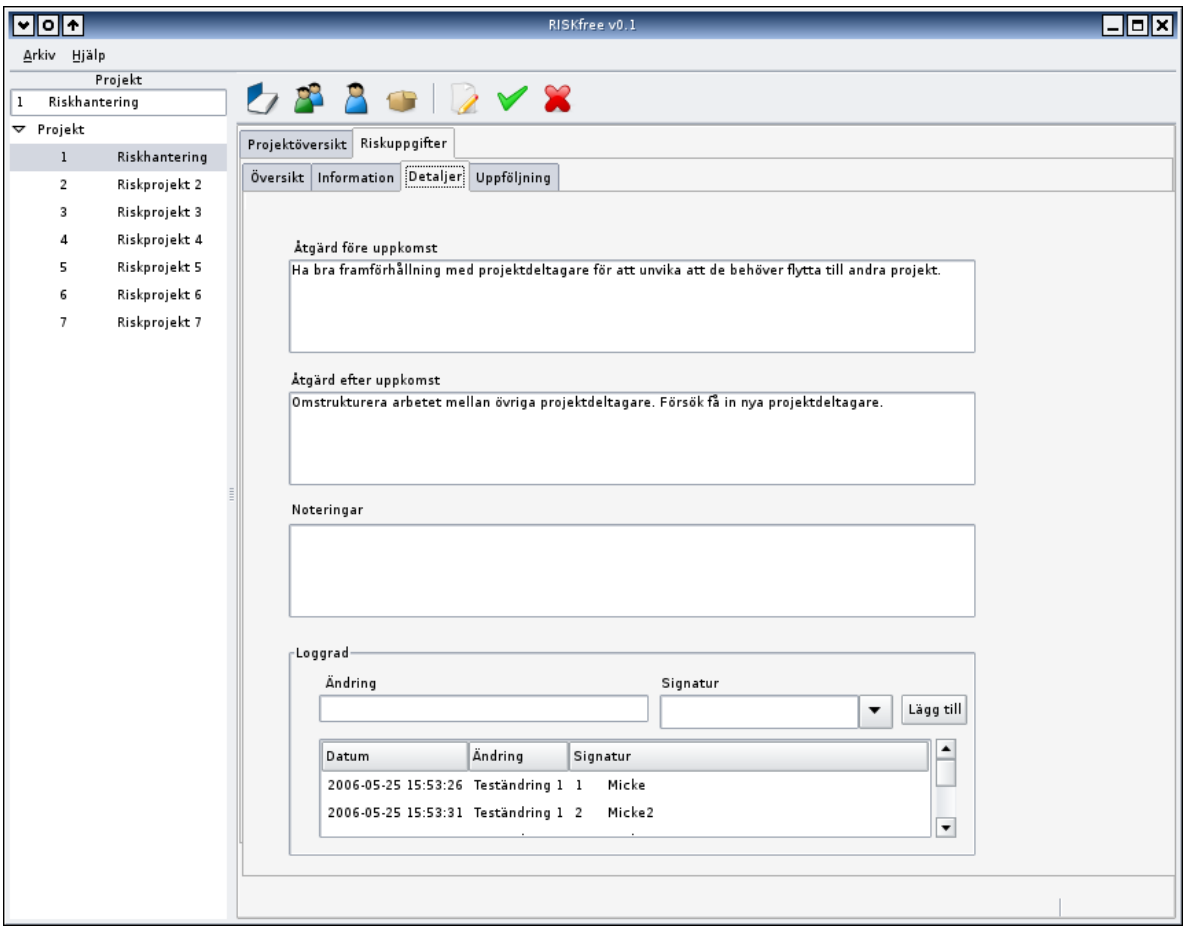
All information som behandlar riskuppgiften är tillåten att ändras, förutom risknummer och projektnummer. Alla ändringar av riskuppgiften sparas eller förkastas med hjälp av snabbknappar i menyraden. För att skapa en ny riskuppgift aktiverar användaren *Ny*-snabbknappen i menyraden, så att en ny riskuppgift skapas för det aktuella projektet med ett unikt risknummer.



Figur 15 Riskvy - Information

Riskvyn innehåller även fliken *Detaljer* [Figur 16] där information presenteras om före- och efteråtergärder för risken, noteringar samt en logglista för att dokumentera ändringar som har skett med riskuppgiften.

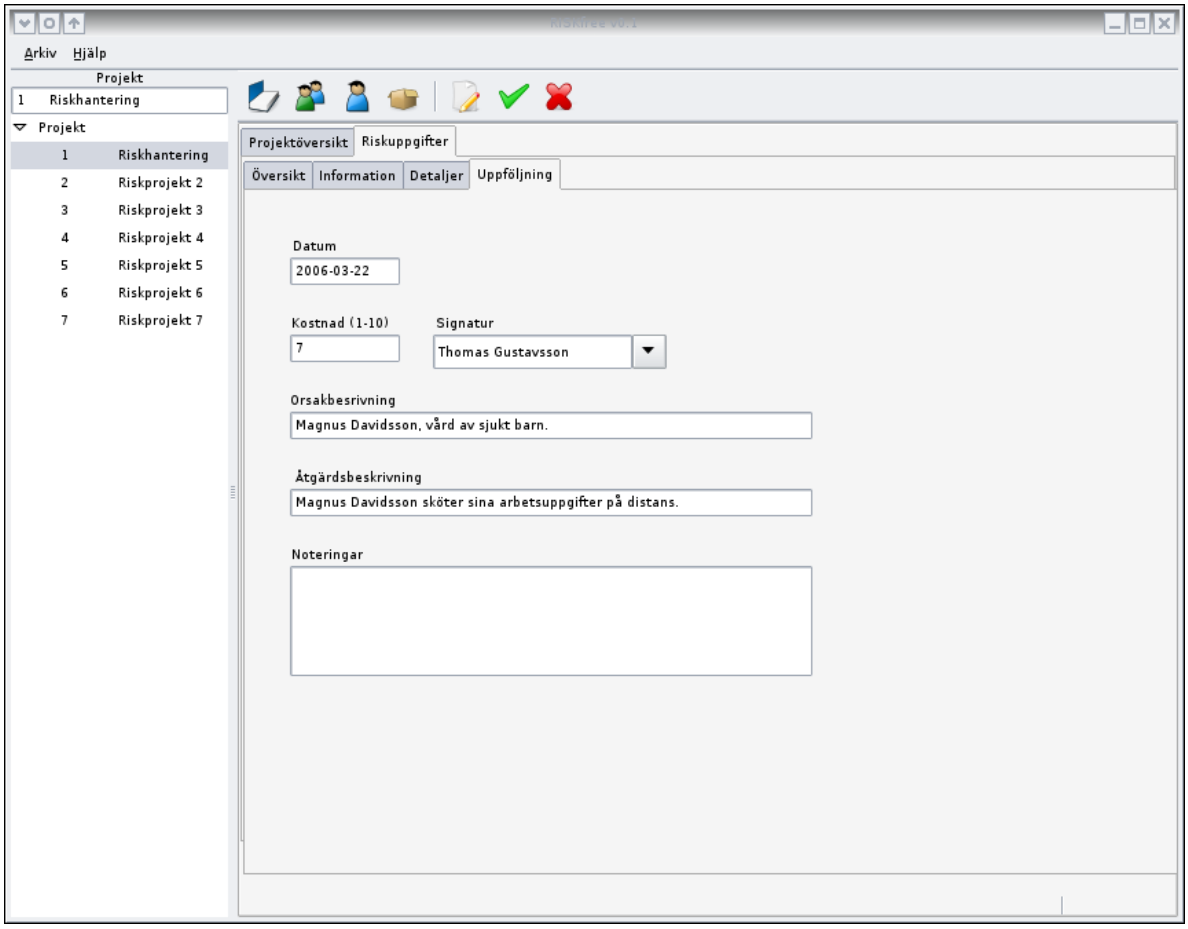
Det primära målet med dokumentation av ändringar är att få dessa spår- och kontrollerbara under projektets livscykel. När en loggrad adderas till riskuppgiften får den automatiskt en tidstämpel, dock måste ändringen signeras av en projektmedlem innan den kan adderas till riskuppgiften med hjälp av knappen *Lägg till*.



Figur 16 Riskvy - Detaljer

Efter att projektet har avslutats bör riskuppgifterna utvärderas där riskuppgiftens uppskattade värde jämförs med de faktiska värdena, såsom sannolikhet och konsekvens. På detta sätt kan projektgrupper skapa sig en kunskapsbank där erfarenheter från tidigare projekt återanvänds och tas tillvara.

Denna uppföljning görs i riskvyn under fliken *Uppföljning* [Figur 17] och här lagras information om kostnadsvärde, orsak, åtgärd och noteringar för riskuppgiften.



Figur 17 Riskvy - Uppföljning

4.8 Designmönster och principer

Både server och klientapplikationen är utvecklade runt flera olika typer av designmönster och principer [5]. Orsaken till detta är att separera beroendet mellan moduler i källkoden, och på så sätt minska påverkan vid ändringar och nyutveckling. Designmönster och principer har även utnyttjats för att skapa en bättre logisk uppbyggnad av källkoden.

Bland annat har Command-mönster använts i server-delen för att uppfylla The Single-Responsibility Principle och The Open-Closed Principle. Detta innebär att funktionalitet lätt kan adderas till servern då de nya klasserna endast behöver implementera interface-klassen för Command. Vidare har mönstret Facade använts för serverns databasklass.

På klientsidan sköter Observer-mönster separationen mellan data och det grafiska gränssnittet. Fördelen med denna lösning är att det grafiska gränssnittet endast kommer att presentera data och behöver inte behandla dessa data på något sätt. Därför sker behandlingen av data i modell-klassen istället, vilket skapar en logisk uppdelning mellan data och presentation.

5 Resultat

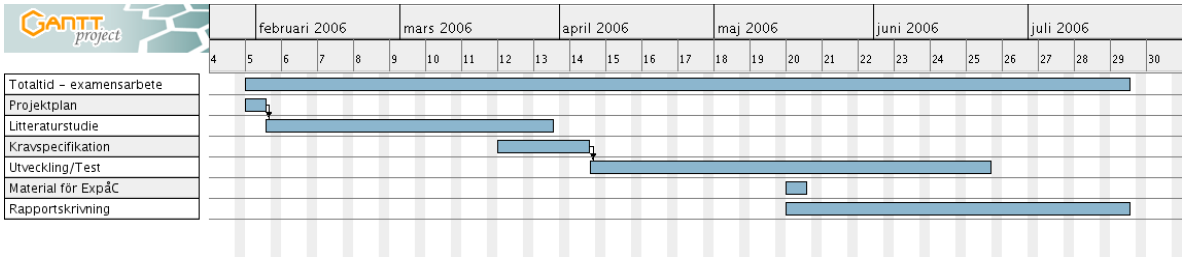
Målet med detta examensarbete är att utveckla ett system för riskhantering inom programvaruprojekt. Då min domänkunskap inom detta område var begränsad krävdes det en grundlig litteraturstudie. Denna litteraturstudie gav mycket information inom riskhanteringsområdet på grund av dess väldokumenterade teoridel. Dock finns det många olika åsikter hur bra riskhantering skall utföras och därför vore fler intervjuer med projektledare/deltagare varit önskvärt för att få en mer verklig bild.

5.1 Tidsplan

I inledningsskedet av examensarbetet tycktes utvecklingsarbetet och implementation av systemet vara ganska begränsat. Men allt efterhand som domänkunskapen ökade så ökade även omfattningen av systemet. Valet av att dela upp systemet i mindre delsystem påverkade också omfattningen på examensarbetet, då källkodens storlek blev betydligt större än beräknat.

Därför borde examensarbetets avgränsning ha varit bättre definierade eller examensarbetet haft en extra deltagare. En lämplig avgränsning vore att endast implementera E/R-diagrammet i databasen. Detta hade förmodligen gjort tidsplanen mer realistisk och sannolikheten för att den ursprungliga tidsplanen hade efterlevts hade ökat. Därför fick den ursprungliga tidsplanen överges och en ny tidsplan [Figur 18] med extra tid för utvecklingsarbete, test och rapportskrivning skapades.

Orsaken till att utvecklingsarbetet krävde mer tid än planerat berodde till stor del av den bristfälliga dokumentationen för Mono, men framförallt saknades dokumentation till GTK# som användes i implementationen av den grafiska klientapplikationen. Men mycket tid har även ägnats till systemdesign för att underlätta efterföljande uppdateringar och tester. Den största orsaken till rapportskrivningen inte höll tidsramen är att implementationsarbetet överskred tidsplanen, vilket i sin tur gjorde det svårt att utföra rapportförfattandet parallellt med implementationen.



Figur 18 Verklig tidsplan

5.2 Projektprocess

I projektplanen definierades vilka processmodeller som skulle användas. Valet föll på att använda den processmodell som passade bäst för den aktuella projektfasen. Den inledande fasen av examensarbetet bestod av följande:

- Skapa en projektplan.
- Genomföra en litteraturstudie inom riskhantering.
- Utifrån litteraturstudien skapa en domänanalys.
- Utifrån litteraturstudie och domänanalys skapa en kravspecifikation.

Då ovanstående punkter är mycket väldefinierade är de även beroende av att de utförs i korrekt ordning, och utifrån dessa kriterier valdes vattenfallsmodellen. Detta var ett utmärkt val av processmodell, då arbetet blir utfört på ett strukturerat sätt utan påverkan från andra delmoment.

Efter att dokumenten hade granskats och satts i baseline inleddes utvecklingsfasen då högnivådesignen utvecklades. Samtidigt övergick processmodellen till en iterativ vattenfallsmodell där berörda dokument uppdaterades samtidigt som utvecklingen utfördes. När utvecklingsarbetet hade nått en viss nivå påbörjades implementationen av systemet. I detta skede infördes även den testdrivna processmodellen. Även här fungerade processmodellerna väl. För varje ny funktion som infördes i systemet utfördes tester för att verifiera dess korrekthet, vilket medförde att inga nya funktioner implementerades innan tidigare funktioner var testade.

Då felaktigheter upptäcktes i dokumentation eller design åtgärdades detta genom en iterativ process.

5.3 Projektdokument

Totalt skapades fyra projektdokument för examensarbetet, där vart och ett inriktade på att beskriva olika delar av examensarbetet. Dessa dokument har kontinuerligt uppdaterats för att säkerställa dess korrekthet. Dokumentation av kravspecifikation och högnivådesign är de dokument som har uppdaterats störst antal gånger under examensarbetet. Detta har dock en naturlig förklaring då dessa dokument är starkt knutna till implementationen av systemet.

För att strikt följa en testdriven utvecklingsmodell i implementationsfasen av systemet skall de olika testfallen dokumenteras först i en testspecifikation med testinstruktioner. Dessa testfall har inte skapats under examensarbetet då arbetsbelastningen ansågs bli för hög i förhållande till vad vinsten vore i detta stadium.

5.4 Design och implementation

Design och implementation har skett i GNU/Linux-miljö med hjälp av Mono, C#, Monodevelop, GTK# och Firebird. Alla dessa applikationer och verktyg är fria under GPL eller med ekvivalent licensmodell.

Normaliseringen av databasen har skett i enlighet med BCNF, detta för att undvika redundans och problem vid operationer på tabeller och relationer. BCNF är en vanligt förkommande normaliseringsform och ansågs vara fullt tillräcklig för detta system.

I implementationsarbetet har olika typer av designmönster använts, i både server- och klientapplikationen, i de fall där dessa har ansetts vara befogade och inte tillfört någon extra komplexitet. Dessa designmönster kommer framförallt att underlätta fortsatt utveckling av systemet.

5.4.1 Nuvarande status

Systemet har nått det stadium där de mest grundläggande funktioner finns implementerade, men mycket funktionalitet kan tillföras. Exempelvis bör möjligheten att återanvända risker från tidigare projekt implementeras. Men även riskdiagram med utvärdering med hjälp av Monte Carlo-simulering kan vara till stor nytta för att minimera osäkerheten vid uppskattning av sannolikhet och konsekvens.

För att återanvändning av risker i tidigare projekt skall vara möjlig behöver risker och projekt kategoriseras på ett korrekt sätt. Den nuvarande implementationen av systemet stödjer detta men genom en djupare analys av hur dessa kategoriseringar skall genomförs kan förmodligen noggrannhet och precision förbättras.

Vidare har heller ingen användarhantering implementerats förutom en länkad lista som är tänkt att sköta de inloggade användarna. Tanken med denna användarehantering var att förfrågningar från klientapplikationen till databasen inte skall behöva skicka användarnamn och lösenord upprepade gånger över närverket.

5.4.2 Problem och brister

Ett stort problem under implementationsfasen har varit bristen på dokumentation inom Mono-projektet, och framförallt vid utvecklingen av det grafiska användargränssnittet med GTK#. Då Mono är ett projekt under utveckling så har dokumentationen av Monos API blivit ganska eftersatt. Men efter mycket sökningar på webbsidor och i forum så hittades lösningar på de flesta problem och i de fallen som inte någon lösning kunde uppbringas så fick designen omarbetas.

Då utvecklingsarbetet uteslutande har skett i Monodevelop och GNU/Linux har bara viss grundläggande kompabilitetstest gjorts med Windows, men enligt dokumentation [16] skall kompabiliteten mellan Mono och Windows inte vara några problem.

6 Sammanfattning

Trots att riskhantering har avhandlats på teoretisk nivå i drygt 25 år så finns det fortfarande brister i den praktiska riskhanteringen inom dagens programvaruprojekt. Detta tyder på att riskhantering i teorin är förhållandevis enkelt att kontrollera men verkligheten speglar en helt annan sida.

Inom ramen för detta examensarbete har jag skapat grunden för ett riskhanteringssystem vars uppgift är att hantera risker inom programvaruprojekt på ett strukturerat och effektivt sätt. De ingående delarna i systemet är en relationsdatabas, en severapplikation samt en grafisk klientapplikation. Serverapplikationen är en distribuerad applikation där klientapplikationen anropar de distribuerade objekten på servern med hjälp av .NET Remoting-teknik.

Sammanfattningsvis har det utvecklade riskhanteringsystemet resulterat i följande punkter.

- Hantering och lagring av projekt- och riskinformation inom programvaruprojekt på ett strukturerat och effektivt sätt.
- Kunskapsbank till framtida projekt.
- Kategorisering av både projekt och risker.
- Möjlighet att återanvända tidigare erfarenheter från projekt och risker.
- Kontinuerlig användning under hela projektets livscykel.

6.1 Slutsatser

Detta examensarbete har skapat grunden för ett riskhanteringssystem som kan användas i utvecklingsprojekt inom programvarubranschen. Riskhantering i denna typ av projekt är ofta eftersatta då andra aktiviteter anses som viktigare och mer effektiva. Trots att drygt 70 % av alla programvaruprojekt misslyckas på ett eller annat sätt, verkar inte riskhantering få den uppmärksamhet som det förtjänar. En orsak till detta kan vara det till antalet få applikationer som existerar på marknaden inom området.

Utvecklingsfasen av applikationen har krävt betydligt mer tid än beräknat och detta beror till stor del på att begränsningarna av applikationen inte var

tillräckligt genomtänkta från början. När den övergripande designen av systemet genomfördes lades störst vikt vid att skapa en så enkel och flexibel lösning som möjligt. Denna grundtanke med att skapa en stabil grund som systemet kan vila på, kommer att underlätta framtida uppdateringar och förändringar i systemet. Då detta var det primära målet för de designbeslut som togs i inledningsskedet av utvecklingsfasen, fick storleken av implemetationsarbetet en något sekundär betydelse. Med denna grundfilosofi som en röd tråd genom implementationsfasen, är det nu möjligt att fortsätta med utvecklingen för utomstående utan att detta skall påverka den nuvarande implementationen av systemet.

Riskhanteringssystemet som har utvecklats är tänkt att användas kontinuerligt under alla de faser som ett utvecklingsprojekt genomgår. Alla personer som har användarrättigheter för att delta i ett specifikt projekt har möjlighet att följa just detta projekts riskhantering, detta gäller även kunder som då får möjlighet att granska och övervaka riskhanteringen inom deras projekt. Systemet är tänkt att underlätta arbetet för dem som ingår i riskhanteringsgruppen i projektet, då riskhanteringen blir mer strukturerad och i förlängningen mer effektiv. Men systemet är även tänkt att ge vanliga projektdeltagare mer insikt i de risker och problem som kan tänkas ingå i deras projekt. Även detta höjer effektiviteten av riskhantering och då framförallt genom ökad medvetenhet och öppenhet runt riskhanteringsfrågor av alla inblandande projektdeltagare.

Genom detta examensarbete har jag fått nöjet att prova på att arbeta med öppen programvara. Detta har gett både positiva och negativa erfarenheter. De positiva sidorna är bland annat att många av dessa programvaror är kompetenta och har minst lika bra funktionalitet som kommersiell programvara. Exempelvis fyller Mono-projektet en viktig funktion för alla C#-utvecklare som vill utveckla .NET-programvara för fler plattformar än bara Windows. Dessvärre har inte dokumentationen för Mono kommit lika långt som implementationen av dess klassbibliotek. Förhoppningsvis kommer detta att förbättras i framtiden, då mycket av utvecklingstiden spenderades till att leta efter andra utvecklares lösningar på relativt vanliga implementationsproblem.

6.2 Vidareutveckling

Den grundläggande strukturen av systemet har implementerats och E/R-diagrammet är fullständigt specificerat i databasen. Dock finns det funktioner i systemet som saknas.

En naturlig funktion som bör implementeras innan systemet används i en reell miljö är användarhanteringen i server-delen. Detta för att möjliggöra kontroll av vilka användare som har rättigheter att ansluta till systemet. För att

implementera en sådan funktion behövs antagligen ett administratörs-gränssnitt vilket då också behöver implementeras.

Säkerheten av kommunikationen mellan de olika delsystemen är också en viktig fråga som bör undersökas innan systemet tas i drift. Utan kryptering av datatrafiken är det mycket enkelt för utomstående att avlyssna datapaketen i nätverket och på så sätt få tillgång till användarnamn och lösenord. När inkräktaren väl har tillgång till dessa uppgifter kan han logga in i systemet utan att någon annan upptäcker intrånget.

Rena användareorienterade funktioner som saknas är riskdiagram, Monte Carlo-simulering och automatisk prioritering av riskuppgifter. Ett riskdiagram ger en bättre grafisk översikt av riskuppgiftens karaktär avseende på sannolikhet och dess riskvärde. Riskdiagram är också en förutsättning för att använda Monte Carlo-simulering.

Klient- och serverapplikation använder sig av .NET Remoting och distribuerad programmering för kommunikation sinsemellan. Detta gör det förhållandevis enkelt att konvertera applikationerna till att använda sig av exempelvis Web Services och låta klientapplikationen köras i en webbläsare istället för en separat applikation.

7 Terminologi

Benämning	Förklaring
.NET	.NET är ursprungligen ett antal produkter och teknologier från Microsoft.
.Net Base Class Library (BCL)	Innehåller klasser för gemensamma funktioner såsom läsning och skrivning av filer, grafikrendering och databasinteraktion.
.NET Remoting	Ett generellt sätt för separata applikationer att utbyta information och data.
API	Application Programming Interface, gränssnitt för att låta system och applikationer kommunicera med varandra.
Baseline	Nivå som ett dokument uppnår när en specifik förutbestämd aktivitet är avklarad, exempelvis en granskning.
BCNF	Boyce-Codd Normal Form, ett sätt att organisera en databas så informationen lagras på ett strukturerat och effektivt sätt.
C#	C-sharp är ett objektorgienterat programspråk utvecklat av Microsoft i samband av skapandet av .NET-plattformen.
Brandvägg	En brandvägg är en programvaru- eller hårdvarubaserad kontrollmekanism för nätverkstrafik.
Common Language Infrastructure	Beskriver bland annat hur applikationer skall kunna köras i flera olika miljöer utan att behövas skrivas om.
DBMS	Database Management System, en eller flera applikationer för att hantera databaser och utföra operationer på data lagrad i databasen.
DCOM	Distributed Component Object Model, en Microsoft teknologi för distribuerade programvarukomponenter att kommunicera med varandra över ett nätverk.
Distribuerad programmering	En programmeringsteknik för att låta delar av en applikation utföras på separata datorenheter.
GPL	GNU General Public License, en licensmodell för programvara som bland annat ger användaren rätt till att studera källkoden och modifiera den, skapa kopior av applikationen.

Benämning	Förklaring
GUI	Graphical User Interface, genom knappar och textfält kan användaren interagerar med ett datorsystem.
GTK#	Ett bibliotek för att skapa grafiska gränssnitt.
http	Hypertext Transport Protocol, ett transportprotokoll för att överföra information på World Wide Web (WWW).
Implementera	Utföra, tillämpa, realisera.
Interface-klass	Klass som specificerar vilka metoder underklassen skall implementera.
LAN	Local Area Network, begrepp som oftast avser ett nätverk baserat på grundläggande protokoll såsom Ethernet eller Token Ring. Ovan på detta körs ofta ett protokoll av högre nivå.
Likformig sannolikhetsfördelning	En likformig sannolikhetsfördelning innebär, i det diskreta fallet, att alla utfall i ett försök (exempelvis ett tärningskast) har samma sannolikhet.
Mono	En .NET-kompatibel utvecklingsplattform baserat på öppen källkod.
Monte Carlo-simulering	En metod för att uppskatta ett specifikt resultat genom multipla samplingar som innefattar slumptalsvariabler.
Redundans	Överflödig information som finns på mer än ett ställe.
SOAP	Simple Object Access Protocol, ett protokoll för att utbyta information i decentraliserade och distribuerade miljöer.
SQL	Structured Query Langage, standardiserat frågespråk för databaser.
Testdrivenmodell	Utvecklingsmodell där testfall av systemet dokumenteras före implementationen av programmet/funktionen.
Vattenfallsmodell	Utvecklingsmodell där väl avgränsade faser utförs i en specifik följd.
UML	Unified Modeling Language, ett visuellt objekt-orienterat generellt språk för modellering av system.
Web services	Webbaserade datorprogram som kommunicerar och samarbetar dynamiskt med andra webbtjänster.

Benämning	Förklaring
Windows	Ett operativsystem för persondatorer.
XML	eXtensible Markup Language, ett dokumentstrukturdefinitionsspråk med "taggar", "attribut", och "attributvärden". XML används för att strukturera och organisera information/data, till skillnad från t.ex. HTML som används till att åskådliggöra och visa data.

8 Referenser

1. Boehm B.W., “Software Risk Management: Principles and Practies”, IEEE Software, Januari 1991.
2. Ceri S., Mandrioli D., Sbattella L., “The Art & Craft of Computing”, Addison-Wesley, 1998.
3. DeMarco T., “Waltzing with Bears”, Dorset House Publishing Co, 2003.
4. Fairley Richard, “Risk Management for software projects”, IEEE Software, Maj 1994.
5. Martin R.C., “Agile Software Development Principles, Patterns, and Practices”, Prentice Hall, 2003.
6. McFarlan W., “Portfolio Approach to Information Systems”, Harvard Business Review, September 1981.
7. Ropponen J., Lyytinen K., ”Components of Software Development Risk: How to Adress Them? A project Manager Survey”, IEEE Transactions on Software Engineering, 2000.
8. Roy G.G., Woodings T.L., “A Framework for Risk Analysis in Software Engineering”, IEEE Software, 2000.
9. Ullman J.D., Widom J., “A First Course in Database Systems”, Prentice Hall, 2002.
10. Wallace L., Keil M., “Software Project Risks and their effect on outcome”, Association for Computing Machinery. Communication of the ACM, April 2004.
11. “.NET Remoting”, http://www.csharpcorner.com/Code/2004/Sept/NetRemoting.asp, Juli 2006
12. “Basics of .NET”, http://www.microsoft.com/net/basics.mspx, Juli 2006
13. “Firebird – Relational Database for the New millennium”, http://www.firebirdsql.org/, Juli 2006
14. “Gtk# Win32 Installer for Microsoft .NET Framework 1.1 SDK”, http://forge.novell.com/modules/xfmod/project/?gtks-inst4win, 2006-07-15
15. “Introduction to Monte Carlo Methods”, http://www.ipp.mpg.de/de/for/bereiche/stellarator/Comp_sci/CompScience/csep/csep1.phy.ornl.gov/mc/mc.html, Juli 2006
16. “Mono”, http://www.mono-project.com/Main_Page, Juli 2006
17. “Monodevelop”, http://www.monodevelop.com, Juli 2006
18. “Monte Carlo Method”, http://www.riskglossary.com/link/monte_carlo_method.htm, Juli 2006
19. “Hem- och fritidsolycksfall i Sverige”, http://www.socialstyrelsen.se, Augusti 2006
20. “Allt bättre säkerhet för fallskärmshoppare”, http://www.skyddsnatet.nu/SRV/Securite.nsf/0/e05a06b17ebb87e0c12570a000217e3a!OpenDocument&AutoFramed, Augusti 2006

9 Appendix

Appendix A, Projektplan

RISKfree

- en applikation för riskhantering -

PROJEKTPLAN

Ansvarig Mikael Gullstrand
Datum 2006-07-11
Varsion 1.0

Versionhistorik

Version	Datum	Ansvarig	Ändring
0.1	2006-03-24	Mikael Gullstrand	Skapad
0.2	2006-04-25	Mikael Gullstrand	Ändrat stycket Riskhantering
1.0	2006-07-11	Mikael Gullstrand	Uppdaterat tidsplanen samt satt i baseline efter granskning

Innehållsförteckning

1 Inledning

I dagens IT-centrerade värld sker utvecklingen av hårdvara och programvara i en rasande takt. Om ett företag skall ha möjlighet att hinna med i denna utvecklingstakt krävs det snabba och effektiva metoder i utvecklingsprocessen. Dessa krav från omvärlden på nya och mer komplexa produkter och tjänster kan innebära nya och stora risker för ett utvecklingsprojekt; både ekonomiska som tidsmässiga. Men riskhantering inom programvaruprojekt är ofta en process som anses få stå tillbaka då kravspecifikationer och utvecklingsarbete har en högre prioritet då dessa ses som mer produktiva. Men för att ett företag skall ha möjlighet att öka kvaliteten och leveranssäkerheten av sina produkter, bör riskhanteringsprocessen bli en integrerad del som genomsyrar ett projekts alla olika faser. För att undvika obehagliga överraskningar bör dessa risker identifieras, övervakas och hanteras på ett strukturerat sätt.

Målet med detta projekt är att utveckla ett databasdrivet system, där information om projektet och dess risker lagras. Ett stort problem med riskhantering är att riskerna kan skilja sig markant från projekt till projekt. Men om informationen från tidigare projekt kategoriseras på ett strukturerat sätt bör denna information vara möjlig att används för senare projekt.

2 Utvecklingsplan

Då de nuvarande kunskaperna inom denna domän är starkt begränsade kommer en iterativ vattenfallsmodell att användas som projektprocess. Detta för att de dokument som skapas under projektets gång, kommer att kräva kontinuerlig omarbetning. Som utvecklingsprocess av applikationen kommer en testdriven utvecklingsprocess att användas. Denna metod är lämplig då applikationen blir förhållandevis liten och kravspecifikationen kan komma att förändras under projektets gång.

2.1 Dokumentaktiviteter

I projektet kommer följande dokument att skapas runt projektet. Dessa dokument kommer behöva omarbetas kontinuerligt under projektets olika faser.

2.1.1 Projektplan (PP)

Detta dokument beskriver projektet på ett övergripande plan samt mål och bakgrund för projektet. Dokumentet innehåller även vilka olika ramverk och standarder som skall användas i projektet. Tidsplanen kommer att justeras under projekts gång men deadlines bör efterlevas så långt som möjligt.

2.1.2 Domänanalys (DA)

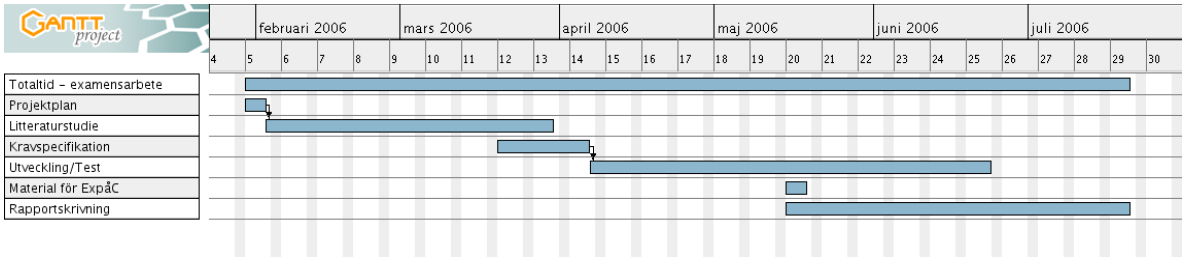
För att kunna skapa en så komplett och relevant kravspecifikation som möjligt för projektet, skall en domänanalys genomföras. Denna domänanalys beskrivs i detta

dokument, där bland annat vilka kraveliciteringstekniker och kravstilar som skall användas specificeras.

2.1.3 Kravspecifikation (SRS)

För att det ska vara möjligt att upprätta en kravspecifikation måste systemets krav identifieras, analyseras och dokumenteras. Hur denna process skall genomföras finns dokumenterat i DA. Det är först efter dessa moment som det är möjligt att skapa en kravspecifikation för systemet. Kvaliteten på kravspecifikationen spelar en nyckelroll för utgången av projektet, då de krav som finns specificerade här ligger till grund för hela systemet. Det är även viktigt att kraven som ställs på systemet skall vara testbara.

3 Tidsplan



Figur 1, Tidsplan för projekt

4 Konfigurationshantering

4.1 Versionshantering

Dokumentation och källkod som skapas under projektets gång kommer att versionshanteras med hjälp av Subversion (SVN). SVN är huvudsakligen för versionshantering av icke binära filer, men det hindrar inte att man kan hanterar binära filer ändå. I detta fall fungerar SVN mer som en backuplösning än versionshantering. SVN gör även det möjligt att gå tillbaka till äldre versioner av filer och även se vad som ändrats och vem som utförde en specifik ändring.

4.1.1 SVN-katalogstruktur

Den grundläggande katalogstrukturen av SVN ser ut på följande sätt.

```
examensarbete/
…/docs – dokument och manualer
…/…/litterature – extern dokumentation
…/…/misc – dokument som inte direkt passar i någon annan katalog
…/…/project – dokument skapade under projektets olika faser
```

/…/…/report – slutgiltig rapport om examensarbetet
…/develop – källkodsfiler
…/…/docs – dokumentation av källkod
…/…/src - källkod

4.1.2 Format för dokumentnamn

Officiella dokumenten kommer att identifieras med hjälp av följande format; *rf_<Typ av dokument>_<Version>.*

Exempelvis:
rf_pp_0.1

Det skall klart framgå vilken baslinje projektet befinner sig på via versionnumret. Versionnummret formateras på följande sätt:

- versionnumret börjar med 0.1 och har ett inkrement på 0.1 vid varje ny version av dokumentet.
- första baslinje har versionsnumer 1.0 och därefter ökas till närmaste heltal vid en ny baslinje.

Ändringshanteringen kommer att ske automatiskt med hjälp av *SVN,* och inga särskilda dokumenten för ändringshantering kommer att behövas.

5 Standard och hjälpmedel

5.1 Specifikation av programhjälpmedel, teknik och metoder

Följande verktyg och hjälpmedel skall användas i projektets olika faser.

5.1.1 Dokumentation

- Textdokument – Openoffice 2.0 (filformat OpenDocumentText)
- Källkod – Doxygen
- E/R-diagram – Dia
- UML-diagram – ArgoUML
- Ganttshema - Ganttproject

5.1.2 Utveckling

- Ramverk – Mono
- Programspråk – C#
- Utvecklingsmiljö – Monodevelop
- Klient GUI – GTK-Sharp
- Databashanterare - Firebird
- Administratörgränssnitt DB - ibWebAdmin

5.2 Design och kodningsstandard

5.2.1 Dokumentation av källkod

All dokumentation av källkod genereras automatiskt med hjälp av Doxygen: Källkoden skall därför dokumenteras enligt doxygen-notationen.
Exempel:

```
/**
 *  A test class. A more elaborate class description.
 */

class Test
{
  public:

    /**
     * An enum.
     * More detailed enum description.
     */

    enum TEnum {
         TVal1,  /**< enum value TVal1. */
         TVal2,  /**< enum value TVal2. */
         TVal3   /**< enum value TVal3. */
        }
      *enumPtr,  /**< enum pointer. Details. */
      enumVar;   /**< enum variable. Details. */

      /**
       * A constructor.
       * A more elaborate description of the constructor.
       */


     Test();


      /**
       * a normal member taking two arguments and returning an integer      value.
       * @param a an integer argument.
       * @param s a constant character pointer.
       * @see Test()
       * @see ~Test()
       * @see testMeToo()
       * @see publicVar()
       * @return The test results
       */
```

```
    int testMe(int a,const char *s);

    /**
    * A pure virtual member.
    * @see testMe()
    * @param c1 the first argument.
    * @param c2 the second argument.
    */
    virtual void testMeToo(char c1,char c2) = 0;

    /**
    * a public variable.
    * Details.
    */
    int publicVar;

    /**
    * a function variable.
    * Details.
    */
    int (*handler)(int a,int b);
};
```

5.2.2 Filnamn och modulnamn

Det kommer att finnas två modulnamn (namespace), rf_srv och rf_cnt. De funktioner och klasser som hör till en specifik modul skall skrivas under respektive modulnamn rf_srv eller rf_cnt. Detta förfarande förhindrar namnkrockar mellan klasser/metoder både i och utanför projektet.

Alla filnamn skall skrivas med små bokstäver, ord skall delas med understräck och börja med rf_.

Exempel på en klass som ligger under modulnamnet rf_srv.

```
namespace rf_srv
{
    class NetworkServer
    {
        private:
        public:
    }
}
```

Exempel på källkodsfiler enligt notationen.
rf_svr_network_server.cs

5.2.3 Notation av källkod

Klass- och metodnamn skall börjar med en versal, och i de fall som klassnamnet består av flera ord skall även dessa börja med en versal. Konstanter skapas som statiska variabler med stora bokstäver. Alla variabelnamn börjar med en gemen och om variabelnamnet innehåller flera ord används en versal i början av varje delord. I övrigt gäller de notations regler som finns beskrivna i C# Code Style Guide [2]. För att få ett enhetligt utseende på källkoden skall alla klassnamn, variabelnamn och övriga entiteter i källkoden vara på engelska.

Exempel:

```
namespace ShowMeTheBracket
{

   public enum Test {
      TestMe,
      TestYou
   }

   public class TestMeClass
   {
      Test test;
      public Test Test {
         get {
            return test;
         }
         set {
            test = value;
         }
      }


      void DoSomething()
      {
         if (test == Test.TestMe) {
            //…stuff gets done
         } else {
            //…other stuff gets done
         }
      }
   }
}
```

5.3 Dokumentmall

Dokument producerade i projektet skapas enligt en given mall (se Appendix A). Första sidan är ett försättsblad med dokumentets namn, datum, version, ansvarig och uppgjord av. Alla övriga sidor skall ha en sidfot med centrerad sidnumrering. Sidan efter försättsbladet skall innehålla versionshistorik med versionsnummer, datum, ansvarig och kommentar till varför uppdatering skett.

6 Riskhantering

Riskhanteringen är en viktig del i projektet och kan få stora konsekvenser för projektets utfall. Riskhantering måste därför kontinuerligt utföras under hela projektets livslängd, då riskbilden ändras under projektets olika faser. Därför skall det avsättas minst en halv timme varje fredag för riskhantering.

Riskerna för projektet dokumenteras i projektets domänanalys [1] och identifieras med ett ID-nummer tillsammans med en uppskattad sannolikhet för att risken inträffar och en uppskattad konsekvensgrad. Konsekvensgraden anger hur stor skada risken skulle orsaka om den drabbade projektet.

7 Referenser

[1] Domänanalys rf_da_0.1
[2] C# Code Style Guide, Version 1.2, Scott Bellware

Appendix
Appendix A, Dokumentmall

Projekt

- underrubrik -

DOKUMENT

```
Ansvarig Namn
Datum    ÅÅÅÅ-MM-DD
Varsion  x.x
```

---------- Sidbrytning ----------

Versionhistorik

Version	Datum	Ansvarig	Ändring
x.x	ÅÅÅÅ-MM-DD	Namn	Beskrivning

---------- Sidbrytning ----------

Innehållsförteckning

---------- Sidbrytning ----------

Överskrift 1
Överskrift 2
Överskrift 3
Brödtext
Källkodsexempel

ID	Överskrift
1	Tabellinnehåll

Appendix B, Domänanalys

RISKfree

- en applikation för riskhantering -

DOMÄNANALYS

Ansvarig Mikael Gullstrand
Datum 2006-07-11
Varsion 1.0

Versionhistorik

Version	Datum	Ansvarig	Ändring
0.1	2006-03-22	Mikael Gullstrand	Skapad
0.2	2006-04-05	Mikael Gullstrand	Lagt till *Task descriptions* som kravstil. Samt lagt till exempel på de olika kravstilarna.
0.3	2006-04-21	Mikael Gullstrand	Diverse ändringar efter möte med Christin.
1.0	2006-07-11	Mikael Gullstrand	Satt i baseline efter granskning.

Innehållsförteckning

Figurförteckning

1 Inledning
RISKfree är en programvara som skall underlätta riskhantering i utvecklingsprojekt inom programvaruområdet. Denna programvara är även tänkt att användas som ett verktyg i den allmänt kallade PUSS-kursen, vid LTH Campus Helsingborg, för att dokumentera och spåra risker. Systemet skall vara uppbyggt så att en vidare expansion av funktioner och tjänster kan implementeras på ett enkelt sätt.

2 Intressenter
RISKfree har ett antal olika intressenter i sin yttre och inre domän, och dessa har indelats i följande kategorier.

2.1 Riskhanteringsgrupp
Riskhanteringsgrupp som deltar i ett utvecklingsprojektet, har intresse av att systemet är användarvänligt och pålitligt. Detta är den intressentgrupp som har störst operativ kontakt med systemet, vilket gör att stor vikt bör läggas på att få systemet lättförståelig, lättanvänt och korrekt då kunskaperna inom riskhantering kan variera från deltagare till deltagare inom riskhanteringsgruppen.

2.2 Kund
Kunden är den som är beställare av projektet. Han har intresse av att riskhanteringen utförs på ett säkert och effektivt sätt för att minimera effekterna av dessa.

2.3 Systemadministratör
Administratören har intresse av att systemadministrationen kan ske på ett säkert och tillförlitligt sätt.

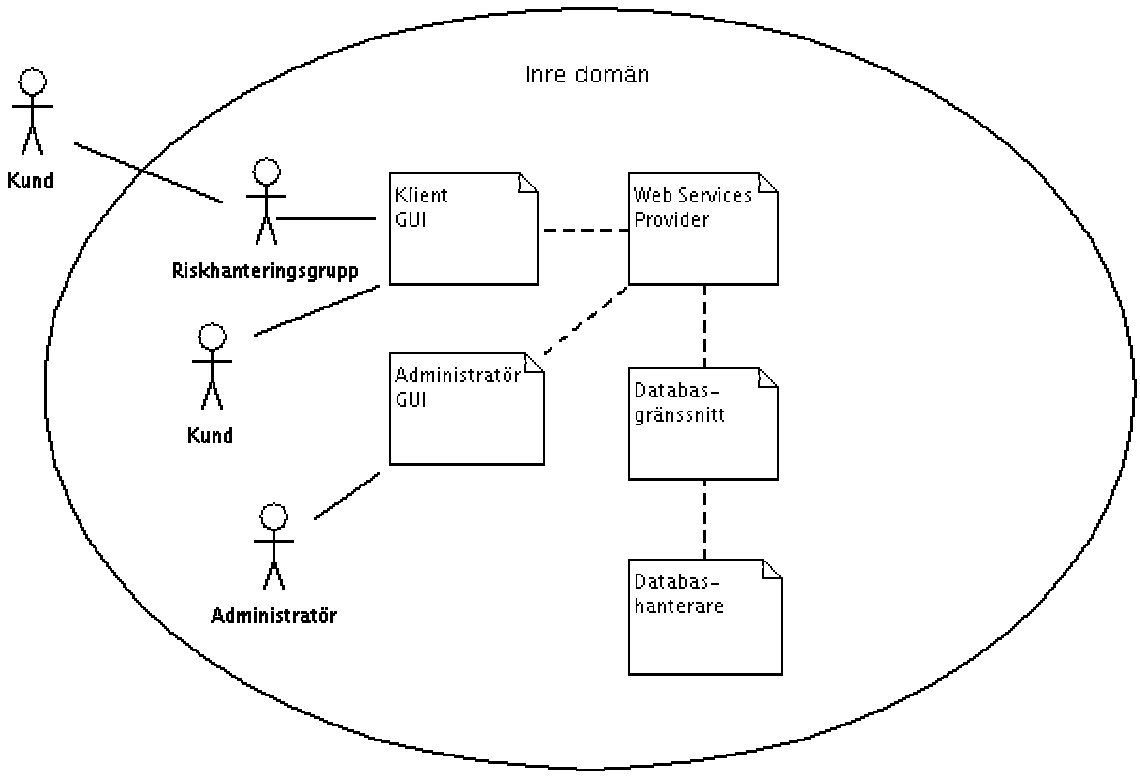
2.4 Externa utvecklare
Utomstående utvecklare har intresse för att vidare expansion av systemets funktioner och tjänster kan implementeras på ett enkelt sätt.

2.5 Företag
Programvaruföretag som önskar att använda sig av ett datoriserat riskhanteringssystem i sina utvecklingsprojekt har intresse för att använda sig av ett lättanvänt och produktivt verktyg.

2.6 Institutionen för telekommunikationssystem i Lund

Institutionen för telekommunikationssystem i Lund, som är ansvarig för examinationen av detta examensarbete. Framförallt är det Christin Lindholm och Mats Lilja som i sina roller som examinator respektive handledare som är intressenter.

3 Domän



Figur 1: Aktörer i den inre- och yttredomänen

3.1 Inre domän

Den inre domänen består av följande aktörer och komponenter.

3.1.1 Aktörer

- Riskhanteringsgrupp – dokumenterar information om de olika risker som skall administreras i projektet.
- Kund – kan följa projektets riskhantering i realtid. Detta ger större förtroende för projektet och kunden kan även upptäcka risker som inte utvecklarna har upptäckt.

- Administratör – administrerar databasen såsom exempelvis lägger till och tar bort användare.

3.1.2 Komponenter

- Web Services Provider – har till uppgift att sammanlänka användargränssnitt och underliggande lager mot databasen.
- Databasgränssnitt – fungerar som ett abstraktionslager mellan databas och Web Services Provider. Detta för att underlätta vid byte av databas.
- Databashanterare – möjliggör den faktiska lagringen av projektinformation.

3.2 Yttre domän

Den yttre domänen har bara en aktör

3.2.1 Aktörer

- Kund – kommunicerar med utvecklarna med önskemål på krav och funktioner i projektet.

4 Kraveliciteringstekniker

För att ha möjlighet att skapa en optimal och relevant kravlista kommer flera olika kraveliciteringstekniker att användas. De krav som uppkommer vid dessa aktiviteter kommer sedan att reduceras och prioriteras efter relevans.

4.1 Litteraturstudie

Litteraturstudie görs på artiklar och böcker inom området riskhantering. Andra dokument som behandlar ämnena sannolikhetsteori, integralkalkyl och programmeringsspråket C# är även av intresse.

4.2 Brainstorm

Brainstorming kommer att användas för att få fram så många möjliga som omöjliga idéer. Dessa idéer kommer sedan noggrant utvärderas beroende på relevans och prioritet för systemet.

5 Kravstilar

De krav som identifieras med hjälp av kraveliciteringstekniker kommer att dokumenteras med nedanstående kravstilar. Beroende på vilken typ av krav som skall dokumenteras kommer lämplig kravstil att användas.

5.1 E/R-diagram
För att visa de krav på vilken typ av data som skall lagras i databasen, kommer E/R-diagram att användas. E/R-diagram gör det enkelt att verifiera att data som hanteras av systemet överensstämmer med kraven. Fördelen med E/R-diagram på domännivå är att dessa även kan användas för beskrivning av produktnivå.

5.2 Virtual windows
Virtual windows är förenklade skärmdumpar, där realistisk information visas men där knappar och menyer saknas. Med hjälp av virtual windows blir det enklare att validera datamodellen som finns beskriven i de olika E/R-diagrammen. Virtual window understryker även vilken typ av information som användaren förväntas ange.

5.3 Task deskriptions
Kravstilen Task descriptions kommer användas för att beskriva produktens krav på domän-nivå. Denna kravstil är lätt att förstå både för användare och utvecklare, samt kraven är lätta att verifiera då denna kravstil innehåller underuppgifter och varianter.

5.4 Feature requirements
För att dokumentera de krav som inte behandlas under ovanstående kravstilar kommer Feature requirements att användas. Fördelen med denna kravstil är att kraven oftast kan direkt översättas till funktioner i systemet.

6 Risker

6.1 Riskanalys
S = Sannolikhet 1-5 där 1 är osannolikt och 5 är mycket sannolikt
K = Konsekvenser 1-5 där 1 är minimala konsekvenser och 5 är katastrofala konsekvenser

6.1.1 Projektgrupp

ID	S	K	Beskrivning av risk	Åtgärd
6.1.1.1	4	3	Kort frånvaro av gruppmedlem.	Tidsplanera med tidsbuffert.
6.1.1.2	1	5	Lång frånvaro av gruppmedlem.	Finns ingen. Projektet kommer att bli försenat.
6.1.1.3	3	3	Gruppen har inte tillräcklig kunskap.	Vidarutbilda.
6.1.1.4	1	4	Gruppmedlem är oengagerad.	Samtal med handledare.

Figur 2: Projektgrupps-relaterade risker

6.1.2 Handledare

ID	S	K	Beskrivning av risk	Åtgärd
6.1.2.1	4	3	Liten frånvaro av handledare.	Planera så att vi har lite spelrum med tiden i de fall handledaren är inblandad.
6.1.2.2	2	4	Lång frånvaro av handledare.	Examinatorn kan ta på sig rollen som handledare.
6.1.2.3	1	3	Kommunikationsproblem med handledare.	Håll kontinuerlig kontakt.

Figur 3: Handledar-relaterade risker

6.1.3 Planering

ID	S	K	Beskrivning av risk	Åtgärd
6.1.3.1	5	3	Underskattning av tidsplan.	Övervaka kontinuerligt.
6.1.3.2	4	4	Underskattning av riskanalys.	Övervaka kontinuerligt.
6.1.3.3	4	3	Underskattning av systemets komplexitet.	Prioritera och implementera de viktigaste bitarna först. Omförhandla med handledare.
6.1.3.4	4	3	Underskattning av storleken av projektet.	Begränsa omfattningen. Omförhandla med handledare.

Figur 4: Planerings-relaterade risker

6.1.4 Projekt

ID	S	K	Beskrivning av risk	Åtgärd
6.1.4.1	3	1	Viktiga filer raderas.	Använd SVN.
6.1.4.2	2	2	Problem med hårdvara.	Testa direkt mot hårdvara vi kommer att använda i projektet.
6.1.4.3	3	2	Problem med tredjeparts programvara.	Testa tidigt. Identifiera ersättningsprogramvara.
6.1.4.4	1	3	Otillräcklig dokumentation av Web Services.	Inhämta information tidigt i projektet.

Figur 5: Projekt-relaterade risker

6.1.5 Programutveckling

ID	S	K	Beskrivning av risk	Åtgärd
6.1.5.1	2	3	Problem med utvecklingverktyg.	Testa tidigt. Identifiera ersättningsverktyg.
6.1.5.2	2	2	Problem med programspråk och ramverk.	Testa tidigt med programvara vi kommer att använda i projektet.
6.1.5.3	2	2	Problem med att köra GUI-klienter både under Windows och Linux.	Testa tidigt. Möjligt att byta till Java.
6.1.5.4	3	4	Problem med tredjeparts komponenter.	Testa tidigt. Identifiera ersättningsprogramvara.
6.1.5.5	2	1	Utveckling av funktioner som inte behövs eller är onödiga.	Börja inte utvecklingen förens kravspecifikationen är klar.
6.1.5.6	2	4	Problem för systemet att stödja flera användare samtidigt.	Testa tidigt. Eventuellt behövs trådhantering.
6.1.5.7	1	4	Otillräcklig dokumentation av Web Services.	Inhämta information tidigt i projektet.

Figur 6: Programutvecklings-relaterade risker

6.2 Hantering av risker

Som i alla andra projekt spelar tiden en avgörande och viktig roll i detta projekt. Vid granskning av tabellerna i kapitel 6.1 så blir det extra tydligt att riskerna i Tabell 3 har störst sannolikhet att inträffa och samtidigt får dom störst konsekvenser. Men man får ej glömma att riskerna i Tabell 3 är intimt knutna till riskerna i Tabell 5. Ifall någon av riskerna i Tabell 5 inträffar så kommer med stor sannolikhet någon av riskerna i Tabell 3 att inträffa. Därför bör dessa två riskgrupper övervakas extra noggrant. För att kontinuerligt göra uppföljning av riskerna i projektet, kommer dessa att utvärderas varje fredag för att se ifall nya risker har tillkommit eller gamla risker kan avskrivas.

7 Terminologi

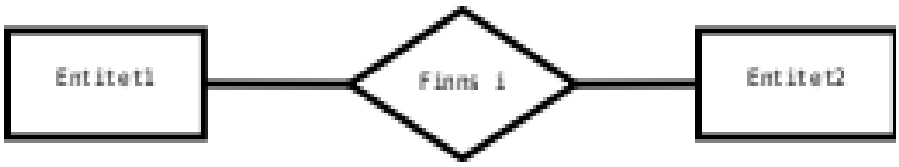
Benämning	Förklaring
PUSS-Kurs	Programvaru Utveckling för Stora System är en projektkurs med 15-20 deltagare. Projektet är att utveckla tilläggtjänster till en befintlig telefonväxel. Kursen genomförs vanligtvis på vårterminen i årskurs två inom ramen för IT-programmen på Campus Helsingborg LTH.
Web Services	Web Services är applikationer eller delar av applikationer som kan göras allmänt tillgängliga över Internet med hjälp av protokoll som är skrivna i XML.
Databashanterare	Databashanterare är en samling information som är organiserad på ett sådant sätt att det är lätt att söka efter och hämta enskilda bitar information, samt ofta även att ändra informationen.
XML	eXtensible Markup Language, ett dokumentstrukturdefinitionsspråk med "taggar", "attribut", och "attributvärden". XML används för att strukturera och organisera information/data, till skillnad från t.ex. HTML som används till att åskådliggöra och visa data.

8. Referenser

[1] RISKfree Projektplan v.0.1 (rf_pp_0.1)

9. Appendix

Appendix A, Exempel på E/R-daigram



Appendix B, Mall för Task descriptions

Uppgift:	T1: Namn på uppgift
Mål:	Målet med uppgiften.
Trigger:	Vad orsakade uppgiften.
Frekvens:	Hur ofta förekommer uppgiften.
Kritiskt:	I vilka lägen är uppgiften kritisk:
Underuppgifter:	
1.	Underuppift 1.
2.	Underuppift 2.
3.	Underuppift 3.
Varianter:	
1a.	Variant
1b.	Variant

Appendix C, Exempel på Feature requirements

SRS1: Systemet skall kunna kopplas till olika DBMS.
SRS2: Systemet skall stödja uppgifterna T1-T5.

Appendix C, Kravspecifikation

RISKfree

- en applikation för riskhantering -

KRAVSPECIFIKATION

Ansvarig Mikael Gullstrand
Datum 2006-07-11
Varsion 1.0

Versionhistorik

Version	Datum	Ansvarig	Ändring
0.1	2006-03-29	Mikael Gullstrand	Skapad
0.2	2006-04-21	Mikael Gullstrand	Ändrade krav: SRS44101, SRS44102, SRS44103, SRS44112, SRS44114, SRS44116, SRS44210, SRS44211 Tagit bort krav: SRS44206 Uppdaterat E/R-diagram
0.3	2006-05-11	Mikael Gullstrand	Uppdaterat E/R-diagram till ver. 0.3
0.4	2006-05-18	Mikael Gullstrand	Uppdaterat E/R-diagram till ver. 0.4
0.5	2006-07-05	Mikael Gullstrand	Lagt till Appendix B. Flyttat krav till appendix B: SRS42002, SRS44105, SRS44108, SRS44110, SRS44115, SRS44116, SRS44117, SRS44119, SRS44203, SRS45205 Tagit bort krav: SRS44207
1.0	2006-07-11	Mikael Gullstrand	Satt i baseline efter granskning

Innehållsförteckning

Figurförteckning

1 Inledning

Detta dokument beskriver de krav som skall stödjas av systemet RISKfree. Systemets uppgift är att på ett enkelt och effektivt sätt lagra och hantera information om risker i programvaruprojekt. Ytterligare information finns i projektplanen [Ref. 1].

2 Referenser

[1] RISKfree Projektplan 0.1

3 Bakgrund och mål

Målet med detta projekt är att utveckla ett databasdrivet system, där information om projektet och dess risker lagras. Ett stort problem med riskhantering är att riskerna kan skilja sig markant från projekt till projekt. Genom att informationen från tidigare projekt kategoriseras och sparas på ett strukturerat sätt, kan man återanvända denna information för att kunna dra mer korrekta slutsatser om det nuvarande projektet.

4 Systemkrav

Detta kapitel beskriver de krav som gäller riskhantering systemet RISKfree. Varje krav har ett unikt nummer som inleds med SRSx, där x är kravets unika nummer. Varje krav innehåller en skall-sats.

4.1 Projektkrav

4.1.1 Utvecklingsmiljö

SRS41101. Monodevelop skall användas som utvecklingsmiljö.
SRS41102. Utvecklingsspråket skall vara C#.
SRS41103. Mono skall användas som .NET ramverk.

4.1.2 Testkrav

SRS41201. Utvidgning
SRS41202. Utvidgning

4.1.3 Leveranskrav

SRS41301. Systemet skall levereras som EXE-filer.

4.1.4 Projektplan och dokumentation

SRS41401. Utvecklingen av systemet skall överensstämma med Projektplanen [1].
SRS41402. Dokumentationen av utvecklingen skall överensstämma med Projektplanen [1].

4.2 Målkrav

Målkrav för systemet är följande:
SRS42001. Systemet skall hantera identifikation av risker.
SRS42002. Utvidgning
SRS42003. Systemet skall hantera prioritering av risker.
SRS42004. Systemet skall hantera uppföljning av risker.

4.3 Domänkrav

SRS43001. Systemet skall understödja Uppgift 1 – 3.

Uppgift:	1: Skapa projekt
Mål:	Skapa ett nytt projekt. Lägg till projektdeltagare.
Trigger:	Företaget har fått ett nytt projekt.
Frekvens:	
Kritiskt:	
Underuppgifter:	
1.	Skapa ett nytt projekt.
2.	Ge projektet ett projekt-id.
3.	Ange projektledare.
4.	Lägg till projektdeltagare.
Varianter:	
1a.	
1b.	

Uppgift:	2: Skapa riskuppgift
Mål:	Skapa en ny riskuppgift i ett projekt. Ge den ett unikt risk-id.
Trigger:	Användaren har upptäckt en ny risk i projektet.
Frekvens:	
Kritiskt:	
Underuppgifter:	
1.	Välj projekt.
2.	Skapa en ny riskuppgift.
3.	Ge den ett unikt risk-id.
Varianter:	
1a.	
1b.	

Uppgift:	3: Uppdatera riskuppgift
Mål:	Uppdatera en riskuppgift i ett projekt. Spara riskdata.
Trigger:	Användaren vill uppdatera informationen för en riskuppgift.
Frekvens:	
Kritiskt:	
Underuppgifter:	
1.	Välj projekt.
2.	Välj riskuppgift.
3.	Spara riskdata i logg.
4.	Uppdatera riskdata.
5.	Spara ny riskdata.
Varianter:	

4.4 Funktionella krav

Detta delkapitel beskriver de funktionella kraven av systemet, uppdelat i systemrelaterade krav samt riskrelaterade krav.

4.4.1 Applikationskrav

SRS44101. Systemet skall ha möjlighet att ha 50 st samtidigt inloggade användare.
SRS44102. Systemet skall ha ett gränssnitt för möjlighet att köra Firebird DBMS.
SRS44103. Systemet skall ha ett GTK# gränssnitt.
SRS44104. Server och DBMS skall inte behövas köra på samma dator.
SRS44105. Utvidgning
SRS44108. Utvidgning

SRS44106. En användare skall ha möjlighet att vara projektdeltagare.
SRS44107. En användare skall ha möjlighet att vara projektledare.
SRS44108. Projektdeltagare skall ha möjlighet att se vilka projekt han deltar i.
SRS44109. Projektdeltagare skall kunna definiera nya kategorier.
SRS44110. Utvidgning
SRS44111. Projektledare skall kunna starta projekt.
SRS44112. Projektledaren i ett projekt skall ange övriga deltagare i projektet.
SRS44113. Projektledaren skall se vilka projekt som administreras av honom.
SRS44114. Projektets projektledare skall ha samma rättigheter som övriga projektdeltagare i projektet.
SRS44115. Utvidgning
SRS44116. Utvidgning
SRS44117. Utvidgning
SRS44118. Systemet skall ha gästkonto för kunder.
SRS44119. Utvidgning

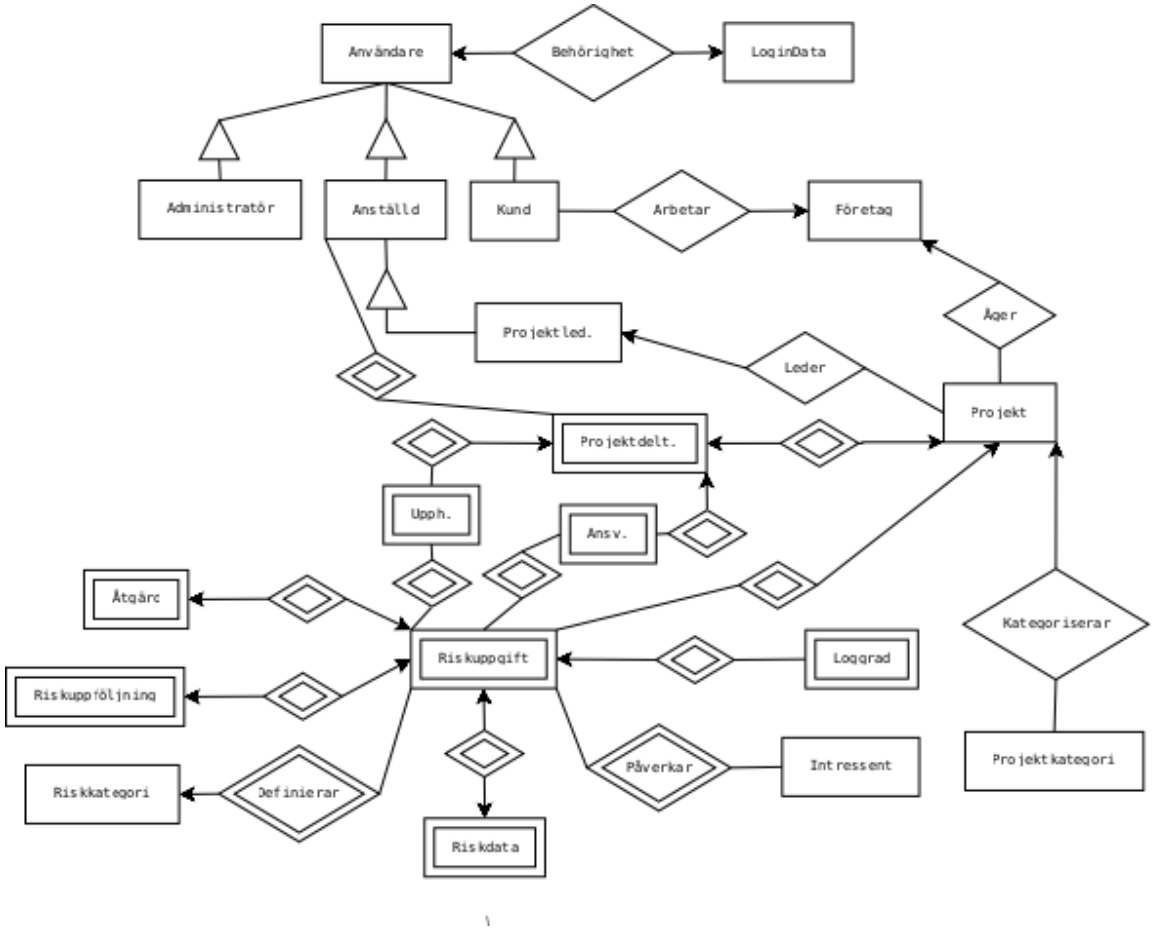
4.4.2 Riskkrav

SRS44201. Risker ska få ett automatiskt löpnummer.
SRS44202. Riskerna skall kategoriseras.
SRS44203. Utvidgning
SRS44204. Prioriteringen av riskerna skall vara möjliga att ändra.
SRS44205. Ändring av riskprioritering skall sparas för att ge möjlighet att se riskens historia.
SRS44206. Borttaget.
SRS44207. Borttaget. Indirekt samma som SRS44208.
SRS44208. Riskerna skall kunna sorteras kategorivis i ett projekt.
SRS44209. Riskerna skall kunna sorteras efter prioritering i projektet.
SRS44210. Projektdeltagare skall ha möjlighet att gradera riskuppgiftens sannolikhet på en 5-gradig skala.
SRS44211. Projektdeltagare skall ha möjlighet att gradera riskuppgiftens konsekvens på en 5-gradig skala.
SRS44212. En riskuppgift skall kunna vara ”projektkritiskt”.
SRS44213. Alla ändringar i en riskuppgift skall lagras i en logg så att användare kan se riskens historia.
SRS44214. Projektdeltagaren skall kunna ange vilken intressent som risken påverkar.
SRS44215. Projektdeltagare skall kunna ta bort en riskuppgift.

4.5 Datakrav

4.5.1 E/R-diagram

SRS45101. Systemet skall understödja nedanstående E/R-diagram.

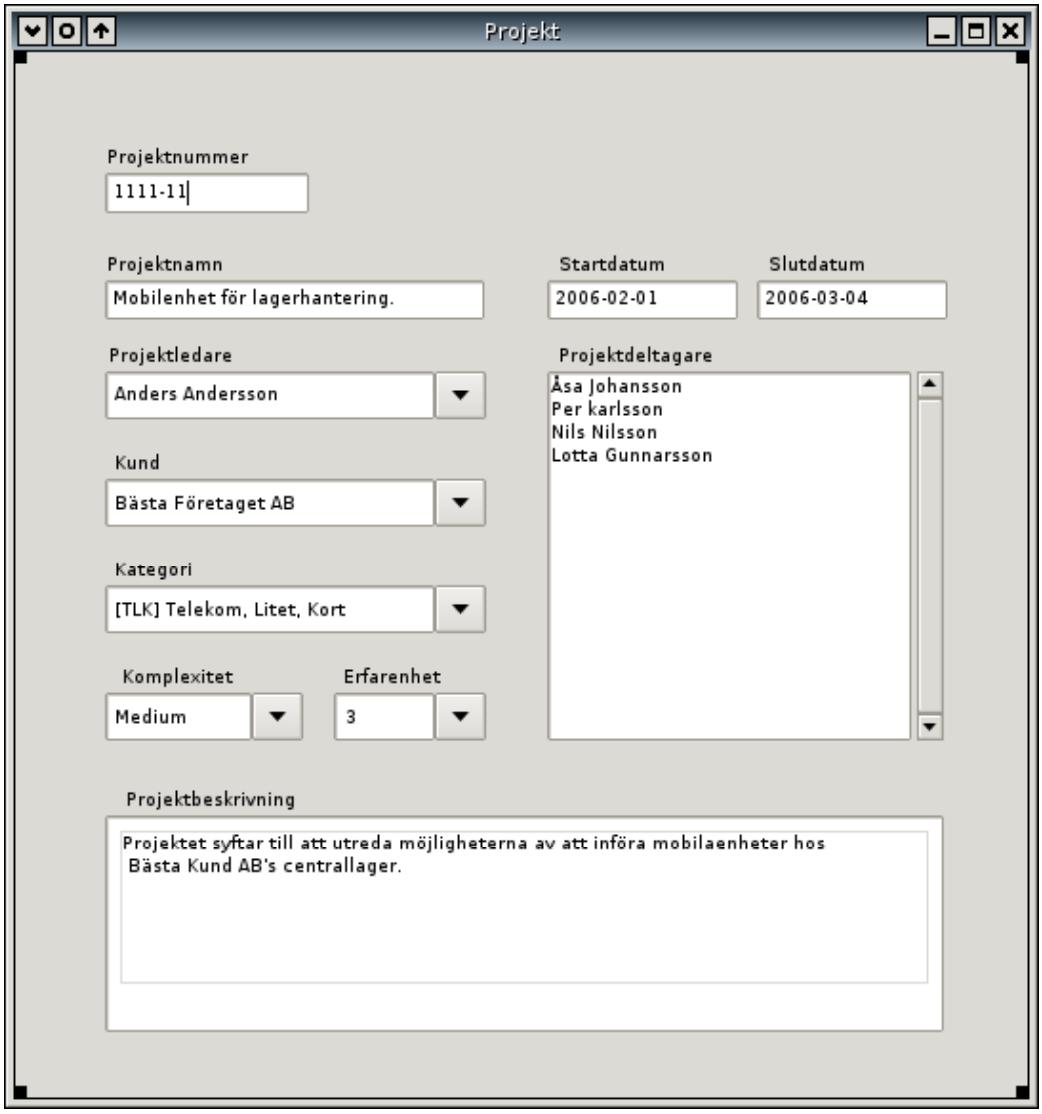


Figur 1: Översikt av relationer i databasmodellen

4.5.2 Virtual window

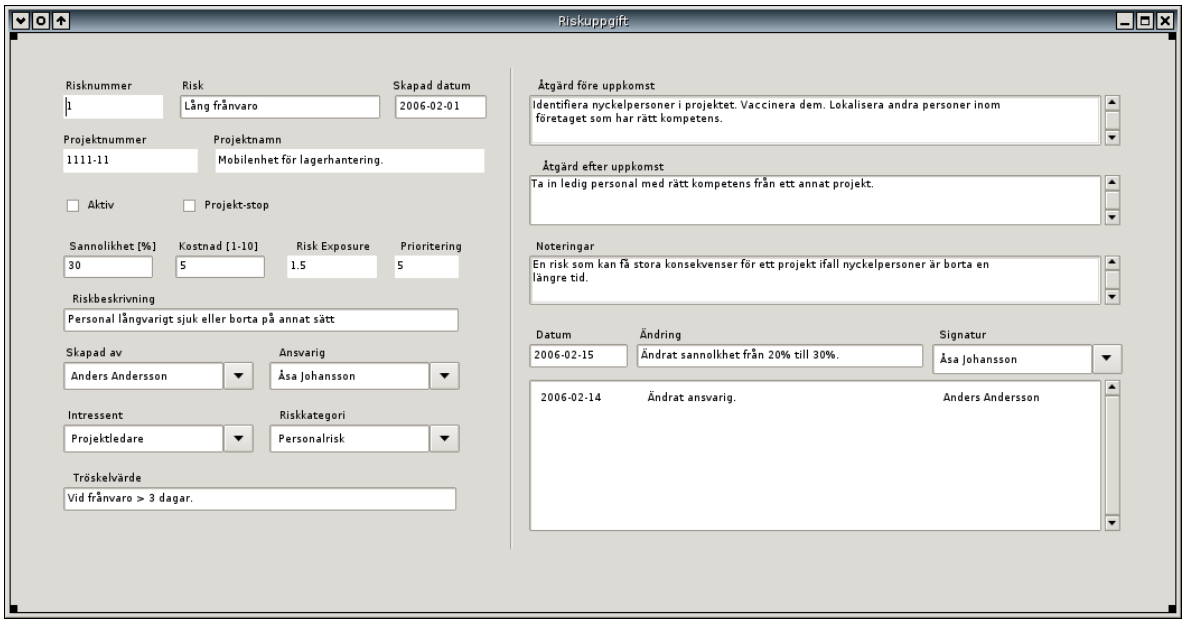
Systemet skall understödja nedanstående Virtual window.

SRS45201. Virtual window för Projekt.



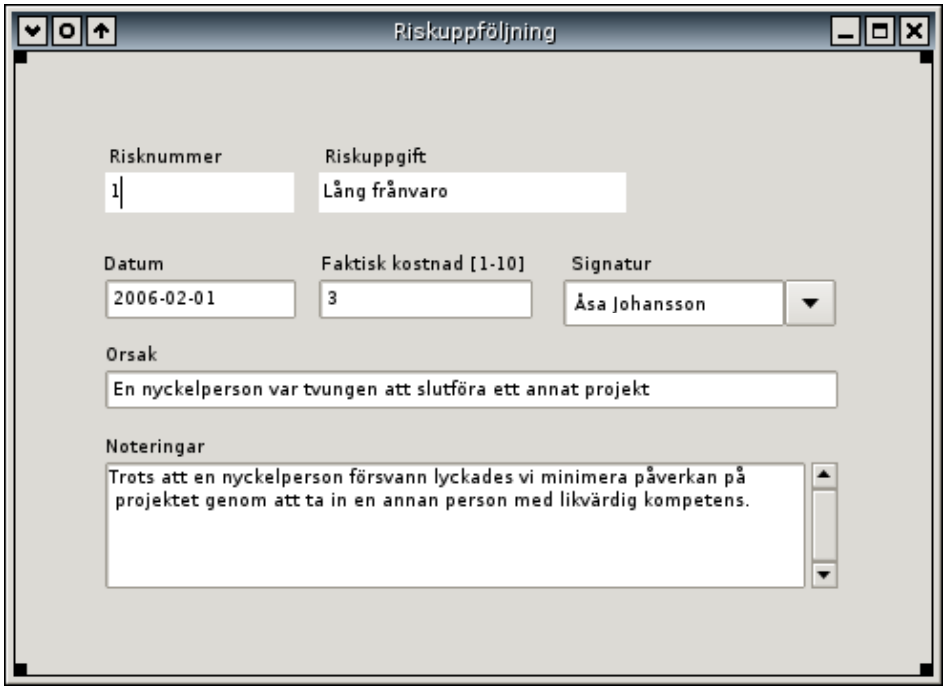
Figur 2, Virtual window för Projekt

SRS45202. Virtual window för Riskuppgift.



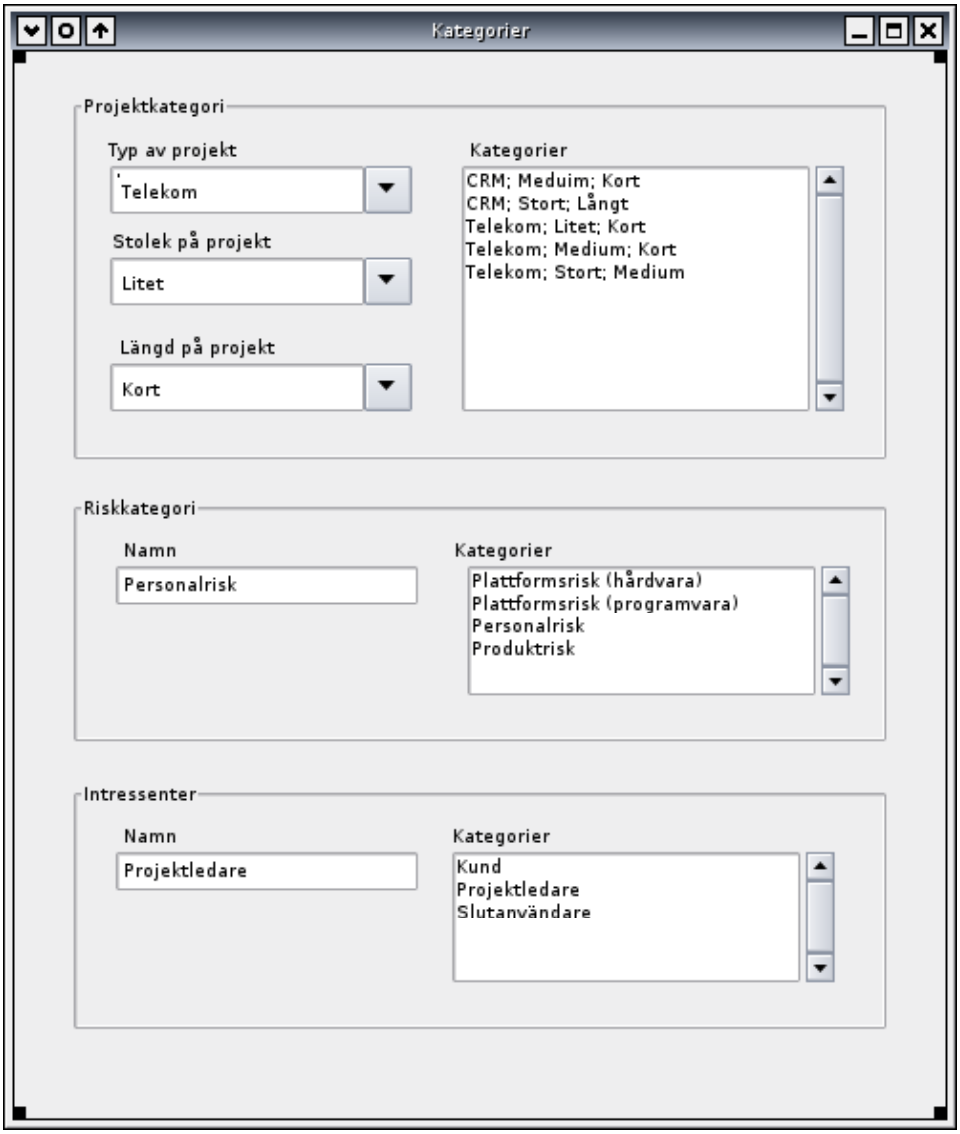
Figur 3, Virtual window för Riskuppgift

SRS45203. Virtual window för Riskuföljning.



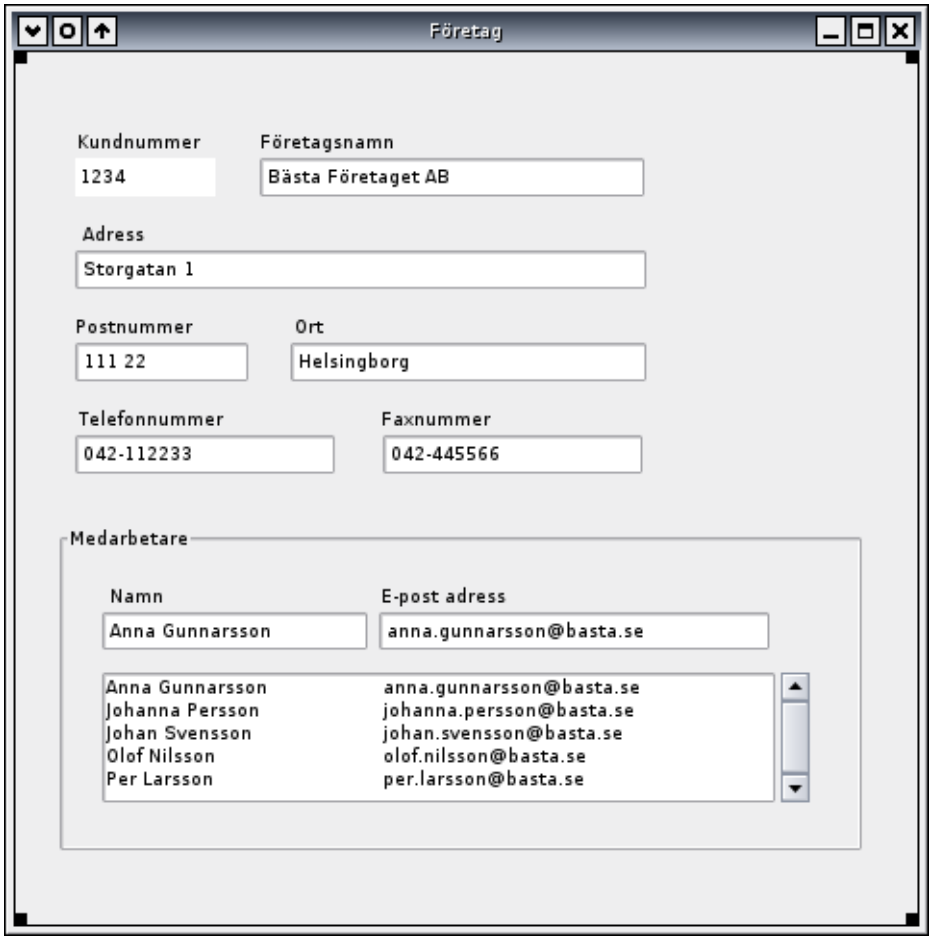
Figur 4, Virtual window för Riskuppföljning

SRS45204. Virtual window för Kategorier.



Figur 5, Virtual window för Kategori

SRS45205. Virtual window för Företag.



Figur 6, Virtual window för Företag

5 Terminologi

Benämning	Förklaring
Multianvändar-applikation	Flera användare kommer ha möjlighet att var inloggade och utföra uppgifter i systemet samtidigt.
DBMS	Database Management System, en samling med program för att hantera en databas.
Klient-GUI	Ett grafiskt gränssnitt som gör det möjligt att se, skapa och ändra information i databasen.
Show-stoppers	En riskuppgift som kan få hela projektet att kollapsa.
E/R-diagram	Ett diagram som beskriver datamodellen och dess relationer.
Virtual window	En prototyp av användargränssnitt fönster som visar den information som skall lagras i databasen.
Riskuppgift	En risk som ingår i ett projekt.
Gästkonto	Kunden får användarnamn och lösenord för möjlighet att granska projektets riskhantering. Kunden skall dock ej ha möjlighet att ändra eller lägga till information i databasen.

Appendix

Appendix A, Strukna krav

SRS44206. Projektet skall ha en 10 i topp-lista av risker.
SRS44207. Riskerna skall kunna visas kategorivis i ett projekt.

Appendix B, Krav för utvidgning

SRS41201. All dokumentation av testfall skall överensstämma med SVVD.
SRS41202. Resultaten av testfallen skall dokumenteras.
SRS42002. Systemet skall hantera analys av risker
SRS44105. När användaren loggar in skall ett användar-objekt skapas på servern.
SRS44108. All inloggning i systemet skall ske med användarnamn tillsammans med lösenord.
SRS44110. Användaren skall kunna ändra sitt lösenord.
SRS44115. Systemet skall administreras via ett administratörskonto.
SRS44116. Administratören skall kunna skapa användare.
SRS44117. Administratören skall kunna stänga av en användare.
SRS44119. Kunder skall bara ha möjlighet att läsa information om sina projekt.
SRS44203. Det skall finnas generella riskuppgifter.

Appendix D, Högnivådesign

RISKfree

- en applikation för riskhantering -

HÖGNIVÅDESIGN

Ansvarig Mikael Gullstrand
Datum 2006-07-11
Varsion 1.0

Versionhistorik

Version	Datum	Ansvarig	Ändring
0.1	2006-04-17	Mikael Gullstrand	Skapad
0.2	2006-05-09	Mikael Gullstrand	Uppdaterat E/R-diagram till ver. 0.3 samt tabelldesign
0.3	2006-05-18	Mikael Gullstrand	Uppdaterat E/R-diagram till ver. 0.4 samt tabelldesign
0.4	2006-07-11	Mikael Gullstrand	Uppdaterat entitetspecifikation samt lagt till klassdiagram
1.0	2006-07-11	Mikael Gullstrand	Satt i baseline efter granskning

Innehållsförteckning

Figurförteckning

1 Inledning

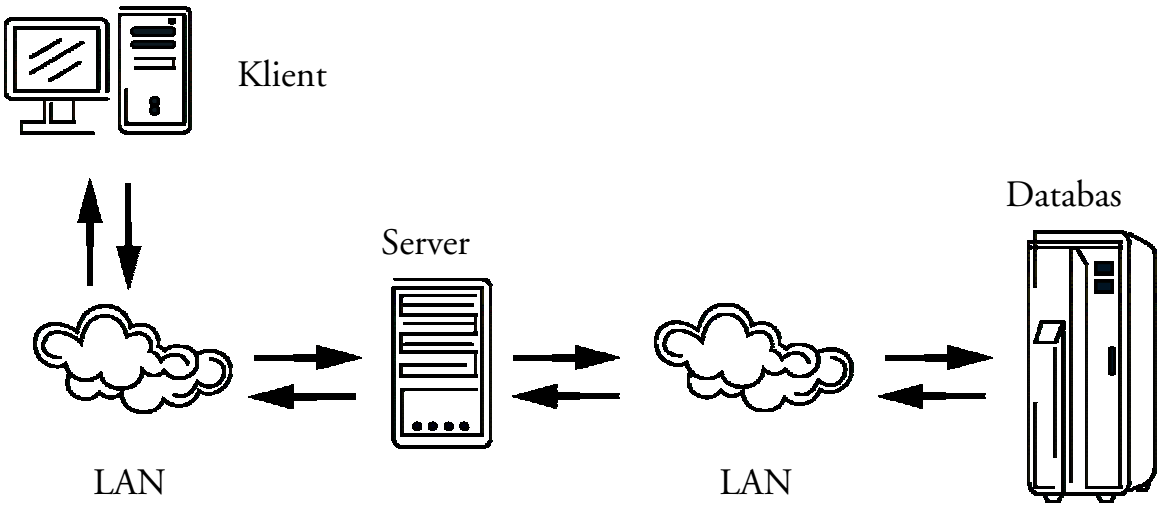
Detta dokument beskriver högnivådesign av riskhanteringsapplikationen RISKfree. Applikationen består av två delapplikationer, en server- samt en klientapplikation. Ytterligare information angående applikationens krav finns i kravspecifikationen [Ref. 1].

2 Referenser

[1] RISKfree Kravspecifikation 0.5

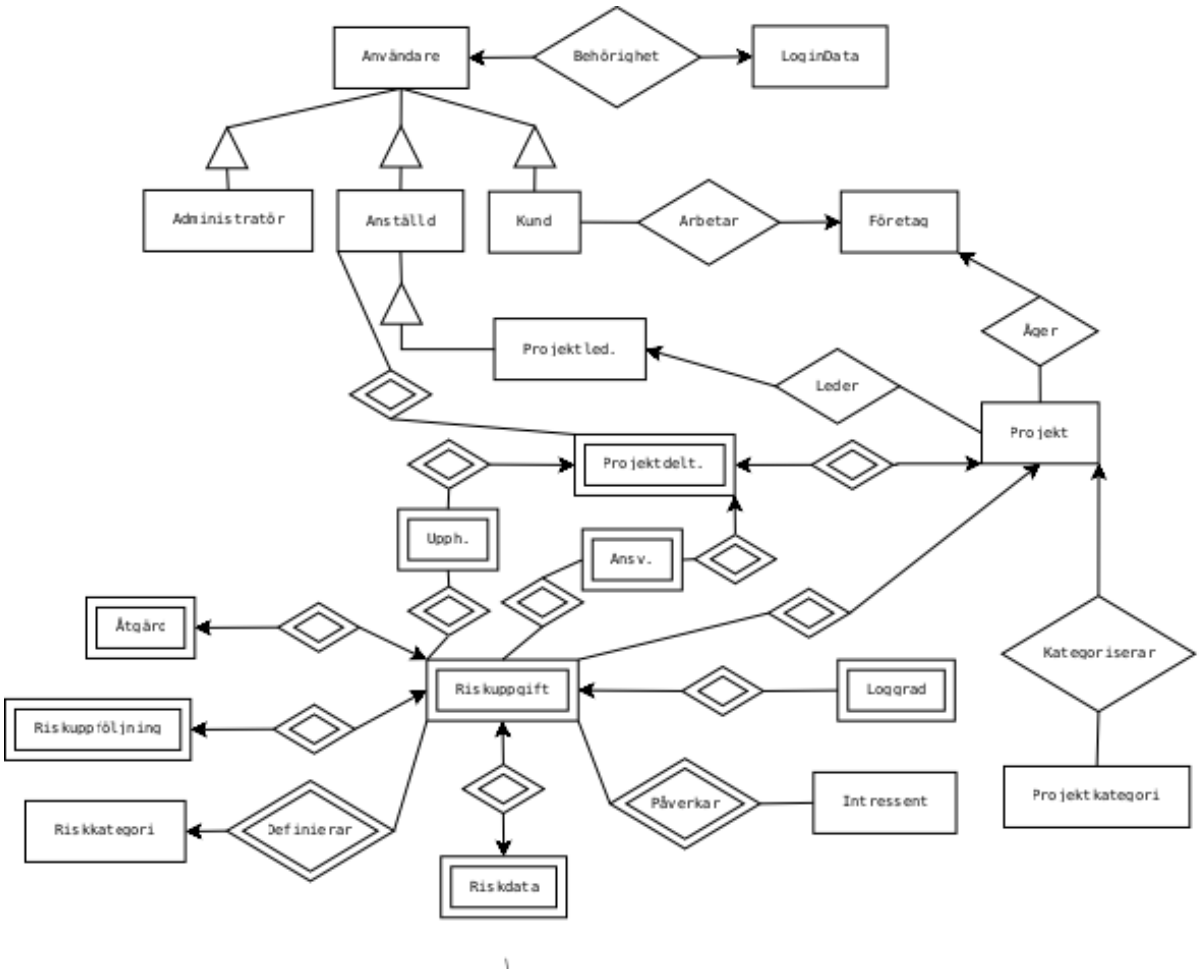
3 Systemarkitektur

RISKfree är en databas-driven applikation som består av en klientapplikation samt en distribuerad serverapplikation som sköter kommunikationen med DBMS. Detta innebär att



Figur 2: Översikt av systemarkitektur

4 Datamodell
4.1 E/R-diagram



Figur 2: Översikt av relationer i databasmodellen

4.2 Entitetspecifikation

4.2.1 Anstalld

Attribut	Typ och längd	Beskrivning
Id	SMALLINT	Anställningsnummer. Finns även i användar-entiteten.
Namn	VARCHAR(25)	Namn på den anställde
Normalisering		
🗝{Id} Id Namn		

4.2.2 Ansv

Attribut	Typ och längd	Beskrivning
Projektnr	SMALLINT	Nummer på projektet.
Risknr	SMALLINT	Nummer på riskuppgift.
Delt_Id	SMALLINT	Nummer på deltagere (anställningsnr eller kundnr.
Normalisering		
🗝 {Projektnr, Risknr, Delt_Id}		

4.2.3 Anvandare

Attribut	Typ och längd	Beskrivning
Id	SMALLINT	Id-nummer för alla användare av systemet.
Namn	VARCHAR(25)	Namn på användaren.
Normalisering		
🗝 {Id} Id Namn		

4.2.4 Atgard

Attribut	Typ och längd	Beskrivning
Projektnr	SMALLINT	Projektnummer som åtgärden gäller.
Risknr	SMALLINT	Risknummer som åtgärden gäller.
Tröskelvärde	VARCHAR(60)	Värde som indikerar när risken har inträffat.
Åtgärd före uppkomst	VARCHAR(200)	Åtgärd för att undvika risken.
Åtgärd efter uppkomst	VARCHAR(200)	Åtgärd då risken har inträffat.
Notering	VARCHAR(200)	Allmäna noteringar för åtgärden.

Normalisering

🗝{Projektnr, Risknr}
Projektnr, Risknr Troskel, AtgardF, AtgardE, Notering

4.2.5 Intressent

Attribut	Typ och längd	Beskrivning
Namn	VARCHAR(25)	Namn på intressent som risken påverkar.
Beskr	VARCHAR(60)	Beskrivande text av intressent.

Normalisering

🗝 {Namn}
Namn Beskr

4.2.6 Kund_Anst

Attribut	Typ och längd	Beskrivning
Id	SMALLINT	Id på kundanställd. Finns även i användar entiteten.
Namn	VARCHAR(25)	Namn på kundanställd.

Normalisering

🗝 {Id}
Id Namn

4.2.7 Kund_Foretag

Attribut	Typ och längd	Beskrivning
Kundnr	SMALLINT	Kundnummer.
Namn	VARCHAR(25)	Namnet på företaget.
Adress	VARCHAR(40)	Adress till företag.
Postadress	VARCHAR(7)	Postadress till företag.
Ort	VARCHAR(25)	Ort där företaget är lokaliserat.
Telenr	VARCHAR(15)	Telefonnummer till företaget.
Faxnr	VARCHAR(15)	Faxnummer till företaget.

Normalisering

🗝 {Kundnr}
Kundnr Namn, Adress; PostAdress, Ort, Telenr, Faxnr

4.2.8 Loggrad

Attribut	Typ och längd	Beskrivning
Projektnr	SMALLINT	Projektnummer som loggraden tillhör.
Risknr	SMALLINT	Risknummer som loggraden tillhör.
Datum	TIMESTAMP	Datum då loggraden skapades.
Andring	VARCHAR(45)	Vilken ändring som utfördes.
Signatur	VARCHAR(25)	Signatur av den som genomförde ändringen.

Normalisering

🗝 {Projektnr, Risknr, Datum}
Projektnr, Risknr, Datum Andring, Sign

4.2.9 Logindata

Attribut	Typ och längd	Beskrivning
Anvandarnamn	VARCHAR(15)	Användarnamn för anvöndaren.
Losenord	VARCHAR(8)	Lösenord för användaren.

Normalisering

🗝 {Anvandarnamn, Losenord}

4.2.10 Projekt

Attribut	Typ och längd	Beskrivning
Projektnr	SMALLINT	Löpnummer på projekt.
Namn	VARCHAR(20)	Namn på projekt.
Beskrivning	VARCHAR(60)	Beskrivande text av projektet.
Startdatum	DATE	Datum för när projektet startade.
Slutdatum	DATE	Datum för när projektet beräknas avslutas.
Komplexitet	SMALLINT	Värde på hur komplext projektet.
Erfarenhet	SMALLINT	Värde på hur stor erfarenhet organisationen har av liknande projekt.

Normalisering

🗝 {Projektnr}
Projektnr Namn, Beskrivning, Startdatum, Slutdatum, Komplexitet, Erfarenhet

4.2.11 Projektdelt

Attribut	Typ och längd	Beskrivning
Id	SMALLINT	Id på användare som deltar i projektet. Finns även i användar-entiteten.
Projektnr	SMALLINT	Projektnummer som deltagaren deltar i.

Normalisering

🗝 {Id, Projektnr}

4.2.12 Projektledare

Attribut	Typ och längd	Beskrivning
Id	SMALLINT	Id på användare som är projektledare. Finns även i användar-entiteten.
Namn	VARCHAR(25)	Namn på projektledare.

Normalisering

🗝 {Id}
Id Namn

4.2.13 Projektkategori

Attributn	Typ och längd	Beskrivning
Id	SMALLINT	Löpnummer på projektkategori.
Namn	VARCHAR(20)	Namn på projektkategori.
Beskrivning	VARCHAR(60)	Beskrivande text av projektkategori.
Typ	VARCHAR(15)	Typ av projekt.
Storlek	SMALLINT	Storlek på projekt.
Langd	SMALLINT	Tidsåtgång till projekt.

Normalisering

🗝 {Id}
Id Namn, Beskrivning, Typ,Storlek, Langd

4.2.14 Riskdata

Attribut	Typ och längd	Beskrivning
Projektnr	SMALLINT	Projektnummer som riskdata tillhör.
Risknr	SMALLINT	Risknummer som riskdata tillhör.
Sannolikhet	SMALLINT	Sannolikhet att risken uppkommer.
Kostnad	SMALLINT	Konsekvens av att risken uppkommer.
Stopp	VARCHAR(5)	Kan risken stoppa hela projektet.
Prioritet	SMALLINT	Prioritet av risken.

Normalisering

🗝 {Projektnr, Risknr}
Projektnr, Risknr Sannolikhet, Kostnad, Stopp, Prioritet

4.2.15 Riskkategori

Attribut	Typ och längd	Beskrivning
Id	SMALLINT	Löpnummer av riskkategori.
Namn	VARCHAR(25)	Namn på riskkategori.
Beskrivning	VARCHAR(60)	Beskrivande text av riskkategori.

Normalisering

🗝 {Id}
Id Namn, Beskrivning

4.2.16 Riskuppg

Attribut	Typ och längd	Beskrivning
Projektnr	SMALLINT	Projektnummer som risken tillhör.
Risknr	SMALLINT	Löpnummer på risk.
Namn	VARCHAR(15)	Namn på risk.
Beskrivning	VARCHAR(60)	Beskrivande text av risken.
Datum	DATE	Datum risken skapades för projektet.
Aktiv	SMALLINT	Är risken aktiv för projektet.

Normalisering

🗝 {Projektnr, Risknr}
Projektnr, Risknr Namn, Beskrivning, Datum, Aktiv

4.2.17 Riskuppf

Attribut	Typ och längd	Beskrivning
Projektnr	SMALLINT	Projektnummer som riskuppföljningen tillhör.
Risknr	SMALLINT	Risknummer som riskuppföljningen tillhör.
Datum	DATE	Datum när riskuppföljningen genomfördes.
Kostn	SMALLINT	Den faktiska konsekvensen för riskuppgiften.
Forsening	VARCHAR(60)	Hur myck försening orsakade risken.
Orsak	VARCHAR(60)	Vad orsakade att risken uppkom.
Notering	VARCHAR(200)	Allmäna noteringar om riskuppföljning.
Signatur	VARCHAR(25)	Signatur av den som genomförde riskuppföljnigen.

Normalisering

🗝 {Projektnr, Risknr}
Projektnr, Risknr Datum, Kostn, Forsening, Orsak, Notering, Signatur

4.2.18 Upph

Attribut	Typ och längd	Beskrivning
Risknr	SMALLINT	Risknummer på den upphittade risken.
Projektnr	SMALLINT	Projektnummer som risken tillhör.
Delt_Id	SMALLINT	Id-nummer på den som hittade risken.
Normalisering		
🔑 {Risknr, Projektnr, Delt_Id}		

4.3 Relationer

4.3.1 Ager

(Kund_Foretag_Kundnr, Projekt_Projektnr)

4.3.2 Arbetar

(Kund_Anst_Id, Kund_Företag_Kundnr)

4.3.8 Behorighet

(Anvandare_Id, LoginData_Anvandarnamn, LoginData_Losenord)

4.3.3 Definierar

(Riskkategori_Id, Riskuppg_Projektnr; Riskuppg_Risknr)

4.3.4 Kategorisering

(Projekt_Projektnr, Projektkategori_Id)

4.3.5 Leder

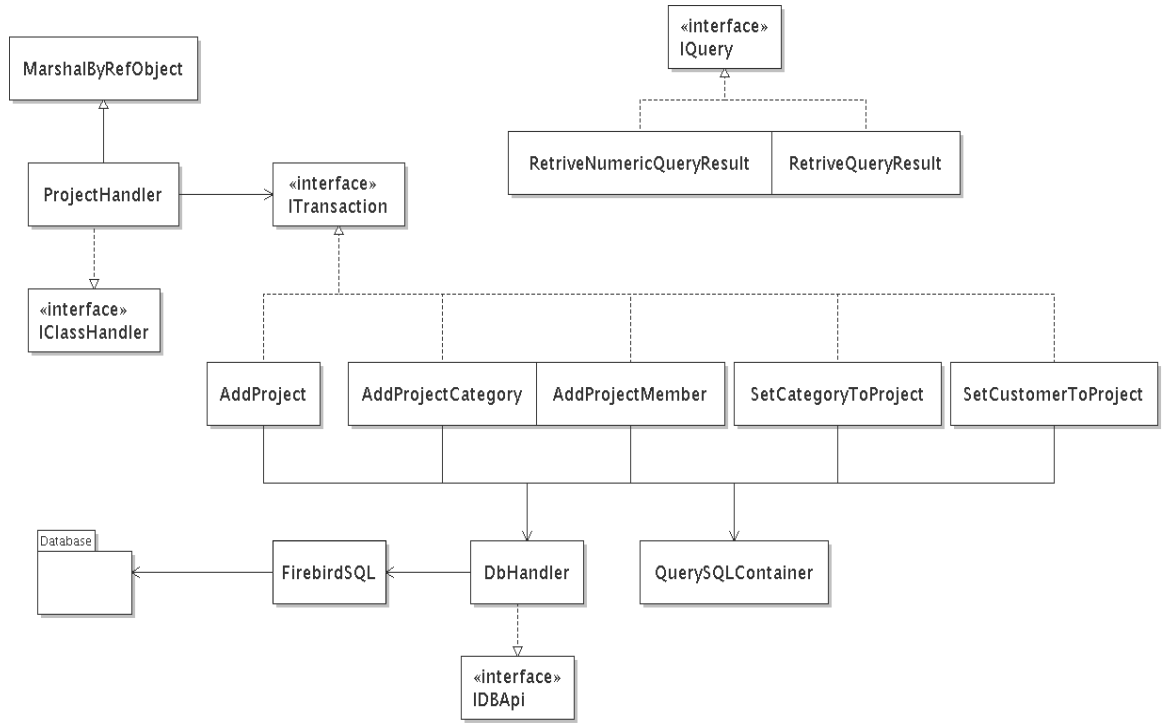
(Projekt_Projektnr, Projektledare_id)

4.3.6 Paverkar

(Intressent_Namn, Riskuppg_Projektnr; Riskuppg_Risknr)

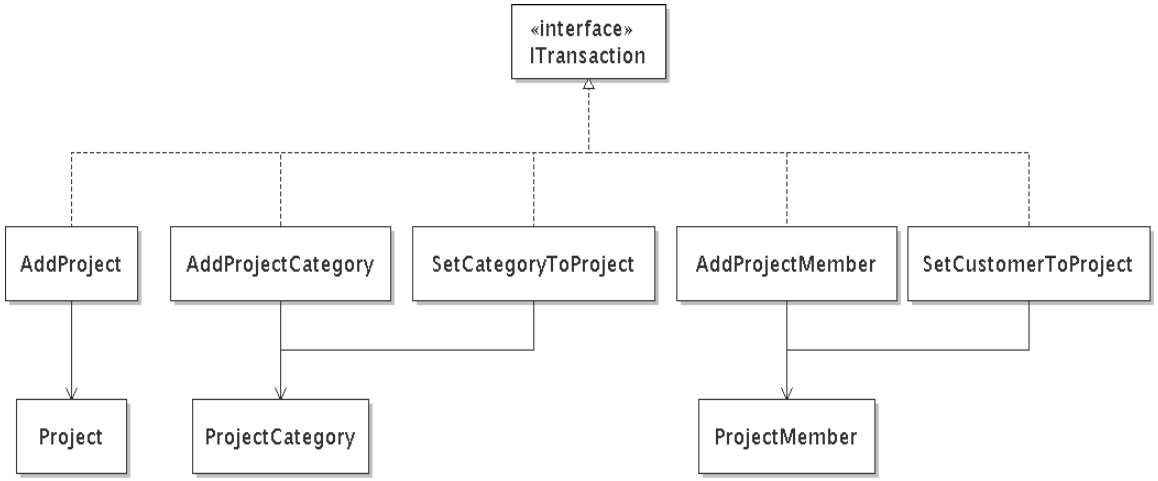
5 Klassdiagram

5.1 Klassdiagram för server-projektklasser

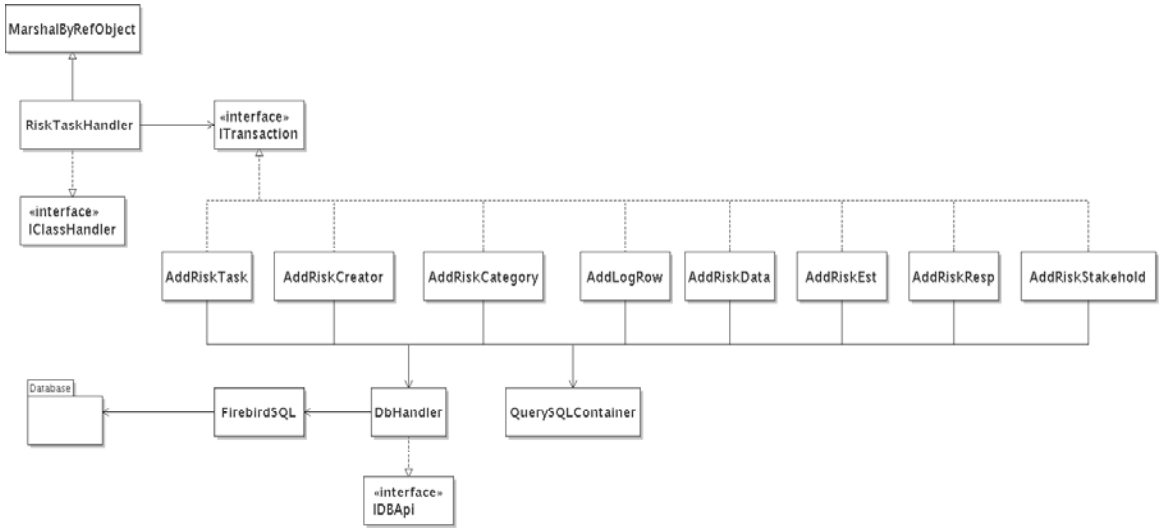


Figur 3, UML-diagram för projektklasser som skriver/läser till databasen

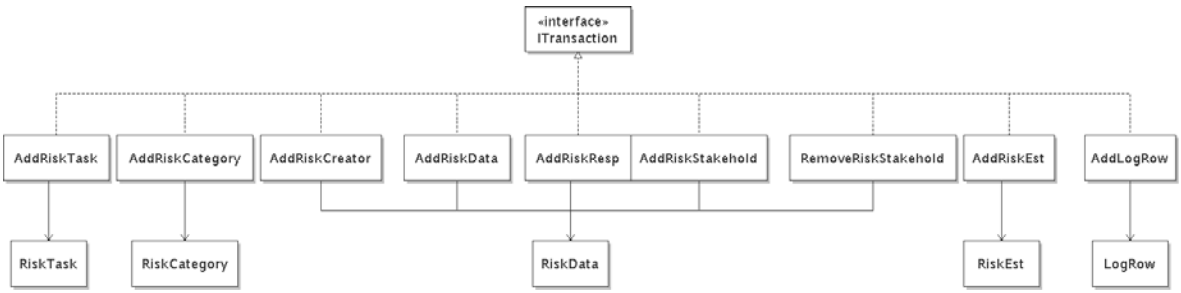
Figur 4, UML-diagram för projektklasser som tillhandahåller lagring



5.2 Klassdiagram för server-riskklasser

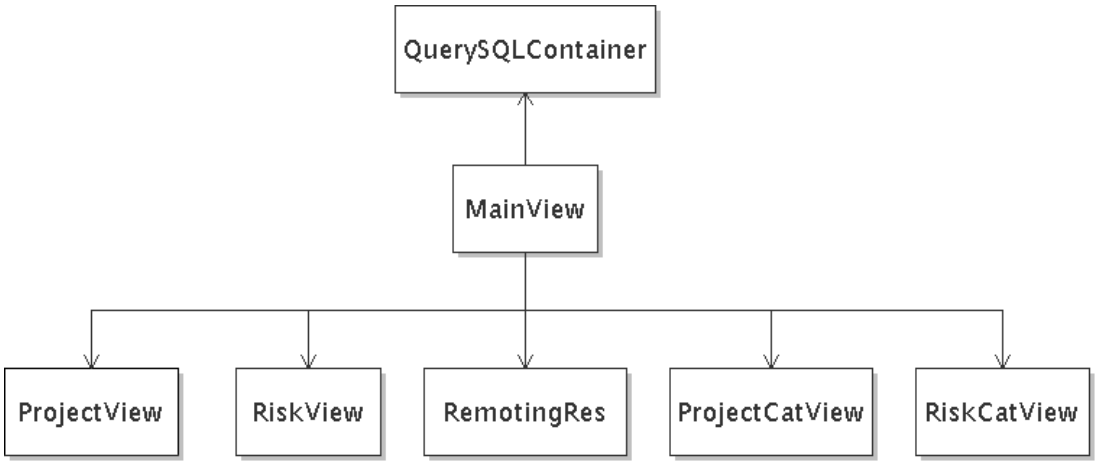


Figur 5, UML-diagram för riskklasser som skriver/läser till databasen

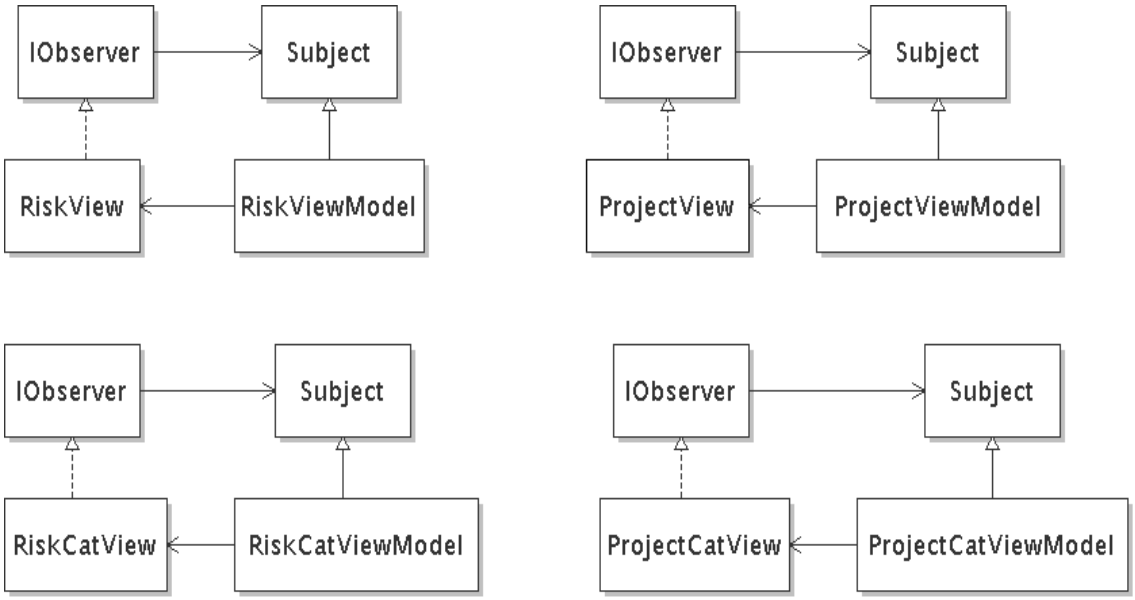


Figur 6, UML-diagram för riskklasser som tillhandahåller lagring

5.3 Klassdiagram för klient-GUI



Figur 7, UML-diagram för vy-klasser i klient GUI



Figur 8, UML-diagram för modell-vyklasser enligt Observer-mönster

6 Terminologi

Benämning	Förklaring
Multianvändar-applikation	Flera användare kommer ha möjlighet att var inloggade och utföra uppgifter i systemet samtidigt.
DBMS	Database Management System, en samling med program för att hantera en databas.
Klient-GUI	Ett grafiskt gränssnitt som gör det möjligt att se, skapa och ändra information i databasen.