MASTER'S THESIS | LUND UNIVERSITY 2015

Vad karaktäriserar komplexa ärenden i mjukvaruprojekt?

Soheil Afghani Khorasgani, Mehmet Fatih Cicek

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2015-36

Vad karaktäriserar komplexa ärenden i mjukvaruprojekt?

En fallstudie på underhåll av ett affärssystem

Soheil Afghani Khorasgani
ada10saf@student.lu.se

Mehmet Fatih Cicek
ada10mci@student.lu.se

29 augusti 2015

Examensarbete utfört på CGI.

Handledare: Markus Borg, Markus.Borg@cs.lth.se
Anders Malmberg, anders.malmberg@cgi.com

Examinator: Per Runeson, Per.Runeson@cs.lth.se

Abstrakt

I de avancerade mjukvarusystem som utvecklas idag krävs det en hel del processer och arbetsmetodik för att säkerställa att produkten håller en bra kvalité. Svårigheten uppstår i att organisationer oftast är tvungna att hantera väldigt många ändringar som kommer från kunder och dessa kan vara av varierande komplexitet, dvs. kräva olika mycket resurser att åtgärda. CGI är ett globalt IT-företag och jobbar mot många kunder. Supportorganisationen i Malmö klassificerar ett ärende som komplext när X antal timmar har tidrapporterats. Genom att implementera ett webbaserat tidrapporteringssystem har vi identifierat de mest tidskrävande ärendena, och utfört djupstudier på dessa. Tillsammans med intervjuer av supportpersonal på företaget har vi identifierat potentiella faktorer som kan förklara varför komplexa ärenden har blivit större och mer tidskrävande än övriga incidenter. Resultatet av intervjuerna överlappade någorlunda med djupstudien vid karaktärisering av komplexa ärenden, 11 av 24 faktorer visade sig vara gemensamma. Att siffran inte blev högre kan bero på att komplexa ärenden inte är så vanliga inom företaget, vilket baseras på resultatet av mätningar från tidrapporteringssystemet och intervjuer med supportpersonal. Några av faktorerna vi har identifierat är:

- Felet går inte att återskapa.
- Arbetet påverkade gammal funktionalitet negativt.
- Allt är anpassat efter en felaktig lösning, som gör att korrekt logik leder till fel på oväntade ställen i koden.
- En ändring i en modul leder till att andra moduler måste modifieras.
- Ändring av kod som redan gått i produktion och fått oförutsedda konsekvenser.

Vi har skapat ett beslutsstöd, i form av de komplexitetsfaktorer som tagits fram, som kan användas av företag som har förvaltningsarbeten för att tidigt i arbetet identifiera ärenden som kan vara tidskrävande, och på så vis omfördela tid och resurser på ett gynnsamt sätt. Om ett ärende påvisar flera av de faktorer som presenterats kan det indikera på att ärendet kommer att bli tidskrävande. Med tidrapporteringssystemet har vi möjliggjort för företaget att ha större insikt i hur mycket tid (och därmed resurser) som läggs ner på varje ärende. I och med att man får en bättre översikt över arbetet kan tidrapporteringssystemet även bidra till att förbättra företagets ärendehanteringskostnader. Vårt resultat kan användas som en bas för vidare forskning, och kan kompletteras med empiriska studier som undersöker huruvida dessa faktorer även kan identifieras i andra supportorganisationer.

Nyckelord: mjukvaruutveckling, ändringshantering, komplexitet, incident

Abstract

In the advanced software systems that are developed today there is need for a lot of processes and proper methods to ensure that the developed product maintains a good quality. The difficulty arises in that the organizations often have to deal with many changes arriving from customers and these can be of varying complexity, i.e. require different amount of resources to address. CGI is a global company and has a large clientele. The support organization in Malmö classifies an incident as being complex when X number of hours have been reported. By implementing a web-based time reporting system, we have identified the most time-consuming incidents, and conducted in-depth studies on these. Furthermore we have interviewed the support staff of the company and used the result of the interviews and our in-depth studies to identify potential factors that could explain why complex incidents become larger and more time-consuming than other incidents. The result of the interviews overlapped somewhat with the depth study in the characterization of complex cases, 11 of the 24 factors were found to be common. The low overlap may be due to the rarity of complex cases within the company, which was derived from the results of measurements of the time reporting system and interviews with the support staff. The following are a few examples we have identified:

- The error can not be reproduced.
- The work has had adverse affects on pre-existing functionality.
- A modification in one module is dependent on changes in other modules.
- Changing code that has already gone into production, and has demonstrated unforeseen consequences.
- Everything is adapted to a faulty solution, which results in the correct logic leading to errors in unexpected places in the code.

We have produced a set of complexity factors which can be used by companies that handle product maintenance, in order to identify cases that can be time consuming at the very early stages of the maintenance work, and thus reallocate time and resources more efficiently. If a maintenance case demonstrates several of the factors presented, it may indicate that the matter will be time consuming. The time reporting system has made it possible for the company to have greater insight into how much time (and thereby resources) that are spent on each case. This in turn helps in improving the company's maintenance costs. Our results can be used as a basis for further research, and can be supplemented by empirical studies investigating whether these factors can also be identified in other support organizations.

Keywords: software development, change management, complexity, incident

Förord

Vi vill tacka våra handledare på CGI och LTH för deras hjälp och värdefulla kommentarer. Vi vill även tacka vår examinator och Krzysztof Wnuk, som hjälpte oss vinkla examensförslaget som CGI hade till ett giltigt examensarbete.

Innehållsförteckning

Bidrag

Allt arbete har delats upp jämnt mellan författarna. Intervjuerna hölls av båda samtidigt, och incidenterna för djupstudien delades upp för invidiuell analys för att effektivisera arbetet. Vi rådfrågade om varandras uppfattning längs vägen för att kunna dra gemensamma slutsatser, och inte låta resultatet påverkas av individuella tolkningar. Vad gäller implementeringsfasen av det webbaserade tidrapporteringssystemet arbetade vi mot ett gemensamt repositorie, men delade inte upp arbetet i olika ansvarsområden. Vi hade en ständig diskussion mellan oss och handledaren om vad som behövde göras härnäst, och delade upp arbetet efter det.

Förklaring av begrepp

Baseline - Ett dokument som sätts i baseline får inte ändras (inklusive dess versionsnummer), så att man kan referera till dess innehåll i framtiden. Detta kan ses som ett kontrakt och försäkrar alla parter om vad som gäller.

Konfigurationshantering - En disciplin i mjukvaruutveckling vars roll är bland annat hantering av konfigurationer och versioner av artefakter.

Formell ändringsbegäran - En ändringsbegäran som vanligtvis kommer in från en kund eller internt inom organisationen, behandlar vanligtvis en ny funktionalitet eller buggfix.

Incident - En ändringsbegäran som består av ett fel som måste åtgärdas (specifikt till CGI).

Kvalitetssäkring - Systematiska processer som kontrollerar att produkten som utvecklas stämmer överens med de krav som ställts upp.

NP-svårt - NP betecknar mängden av beslutsproblem som kan lösas i polynomiell tid av en icke-deterministisk Turingmaskin. Ett problem sägs vara NP-svårt om en algoritm för att lösa det kan skrivas om till en som kan lösa alla andra NP-problem.

Påverkansanalys - Aktivitet som undersöker vilka artefakter som blir påverkade av en ändringsbegäran och till vilken grad.

Riskidentifiering - En aktivitet inom risk management som går ut på att identifiera risker som finns i ett projekt.

Riskhantering - En process som går ut på att skydda projektet så mycket som möjligt mot eventuella problem som kan uppstå under utvecklingen.

Requirements Engineering - Kravhantering

Ändringsbegäran - Se ”Formell ändringsbegäran”.

Ärende - Se ”Incident”.

Ärendehanteringssystem - Ärendehanteringssystem är system som används för att administrera ärenden i ett företag. Det finns flera exempel av sådana system såsom Bugzilla, Remedy och Trac.

Kapitel 1
Inledning

1.1 Introduktion och problembeskrivning

IT-företag som befinner sig i en konkurrensutsatt marknad måste vara bra på att hantera de problem som uppstår under utvecklingen för att säkerställa att kunderna blir nöjda, och därmed behålla en ledande position i branschen. Mjukvarusystem kan vara allt från enklare drivrutiner till större system och företagen strävar efter att leverera mjukvara med bra kvalité. Det är väldigt vanligt att det kommer in ändringar under utvecklingen, både från kund och inom utvecklingsteamet, och kan i många fall vara tidskritiska. Detta ställer ett stort krav på arbetsprocessens effektivitet och disciplin. Därför är det viktigt att undersöka de faktorer som kan påverka produktionen. Ju större ett projekt är desto längre tid kan det ta att analysera ett ändringsförslag (*påverkansanalys*). Ett av målen med examensarbetet är att undersöka vad som kännetecknar komplexitet i ett ärende och hur detta påverkar olika faktorer som till exempel beslutsprocessen och ledtiden. Ledtid definieras som tiden mellan att ett ändringsförslag rapporterats och tidpunkten för när ett beslut tas.

CGI är ett globalt IT-företag som finns utspritt i 40 olika länder. I CGIs supportorganisation i Malmö används *ärendehanteringssystem* Remedy för att registrera SAP-ärenden som kommer från en av de större kunderna. SAP (Systems Applications Products) är ett affärssystem och används av kunden för att erbjuda olika tjänster till dess användare. I Remedy saknas dock möjligheten att rapportera nedlagt tid per ärende och därmed har de svårt att se när ett ärende når gränsen för att bli ett komplext ärende, eftersom de definierar komplexa ärenden efter nedlagd tid. Den exakta definitionen för komplexitet, enligt företagets definition, är konfidentiellt och därför benämns gränsen som X antal timmar i rapporten.

Syftet med examensarbetet var ursprungligen att skapa ett system för att rapportera tid per ärende för CGIs supportpersonal. Eftersom man vill använda systemet för att avgöra när ärenden blir komplexa utökades examensarbetet till att även undersöka vad som definierar

ett komplext ärende, förutom tiden man lägger ner på det. Utvecklingen av systemet blir således en förutsättning för examensarbetet för att kunna samla relevant data om *formella ändringsbegäran* och utföra analyser. I dagsläget finns det få studier som undersöker komplexiteten i formella ändringsbegäran. Målet med denna studie är att bidra till större insikt inom ärendehantering genom att utforska följande forskningsfrågor:

RQ1[1]: Hur vanlig är förekomsten av komplexa ärenden, det vill säga mer än X timmar hanteringstid?

RQ2: Vad karaktäriserar komplexa ärenden, det vill säga finns det något som utmärker dem?

RQ3: Finns det något samband mellan komplexitet, ledtid och beslutsprocess i de ärenden som undersöks?

RQ4: Hur kan man bättre hantera ärendehanteringskostnader och prediktera vilka ärenden som blir komplexa?

Alla forskningsfrågor avser supportärenden för SAP-system som behandlas på företaget.

1.2 Rapportens struktur

Rapporten har följande upplägg:

Förklaring av begrepp innehåller en ordlista med vanliga begrepp inom kravhantering och närliggande fält som vi har använt i rapporten.

Inledning ger en introduktion till ämnet och problembeskrivningen.

Fallbeskrivning beskriver företaget och den process för ärendehantering som används.

Bakgrund och relaterat arbete tar upp relevant fakta om krav-och ärendehantering och tillhörande processer samt relaterat arbete.

Metod beskriver vår metodik och angreppssätt.

Resultat och diskussion tar upp en diskussion om resultatet vi har kommit fram till och hur detta förhåller sig till frågeställningen.

Sammanfattning och slutsats sammanfattar rapporten i korthet och vilka slutsatser vi har kommit fram till.

Källförteckning innehåller alla referenser som har använts i rapporten.

Appendix A är en intervjuguide som användes under intervjuerna på CGI.

Appendix B beskriver överskådligt tidrapporteringssystemet vi har utvecklat.

[1] RQ står för Research Question, vilket är den engelska termen för ”forskningsfråga”.

Kapitel 2
Fallbeskrivning

Kapitlet ger en kort beskrivning av företaget. Sedan följer en beskrivning av ärendehanteringssystemet och processen för förvaltningsarbetet. Med förvaltningsarbete syftar vi på arbetet som utförs av CGIs supportorganisation när ett fel eller en brist upptäcks i kundens system, på begäran av kunden. Slutligen ges en beskrivning av SAP, som är det system som används av kunden. Alla ärenden som hanteras av CGIs supportorganisation berör detta system.

2.1 CGI i korthet

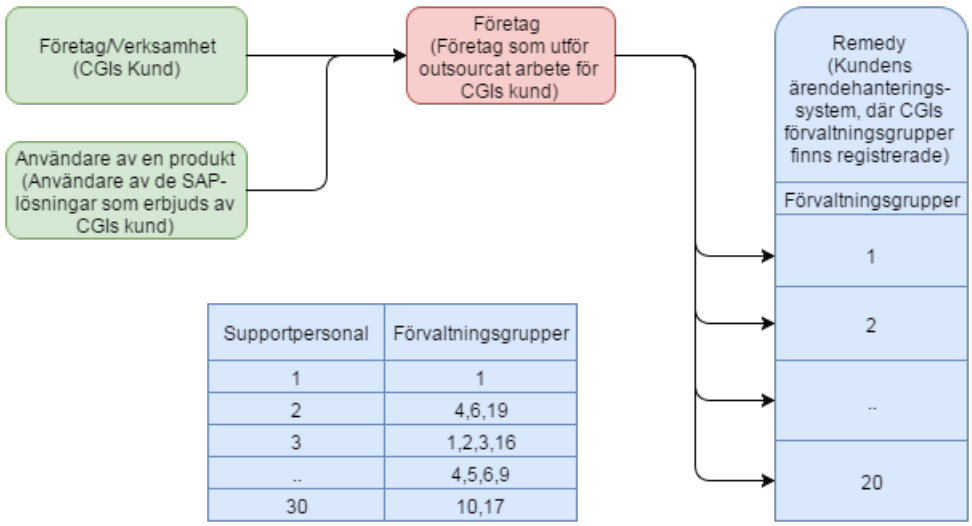
CGI är Sveriges största och världens femte största företag inom IT-och affärsprocesstjänster med dryga 70000 anställda i 40 olika länder, med huvudkontor i Kanada [2, 19]. I Sverige har CGI drygt 4000 anställda spritt på ett 30-tal olika orter, där Malmö-kontoret är ett av de största med ungefär 500 anställda [2]. Företaget har expertis inom flera olika branscher och erbjuder tjänster inom bland annat affär och IT-konsulting, systemintegration, applikationsutveckling och förvaltning, infrastruktur, affärsprocesstjänster och ip-baserade lösningar [19]. År 2013 hade CGI en omsättning på cirka 10.1 miljarder kanadensiska dollar, och en uppskattad orderstock på 18.7 miljarder kanadensiska dollar. WM-data var ett Sverigebaserat företag som 2006 köptes upp av Logica. Logica blev i sin tur uppköpt av CGI för ett par år sedan och på så sätt etablerade företaget sig på den svenska marknaden [2].

2.2 Remedy och ärendehantering

Denna studie utgår ifrån data från ett ärendehanteringssystem på CGI. Ärendehanteringssystemet heter Remedy och används av ett av företagets kunder [15]. CGI använder Re-

medy för att hantera support av ett antal applikationer för kunden. Applikationerna är konstruerade som instickspogram till SAP (se kapitel 2.3) som är ett världsledande affärssystem [79, 48]. Ärenden består av buggfixar och i särskilda fall mindre vidareutveckling. I den studerade utvecklingsorganisationen rapporteras det in cirka 250 ärenden per månad. Ärenden i systemet delas in i olika förvaltningsgrupper beroende på ärendetyp. Supportpersonalen är indelad enligt samma förvaltningsgrupper för att möta upp dessa med rätt kunskaper.

Supportpersonalen består av ett 30-tal personer som ansvarar för supportarbetet. Personalen har varierade kunskaper och har därmed olika roller. En del är programmerare medan andra har kunskap om hur de olika instickspogrammen är uppbyggda och hur dessa utnyttjas i SAP, och hjälper till med problem som kräver större kännedom om systemet. De personer som har insikt i mer än en modul medverkar i flera förvaltningsgrupper. Ärenden som rapporteras in härstammar från kunden och användare av de produkter som förvaltas. Ett separat företag ansvarar för att distribuera alla ärenden till Remedy och klassificerar dem enligt rätt typ och grupp, se Figur 2.1. Det händer ofta att en del ärenden hamnar i fel förvaltningsgrupper [21]. Supportpersonalen har därför möjlighet att flytta ärenden mellan grupper.



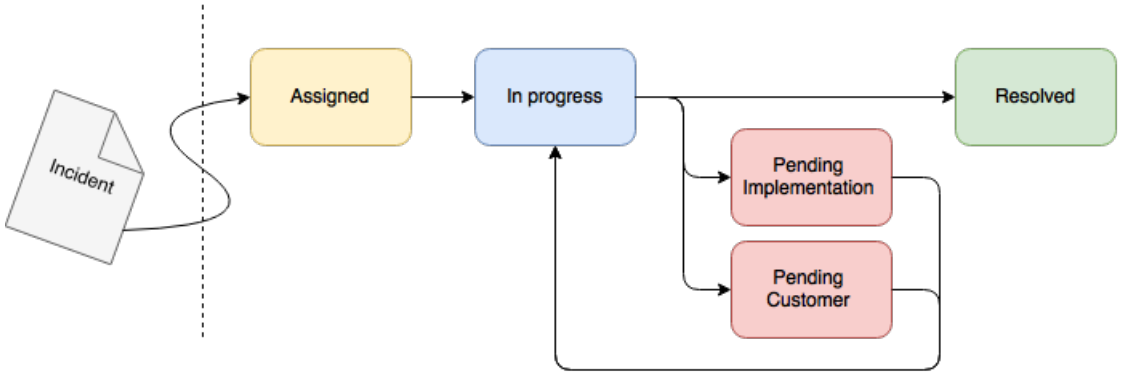
Figur 2.1: Ett flödesdiagram över ärenden.

När en utredare (person i supportpersonalen) påbörjar ett ärende är ett vanligt första steg att skaffa sig en egen bild av problemet och inte enbart gå på problembeskrivningen. Det kan hända att problembeskrivningen inte stämmer överens med verkligheten, och skapar man inte en egen bild kan onödig tid gå till spillo. När man har verifierat att problemet kan återskapas och kontrollerat detta mot SAP-miljön kan man påbörja sin lösning av ärendet. När man eventuellt har ändrat i utvecklingsmiljön samt testmiljön och testerna från både supportpersonal och verksamhet går igenom sammanställs alla ändringar och tester enligt kundens dokumentmallar. Detta granskas av en IT-arkitekt hos kunden och om inga brister hittas sätter man ändringarna i produktion.

I vilken ordning ärenden prioriteras beror mycket på deadline. Enligt avtal finns det krav

på när ett ärende ska vara klart, och prioriteringar görs oftast enligt dessa krav. Om man missar deadline för ett ärende får den automatiskt en lägre prioritet. Mycket handlar också om att bygga upp ett bra förtroende för att representera CGI, och därför kan det hända att ärenden från vissa personer kommer högre upp i prioritetlistan [50].

Ärenden går igenom olika tillstånd när de behandlas av supportpersonalen, se Figur 2.2. När ett ärende har tilldelats en ansvarig supportpersonal får den status ”Assigned”. Supportpersonalen kan tilldela ett ärende till sig själv eller tilldela ett ärende till andra inom samma grupp (detta är mindre vanligt, och uppstår om man vet att en särskild person har lämplig kompetens för att lösa problemet). När den ansvarige påbörjar arbetet övergår status till ”In progress”. Om arbetet stöter på ett hinder som kräver kommunikation med kund övergår status till ”Pending Customer” tills det att hindret har åtgärdats. På liknande vis övergår status till ”Pending implementation” om lösningen kräver testning eller ska in i produktion. När ärendet slutligen är åtgärdat övergår status till ”Resolved” (detta tillstånd kan endast nås från ”In progress”).



Figur 2.2: Ett flödesdiagram över status för ett ärende. Den streckade linjen indikerar att ett ärende har hamnat i Remedy.

Ett ärende klassificeras som komplext när man har lagt ner X antal timmar och antalet timmar regleras av avtalen som CGI har med kunden. Systemet vi studerar hanterar endast SAP-ärenden och dessa har ett fastpris som motsvaras av ungefär fem timmars arbete. Om ärendet slår över till ett komplext ärende påbörjas löpande debitering. Vid det här tillfället rapporterar supportpersonalen till kunden vad som har gjorts och varför detta har tagit lång tid. Kunden får även en beskrivning av det som återstår och ett estimat på hur lång tid detta bör ta. Sedan avgör IT-arkitekten hos kunden om man skall fortsätta med ärendet eller inte. Om ett (komplext) ärende visar sig vara mycket större än förväntat brukar det oftast gå över till ett utvecklingsärende, vilket hanteras i en separat process.

2.3 SAP

SAP är ett affärssystem som kan ge stöd till flera affärsområden inom ett företag. Systemet är en så kallad ”business suite”, känt som SAP eftersom det har skapats av ett företag med samma namn. Ett affärssystem är ett programpaket med integrerade IT-system som

utnyttjas till informationshantering och för att tillgodose ett företags behov av styrning och administration. Systemet är skrivet i programmeringsspråken Java och ABAP, som också är utvecklat av företaget SAP. En av anledningarna till att systemet är så populärt är att det går att anpassa mot de flesta branscher tack vare dess möjligheter för utbyggnad [29]. Företag som utnyttjar systemet kan välja att bygga ut olika subsystem genom att skapa insticksprogram med egna subrutiner. Detta erbjuder stora möjligheter för företagen att skräddarsy egna lösningar som är lämpliga för sina specifika ändamål. Ett kompletterande koncept är egenutveckling av webbgränssnitt [14, 24]. Företagen utnyttjar systemet efter eget behov och detta innebär att en del funktionaliteter inte är nödvändiga. Genom att skapa egna webbgränssnitt kan företagen dölja överflödig funktionalitet och erhålla ett gränssnitt som är enkelt och koncist samtidigt som man har stor flexibilitet och möjligheter för expansion.

Kapitel 3
Bakgrund och relaterat arbete

Det här kapitlet beskriver bakgrundsfakta som är relaterat till ärendehantering för att skapa en gemensam bild över olika begrepp, processer och metodik och avslutas med relaterade forskningsartiklar och arbeten som har utförts inom området.

3.1 Bakgrund

Mjukvaruföretag som utvecklar produkter får ständigt in information om behov från kunder som ska uppfyllas. Alla dessa behov formuleras om till funktionella krav och fastställs i ett kravdokument som godkänns av involverade parter, och dokumentet hamnar i *baseline* [39]. Ändringar av krav eller tillägg av ny funktionalitet efter att kravdokumentet har satts i baseline måste göras via formella ändringsbegäran. Det råder inget tvivel om att ändringar kan ske vid alla faser i ett projekt, och att hålla reda på alla dessa krav medför ett stort ansvar på företaget och arbetsdisciplinen inom mjukvaruprojekten [38]. En arbetsprocess inom mjukvaruutveckling som ansvarar för att hantera inkommande krav är *kravhantering*. Utöver kravhantering, som ansvarar för alla krav i ett mjukvaruprojekt, är *ändringshantering* ännu en viktig disciplin som ansvarar för att underlätta ändringar vid olika stadier av utvecklingsprocessen. Slutligen är *konfigurationshantering* en kompletterande disciplin som skapar en bro mellan kravhantering och ändringshantering och andra discipliner [39].

3.1.1 Kravhantering

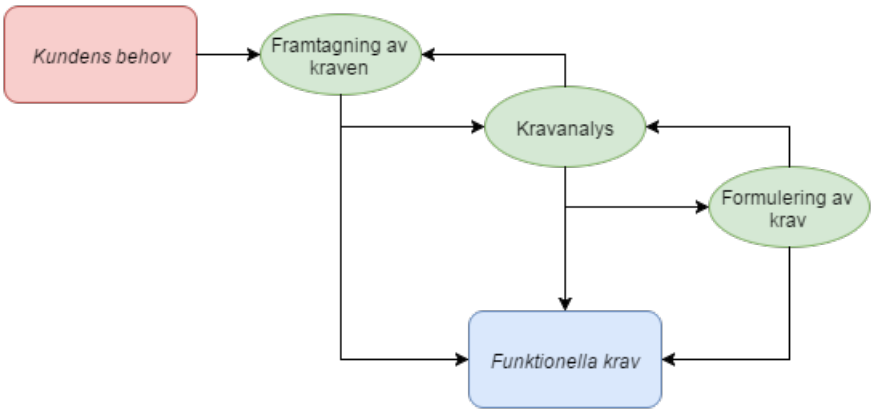
Kravhantering är en viktig disciplin inom mjukvaruutveckling som ansvarar för utvecklingsprocessens tillförlitlighet genom att erbjuda verifikation av kraven, förvaltning av produkten under utvecklingsprocessen och efter leverans samt rättsäkerhet för både kund och leverantör [40].

Innan utveckling av ett system påbörjas går arbetet ut på att identifiera kraven för produkten och presentera dessa för kunden för validering [40, 66]. Denna process inkluderar framtagning av kraven, såväl som indelning och utvärdering för att försäkra sig om att kraven går att uppfylla och att de är tillräckligt tydliga för att kunna användas under utvecklingen [7, 66], se Figur 3.1.

En viktig aktivitet i denna process är framtagning av krav. Detta innebär att man samlar in krav från såväl kund som användare och andra berörda intressenter. Detta inkluderar även indirekta insamlingsmetoder som marknadsstudier och prototyptestning för att ta reda på vilka krav som inte är lika självklara [7, 66]. Ytterligare två aktiviteter är kravanalys och formulering av krav.

Kravanalys består av ett antal subaktiviter, däribland prioritering och klassificering av krav. Huvudsyftet med kravanalys är att ta reda på varför kunden vill ha de krav som angetts genom att ta reda på syftet med produkten och identifiera användarna, och på så sätt få en bättre bild av de enskilda kraven [7, 66].

Vid formulering av krav sätts fokus på de krav som behöver ställas upp för utvecklingen snarare än de formuleringar som kommer in från kunder och användare. Detta innebär att kraven översätts och utvecklas till funktionella krav som är mer lämpliga för utvecklingsmiljön [7, 66]. Dessutom kontrollerar man även att kraven inte är motstridiga.



Figur 3.1: Ett schema över hur de tre huvudaktiviteterna inom kravhantering kan utföras. [7]

Det finns många metoder och processer för hur de tre aktiviteterna ovan kan utföras [7, 36, 84]. Vi går inte in så mycket på detta, men det är värt och nämna att framtagningen av kraven är en process som kräver noggranna undersökningar och bearbetningar av krav för att framställa en bra baseline för utvecklingen. Har man bristande kravhantering redan från början av arbetet kan detta leda till att det krävs stora omarbetningar och kan medföra stora kostnader för projektet [40]. Detta är även viktigt för att ha ett kontrakt gentemot en kund som försäkrar att båda parter är överens om vilka förväntningar de kan ha på produkten som utvecklas [40, 66].

Kravhantering är en kontinuerlig process som fortsätter även efter att kraven formulerats och godkänts av kunden, bland annat för att förenkla läsbarheten i och spårbarheten mellan

kraven [67]. Detta görs genom att kraven uppdateras med nya revisioner och omprioriteras, och ändras i enlighet med nya förändringar (*ändringsbegäran*). Syftet med detta är att erbjuda stöd till andra discipliner och processer i utvecklingen, som bland annat ändringshantering, konfigurationshantering och *kvalitetssäkring* [40, 66].

Ett område som kravhantering försöker underlätta är förändringsarbete [40]. I ett mjukvaruprojekt är förändringar oundvikliga eftersom det ständigt kommer in nya funktionella krav och buggfixar i form av ändringsbegäran. Utöver dessa avsiktliga ändringar kan det förekomma situationer då man är tvungen att utföra stora ändringar som kan påverka en stor del av systemet och i sin tur ge upphov till ytterligare ändringar. Ett exempel är att man kommer fram till att den övergripande systemarkitekturen är bristande och behöver omstruktureras. Ett annat exempel är att mjukvaran som utvecklas är anpassad för en specifik hårdvara som kommer från en extern leverantör, och om leverantören slutar utveckla denna produkt kan detta ha en stor påverkan på utvecklingen, och ge upphov till att mjukvaran måste anpassas till en ny hårdvara. Kravhantering försöker sprida kunskap i utvecklingen genom att definiera och dokumentera krav, det vill säga målen för utvecklingen [67], och på så sätt erbjuda projektmedlemmarna en säker grund att förhålla sig till [40]. Vi diskuterar vidare om hur ändringar hanteras i avsnitt 3.1.2.

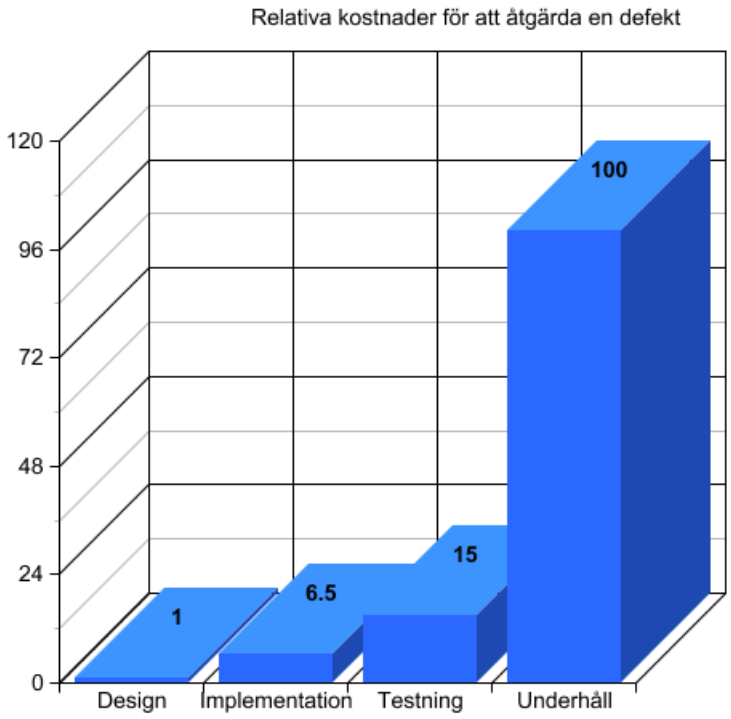
Framställning och underhåll av kravdokument är även viktigt för andra aktiviteter, till exempel *riskidentifiering* och testning [40, 41]. Riskidentifiering utgår ifrån de krav som är uppsatta för projektet och identifierar olika risker som varje krav kan medföra. Detta utförs för att man ska kunna skydda sig mot problem som kan uppstå under utvecklingen, antingen genom att motverka eller genom att vara medveten om och på så sätt förbereda handlingsberedskap. Riskidentifiering är en aktivitet som ingår i *risk management* [41]. Även vid testning utgår man från enskilda krav för att framställa testfall, detta för att försäkra om att alla krav verifieras.

3.1.2 Ändringshantering

Ändringshantering är en av de viktigaste aspekterna i utvecklingsprocessen [38]. Som namnet antyder är ändringshantering en process som hanterar ändringar av krav och mjukvara under kravhanteringsprocessen och utvecklingsprocessen [72]. Ändringar kan upplevas vara riskfyllda, speciellt när ett projekt närmar sig sitt slut, och tappra försök för att förhindra förändringar (efter en viss tidpunkt) har gjorts i olika projekt [38]. Erfarenhet visar dock att det oftast inte är möjligt att sätta en sådan begränsning, utan ändringar krävs genom hela projektet av olika anledningar, och därför får ändringshantering en viktig roll.

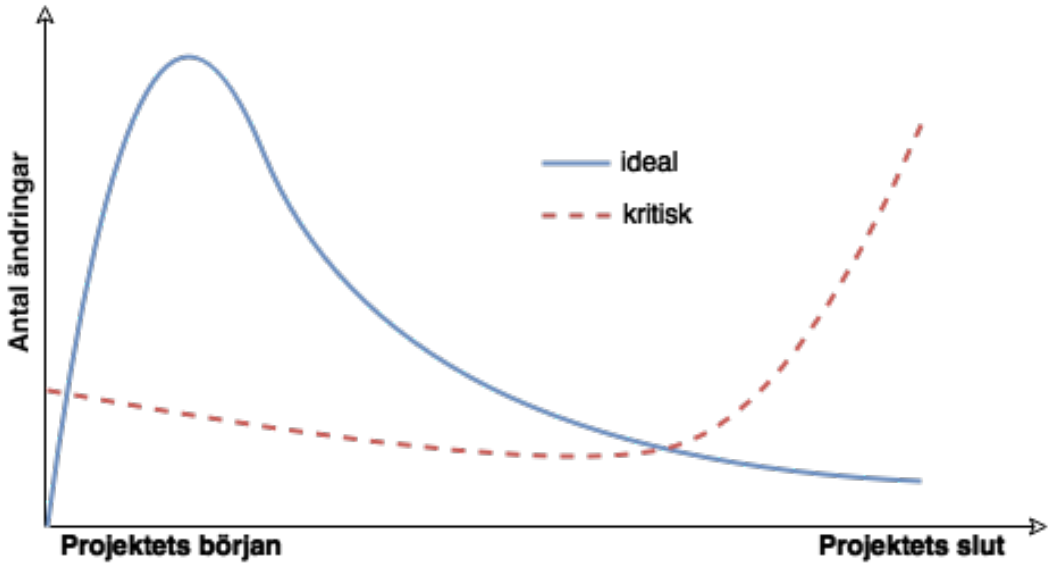
Det kan finnas många olika anledningar till varför man vill införa en ändring. Budgetändringar, omfördelning av resurser, schemaändringar, marknadsförändringar eller system- och arkitekturändringar är endast ett fåtal exempel som kan påtvinga ändringar i systemet. Bortglömda, motstridiga/felaktiga och felformulerade krav samt otydliga specifikationer påtvingar också förändringar, men risken för detta minskar med rätt tillämpad kravhantering [38]. Detta visar hur viktigt kravhantering är, eftersom kostnaden för att åtgärda ett fel som orsakas av ett felaktigt krav är mycket högre än ändringar som orsakas av system, arkitektur-och implementeringsfel [72]. Tyvärr ser projektledare ständiga ändringar

av krav som en av de främsta orsakerna till misslyckanden [72]. Det finns empiriska belägg som påstår att kvaliteten på kravhanteringsprocessen återspeglar kvaliteten på kraven och därmed har en effekt på projektet [72]. Om denna arbetsprocess tillämpas på rätt sätt ökar man chansen för framgång av ett projekt [72].



Figur 3.2: Kostnaden för att åtgärda en defekt ökar drastiskt med tiden [20, 70, 71, 83].

Som man kan se i Figur 3.2 ökar kostnaden för att genomföra en ändring med tiden, och är som störst vid slutet av projektet. Detta gör att implementering av många ändringar kan hämma ett projekts framgång, speciellt om dessa förekommer vid slutet av projektet [72]. Tyvärr lägger man oftast inte tillräckligt mycket fokus och arbete på att producera genomtänkta och välskrivna specifikationer i början, utan nöjer sig med att utgå från äldre specifikationer som mall och modifiera dessa för en snabb uppstart av ett projekt. Detta bidrar till inkonsekvens och misstag som upptäcks alldeles för sent, vid en tidpunkt då korrigering kostar som mest [72]. Det finns en studie som visar att endast en femtedel av alla krav är oberoende av varandra och detta medför att ändring av ett krav oftast bidrar till ändringar i andra krav [10, 17]. Om detta försummas kan det resultera i att man också försummar några av de faktiska effekterna av en förändring [10]. Därför är det mer kostnadseffektivt att upptäcka många fel i början av ett projekt där man endast har dokumentation att korrigera för att hålla nere kostnaderna [38], se Figur 3.3 för antalet ändringar i ett idealt respektive kritiskt projekt. Om detta inte är fallet innebär det att man har skrivit alldeles utmärkta specifikationer, eller mer vanligtvist, inte analyserat tillräckligt bra. Oavsett vad kräver situationen att man dokumenterar alla ändringar som utförs så att alla kan ta del av det [38].

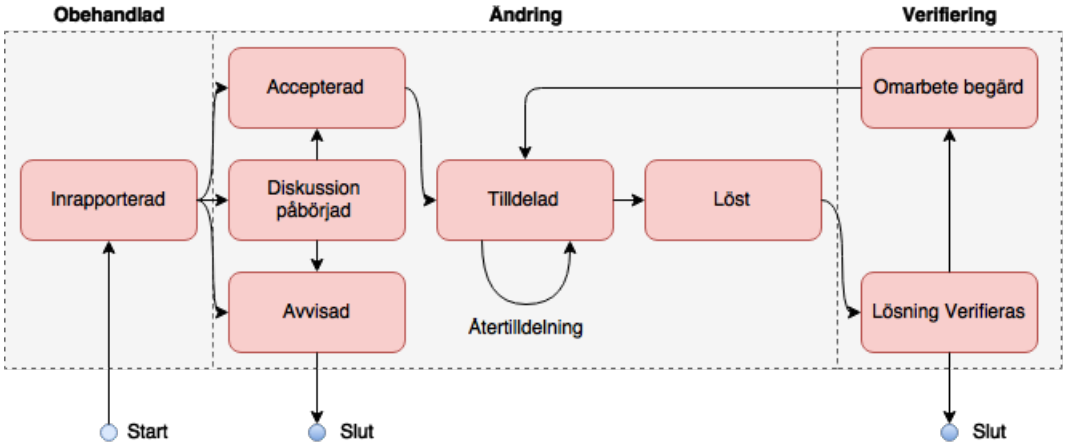


Figur 3.3: Antal ändringar genom ett projektets gång [38].

Ändringshantering kan delas in i två faser. I den första fasen har man väldigt fria händer vid produktion av kraven och kravdokumentet genom kontinuerliga ändringar. Detta är innan kravdokumentet har satts i baseline och har därför stark anknytning till utveckling av kraven (se tidigare avsnitt) medan den andra fasen startar efter att kravdokumentet har satts i baseline. Detta innebär att alla ändringar som utförs i denna fas är kontrollerade, genom att man skickar in en ändringsbegäran som sedan behandlas och antingen godkänns eller avböjs. Den andra fasen är väldigt relevant till denna studie eftersom vi utvärderar sådana formella ändringsbegäran [38].

En formell ändringsbegäran kan se olika ut beroende på vilket ärendehanteringssystem man använder. I regel eftersträvar man information såsom id, datum, prioritet på ändringen, vem som begär ändringen och till vem det tilldelas, en beskrivning som förklarar problemet och anledning till ändringen, vilket datum ett beslut togs och vad beslutet grundades på [38], se Figur 3.5. Studier visar att tydliga och välskrivna ändringsbegäran kan förebrygga en del utmaningar som finns inom ärendehanteringssystem [18]. Figur 3.4 illustrerar vilka stadier en ändringsbegäran kan gå igenom.

Under den första delen av processen skapar man en ny ändringsbegäran och fyller i nödvändig information, och vilka attribut som krävs beror på konfigurationen av det repositorie som används. Andra delen av processen följs av påverkansanalys, som är en viktig del av ändringshantering, och går ut på att identifiera vad som behöver modifieras för att genomföra en ändring och ta fram konsekvenserna om en sådan ändring skulle implementeras [10]. Analysen som utförs blir sedan en grund till vilket beslut man tar. De som brukar utföra sådana analyser är oftast projektledare eller erfarna utvecklare som kan bedöma



Figur 3.4: Ett generellt flödesdiagram för en ändringsbegäran [18].

huruvida beslutet om ändringen påverkar planeringen, kostnaden och den övergripande systemkvalitén [38]. Om ändringen accepteras tilldelas ändringsbegäran till en lämplig utvecklare [18]. Slutligen följs processen av en verifikationsfas, där man verifierar att ändringen är rätt implementerad.

Även om fördelarna är stora med ovanstående process finns det en del utmaningar att ta hänsyn till. Duplicerade ändringsbegäran i repositoriet eller tilldelning av en ändringsbegäran till en lämplig utvecklare är till exempel några av utmaningarna. Mycket av arbetet som sker måste hanteras manuellt, och om man tar hänsyn till att flera hundra ändringsbegäran kan rapporteras per dag kommer man fram till att det innebär mycket jobb och ett stort ansvar [18].

3.1.3 Konfigurationshantering

Som vi nämnt tidigare i avsnitt 3.1 agerar konfigurationshantering som en bro mellan olika discipliner. Detta beror på att konfigurationshantering fokuserar på artefakter som redan existerar inom andra discipliner, och har som syfte att underhålla dessa på en mer detaljerad och specialiserad nivå än vad som antyds i de olika disciplinerna. Dessa artefakter beskrivs som configuration items (CI). En CI kan vara ett kontrakt, ett tekniskt dokument eller diverse dokument som beskriver hur olika procedurer ska utföras och olika standarder används för olika moment i utvecklingen. Även källkod samt exekverbar kod som levereras till en kund anses vara en eller flera CI, beroende på om det är indelat i moduler. Något som är gemensamt för alla dessa artefakter är att de behöver någon form av separat identifiering, testning, versionshantering och underhåll [11, 25, 39, 45].

Ett stort fokusområde inom konfigurationshantering är kontrollerade ändringar. Kontrollerade ändringar innebär att man kan utföra ändringar på ett säkert sätt utan att påverkas av oförutsedda biverkningar och bibehålla en välstrukturerad arbetsmiljö, vare sig det gäller ändringar i kod eller diverse dokument. Tre byggstenar för att uppnå detta mål är att hålla reda på versioner för varje CI, skapa baselines med jämna mellanrum och hålla reda på de olika konfigurationer, det vill säga kombinationer av olika CI som man skapat

Projektformulär för ändringsbegäran		
Ändringsbegärans nummer	Rapporterad av	Datum för begäran
Beskrivning av ändringsbegäran		
Beskrivning av ändringsbegäran (Inkludera påverkade objekt, leverabler och nya objektiv och leverabler)		
Anledning eller teknisk motivering för ändringsbegäran		
Prioritet ☐ Högst ☐ Hög ☐ Mellan ☐ Låg		
Ändringens inverkan		
Påverkansanalys		
Inverkan på projektets krav ☐ Inom räckvidd ☐ Utom räckvidd		
Inverkan på projektets risker		
Inverkan på projektetplan		
Inverkan på projektets beräknade budget		
Inverkan på projektets konfiguration		
Alternativ		
Rekommendation		
Mötesbeslut		
Beslut ☐ Godkänd ☐ Parkerat ☐ Nekad	Datum för beslut	
Beslutat av ☐ Projektledare ☐ Projektansvarig ☐ Verkställande projektansvarig ☐ Övrig		
Skäl för beslut		

Uppföljning av ändringsbegäran (Uppdateringar till projektets baseline)					
Kravdokument	Ja	N/A	Ansvarig	Datum	Kommentarer
Schema (WBS)	Ja	N/A	Ansvarig	Datum	Kommentarer
Riskhanterings plan	Ja	N/A	Ansvarig	Datum	Kommentarer
Konfigurationhanterings plan	Ja	N/A	Ansvarig	Datum	Kommentarer
Kommunikations plan	Ja	N/A	Ansvarig	Datum	Kommentarer

Figur 3.5: Ett exempel på hur ett formulär för en ändringsbegäran kan se ut [87].

[11, 25, 39]. En viktig särskiljning som man gör mellan olika versioner är att man skiljer på varianter, vilket är alternativa implementationer av en artefakt beroende på exempelvis olika hårdvarukompabiliteter och revisioner som representerar hur en artefakt ser ut vid olika tidpunkter [11, 51].

Konfigurationshantering strävar efter att alla dokument och all kod ska hanteras på ett

kontrollerat sätt, att man ska ha en tydlig översikt över historiken för dessa artefakter och att man ska kunna spåra ändringar i en artefakt till den som är ansvarig för ändringen, anledningen till ändringen samt annan intressant data, och ytterligare kunna spåra vidare ändringen till andra artefakter som påverkats. På så sätt har man en tydlig översikt över utvecklingen och tillåter för de olika disciplinen i utvecklingen att samarbeta på ett effektivt sätt [11, 25, 39].

3.1.4 Beslutshantering

Med tillräckligt mycket kunskap om kravhantering och hur ändringsprocesser går till kan vi fördjupa oss i vilka beslutsproblem som finns inom kravhantering och på vilka nivåer dessa beslut tas. Endast ett enda beslut kan ibland räcka för att ändra en hel projekts riktning (positiv som negativ), och därför är det viktigt att förstå hur och när dessa tas och vad den underliggande metodologin är för att förbättra processen [9].

Forskning kring beslutshantering är relativt nytt inom kravhantering men har börjat få mer uppmärksamhet. Detta beror på den omfattande forskningen som görs för att förbättra kravhanteringsprocessen, där medvetenheten om betydelsen av beslutsstöd har framkommit [5]. Även om beslutshantering inte har uppmärksammats tillräckligt mycket kvarstår faktumet att det är en väldigt viktig del som avsevärt påverkar kravhanteringsprocessen [4, 9, 28]. DeGregorio [26] utförde en forskning om beslutshantering på Software & System Engineering Laboratory of Motorola Labs och kom fram till följande:

“[...] The most successful companies in the future will be the ones which leverage their intellectual capital generated by the decision making process and would link this process to the essential supporting information.“

Slutsatsen som DeGregorio kom fram till innebär egentligen att framgångsrik kravhantering inte är möjlig utan beslutshantering och detta bekräftas även av andra forskare. Evans, Park och Alberts betonar vikten på att se krav som designbeslut [28]. Regnell [75] har liknande åsikter och uttrycker detta med “Requirements mean decisions!”. Ruhe [78] väljer att beskriva “software planning”, “development” och “evolution process” som en kontinuerlig problemlösning och beslutsfattande aktivitet medan Aurum och Wohlin [9] beskriver de ingående aktiviteterna i kravhantering som en beslutstagande process.

Ett beslutsproblem kan antingen vara strukturerat eller ostrukturerat [82]. Med strukturerad avser man sådana problem som har en välidentifierad process för att nå ett beslut och brukar oftast innefatta repeterande problem som man har goda kunskaper om. Ostrukturerade beslutsproblem innefattar sådana problem där den associerade processen fortfarande är oklar. Detta brukar vara nya beslutsproblem man aldrig, eller väldigt få gånger, har stött på och därför saknar man oftast ingående kunskaper om hur man bör hantera sådana situationer. Semi-strukturerade beslutsproblem är sådana som inte har en etablerad process för att lösa ett problem men några initiala riktlinjer har utvecklats. Situationen är typisk inom kravhantering, där de flesta av de viktiga beslutsproblem inte är nya, men de tillhörande processerna är vanligtvis tvetydiga [5, 8].

Som vi nämnt i början av det här stycket kan man ta ett beslut på olika nivåer. Dessa nivå-

er brukar definieras som *strategic, tactical* och *operational* [6]. Strategiska beslut tas på organisationsnivå och kan beröra organisationens eller produktens mål. Beslut som tas på den här nivån är långsiktiga och har en stor omfattning och påverkan. De kan till exempel påverka alla aktiviteter i en organisation. Strategiska beslut betraktas vara ostrukturerade och några typiska problem på den här nivån är identifiering av affärsmål och val av kravhanteringsprocesser.

Taktiska beslut berör oftast planeringsfrågor för att uppnå de mål som beslutats i den strategiska nivån. Besluten som tas på den här nivån har inte lika stor omfattning och påverkan som föregående nivå. Taktiska beslut anses vara semi-strukturerade och några typiska exempel är projektplanering (och relaterade frågor som schemaläggning av olika uppgifter), identifiering av intressenter (ställningstagning till frågor berörande vem som är en intressent, vilken nivå av deltagande man bör förvänta sig av en intressent eller huruvida man bör prioritera dem), val av krav (vilka krav som bör implementeras) och release planning (som visar sig vara ett *NP-svårt* problem, se nästa avsnitt).

Operativa beslut tas på den lägsta nivån och omfattar individuella beslut som tas av kravingenjörer, utvecklare och testare och berör specifika uppgifter för att genomföra projektet enligt planeringen. Några typexempel på frågor som kräver ett beslut på den här nivån är när man till exempel skall sluta testa, hur man skall designa olika moduler eller vilka mönster man bör använda för att uppnå en arkitektur av en viss kvalité. Sådana problem är mer vardagliga och påträffas mycket oftare, och därför anses operativa beslutsproblem vara mer strukturerade och inte lika viktiga som övriga beslut som tas på högre nivåer, därmed enklare att hantera (undantag finns som till exempel acceptanstestning som är ett operativt beslut men mycket viktigt för både kund och företag). Utöver tvetydiga processer kan det finnas fler faktorer som påverkar beslutshantering, såsom politik och makt inom organisationen [59] och detta bidrar till att öka risken för att ta ett felaktigt beslut med konsekvenser som kan vara dramatiska. Projektets storlek påverkar också vilka utmaningar man kan stå inför, speciellt om man har flera intressenter i projektet med varierande syn på vad som exempelvis genererar mer vinst eller vilka krav som bör vara mer prioriterade [13]. Dessutom är det sannolikt att man står inför utmaningar såsom dåligt strukturerade problem, skiftande mål, otillräcklig information, komplexa beroenden mellan olika krav och långa återkopplingstider vilket försvårar för individer att ta lämpliga beslut [4, 5, 22].

3.1.5 Estimering och komplexitet

En viktig förutsättning för att lyckas konkurrera inom IT-marknaden är att kunna hålla sig inom en budget och uppfylla uppsatta deadlines. Detta är väldigt viktigt både för den interna arbetsprocessen i ett företag, och för att ha nöjda kunder och användare och på så sätt skapa efterfrågan för företagets tjänster och produkter. Det är därför viktigt att företag har processer uppsatta för att estimera kostnader och resurser för sina projekt. Det finns många etablerade metoder för att utföra sådana estimat, men trots detta visar studier att en stor andel av mjukvaruprojekt skjuter över den planerade budgeten och missar uppsatta deadlines [44, 61, 63, 65]. Anledningar är dels att företagen överskattar sin förmåga att utföra korrekta estimat (både för att man har för lite kunskap om avgörande estimeringsfaktorer samt på grund av att man har bristande kravhanteringsprocesser som i sin tur påverkar

estimeringen), dels för att många av de metoder som används i en del företag är föråldrade och har stora felmarginaler [30, 33, 34, 44, 63].

Många äldre estimeringsmetoder fokuserar på programkod som utgångspunkt för att uppskatta komplexiteten för mjukvaran och använder denna som ett mått för att etablera en budget och tidsplan. Detta innebär förstås att man behöver historisk data för att utföra dessa estimeringar eftersom man inte har någon programkod i början av ett projekt [64]. Metoderna utgår antingen ifrån en algoritm med inparametrar som beräknar en komplexitet eller heuristiska metoder. Exempel på parametriska metoder är Cocomo [16], Halstead software difficulty metric [35], Mc Cabe Cyclometric complexity metrics [53], Klemola's KLCID complexity metric [46], Wang's cognitive functional complexity [89] och Kushwaha's Cognitive Information Complexity Measure [47] . Det finns många fler sådana tekniker men grundtanken bakom dem är densamma. Exempel på heuristiska metoder är Expert Judgment Method [62], Tumregler och Delphi Technique [64]. Problemet med pararmetriska och heuristiska metoder är att man måste kombinera flera av dessa för att erhålla relativt bra estimeringar, och man är beroende av tidigare erfarenheter [64].

Nyare forskning inom estimering fokuserar på att hitta estimeringsmetoder som har kravdokumentet som utgångspunkt. En fördel med denna tillvägagång är att man inte längre är beroende av historisk data utan kan utgå från de krav som ställts upp för det projekt som uppskattas. Sharma och Kushwaha föreslår en sådan teknik som utnyttjar IEEE software requirement specification document [1] som utgångspunkt för att bedöma komplexiteten hos enskilda krav i ett kravdokument [80]. Tekniken verkar lovande i jämförelse med kodbaserade tekniker [81]. En liknande studie av Paviotti och Martins utnyttjar kunskaper från erfarna yrkesmän för att producera kriterier för att bedöma komplexiteten i krav [77]. Däremot kräver sådana tekniker att man har mogna kravhanteringsprocesser på företaget eftersom kravdokumenten påverkar estimaten.

3.2 Relaterade arbeten

I det här avsnittet sammanfattas tidigare arbeten som är relaterade till vår studie. Vi har valt att undersöka arbeten som baseras på ändringsbegäran och hur dessa påverkar ledtid och beslutet som tas, såväl som arbeten som fokuserar på kravkomplexitet. Anledningen till att fokus läggs på kravkomplexitet är för att det har direkt påverkan på ärendehanteringsprocessen. Vi fokuserar på såväl teoretiska som tillämpade arbeten. Slutligen följer en summering av nuvarande forskning och vad vårt arbete bidrar med.

3.2.1 Tidigare forskning

Wnuk et al. [91] har skrivit en forskningsartikel med fokus att identifiera olika parametrar som påverkar ledtid och beslutstagning. Forskningen baseras på data från ett företag och kompletteras med formulärundersökning. Parametrarna som undersöks är komplexitet, release number och kundbetydelse och hur dessa påverkar beslutstagningen och ledtiden. Dessutom undersöks huruvida ledtiden påverkar beslutstagningen. Resultatet baseras på statistiska tester och med hjälp av hypotesställningar kommer man fram till resultat som verifieras med enkätundersökning [91].

Regnell et al. [74] delar in kravhantering i fyra olika situationer beroende på hur många krav som behandlas i ett företag. Den minst omfattande av dessa kallas för SSRE (small-scale requirements engineering) och beskriver projekt som hanterar ett tiotal krav. De övriga situationerna kallas för MSRE (medium-scale requirements engineering) med hundratals krav, LSRE (large-scale requirements engineering) med tusentals krav och VLSRE (very large-scale requirements engineering) med tiotusentals krav. Fokus i artikeln ligger på hur större projekt som kräver VLSRE kan förbättra dess kravhanteringsprocesser. Författarna har utfört en studie på ett mobilföretag som har övergått från LSRE till VLSRE för att visa hur denna uppskalning kan påverka kravhanteringen och vilka anpassningar detta kan kräva. Artikeln avslutas med att tre forskningsområden föreslås som anses vara relevanta för att få vidare insyn i hur större projekt kan främjas av liknande uppskalningar. Denna artikel är relevant för vårt ändamål eftersom det behandlar hur komplexiteten växer med antalet krav.

Cavalcanti et al. [18] har publicerat en studie vars huvudsyfte är att ta fram svårigheterna och möjligheterna med att utnyttja ärendehanteringssystem. Ärendehanteringssystem har en viktig roll i underhåll av mjukvarusystem men kommer också med en viss kostnad. Skribenterna undersökte 142 studier och klassificerade dessa i olika områden (research area) och ämnen (topic). “Change request effort estimation” är ett av områden som kartlades med tillhörande ämnen som “Påverkansanalys” och “Time to fix” vilket är attribut vi finner är viktiga när man analyserar hur komplext ett ärende är. Den här systematiska mappningen användes för att analysera varje ämne i sig för att ta fram existerande svårigheter och huruvida verktyg och onlinetjänster löser dessa. Det här arbetet gav oss en insikt i hur omfattande ändringshantering kan vara och vilka problem man kan stå inför.

Wnuk et al. [76] har utfört en fallstudie på tre företag med olika produktionsskalor, och fokuserat på hur kravhanteringsartefakter hanteras. 13 enkätintervjuer, i samband med diskussioner mellan intervjuaren och intervjupersonen har utförts på vardera företag och resultatet har sammanställts för att visa skillnaderna mellan företagen. Enkäterna riktades till personal som är delaktig i olika delar av utvecklingsprocessen. Studien syftar till att visa hur problemen och angreppssätten skiljer sig beroende på produktionsskalan. Detta är intressant för vår del eftersom det ger oss ytterligare insikt i faktorer som kan påverka komplexiteten i kraven.

Bagnall et al. [12] har undersökt problematiken gällande vilka features som skall vara med i nästa release. Företag som utvecklar och underhåller komplexa programvarusystem måste bestämma de funktionaliteter som bör läggas till i deras system som en del av nästa utgåva. Det gäller att hitta en balans sådan att man lyckas tillfredsställa kundens önskemål samtidigt som man säkerställer sig om att man har tillräckliga resurser för att genomföra den nödvändiga utvecklingen, vilket påverkar de beslut man tar vid utvärdering av ändringsbegäran. Denna situation är modellerad i detta arbete och problemet med att välja optimal mängd av funktioner visar sig vara NP-svårt.

An Ngo-The och Ruhe [5] beskriver grundläggande fakta om beslutsvetenskap (decision science) och hur lite uppmärksamhet detta har fått inom kravhantering. Det finns en del kända beslutsproblem inom fältet, som till exempel identifiering av affärsmål, val av kravhanteringsprocesser, identifiering av intressenter och val av krav, och dessa problem kräver

att man tar beslut på olika nivåer (allt från organisation till individuella val). Ett beslut kan vara mycket avgörande för ett projekt, och mjukvaruutveckling drivs av beslut som tas dagligen. Därför tycker författarna att man bör ta fram beslutstöd som kan hjälpa till att ta fram optimala eller tillräckligt bra beslut. Målet med beslutsstöd är inte att ersätta mänskliga bedömningar och expertis, utan att hjälpa människor att göra bättre beslut. Författarna har gjort ett omfattande arbete och tagit fram ett klassificeringsschema (classification scheme) av relaterade arbeten. Man lyckades endast identifiera 44 forskningspapper efter att ha filtrerat alla arbeten utifrån uppsatta kriterier. Målet var att försöka få en bättre förståelse av vilka beslutsproblem som får större uppmärksamhet inom kravhantering och vilka som försummas. Man kom bland annat fram till att betydelsen och utmaningarna med beslutsbefattning inte har uppmärksammats tillräckligt i forskarvärlden.

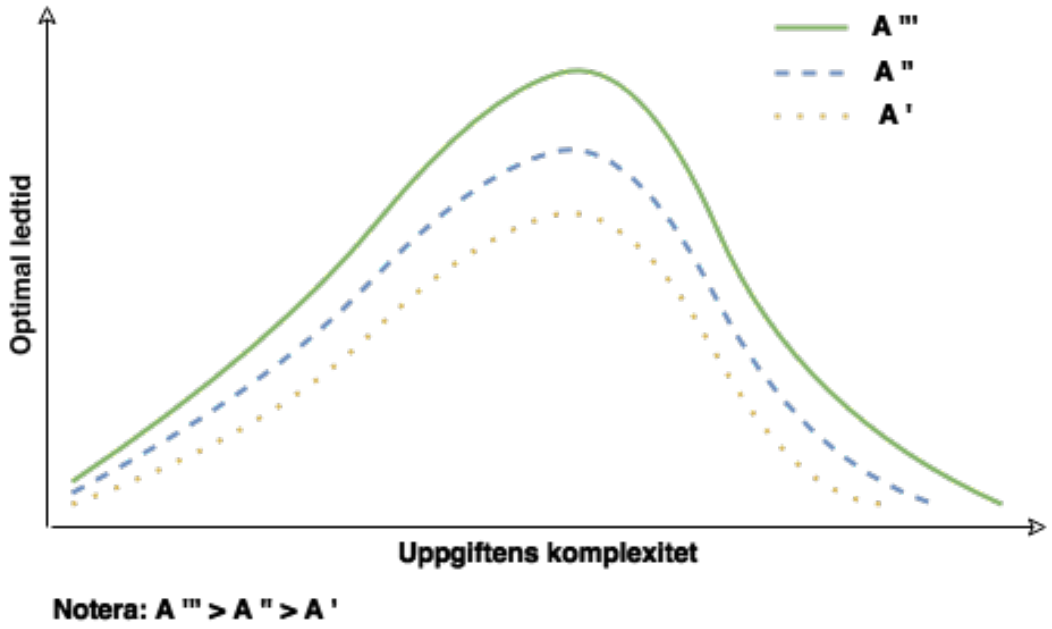
Sharma och Kushwaha [80, 81] har publicerat en studie som undersöker komplexiteten hos mjukvaror med kravdokumentet som utgångspunkt. Denna metod för estimering av komplexitet föreslås vara användbar för planeringssyften vid tidiga utvecklingsstadier. Metoden valideras genom jämförelse mot ett flertal beprövade metoder för komplexitetsestimering med programkod som utgångspunkt. Studien tar upp aspekter i kravdokument som är intressanta för vår studie, även om vi fokuserar på enskilda eller mindre grupper av krav.

Paviotti och Martins [77] presenterar ett antal kriterier för att mäta komplexiteten hos enskilda krav i ett kravdokument. Kriterierna är framtagna genom forskning inom kravhantering och har validerats genom en fallstudie och enkätsundersökning där yrkesmän med mycket erfarenhet inom mjukvaruutveckling har bedömt kriterierna. Studien är starkt anknuten till vårt forskningsområde då det angriper samma problemställning, dock anser vi att det krävs ytterligare studier som baseras på verklig data från arbetsmarknaden för att ytterligare validera kriterierna.

Hogarth [37] beskriver sambandet mellan ledtiden och komplexiteten i en uppgift med hjälp av en matematisk modell. Modellen utgörs av variabler och ekvationer som används till att kvantifiera komplexiteten och kostnaden i beslutsprocessen. Komplexitet definieras som en växande funktion av två variabler som används i modellen, nämligen antalet egenskaper/dimensioner per alternativ och svårigheten att välja mellan olika alternativ. Slutsatsen man erhöll var följande: Ledtiden ökar med komplexiteten tills man når en viss punkt. Punkten beskriver människans begränsade informationsbehandlingsförmåga och när detta har nått sitt maxima börjar ledtiden att minska med den ökande tidskomplexiteten eftersom man inte längre har kapaciteten till att ta in all information, se Figur 3.6.

3.2.2 Summering och bidrag

Resultatet från relaterade arbeten kan sammanfattas på följande vis. Allteftersom företagen utvecklas och tar sig an större leveranser uppstår ett behov av större förståelse och noggrannhet för de processer som utnyttjas. Individer som är med i dessa processer ansvarar för att saker och ting ska gå till på rätt sätt, men det finns en gräns för människans förmåga att hantera komplexa problem. Det behövs därför beslutsstöd som man kan förlita sig på för att säkerställa att man kan uppfylla kundens förväntningar utan att stiga över dem resurser som finns tillgängliga. Komplexitet och kundbetydelse är två faktorer som kan ha stark anknytning till ledtiden och beslutstagningsprocessen, och är därför intressanta att ta



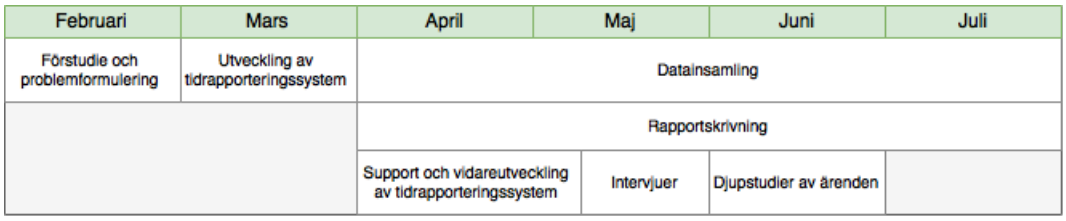
Figur 3.6: En graf som visar sambandet mellan ledtid och komplexitet. Notera att ju färre alternativ man har att utgå ifrån, desto kortare blir ledtiden [37].

hänsyn till vid konstruktion av ett beslutssöd.

Som det framgår finns det en del studier som undersöker komplexiteten kring ändringsbegäran men området är ännu färskt och utspritt med olika angreppssätt. Det krävs ytterligare studier för att kartlägga komplexiteten och identifiera vilka aspekter som kan utnyttjas till att förbättra beslutsprocessen och estimering av resurser. Dessutom måste resultatet från befintliga studier knytas ihop för att få en klar bild av de framsteg som gjorts. Vi har som mål med detta arbete att erbjuda ytterligare mognad till tidigare forskning och eventuellt styrka en del av de argument som redan finns.

Kapitel 4
Metod

I det här kapitlet beskriver vi bland annat hur vi utvecklade tidrapporteringssystemet. Systemet är nu i drift och används av supportpersonalen, och utnyttjades i examensarbetet för mätning och identifiering av de mest tidrapporterade incidenterna. Insamling av nödvändig data behövdes för att kunna undersöka RQ1 och RQ4. Efter att utvecklingsprocessen har presenterats följer en beskrivning av vilken typ av data som har tagits fram och hur datan har bearbetats för att undersöka RQ2 och RQ3. Två källor har utnyttjats för att erhålla ett underlag för de resonemang som görs i undersökningen, dessa är i form av intervjuer och analys av ärenden från ett ärendehanteringssystem. Anledningen till att vi utför intervjuer är för att detta erbjuder en extra dimension som kan utnyttjas för att förstärka validiteten i det resultat som erhålls vid analys av ärenden.



Figur 4.1: En översikt över arbetet inom ramen för examensarbetet.

4.1 Förstudie och problemformulering

Innan ett examensarbete kan inledas måste man skriva ett måldokument och tidsplan som innehåller bakgrund och motiv till examensarbetet, angreppssätt, vetenskaplig grund, kunskapsutveckling inom ämnet samt en tidsplan över arbetet. Ett godkänt måldokument är en förutsättning för att kunna påbörja examensarbetet. Ett examensarbetet måste bidra med

kunskapsutveckling inom ett ämne enligt datainstitutionens nya och förändrade regler och detta påverkar naturligtvis problemformuleringen. Examensförslaget som CGI hade var endast tekniskt fokuserat, och inte giltigt ur institutionens syn, och mycket av arbetet gick åt till att vinkla förslaget. Efter möte med Krzysztof Wnuk, Flavius Gruian och CGI representanter lyckades vi enas om ett examensarbete som både företaget och universitet får nytta av. Under tiden friskade vi upp våra kunskaper i de teknologier som behövde användas i den praktiska delen av examensarbetet samt läste relevanta forskningsartiklar och böcker efter rekommendation av dåvarande examinator Krzysztof Wnuk.

4.2 Utveckling av systemet

Vi fick en övergripande kravställning på systemet från CGI men hade fria händer vad gäller den tekniska delen. En databas som innehåller tabeller för åtminstone incidenter, tid per incident och användardata behövdes för att lagra information och databasen behöver dagligen uppdateras med data från kundens ärendehanteringssystem. Vid problem av integration skall en logg skapas och informera administratör om vad som har skett. Efter möte med handledaren[1] bestämde vi oss för att använda MS SQL [57] som databas och ASP.NET MVC 4.0 [54] som ramverk för att skapa det webbaserade tidrapporteringsystemet.

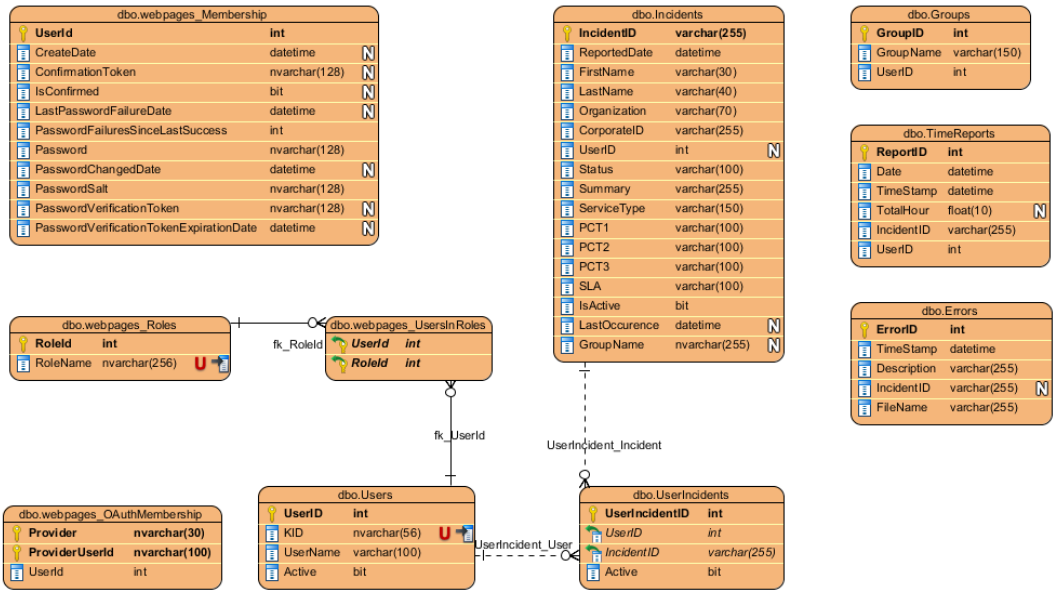
Vi hade inga designkrav för systemets utseende utan hade i uppgift att ge det ett acceptabelt utseende som kan vidareutvecklas längre fram. Vi valde därför att utnyttja ramverket Bootstrap [88] för den övergripande designen och ett insticksmodul skrivet i Javascript som heter FooTable [32], för att få interaktiva tabeller med sökning, sortering och paginering. Både Bootstrap och FooTable strävar efter att erbjuda en snabb och användarvänlig upplevelse för olika enheter, såväl datorer och mobila applikationer, eftersom de stödjer responsiv webbdesign [43, 60].

4.2.1 Utvecklingsfas

Vi skapade en databas-design utifrån textfilen vi fick av handledaren, som är ett utdrag från kundens ärendehanteringsystem och innehåller all information som systemet behövde lagra. Textfilen är uppdelad i olika kolumner och varje rad beskriver en incident i detalj. Utifrån textfilen och kommunikation med handledaren skapade vi tabeller som representerar användare, incidenter, tidrapporter, grupper och felmeddelande och dessa tabeller uppdaterades efterhand som utvecklingen pågick, se Figur 4.2 för den slutgiltiga lösningen. När databas-designen var klar och SQL databasen var uppsatt gick vi över till att implementera integrationsdelen av systemet, vars mål är att läsa in en textfil från ärendehanteringssystemet och uppdatera befintliga incidenter i databasen, och lägga till nya incidenter om dessa inte finns integrerade ännu. Vi valde att implementera integrationslösningen som en konsol-applikation i C#, och skapade en modell av den befintliga, lokala databasen med hjälp av Entity Framework v6.0. Detta autogenererade C#-klasser som representerar tabellerna i databasen, och medförde att vi kunde implementera allt i C# för att lägga till,

[1] Handledaren på CGI. Från denna punkt och framåt i texten används uttrycket ”handledaren” för att benämna vår handledare på CGI.

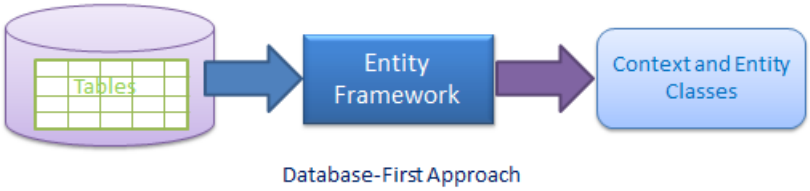
ta bort och modifiera information. Entity Framework skapade även en så kallad Context-klass som representerar databasen och innehåller egenskaper för tabellerna, se Figur 4.3. Med hjälp av dessa klasser kan man enkelt utföra förfrågningar mot databasen utan att att använda SQL eftersom alla förfrågningar översätts till SQL-förfrågningar ”under huven”. Detta är mycket smidigt eftersom hela applikationen blir utvecklat i ett enda språk och allt förblir objektorienterat, vilket har sina fördelar.



Figur 4.2: Den slutliga databasdesignen.

Integrationslösningen, som vi kallar för XMLParsing, består av drygt 400 rader kod. Vad applikationen gör i stora drag är att gå igenom Excel-filen, rad för rad, och sparar undan alla cellvärden på den aktuella raden i en vektor. Kombinationer av dessa cellvärden används för att avgöra om en viss grupp, användare, eller incident existerar, och om sådan är fallet uppdateras motsvarande entitet i databasen. Om entiteten däremot inte finns läggs den till i databasen. UserIncident-tabellen i databasen får en viktig funktion i integrationslösningen, eftersom den binder specifika användare till incidenter. Varje användare kan ha incidenter kopplade till sig, men kan också se incidenter som tillhör gruppen eller grupperna som användaren är med i. I och med att man kan ta bort incidenter från vyn när man har jobbat färdigt med dem vore det olämpligt att ta bort en incidenten från Incident-tabellen, eftersom detta skulle påverka alla användare. Istället skapar vi en koppling mellan en användare och en incident i UserIncident-tabellen, och tar bort denna koppling om en specifik användare väljer att ta bort incidenten. Därmed påverkas inte övriga användare som har samma incident kopplade till gruppen de tillhör.

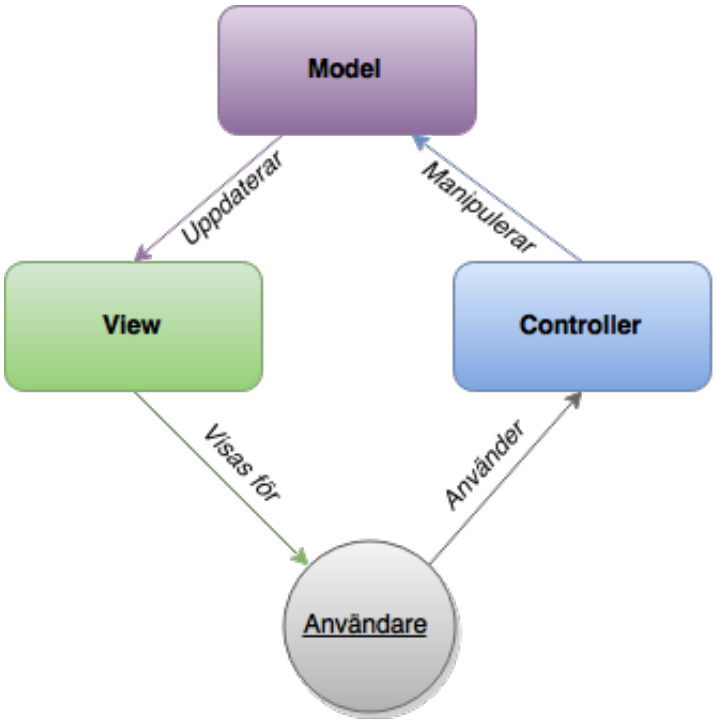
Excel-filen som används i applikationen måste ha en viss format på filnamnet, nämligen *Status_yyMMdd*, där yy är akutell år, MM är aktuell månad och yy är aktuell dag, eftersom datumet används för att läsa in aktuell Excel-fil. Detta är viktigt eftersom integrations-



Figur 4.3: Entity Framework database-first [27].

lösningen automatiserades, och körs varje dag på en virtuell maskin. Lösningen försöker öppna den aktuella Excel-filen 20 gånger med 30 minuters mellanrum. Om inläsningen misslyckas stängs programmet ner, för att försöka igen nästa dag. Supportpersonalen på CGI är ansvarig och ser till att Excel-filen hamnar på rätt plats i den virtuella maskinen.

När integrationslösningen var någorlunda färdigimplemeterad gick vi över till att implementera det webbaserade tidrapporteringssystemet. Vi utvecklade systemet i ASP.NET MVC 4 eftersom vi hade tidigare kunskap om detta ramverk och utvecklingsspråket som är C#. Som namnet tyder bygger ramverket på designmönstret ”Model - View - Controller”, vilket innebär en separation i tre ansvarsområden (se figur 4.4). View ansvarar för den grafiska representationen medan all ”business logic” ligger i Model, inklusive databaskommunikation, och Controller fungerar som en slags brygga mellan View och Model och sköter all interaktion däremellan. En sådan separation är till vår fördel eftersom vi kan till största del arbeta oberoende av varandra, och därmed bli mer effektiva. Dessutom blir koden renare och enklare att underhålla, och förenklar vårt arbete vid felsökning.



Figur 4.4: En översikt för hur Model, View och Controller samarbetar [52].

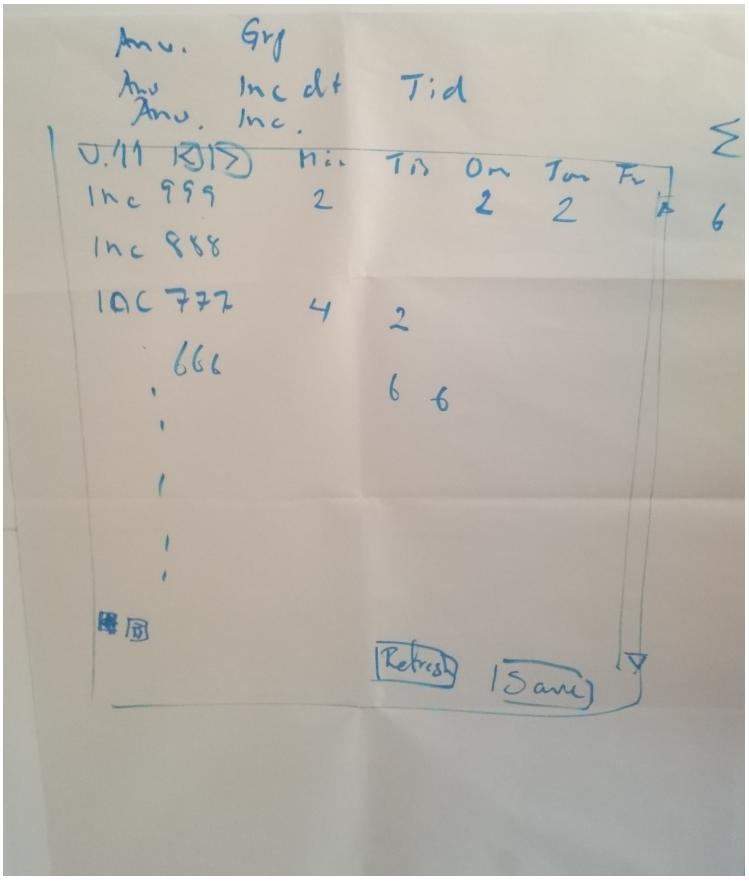
När vi skapade projektet valde vi en mall som har stöd för autentisering, vilket ger oss all den funktionalitet och kod som krävs för att skapa nya konton, logga in och byta lösenord. Integrationslösningen skapar användare utifrån Excel-filen som läses in, personal som har tilldelats ett ärende får ett konto enligt ett ID som anges i filen. Genom att skapa en administratörspanel och generera lösenord för dessa nyskapade konton får endast behöriga tillträde till systemet.

Vi började med att skapa en model av databasen, precis som i integrationslösningen, med hjälp av Entity Framework Database-first, och skapade därefter en IncidentController. IncidentController användes för att generera vyer som kan visa alla incidenter och visa detaljer om en viss incident. Om vyn är en så kallad *strongly-typed view* kan man utnyttja *Scaffold template* som är färdiga mallar för att till exempel skapa, lista eller editera modellen, och detta utnyttjade vi för att snabbt komma igång.

Vi gick sedan över till att skapa sidan för tidrapportering per ärende, som även har stöd för att lägga till incidenter om de inte blivit integrerade ännu. Det finns även stöd för lätt manövrering mellan olika veckor och support för sökning och borttagning av incidenter. Eftersom vi inte hade några prototyper att utgå ifrån fick vi själva tänka ut hur tidrapportering borde se ut (utöver de funktionella krav som hade ställts på sidan), men vår uppfattning stämde inte alltid överens med handledarens och supportpersonalens önskemål och därför gick lite tid förlorad. Efter diskussion kom vi fram till en prototyp som visas i Figur 4.5. Alla anrop som sker på denna vy går till TimeReportController. För att optimera användarupplevelsen gick vi över till att använda AJAX-request som är asynkrona anrop till servern, vilket innebär att servern (TimeReportController) svarar med partiella vyer och uppdaterar endast en portion av sidan istället för hela vyn som bidrog till snabbare svarstider och bättre upplevelse. Denna teknik utnyttjades även på övriga sidor.

En AdminController skapades tillsammans med ett administratör-konto vars syfte är att kunna se och hantera kontoadministration, integrationsloggar och inloggningsstatistik. Vidare kan en administratör även generera nya lösenord för användare och se hur mycket tid en person har registrerat under vald period. Precis som i övriga Controller-klasser utnyttjade vi databasmodellen för att skicka data vidare till administratörsvyerna och dessa vyer skapade vi via Scaffold template. Vyerna begränsades till endast administratörskontot genom att utnyttja *Roles* klassen och använda så kallade *Data annotations* på Controller-nivå. Data annotations kan ses som attribut som kan sätta begränsningar på metoder eller hela klasser. Man kan till exempel tillåta anonyma användare att besöka en viss sida med *[AllowAnonymous]* eller begränsa sidan till autentiserade användare med *[Authorize]*, och man kan dessutom förse DataAnnotations med ytterligare attribut för att till exempel endast tillåta vissa roller eller vissa användare.

Vi avslutade utvecklingsfasen med refaktorisering av HTML-kod i vyerna och C#-kod i Controller-klasserna. Eftersom både tidrapporteringssystemet och integrationslösningen utnyttjade samma model för databaskommunikation skapade vi ett *bibliotek* (DLL fil) för modellen, för att slippa underhålla samma filer på två olika platser. Vi optimerade även förfrågningarna som utfördes mot databasen, eftersom vissa förfrågningar tog väldigt lång tid. Vi skapade även så kallade *ViewModels*, som är klasser som representerar modeldata i en specifik vy. Dessa är väldigt användbara om modell-klassen innehåller väldigt många



Figur 4.5: Efter möte med handledaren på CGI fick vi en idé om hur tidrapportering borde se ut.

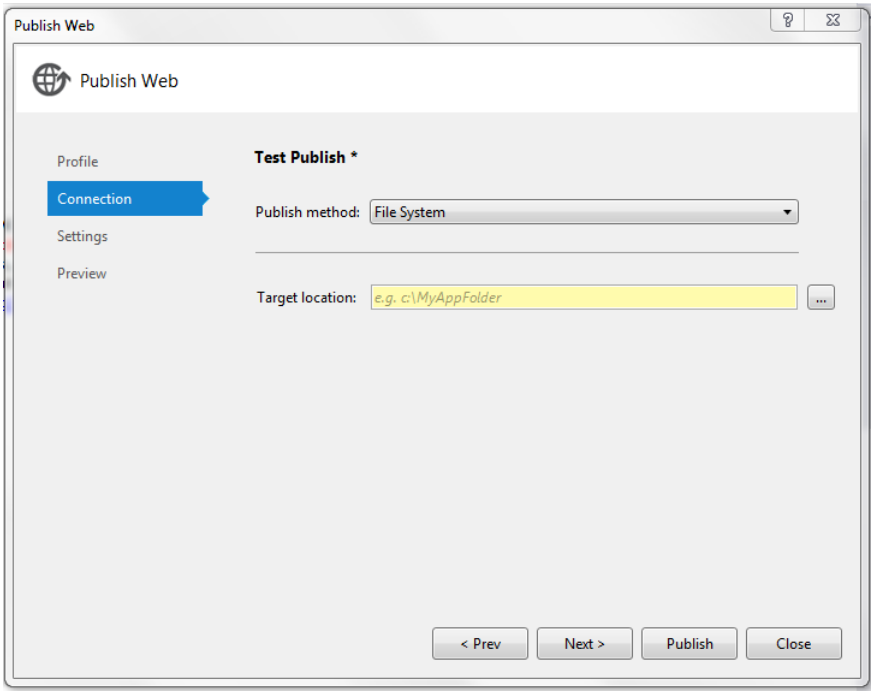
egenskaper, men man endast behöver ett fåtal av dem för en specifik vy.

4.2.2 Produktionsfas

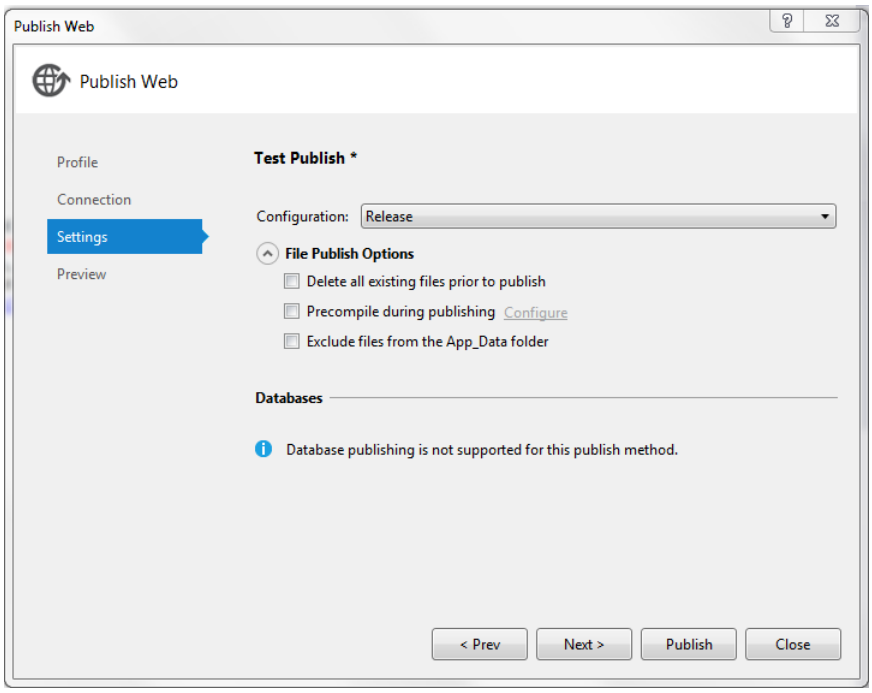
När systemet var färdigutvecklat behövde vi en lämplig serverdator samt en databas för att tillgängliggöra webbapplikationen för suppportpersonalen. Microsoft Azure är en platform för molnbaserade system, med möjlighet för användare att installera hemsidor och datorapplikationer [55]. Innan systemet kunde publiceras på Azure ville vi kontrollera att det inte fanns några oförutsedda problem med publiceringsprocessen, och för att göra det enklare att undersöka detta föreslog handledaren på CGI att vi skulle använda en intern serverdator på företaget.

Den interna serverdatorn hade en IIS server (Microsofts webbservertjänst) installerad [56]. För att kunna publicera webbapplikationen på IIS servern behövde vi paketera applikationen tillsammans med dess inställningar till ett format som servern kan hantera. Eftersom både utvecklingsmiljön Visual Studio och IIS är skapade av Microsoft var detta hyfsat trivialt. Microsoft har möjliggjort för utvecklare att paketera webbapplikationer för publicering på en IIS server på ett enkelt sätt genom att ange publiceringsmetod och inställningar för publicering i Visual Studio, se Figur 4.6 och 4.7. Vi hade inte möjlighet att publice-

ra direkt till IIS servern och skapade därför ett filsystem enligt Figur 4.6 och överförde denna manuellt till serverdatorn. Detta bidrog till att databasen inte följde med i filsystemet, vilket framgår i Figur 4.7, och av denna anledning skapade vi databasen manuellt på serverdatorn med hjälp av SQL Server Management Studio [58].



Figur 4.6: Val av publiceringsmetod för ett testprojekt.



Figur 4.7: Val av publiceringsinställningar för ett testprojekt.

Publiceringen av webbapplikationen på Azures webbserver visade sig vara väldigt enkelt eftersom vi kunde göra det via Visual Studio med ett litet tillägg. Den här gången utnyttjade vi en publiceringsprofil från Azure som innehöll publiceringsmetod och inställningar som krävdes för att publicera applikationen. Vi skapade även en virtuell miljö på Azure (ett virtuellt Windows 7 operativsystem) och laddade upp och schemalade integrationslösningen på denna. Det enda som återstod var att möjliggöra ett enkelt sätt att ladda upp Excel-filerna på den virtuella miljön. Vi satte därför igång en FTP-server på den virtuella miljön med hjälp av FTP-klienten FileZilla [31] och skapade ett konto som CGI personalen kunde utnyttja för att koppla upp sig direkt till den mapp där Excel-filerna behövde laddas upp. FTP är ett protokoll som utnyttjas för att skicka filer mellan en klientdator och serverdator [3, 49].

4.3 Genomförandet av intervjuer

Fyra personer har intervjuats på företaget, se Tabell 4.1. Intervjupersonerna har anonymiserats på så sätt att de benämns som person A-D. Gemensamt för alla intervjupersoner är att dessa jobbar med SAP-ärenden mot samma verksamhet, men har olika roller inom företaget. Vi har intervjuat personer med skilda kunskaper och inriktningar inom företaget för att kunna täcka fler perspektiv och därmed förstärka validiteten i våra slutsatser. Personerna har bland annat fått beskriva hur vanligt det är med komplexa ärenden i företaget och vilka attribut de tycker karaktäriserar ett komplext ärende utifrån egna erfarenheter. Svaren användes för jämförelse med resultatet från djupstudien och lade grunden till resultatet för RQ1 och RQ2. Vi har fått information om vilka grupper intervjupersonerna tillhör, hur stora dessa är samt genomsnittstiden per ärende för respektive grupp. Detaljerna i denna information är dock konfidentiella och kan därför inte redovisas i rapporten.

Intervjuerna har utförts enskilt med varje intervjuperson för att minska yttre påverkan, och säkerställa att varje individ känner sig trygg med att uttrycka sina åsikter och synpunkter. Detta är viktigt då studier visar på att gruppintervjuer kan leda till att intervjupersonerna påverkar varandra [85]. Intervjuerna var strukturerade på så sätt att en person ställde frågorna, medan en annan person diskuterade frågorna med intervjupersonen på ett mer avslappnat och ostrukturerat vis. På så sätt säkerställs att alla frågor besvaras samtidigt som intervjupersonen får tillfälle att diskutera och tänka igenom sina svar på varje fråga. Studier kring intervjuer visar att intervjuer som utförs av två intervjuare brukar leda till att intervjupersonen är mer öppen för diskussion [42]. För att försäkra att data från intervjuerna inte går förlorad spelade vi in intervjuerna och transkriberade dessa till textform efter varje intervjutillfälle. Intervjupersonen fick sedan möjlighet att förtydliga sina påståenden i texten för att undvika missförstånd, detta eftersom det inte alltid går att fånga upp innebörden från en muntlig formulering i textform [68]. Dessutom möjliggör inspelning för intervjuaren att fokusera på intervjupersonen utan att behöva oroa sig för att dokumentera alla svar [69, 73, 86, 90].

Intervjuerna fokuserar inte på personlig data om intervjupersonen utan fokuserar endast på arbetsmässig erfarenhet, och de processer och system som finns implementerade i företaget. Av denna andledning riskerar vi inte att hämma intervjupersonens svar eftersom frågorna inte anses vara känsliga [23].

Intervjuperson	Erfarenheter & Arbetsuppgifter
A	Systemvetare från Lunds universitet. Har jobbat med systemutveckling specifikt mot energibranschen. 10 års erfarenhet med SAP.
B	Arbetar med webblösningar för SAP, både interna och externa lösningar. Har kantidatexamen inom informatik. 13 års erfarenhet med SAP.
C	Agerar som länk mellan de tekniska resurserna och verksamheten som har någon sorts problem, läser igenom kod för att hitta orsak till felet. Jobbar med mätvärdeshantering. 14 års erfarenhet med SAP.
D	Agerar som länk mellan de tekniska resurserna och verksamheten som har någon sorts problem, läser igenom kod för att hitta orsak till felet. Hjälper även till vid testning. 12 års erfarenhet med SAP.

Tabell 4.1: Personer som intervjuats.

Intervjuguiden har bifogats i Appendix A.

4.4 Analys av ärenden

Vi har valt ut de nio mest tidrapporterade ärendena som underlag för vår djupstudie, dock har endast åtta av dessa visat sig lämpliga för utvärdering, se Tabell 4.2. Anledningen till att vi endast valde ut nio ärenden är dels för att hinna med analysen av dessa, dels eftersom det inte fanns många fler ärenden som var storleksmässigt intressanta att undersöka. Vi undersökte potentiella faktorer som påtvingade dessa incidenter att växa i tid, och definierade detta som potentiella komplexitetsfaktorer. Vi ser detta som faktorer som kan tänkas definiera komplexitet i ett ärende.

Analysen baseras på två informationskällor. Varje ärende har en problembeskrivning och en arbetslogg. Problembeskrivningen består av den information som rapporteras i Remedy i samband med att en incident skapas. Arbetsloggen innehåller alla statusuppdateringar och kommunikation mellan personal och kund som sköts via mejl. Vi har studerat problembeskrivningen och vid behov även kommunikationen i arbetsloggen för att få en översikt över vad varje ärende går ut på. Sedan har vi studerat kommunikationen, med fokus på vilka som är delaktiga i ärendet, när meddelanden skickats och hur lång tid som har passerat mellan meddelanden, vilka problem som har tagits upp i kommunikationen, samt hur och när statusen förändrats för samtliga ärenden. Slutligen har vi utnyttjat denna information för att plocka ut eventuella aspekter och problem som tycks ha påverkat tiden för att lösa ärendena och därmed medfört att dessa har blivit större än övriga ärenden.

ID	Spenderad tid	Öppnat	Stängt
1	19.25h	2015-02-19	-
2	40h	2015-03-20	-
3	17h	2015-03-27	2015-04-30
4	28.5h	2015-05-22	-
5	30h	2015-01-30	2015-05-26
6	17h	2015-02-02	2015-05-28
7	18h	2015-03-06	-
8	17h	2015-03-06	2015-05-19

Tabell 4.2: Incidenter som har använts för djupstudien. De incidenter som inte har någon sluttid har studerats innan de avslutats.

Kapitel 5
Resultat och diskussion

I det här avsnittet diskuterar vi forskningsfrågorna vi lade upp i första kapitlet och besvarar dem utifrån intervjuerna och djupstudierna vi har utfört. Frågor som vi inte har kunnat besvara på grund av brist på data lämnar vi istället över som möjliga forskningsfrågor i avsnitt 5.6.

5.1 RQ1: Andel komplexa ärenden

Första forskningsfrågan undersöker hur vanlig förekomsten av komplexa ärenden är på företaget. Med komplexa ärenden menas de ärenden som överstiger den avtalade tiden med kund och har en egen process att fullfölja. Vi har använt intervjuerna som underlag för att besvara den här frågan och jämfört resultatet med mätning av systemet för verifiering. Enligt person A blir ungefär 1 av 200 ärenden komplexa enligt företagets definition på komplexitet (X antal timmar). Denna person hävdar att de flesta ärenden blir halvstora, men inte riktigt når upp till gränsen för att klassificeras som komplexa.

Person B är relativt ny på företaget, men har jobbat med SAP-ärenden i 13 år sedan tidigare. Supportgruppen som person B sitter i är ganska liten, och sköter inte samma typer av ärenden som person A. Redan i början av intervjun påpekade personen att denna inte har jobbat med komplexa ärenden inom företaget, och kunde tyvärr inte ge oss en uppskattning över hur många ärenden som blir komplexa. Men personen påpekade att inget ärende har blivit komplext i supportgruppen som denna jobbar för under personens anställdningsperiod (> 1 år), vilket stämmer bra överens med det person A har sagt. Person B hävdar att det är vanligare att ärenden går över till att bli utvecklingsärenden, vilket oftast omfattar ny funktionalitet, i dennas supportgrupp. Utvecklingsärenden, som används till syfte att utveckla ny funktionalitet till de befintliga systemen, skiljer sig från SAP-ärenden och har en egen process, och supportpersonalen på CGI hanterar endast SAP-ärenden. Därför har vi inte haft möjligheten att analysera och utvärdera utvecklingsärenden.

Person C har jobbat med verksamheten sedan den startade och har mycket erfarenhet inom supportorganisationen. Under intervjun fick vi aldrig en uppskattning på antalet komplexa ärenden, men när vi frågade om kännetecken och attribut som karaktäriserar komplexa ärenden kunde personen inte riktigt ge svar på frågan eftersom de sällan jobbar med dem, enligt personens egna erfarenhet. Person C är med i samma supportgrupp som person A och är en av de som har rapporterat in flest antal timmar inom gruppen. Person C's erfarenhet tyder på att det är få ärenden som blir komplexa, men enligt personen är det vanligare att ett komplext ärende godkänns och fullföljer processen än att det förkastas.

Person D har jobbat med SAP i 12 år och har funnits på företaget sedan Logica-tiden. Enligt person D jobbar de i princip aldrig med komplexa ärenden i den grupp denna tillhör, och personligen var det över fem år sedan personen jobbade med ett sådant ärende. Personen påpekade även under intervjun att komplexa ärenden är mer sällsynt för deras grupp jämfört med person A's grupp. Person D nämnde även under intervjun att det har varit dålig uppföljning på ärendena, och att man inte riktigt haft koll på när ett ärende blir komplext. Därför uttryckte personen att systemet vi har implementerat kan hjälpa till att identifiera fler komplexa ärenden.

Summerar man ihop alla intervjupersoners åsikter är slutsatsen vi kan dra att det är väldigt få ärenden som blir komplexa i den studerade organisationen. Mätningar i systemet visar att fördelningenen av antalet timmar som spenderas per ärende stämmer överens med det intervjupersonerna har sagt, och att det är väldigt få incidenter som kommer upp till den gräns som har satts av företaget för att klassificeras som komplext.

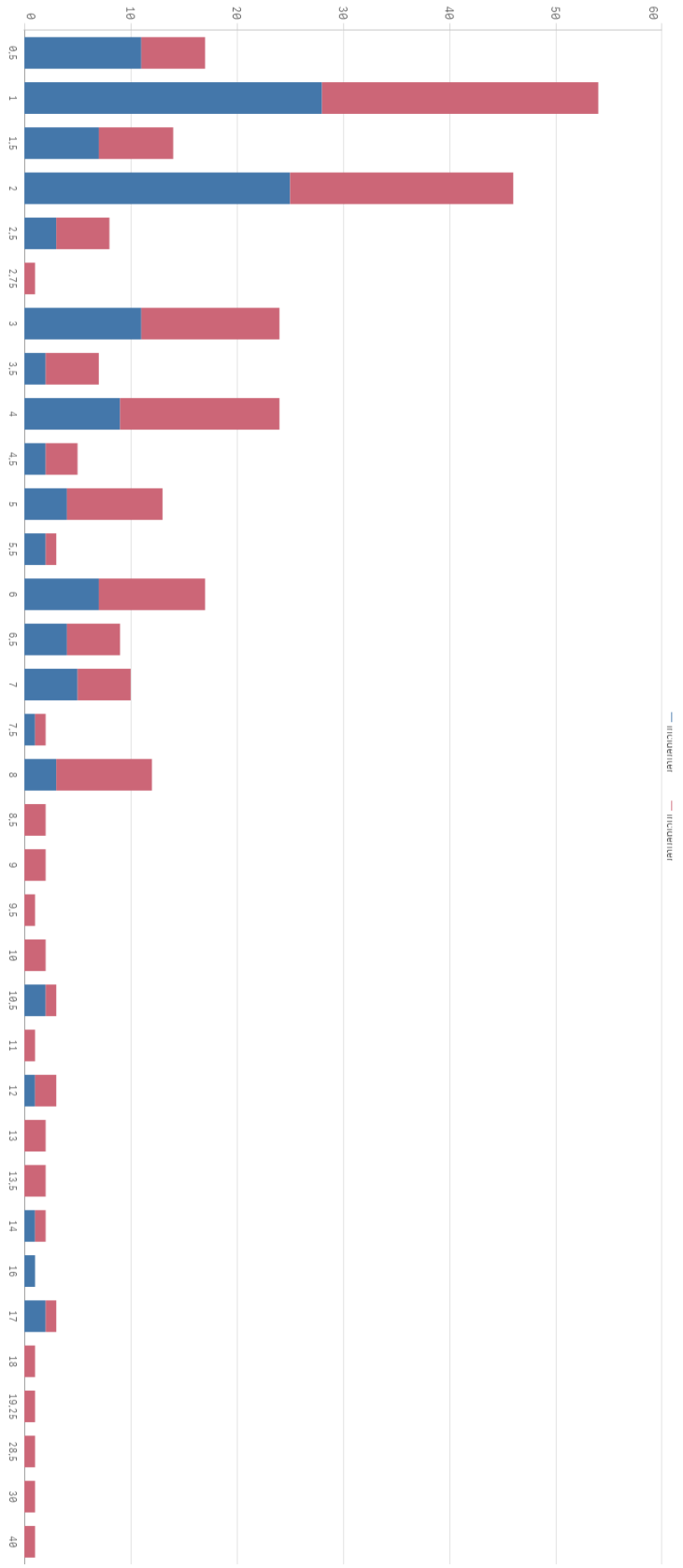
Figur 5.1 visar fördelningen på tidsrapporterade ärenden som togs under perioden april till juni 2015. Grafen innehåller både öppna och stängda ärenden. Ett ärende betraktas som stängt om integrationslösningen inte har uppdaterat ärendet på två veckor, det vill säga om ärendet inte har funnits med i Excel-filen som bearbetas dagligen av lösningen de senaste två veckorna[1]. Enligt figuren är en och två timmar per ärende bland de vanligaste och majoriteten av incidenterna ligger under fem timmar, vilket är en bit ifrån den gräns företaget har satt för komplexitet (X antal timmar). Detta bekräftar det som har sagts under intervjuerna och slutsatsen vi har kommit fram till angående första forskningsfrågan.

5.2 RQ2: Karaktärisering av komplexitet

Andra forskningsfrågan har undersökts genom intervjuer med supportpersonalen på CGI och analys av incidentloggar. Under intervjuerna har personerna fått beskriva vilka attribut de tycker karaktäriserar ett komplext ärende utifrån egna erfarenheter. Vi har kompletterat detta med analys av incidentloggarna, genom att följa arbetsutvecklingen i diskussionerna och noterat intressanta faktorer som kan ha påverkat komplexiteten. Resultatet från intervjuerna och analysen av incidentloggarna framgår i Tabell 5.1 och 5.2.

De faktorer som anmärkts av supportpersonalen (se tabell 5.1) har delats upp i fyra grupper för att förenkla diskussionen. Grupperingen utfördes genom att utvärdera vilka punkter

[1]För att skilja på öppna och stängda incidenter lade vi till kolumnen *LastOccurrence* i tabellen *Incident*, som håller reda på det senaste datumet ett ärende uppdaterades.



Figur 5.1: Antalet timmar (x-led) som spenderas per ärende för öppna (övre del) och stängda (nedre del) incidenter (y-led).

Komplexitetsfaktor	Intervjuperson(er)
1. Ändring av kod som redan gått i produktion och fått oförutsedda konsekvenser	A
2. Gruppbyte, efter undersökning kommer man fram till att ärendet ska till någon annan person/grupp	A, B
3. Om det är något som är svårt att testa	A
4. Arbetet behandlar kod som skrivits av annan person som inte är tillgänglig för konsultation	A,C
5. Felet går inte att återskapa	B
6. Dåligt dokumenterad kod	B, D
7. Duplicerade ärenden leder till extra arbete om de inte detekteras i tid	B
8. Felaktig information från kunden angående felet leder till onödigt arbete på fel ställe	C
9. Uppgiften lämnas över från en person till en annan, detta kräver extra kommunikation och uppföljning	C, D
10. Otydliga instruktioner, kunden kommer med synpunkter vid slutet av arbetet	C
11. Många delar av koden behöver ändras	C, D
12. Den existerande koden har hög komplexitet, programmet är stort	C, D
13. Arbetet beror på någon annan vilket leder till ökad väntetid	D
14. Kunden behöver rådfrågas men är upptagen eftersom man prioriterar andra ärenden	D
15. Den krävda kunskapen är fördelad bland flera personer	D
16. En ändring i en modul leder till att andra moduler måste modifieras	D
17. Olika kontrakt leder till extra arbete för att undersöka hur arbetet ska utföras och av vilken personal	D

Tabell 5.1: Resultat från intervjuer: faktorer som bidrar till att göra ett ärende komplext.

som är nära relaterade enligt vår egen uppfattning. Dessa är faktorer som påverkas av:

- **Kodegenskaper**, vilket inkluderar faktor 1, 3, 5, 7, 11, 12 och 16.
- **Process och verktygsmognad**, vilket inkluderar faktor 2, 8, 10, 13 och 17.
- **Kunskapsfördelning**, vilket inkluderar faktor 4, 6, 9 och 15.
- **Kundens prioriteringar**, vilket inkluderar faktor 14.

Den första gruppen styrs av egenskaper i koden såsom komplexitet och storlek, vilket är faktorer som kan undersökas vid påverkansanalys redan innan arbetet påbörjats. Faktorer i denna grupp som eventuellt kan vara svåra att detektera vid påverkansanalys är testnings- och återskapningsbegränsningar. Dock anser vi att dessa faktorer kan undersökas i början av arbetet för att avgöra om de kommer att ha en negativ inverkan på komplexiteten.

Den andra gruppen består av faktorer som till stor del hänger ihop med arbetsprocessen, det vill säga hur rapportering av ärenden hanteras, hur ärenden felsöks och hur felet blir löst. Därför skulle man kunna undvika dessa genom att uppmuntra tydligare kommunikation och bättre planering av resurser i arbetsprocessen. Ärenden som hamnar i fel grupper skulle kunna hanteras med hjälp av maskininlärningsprocesser i ärendehanteringsverktyget, denna lösning diskuteras vidare av Cavalcanti et al [18]. Även om en automatiserad lösning inte garanterar att alla ärenden hamnar i rätt grupper så kan det hjälpa till att minska den manuella arbetsbördan.

Den tredje gruppen består av problem som vi anser inte kan kontrolleras till fullo. Detta eftersom kunskapspridning är begränsad då personal byter jobb, arbetsuppgifter eller går på semester vilket leder till att en del kunskap måste tas in på nytt av en ersättare. Dålig dokumenterad kod är den faktor i gruppen som kan påverkas mest, genom att uppmuntra bättre dokumentation, dock är detta exempelvis inte tillämpligt i de fall då koden i fråga har utvecklats av ett annat företag. Ytterligare en intressant faktor i denna gruppen är de fall då den nödvändiga kunskapen är fördelad bland olika personal, eftersom detta kan undersökas till viss del vid påverkansanalys.

Den fjärde gruppen behandlar kundprioriteringar som även diskuteras i [91]. Eftersom det är viktigt att hålla kunden nöjd bör man ta hänsyn till hur man prioriterar olika arbetsuppgifter och ta detta i beräkning vid planeringsstadiet innan arbetet påbörjas.

Punkterna som sammanfattar våra upptäckter i Tabell 5.2 har tagits från djupstudier av de mest tidsrapporterade incidenterna. Efter att ha jämfört dessa faktorer med de som presenterats i tabell 5.1 har vi kommit fram till att 11 av totalt 24 faktorer som tagits fram via analysen överensstämmer med vad personalen föreslagit.

Tabell 5.2: Resultat från analys av ärenden.

Komplexitetsfaktor	Incident(er)
1. Arbetet påverkade gammal funktionalitet negativt	1

2. Man fick avvakta arbetet med en incident eftersom det blockerade annan utveckling med högre prioritet	1
3. Bristande dokumentation över befintlig lösning som skulle ändras	2
4. Olika klasser i testmiljön och produktionsmiljön vilket lett till komplikationer och förvirring	3
5. Stor kvantitet data behandlas av den funktionalitet som implementerats vilket leder till att det blir svårt att se var felet kan ha skett	3
6. Lösningen går inte att testa i en testmiljö, därför tar man stickprov för att se om det fungerar i huvudsystemet vilket tar extra tid och arbete	3
7. Datastrukturen innehåller många egenskaper vilket gjort det svårt att felsöka dessa, en person föreslog att man delade upp dessa egenskaper i flera klasser för att förenkla felsökning	3
8. Arbetet är tidskritiskt och fokus läggs därför på att fixa symptomen i första hand istället för att reparera orsaken till felet	4
9. Många scenarion som ska hanteras och olika personer hos kund behöver rådfrågas för varje	4
10. En del funktionalitet måste sättas ur bruk vid testning vilket resulterar i att man måste anmäla i god tid för att utföra tester och på något sätt kompensera för funktionaliteten som tas ur bruk	4
11. Allt är anpassat efter en felaktig lösning, som gör att korrekt logik leder till fel på oväntade ställen i koden	5
12. Den bästa lösningen kräver att man utför ändringar på många ställen i koden vilket blir väldigt tidskrävande, komplext och kostsamt	5
13. Byte av personal under arbetet	6, 7
14. Brist på information kring problemet	6
15. Misstolkning av kommunikation via e-post	6
16. Testning av incidenter förskjuts för att kunna testa dessa gemensamt när alla är klara	1
17. Oklart vem som skulle ha hand om ärendet	2

18. Felet är svårt att återskapa eftersom man vill undvika att användarna påverkas av arbetet som utförts för att lösa ärendet	2
19. Ärendet behandlade flera olika fel men antogs bero på samma ursprungskälla vid början av arbetet	2
20. Den person som hade mest insikt i det behandlade området var upptagen med ett annat ärende och blev därför rådfrågad via e-post med jämna mellanrum	2
21. Flera personer är involverade i lösningen med varsitt ansvarsområde vilket kräver mer kommunikation och ansvarsfördelning	3
22. Lång väntetid då man väntar på bekräftelse från slutanvändaren	3
23. Arbetet kräver verifikation från två parter både vid användning av resurser för felsökning och tillåtelse att jobba vidare	4
24. Problem med att återproducera felet eftersom personen inte hade fullständig kunskap i det som undersöktes och var därför tvungen att rådfråga andra	8

De 12 första faktorerna i Tabell 5.2 beror på begränsningar i koden. Sju stycken av dessa faktorer rapporterades även av intervjupersonerna, se Tabell 5.1, vilket ökar validiteten för dessa faktorer. En av dessa är faktor 3, bristande dokumentation, som föreslagits av person B och D under intervjuerna. Ännu en faktor är nummer 5 där stor kvantitet av data behandlas, vilket föreslår att koden har hög komplexitet, detta har påpekats av person C och D. Faktor 6 berör svårtestad kod som man är tvungen att testa på ett speciellt sätt och detta nämndes även av person A under intervjun. Faktor 7 behandlar komplex kod liksom faktor 5. Faktor 9 kräver återkoppling från flera personer, detta har nämnts av person D i form av att kunskapen är fördelad bland flera personer. Faktor 10 berör svårtestad kod liksom faktor 6. Faktor 12 handlar om att man måste ändra på många ställen i koden. Detta har nämnts under intervjuerna av person C och D.

De 12 sista faktorerna i Tabell 5.2 är relaterade till externa begränsningar utanför koden. Fyra av dessa faktorer stämmer överens med det som nämnts under intervjuerna, nämligen faktor 13, 18, 21 och 24. Faktor 13 handlar om att man byter personal under arbetet vilket kräver extra kommunikation och uppföljning. Detta har nämnts av person C och D under intervjuerna. Faktor 18 berör ett ärende där man var tvungen att manipulera befintlig data i systemet för att återskapa felet fast man undvek detta eftersom det skulle ha påverkat användarna. Detta innebär att man i princip inte har möjlighet att återskapa felet, vilket är något som har påpekats av person B. Faktor 21 behandlar ett ärende där flera personer är involverade i arbetet, vilket har lett till fler diskussioner och uppdelning av arbetet som

kan tänkas ta extra tid. Person D har nämnt detta under intervjun i form av att den krävda kunskapen är fördelad bland flera personer. Faktor 24 berör en incident där man haft problem med att återskapa felet vilket är någorlunda relaterat till faktor 18.

Vi har lyckats komma fram till en del potentiella komplexitetsfaktor både via intervjuer med personalen och genom att analysera ärenden. Däremot har vi inte haft möjlighet att utvärdera dessa faktorer och avgöra hur pass pålitliga de är. Detta både eftersom stödet inte finns för detta i dem ärenden som vi undersökt och på grund av att det kräver granskning av komplexa ärenden från en lång tidsperiod vilket vi inte heller har haft tillgång till. Detta beror också på att vi inte har kunnat ta del av utvecklingsmiljön, testmiljön och produktionsmiljön som används av både kund och CGI för att göra en egen bedömning.

5.3 RQ3: Samband

Vårt mål har varit att undersöka den tredje forskningsfrågan med hjälp av intervjuerna. Däremot har vi inte haft någon möjlighet att utvärdera den första delen av frågan, huruvida komplexiteten påverkar ledtiden, eftersom de ärenden som undersökts inte har någon formell ledtid. Den enda definitionen på ledtid som funnits i personalens arbete är förarbetet för ett ärende, där de sätter sig in i problemet och försöker återskapa felet. Detta innebär att vi skulle undersöka hur lång tid det tar för personalen att sätta sig in i ett komplext ärende vilket endast utgör en del av den ledtid som definierats i frågeställningen, eftersom syftet med denna frågeställning har varit att undersöka hur lång tid det tar att komma fram till ett beslut för huruvida man ska lösa ett ärende eller inte.

Vad gäller den andra delen av forskningsfrågan, det vill säga huruvida komplexiteten påverkar beslutsprocessen så har vi kommit fram till ett bristande svar som är väldigt specifikt till CGIs arbetsprocess. Eftersom de ärenden som undersökts är felrapporter så vill kunden ha dessa lösta och därför behandlar man alla de ärenden som kommer in utan någon beslutsprocess. Beslutsprocessen förekommer då ett ärende har pågått i ett bestämt antal timmar (förutsatt att ärendet blir komplext), och vi har inte haft möjlighet att ta del av denna process då själva beslutet tas av CGIs kund som vi inte har varit i kontakt med. Person A minns däremot inte att något ärende nekats av kunden och person C påstår att de flesta ärenden accepteras av kunden. Detta skulle innebära att komplexiteten inte påverkar beslutet alls. Men detta är i slutändan endast spekulationer eftersom vi inte har tillgång till någon som helst data från kundens beslutsfattande, utöver om det resulterat i ett godkänt ärende eller ej.

Det behövs ytterligare forskning för att kunna svara på vår tredje frågeställning, se avsnitt 5.6.

5.4 RQ4: Förbättring

Systemet vi har implementerat används inte bara för tidrapportering och mätning av ärenden, utan berör även den fjärde forskningsfrågan. Innan man gick över till Remedy användes ett annat ärendehanteringssystem på CGI, som hade stöd för tidrapportering, men

detta system togs ur bruk för ett par år sedan. Remedy saknar denna funktionalitet och supportpersonalen har fått använda alternativa lösningar för att hålla koll på tiden de har spenderat för respektive ärende, som till exempel personliga Excel-filer. Enligt handledaren själv var det mycket intuition och magkänsla som styrde, och man hade inte riktigt koll på när ärenden blev komplexa, och därav var man i behov av ett system där man enkelt kan tidrapportera och få en överskådlig bild på hur statusen på ärenden ser ut. Under intervjun frågade vi om personernas åsikter angående detta och fick liknande respons från alla. Person A konstaterade att man fick förlita sig på magkänsla vid tidrapportering och att lösningen med Excel-filer inte fungerade så bra. Person D anser att man inte har haft någon uppföljning för när ärenden blir komplexa, och det kan vara anledningen till att man har noterat ganska få av dem. Därför tycker person D att det är väldigt bra att man kan skriva in antalet timmar per ärende. Båda personernas åsikter tyder på att man tidigare inte har kunnat identifiera komplexa ärenden i tid eller förutse vilka ärenden som tenderar att bli komplexa.

Person C var mer specifik och beskrev varför Excel-lösningen inte fungerade särskilt bra. I Excel-filen skrev personen upp ärendena som denna hade tilldelats, och alla dagar och hur många timmar man hade jobbat på respektive ärende. I slutändan fick man gå igenom filen och jämföra om man har fått ihop lika många timmar som man egentligen har jobbat. Personen tyckte att detta blev opraktiskt (vilket övriga också tyckte, inklusive handledaren), och det var därför handledaren ville ha applikationen vi utvecklade. Personen konstaterade att han förstod handledarens syfte och intentioner med applikationen, eftersom situationen var ett problem för dem. Personen sammanfattade det som: *”Vi måste tjäna pengar, vi får ju liksom inte ge bort någonting”* under intervjun och var inne på hur systemet vi implementerat kan förbättra ärendehanteringskostnader med bättre uppföljning och större insikt i hur statusen på ärendena ser ut. Personen passade även på att jämföra vårt system med tidrapporteringsfunktionaliteten på det gamla ärendehanteringssystemet, och hävdade att det var en liknande hantering men lösningsstrategin var annorlunda. Personen i fråga tyckte att vårt sätt att lösa det var att föredra, eftersom överskådligheten blev bättre.

Det vi har fått ut från intervjuerna och diskussion med handledaren på CGI kan sammanfattas som följande: Med hjälp av systemet kan man få en bättre uppföljning och se vilka incidenter som blir komplexa. Detta är viktigt ur företagets syn eftersom ärenden som klassificeras som komplexa har en annan process och styrs av andra avtal, och av ekonomiska skäl är det viktigt att identifiera **alla** dessa ärenden. På så sätt kan man även förbättra ärendehanteringskostnader. Vi föreslog tillsammans med handledaren på LTH att även utveckla en modell som kan prediktera vilka ärenden som kommer att bli komplexa med hjälp av *machine learning*. Tanken var att utföra mätningar och se med vilken noggrannhet modellen predikterade. Vi kom fram till att detta inte var möjligt eftersom vi inte hade tillgång till den data som krävs för att träna upp modellen.

5.5 Forskningens validitet

Den första forskningsfrågan har besvarats genom intervjuer och observationer av de ärenden som tidrapporterats från det att tidrapporteringssystemet satts i bruk fram till examensarbetets slut, cirka tre månaders tid. Då intervjuerna baseras på personalens erfarenhet kan

det antas att dessa estimerar sina svar utifrån den arbetssätt som de haft innan systemet satts i bruk. Detta innebär att den definition de haft för komplexa ärenden kan ha skiljt sig från den verkliga situationen eftersom man inte haft fullständig inblick i hur lång tid ett ärende tar. Detta är en faktor som kan ha påverkat det resultat som uppnåtts. Dessutom har systemet endast varit i bruk under tre månader, och det har tagit en del tid i början för personalen att vänja sig vid att använda systemet. Det skulle vara intressant att undersöka hur statistiken ändrats om ett halvår då det förts in fler ärenden i systemet. Detta är två faktorer som vi tycker påverkar validiteten i denna forskningsfråga, och vi har haft dessa i åtanke då vi fokuserat på att få igång systemet så fort som möjligt.

Vi har undersökt den andra forskningsfrågan med hjälp av intervjuer och genom att analysera åtta komplexa ärenden, och producerat potentiella komplexitetsfaktorer utifrån dessa. De faktorer som nämnts av intervjupersonerna baseras på deras erfarenheter av att jobba med supportärenden. Detta begränsar deras insikt eftersom de inte sitter med utvecklingsärenden, då dessa utgör en stor del av ändringsbegäran inom mjukvaruutveckling. Däremot utför de mindre utvecklingsprojekt bland de ärenden som behandlas. Vi har endast haft möjlighet att intervjua fyra personer, detta beror på att intervjupersonerna är konsulter som kan sitta och jobba både på CGI och hos kund vilket gör det svårt att boka många intervjuer. Detta är ännu en faktor som kan ha påverkat resultatet, speciellt eftersom personalen jobbar i olika grupper med egna ansvarområden och har därför områdesspecifika erfarenheter. De ärenden som analyserats begränsades till åtta stycken delvis på grund av tidsbrist men även eftersom det inte fanns många ärenden som var tillräckligt stora att analysera. Detta begränsar resultatet eftersom det inte går att generalisera till alla typer av ärenden och eftersom undersökningen skulle kunna byggas vidare för att erhålla en starkare validitet. Analysen har utförts efter intervjuerna, vilket möjligtvis har påverkat vårt synsätt när vi undersökte komplexitetsfaktorer, dock tycks detta inte avspeglas i resultatet eftersom det endast är 11 av 24 faktorer som överlappar. Ännu en påverkande faktor som nämnts tidigare i diskussionen är att vi inte har haft möjlighet att utvärdera de komplexitetsfaktorer som vi tagit fram för att avgöra hur pass stor effekt de har på komplexiteten och hur ofta de förekommer.

För att besvara den fjärde forskningsfrågan har vi skapat ett tidrapporteringssystem som medför en bra översikt över hur arbetet med varje ärende utvecklas enligt vad handledaren och supportpersonalen begärt. Eftersom vi har haft fokus på att få igång systemet har vi ursprungligen endast utgått ifrån handledarens beskrivningar men efter att systemet satts i bruk har vi modifierat det till att ge en bättre användarupplevelse för personalen men även en bättre översikt för handledaren som behöver hålla reda på statistiken med jämna mellanrum. Det finns möjligheter för att bygga vidare systemet till att generera automatisk statistik och på så sätt få en snabbare översikt över hur ärenden utvecklas med tiden.

5.6 Möjligheter för vidare forskning

Vi har inte kunnat besvara vår tredje forskningsfråga, och detta är en intressant aspekt att studera i framtida studier för att se om komplexitet har någon som helst relation till ledtid och beslutsprocess. Detta kräver dock data och arbetsprocess som gör det möjligt att undersöka en sådan forskning. Man kan möjligtvis utgå från Hogarth's matematiska

model för att se om den stämmer överens med verkligheten, eftersom Hogarth's arbete saknar empiriskt stöd [37].

Ytterligare en sak som man kan forska vidare på är huruvida man kan prediktera tillväxten och på förhand säga vilka incidenter som kommer att bli komplexa med en viss noggrannhet. Detta är möjligt med machine learning, om man bygger upp en modell och tränar upp denna med gamla komplexa ärenden. Detta är något som möjligtvis hade gynnat företaget och det vore även intressant att utföra mätningar och se med vilken noggrannhet modellen predikterar. Detta kräver förstås lämplig data vilket vi inte hade tillgång till, och därmed lämnar vi över detta som förslag på möjlig vidare forskning.

Kapitel 6
Sammanfattning och slutsats

Studier visar att en stor andel av mjukvaruprojekt skjuter över den planerade budgeten och missar uppsatta deadlines. Det är därför viktigt att man har koll på hur mycket resurser olika uppgifter tar upp. Vårt examensarbete fokuserar på att ta fram kriterier som kan användas till att avgöra om ett ärende är komplext, i syfte att uppnå bättre kontroll över kostnader.

I de avancerade mjukvarusystem som utvecklas idag krävs det en hel del processer och arbetsmetodik för att säkerställa att produkten håller en bra kvalité. Företag som är marknadsdrivna måste ta snabba beslut för att kunna konkurrera på marknaden. Svårigheten uppstår i att organisationen oftast är tvungen att hantera väldigt många ändringsförfrågningar som kommer från kunder och dessa kan vara av varierande komplexitet, dvs. kräva olika mycket resurser att åtgärda. CGI är ett globalt IT-företag och jobbar mot många kunder. Supportorganisationen i Malmö klassificerar ett ärende som komplext när ett fördefinierat antal timmar har tidrapporterats. Vi har utvecklat ett tidrapporteringssystem, i syfte att användas vid support av ett SAP-system som förvaltas för ett av företagets kunder. För att kunna undersöka nedanstående forskningsfrågor utnyttjade vi dels systemet för att identifiera intressanta ärenden att analysera, samt personalen på företaget i form av intervjuer. För att kunna analysera ärenden hade vi en insamlingsperiod av data, där personalen fick använda systemet och registrera tider. Detta kompletterades med djupstudie av de mest tidskrävande ärendena för att se om det finns potentiella mönster eller faktorer som kan förklara varför dessa ärenden har blivit större än övriga. En pilotversion av tidrapporteringssystemet är satt i drift sedan april och har använts av ett 30-tal supportingenjörer på CGI.

RQ1: Hur vanlig är förekomsten av komplexa ärenden, det vill säga mer än X timmar hanteringstid?

RQ2: Vad karaktäriserar komplexa ärenden, det vill säga finns det något som utmärker dem?

RQ3: Finns det något samband mellan komplexitet, ledtid och beslutsprocess i de ärenden som undersöks?

RQ4: Hur kan man bättre hantera ärendehanteringskostnader och prediktera vilka ärenden som blir komplexa?

Vi kom fram till att komplexa ärenden inte är vanliga inom företaget. Med hjälp av tidrapporteringssystemet identifierandes de mest tidskrävande ärendena för djupstudien. Arbetet resulterade i en lista med komplexitetsfaktorer som tagits fram från våra intervjuer med personalen, och två listor med faktorer som erhållits från de analyserade ärendena och har kategoriserats i följande grupper: ”**Kodegenskaper**”, ”**Process och verktygsmognad**”, ”**Kunskapsfördelning**”, och ”**Kundens prioriteringar**”. Elva av de 24 faktorer som tagits fram genom analysen rapporterades även av supportpersonalen under intervjuerna. Vi lyckades inte undersöka den tredje forskningsfrågan eftersom nödvändig information inte var tillgängligt för att kunna utföra en noggrann bedömning. Vi har föreslagit metoder för att prediktera vilka ärenden som kommer att bli komplexa men inte haft möjlighet att implementera dessa metoder eftersom det kräver historisk data, som vi inte har haft tillgång till.

Vi har skapat ett beslutsstöd, i form av de komplexitetsfaktorer som tagits fram, som kan användas av företag som har förvaltningsarbeten för att tidigt i arbetet identifiera ärenden som kan vara tidskrävande, och på så vis omfördela tid och resurser på ett gynnsamt sätt. Om ett ärende påvisar flera av de faktorer som presenterats kan det indikera på att ärendet kommer att bli tidskrävande. Detta har främst CGI användning av eftersom studien baseras på de ärenden som hanteras av företaget. Med tidrapporteringssystemet har vi möjliggjort för företaget att ha större insikt i hur mycket tid (och därmed resurser) som läggs ner på varje ärende. I och med att man får en bättre översikt över arbetet kan tidrapporteringssystemet även bidra till att förbättra företagets ärendehanteringskostnader.

Med vår rapport bidrar vi med större insikt inom ärendehantering och komplexa ärenden. Våra resultat kan användas som en bas för vidare forskning, och kan kompletteras med empiriska studier som undersöker huruvida dessa faktorer även kan identifieras i andra supportorganisationer.

Källförteckning

[1] IEEE Recommended Practice for Software Requirement Specifications, New York. *IEEE Computer Society*, 1994.

[2] 8till5.se. http://8till5.se/2014/11/25/konsultbolaget-cgi-gor-affar-med-skanetrafiken, 2015.

[3] Aggarwal, Sabnani, Gopinath. A new file transfer protocol. *AT&T Technical Journal, Volym 64, Upplaga 10, s.2387-2411*, 1985.

[4] Alenljung, Persson. Portraying the practice of decision-making in requirements engineering: a case of large scale bespoke development. *Requirements Engineering Volym 13, Upplaga 4, s.257–279*, 2008.

[5] An, Ruhe. Decision support in requirements engineering. *Engineering and Managing Software Requirements, s.267-286*, 2005.

[6] Anthony. *Planning and control systems: A framework for analysis*. Harvard University, Boston, USA.

[7] Attarha, Modiri. *Sage journals*.

[8] Aurum, Martin. Requirements elicitation using solo brainstorming. *Proceedings of 3rd Australian Conference on Requirements Engineering (ACRE'98), Melbourne, Australia, s.29-37*.

[9] Aurum, Wohlin. The fundamental nature of requirement engineering activities as a decision making process. *Information and Software Technology 45(14), s.945-954*, 2003.

[10] Aurum, Wohlin. *Engineering and Managing Software Requirements*. Springer, 2005.

[11] Babich. *Software Configuration Management: Coordination for Team Productivity*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1986.

[12] Bagnall, Rayward-Smith, Whittley. The next release problem. *Information and Software Technology, Volym 43, upplaga 14, s.883–890*, 2001.

[13] Berander, Andrews. Requirements prioritization. *Aurum A, Wohlin C (eds) Engineering and Managing Software Requirements, Springer Berlin Heidelberg, s.69–94*, 2005.

[14] Berdie, Osaci, Prostean, Cristea. Web programming features on integrated system sap. *Applied Computational Intelligence and Informatics (SACI), 2011 6th IEEE International Symposium, s.227-230*, 2011.

[15] bmc. http://www.bmc.com/it-solutions/remedy-itsm.html, 2015.

[16] Boehm et al. The cocomo 2.0 software cost estimation model. *American Programmer*, 1996.

[17] Carlshamre, Sandahl, Lindvall, Regnell, Natt och Dag. An industrial survey of requirements interdependencies in software product release planning. *Proceedings of the 5th International Symposium on Requirements Engineering, 27-31 August, Toronto, Canada, s.84-91*, 2001.

[18] Cavalcanti, Anselmo,Machado, Vale, Almeida. Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal of Software: Evolution and Process, Volym 26, Upplaga 7, s. 620–653*, 2014.

[19] CGI. http://www.cgi.se/, 2015.

[20] Charles J. Kolodgy. The Case for Building in Web Application Security from the Start. *IDC Analyze the Future*, 2011.

[21] Chen, Univ, Wang, Liu. Improving bug assignment with bug tossing graphs and bug similarities. *Biomedical Engineering and Computer Science (ICBECS), 2010 International Conference on, s.1-5*, 2010.

[22] Cleland-Huang, Settimi, BenKhadra, Berezhanskaya, Christina. Goal-centric traceability for managing non-functional requirements. *Proceedings of the 27th international conference on Software engineering, ACM, New York, NY, USA, ICSE '05, s.362–371*, 2005.

[23] Cohene, Easterbrook. Contextual risk analysis for interview design. *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference, s.95-104*, 2005.

[24] Cristea, Prostean, Muschalik, Tirian. The advantages of using sap netweaver platform to implement a multidisciplinary project. *Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference, s. 383-386*, 2010.

[25] Daniels. *Principles of Configuration Management*. Advanced Applications Consultants, Inc., 1985.

[26] DeGregorio. Enterprise-wide requirements and decision management. *Proceedings of 9th International Symposium of the International Council on System Engineering, Brighton*, 1999.

[27] Entity Framework Tutorials. http://www.entityframeworktutorial.net/images/ef5/databasefirst.png, 2015.

[28] Evans, Park, Alberts. Decisions not requirements: Decision-centered engineering of computer-based systems. *Proceedings of International Conference on Engineering and Computer-Based Systems, s.435-442*, 1997.

[29] Farhoomand. Opening up of the software industry: The case of sap. *Management of eBusiness, WCMeB, Eighth World Congress, s.8*, 2007.

[30] Ferens. The conundrum of software estimation models. *Aerospace and Electronics Conference, 1998. NAECON 1998. Proceedings of the IEEE 1998 National, s. 320-328*, 1998.

[31] FileZilla. https://filezilla-project.org/, 2015.

[32] Foo Plugins, FooTable. http://fooplugins.com/plugins/footable-jquery/, 2015.

[33] Grimstad. Understanding of estimation accuracy in software development projects. *Software Metrics, 2005. 11th IEEE International Symposium, s. 2-42*, 2005.

[34] Grimstad, Jørgensen, Moløkken-Østvold. The clients' impact on estimation accuracy in software development project. *Software Metrics, 2005. 11th IEEE International Symposium, s. 10*, 2005.

[35] Halstead. *Elements of Software Science*. Elsevier North, New York, 1977.

[36] Hickey, Davis. Requirements elicitation and elicitation technique selection:a model for two knowledge-intensive software development processes. *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on, s.*, 2002.

[37] Hogarth. Decision time as a function of task complexity. *Utility, Probability, and Human Decision Making Theory and Decision Library, Volym 11, s.321-338*, 1975.

[38] Hood, Wiedemann, Fichtinger, Pautz. *Change Management interface, Requirements Management (The Interface Between Requirements Development and All Other Systems Engineering Processes)*. Springer, 2008.

[39] Hood, Wiedemann, Fichtinger, Pautz. *Configuration Management interface, Requirements Management (The Interface Between Requirements Development and All Other Systems Engineering Processes)*. Springer, 2008.

[40] Hood, Wiedemann, Fichtinger, Pautz. *Introduction to Requirements Management, Requirements Management (The Interface Between Requirements Development and All Other Systems Engineering Processes)*. Springer, 2008.

[41] Hood, Wiedemann, Fichtinger, Pautz. Risk management interface, requirements management (the interface between requirements development and all other systems engineering processes), 2008.

[42] Hove, Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. *Software Metrics, 2005. 11th IEEE International Symposium*, 2005.

[43] Jiang, Zhang, Zhou, Jiang, Zhang. Responsive web design mode and application. *Advanced Research and Technology in Industry Applications (WARTIA), 2014 IEEE Workshop on, s.1303-1306*, 2014.

[44] Jørgensen, Grimstad. Over-optimism in software development projects: “the winner’s curse". 2005.

[45] Kelly. *Configuration Management: The Changing Image*. McGraw-Hill, New York, 1995.

[46] Klemola, Rilling. A cognitive complexity metric based on category learning. *Cognitive Informatics, 2003. Proceedings. The Second IEEE International Conference on, s.106-112*, 2004.

[47] Kushwaha Misra. Cognitive information complexity measure: A metric based on information contained in the software. *WSEAS Transactions on Computers, Volym 5, Upplaga 3*, 2006.

[48] Leimbach. The sap story: Evolution of sap within the german software industry. *Annuals of the History of Computing, IEEE, Volym 30 , Upplaga 4, s.60-76*, 2008.

[49] Linington. File transfer protocols. *Selected Areas in Communications, IEEE Journal on, Volym 7, Upplaga 7, s.1052-1059*, 1989.

[50] Magazinius, Feldt. Confirming distortional behaviors in software cost estimation practice. *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on, s.411-418*, 2011.

[51] Mahler. *Variants: Keeping things together and telling them apart*. John Wiley & Sons, Inc. New York, NY, USA, 1995.

[52] Masound, Halabi. Asp.net and jsp frameworks in model view controller implementation. *Information and Communication Technologies, ICTTA ’06, Volym 2, s.3593-3598*, 2006.

[53] Mc Cabe. A complexity measure, ieee transactions on software. *Software Engineering, IEEE Transactions on, Volym 2 , Upplaga 4, s. 308-320*, 1976.

[54] Microsoft, ASP.NET MVC 4.0. http://www.asp.net/mvc/mvc4, 2015.

[55] Microsoft, Azure. http://azure.microsoft.com/sv-se/, 2015.

[56] Microsoft, Internet Information Services (IIS). https://www.iis.net/learn, 2015.

[57] Microsoft, MS SQL. https://www.microsoft.com/en-us/server-cloud/products/sql-server/, 2015.

[58] Microsoft SQL Server Management Studio. https://msdn.microsoft.com/en-us/ms174173.aspx, 2015.

[59] Milne, Maiden. Power and politics in requirements engineering: Embracing the dark side. *Requirements Engineering, Volym 17, Upplaga 2, s.83–98*, 2012.

[60] Mohorovicic. Implementing responsive web design for enhanced web presence. *Information & Communication Technology Electronics & Microelectronics (MIPRO), 36th International Convention on, s. 1206-1210*, 2013.

[61] Moløkken, Jørgensen. A review of software surveys on software effort estimation. *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on, s. 223-230*, 2003.

[62] Moløkken-Østvold, Jørgensen. Group processes in software effort estimation. *Empirical Software Engineering, Volym 9, Upplaga 4*.

[63] Moløkken-Østvold, Jørgensen, Gallis, Lien, Tanilkan, Hove. A survey on software estimation in the norwegian industry. *Software Metrics, 2004. Proceedings. 10th International Symposium on, s. 208-219*, 2004.

[64] Nasir. A survey of software estimation techniques and project planning practices. 2006.

[65] Nassif, Capretz, Ho. Software effort estimation in the early stages of the software life cycle using a cascade correlation neural network model. *Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on, s.589-594*, 2012.

[66] Nilsson. Requirements engineering body of knowledge version 1.0. *Requirements Engineering Qualitfication Board*, 2013.

[67] Nuseibeh, Easterbrook. Requirements engineering: A roadmap. *Proceedings of the Conference on the Future of Software Engineering, s. 35-46*, 2000.

[68] Oliver, Serovich, Mason. Constraints and opportunities with interview transcription: Towards reflection in qualitative research. *Soc Forces, Volym 84, Upplaga 2, s.1273–1289*, 2005.

[69] Patton. *Qualitative evaluation and research methods*. Sage Publications: Newbury Park, CA., 1990.

[70] Paul. The need for secure software. *ISC2*.

[71] Paul. *Official (ISC)2 Guide to the CSSLP CBK, Second Edition*. Taylor & Francis Group, 2014.

[72] Ramzan, Ikram. Requirement change management process models: Activities, artifacts and roles. *Multitopic Conference, 2006. INMIC '06. IEEE, s. 219-223*, 2006.

[73] Rapley. *Interviews, In: Seale, C., Gobo, G., Gubrium, J., and Silverman, D. Qualitative research practice*. Sage Publications: Thousand Oaks, CA., 2004.

[74] Regnell, Berntsson Svensson, Wnuk. Can we beat the complexity of very large-scale requirements engineering? *Requirements Engineering: Foundation for Software Quality Lecture Notes in Computer Science, Volym 5025, s.123-128*, 2008.

[75] Regnell, Paech, Aurum, Wohlin, Dutoit, Natt och Dag. Requirements mean decisions! – research issues for understanding and supporting decision making in requirements engineering. *Proceedings of First Swedish Conference on Software Engineering Research and Practice (SERP'01), Ronneby, Sweden, s.49-52*, 2001.

[76] Regnell, Wnuk, Berenbach, Brian. Scaling up requirements engineering – exploring the challenges of increasing size and complexity in market-driven software development. *Requirements Engineering: Foundation for Software Quality Lecture Notes in Computer Science Volym 6606 s.54-59*, 2011.

[77] Roberto Paviotti, Galvão Martins. Mcref: A metric to evaluate complexity of functional requirements, the eighth international conference on software engineering advances. *ICSEA 2013 : The Eighth International Conference on Software Engineering Advances*, 2013.

[78] Ruhe. Software engineering decision support – methodology and applications. *Innovations in Decision Support Systems. Advanced Knowledge International, Tonfoni G, Jain L (Eds.) Adelaide, Australia, s.144-171*, 2003.

[79] SAP. http://www.sap.com/sweden/index.html, 2015.

[80] Sharma, Kushwaha. A complexity measure based on requirement engineering document. *Journal of computer science and engineering, Volym 1, Upplaga 1, s.*, 2010.

[81] Sharma, Kushwaha. Complexity measure based on requirement engineering document and its validation. *Computer and Communication Technology (ICCCT), 2010 International Conference on, s.608-615*, 2010.

[82] Simon. *The new science of management decisions*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1977.

[83] Soni. Defect prevention: Reducing costs and enhancing quality. *Six Sigma*, 2009.

[84] Sutcliffe, Sawyer. Requirements elicitation:towards the unknown unknowns. *Requirements Engineering Conference (RE), 2013 21st IEEE International, s. 92 - 104*, 2013.

[85] Tajfel. *Social identity and intergroup relations*. Cambridge University Press, Cambridge, England.

[86] Taylor, Bogdan. *Introduction to qualitative research methods*. John Wiley & Sons, NY, 1984.

[87] TechnologyUK. http://www.technologyuk.net/computing/project_management/images/change_request_form.gif.

[88] Twitter, Bootstrap. http://getbootstrap.com/, 2015.

[89] Wang. Measurement of the cognitive functional complexity of software. *IEEE International Conference on Cognitive Informatics, s.67-74*, 2003.

[90] Weiss. *CLearning from Strangers*. Free Press, NY., 1994.

[91] Wnuk, Kabbedijk, Brinkkemper, Regnell, Callele. Exploring factors affecting decision outcome and lead-time in large-scale requirements engineering. *Journal of Software: Evolution and Process*, 2013.

Appendices

Kapitel A
Intervjufrågor

Intervjun inleds med följande:

Vi har tecknat ett sekretessavtal. Du behöver inte oroa dig för vad du berättar för oss. Du är garanterad anonymitet. Allt du säger kommer att vara anonymt. Bara vi vet vem som sagt vad, och vi kommer som ett första steg anonymisera informationen. Du gör den här intervjun frivilligt. Du kan hoppa över de frågor du inte vill besvara, och du kan stoppa intervjun när som helst. Den information som samlas in under den här intervjun kommer att lagras säkert. Slutligen undrar vi om det är okej om vi spelar in intervjun? Du kommer att få en utskrift av intervjun innan vi behandlar något som sagts. Du kommer då att ha en chans att ta bort information eller klargöra dig själv.

Generella frågor om intervjuaren

1. Vad för bakgrund har du?
2. Vad har du studerat?
   (a) Vilken utbildning?
   (b) Specialiseringar?
3. Vilka arbetslivserfarenheter har du?
   (a) Vilka roller har du haft på tidigare arbetsplatser?

Frågor om personens roll i företaget

4. Hur länge har du jobbat inom organisationen?
5. Vilken roll har du inom organisationen (utvecklare, testare, projektledare etc)?

6. Vilka branscher har du jobbat inom? Vilka applikationer jobbar du med inom kundens verksamhet?

Frågor om ärendeshantering på generell och individuell nivå

7. Kan du beskriva ärendehanteringsprocessen på CGI, hur man går tillväga och plockar ärenden (generellt inom företaget). Nöjd med processen eller finns det förbättringsmöjligheter?
8. Hur prioriterar man bland sina ärenden?
9. Vad är det för information ni förmedlar till kunden när ni visar att ett ärende är komplext (förutom antalet timmar)?
   (a) Vilka typer av komplexa ärenden brukar gå igenom och vad är det som stoppas (andelen som stoppas i jämförelse mot andelen som går igenom)?
   (b) Finns det något mönster i detta?
10. Vad för typ av information anser du är nödvändig som beslutsstöd vid behandling av ett ärende?
11. Hur ser processen ut när ett ärende har taggats som komplext?
    (a) Finns det någon uppföljning?
    (b) Kan ärendet stoppas när som helst?
12. Hur vanligt är det med komplexa ärenden (uppskattningsvis)?

Personliga uppfattningar

13. Vilka attribut karaktäriserar ett komplext ärende?
14. Vad är det som gör ett ärende komplext, finns det mönster i det (erfarenhetsmässigt)?
15. Vad tycker du om det befintliga ärendehanteringssystemet?
    (a) Finns det funktionalitet som inte används idag men som man kunnat använda för att förbättra ärendeshanteringen?
    (b) Vilka brister finns det och vad är det som är bra?

Intervjun avslutas med att fråga om det finns fler intressanta frågor att diskutera och tips på vem vi ytterligare kan intervjua.

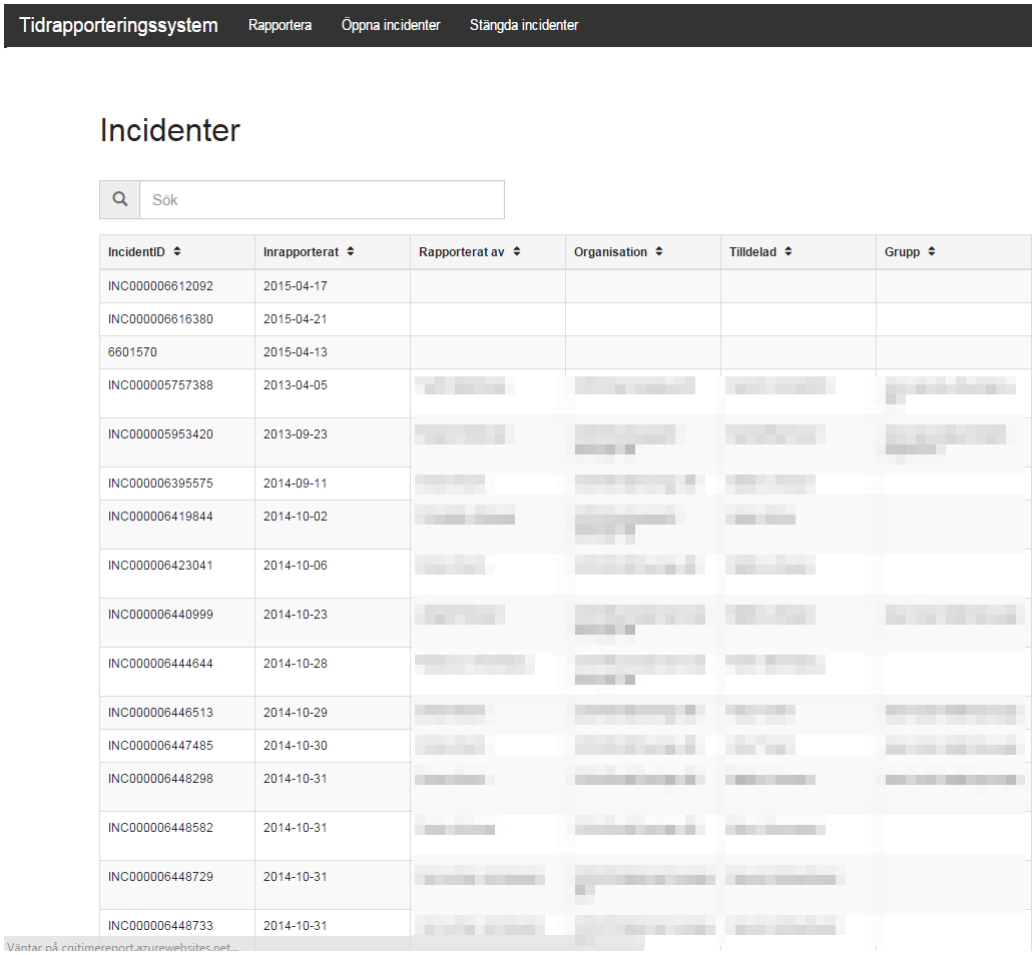
Kapitel B
Webbaserat tidrapporteringssystem

B.1 Introduktion och kravställning

Examensarbetet inleddes med att implementera ett webbaserat tidrapporteringssystem som ska användas av CGI. Systemet ska integreras med ett befintligt ärendehanteringssystem (Remedy) som tillhör en av CGI:s kunder. Övergripande kravställning på systemet är enligt projektbeskrivningen från CGI följande:

En databas ska skapas som innehåller åtminstone tabeller för incidentdata, tid per incident och användardata. Data ska dagligen laddas från kundens ärendehanteringssystem och uppdatera en databas med uppdaterad information över de incidenter som är aktuella. Nya incidenter läggs till, gamla incidenter uppdateras. Logg ska finnas som rapporterar historik och problem vid integration. En webbaserad klient ska skapas med följande grundkrav:

- *Inloggning från både CGI och kundnätverk som måste säkras från obehörigt intrång*
- *Dialog för rapportering av tid per incident*
  - *Incidenter ska kunna sökas upp och tas bort från dialogen*
  - *Rapportering ska kunna ske per vecka*
  - *Lätt manövrering mellan veckor*
  - *Summering per incident skall visas*
- *Incidenter ska kunna manuellt läggas upp om de inte ännu blivit integrerade*
- *En administrationsdialog ska skapas som kan se och hantera*
  - *Kontoadministration*



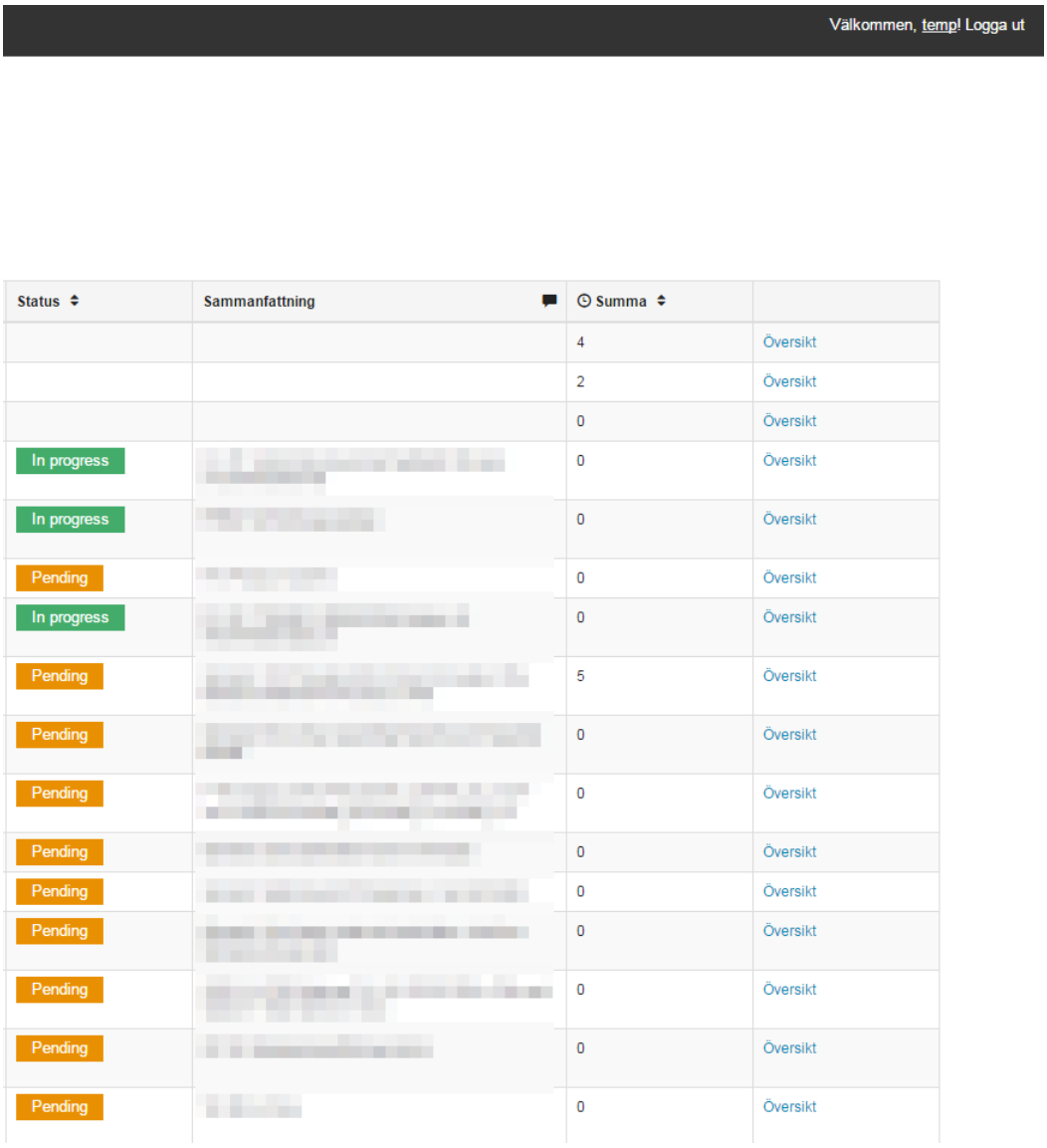
Figur B.1: All-incident sidan, del 1

- *Integrationsloggar*
- *Inloggningsstatistik*

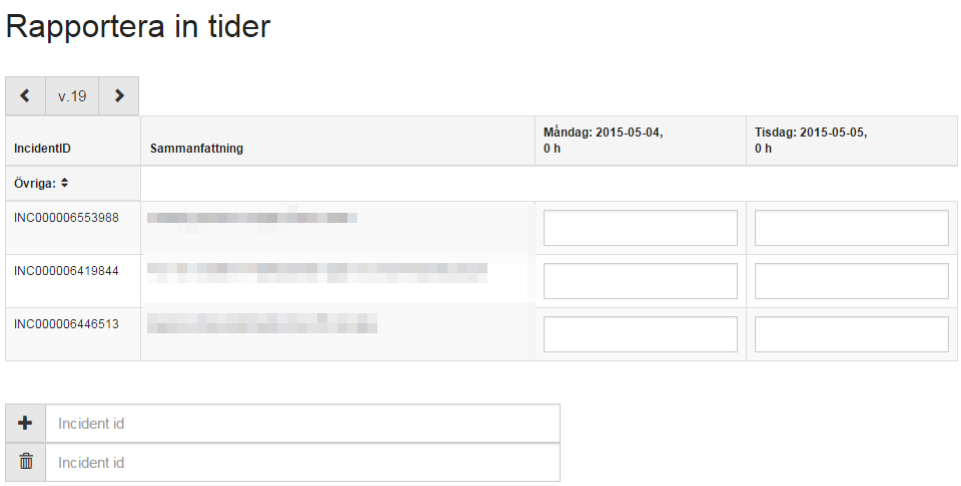
Varför CGI är i behov av ett sådant system beror på att Remedy saknar stöd för tidrapportering av ärenden, och därmed har supportpersonalen på CGI svårt att avgöra när ett ärende når gränsen för att bli ett komplext ärende. Systemet är en förutsättning för att vi skall kunna samla in tidrapporter i samband med tillhörande data för att försöka utreda hur komplexa ärenden kan se ut och hur organisationen väljer att hantera dessa.

Vi avslutar kapitlet med några bilder från systemet vi har implementerat (*tyvärr blev vi tvungna att dela upp bilderna i två halvor eftersom de annars inte fick plats i mallen*). Incidenter-sidan visar alla tillgängliga incidenter som finns i databasen.

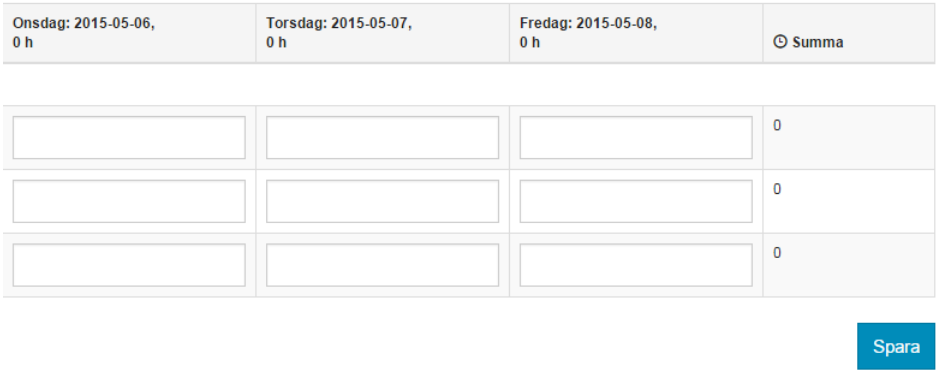
Vill man rapportera på en viss incident går man in på rapporteringssidan, och fyller i antalet timmar man har jobbat. Incidenten kan vara tilldelat gruppen eller personen själv. Man har även möjlighet att lägga till en incident om denna inte har hunnit integreras i systemet eller ta bort en incident om man är färdig med den (se figur 4.3).



Figur B.2: All-incident sidan, del 2

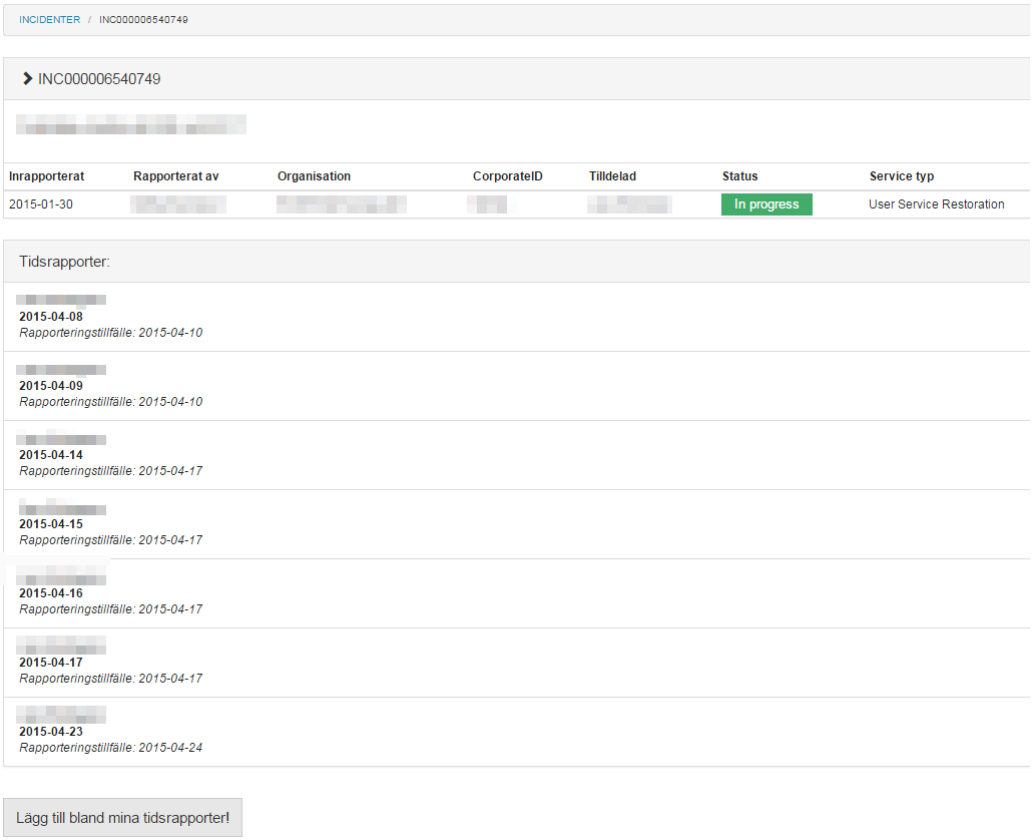


Figur B.3: Rapporteringsdialogen med veckovis manövrering, de1 1

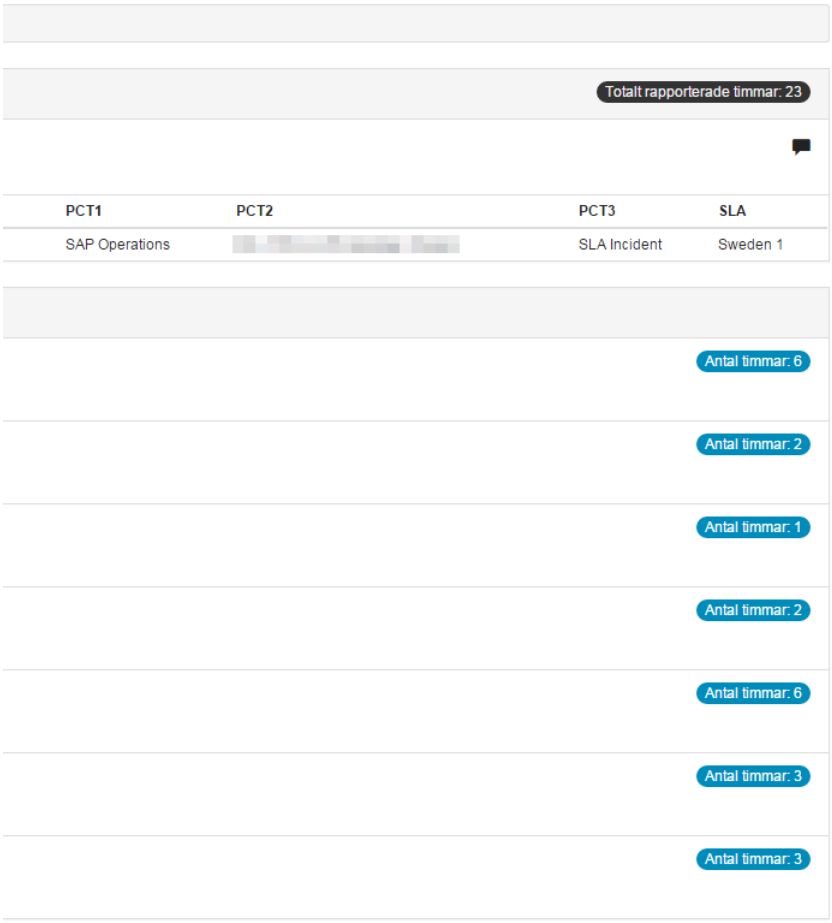


Figur B.4: Rapporteringsdialogen med veckovis manövrering, de1 2

Vill man få en översikt över en incident trycker på man översiktsknappen på incident-sidan och då hamnar på följande sida:

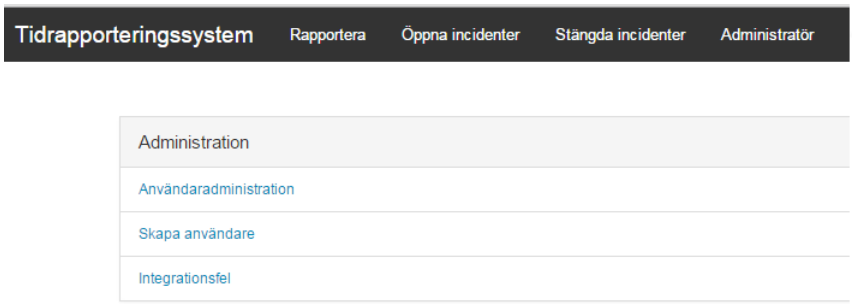


Figur B.5: Översiktssidan för en incident där all tillhörande data visas, del 1



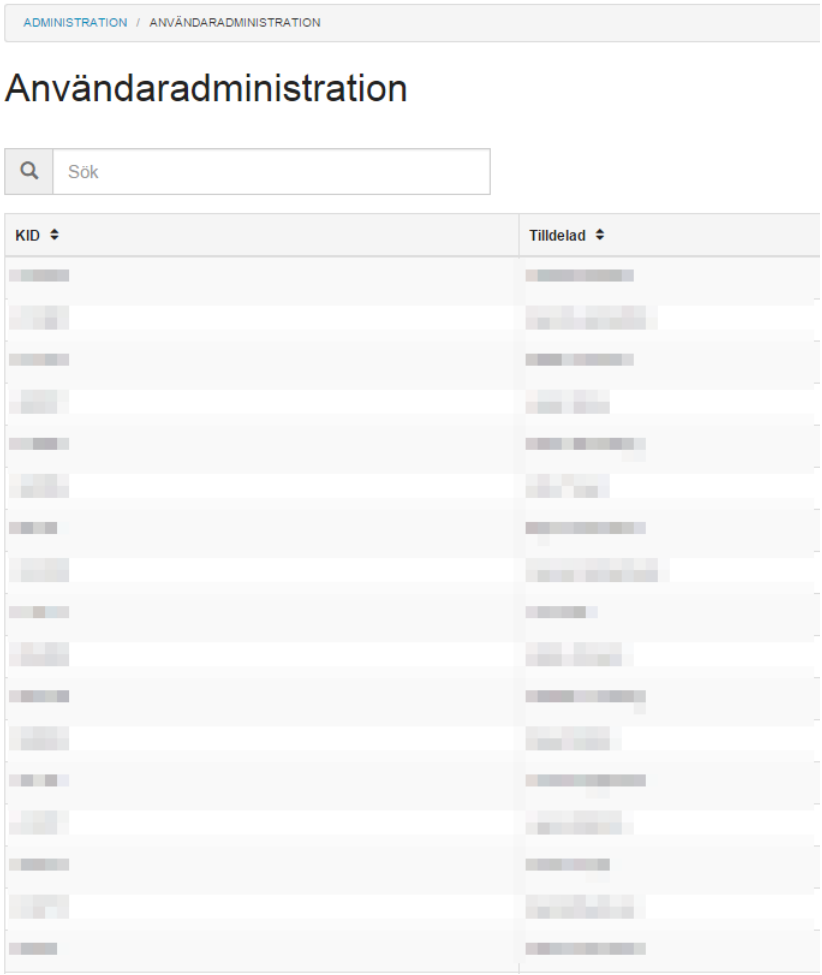
Figur B.6: Översiktssidan för en incident, del 2

När man loggar in som administratör möts man av följande sida:

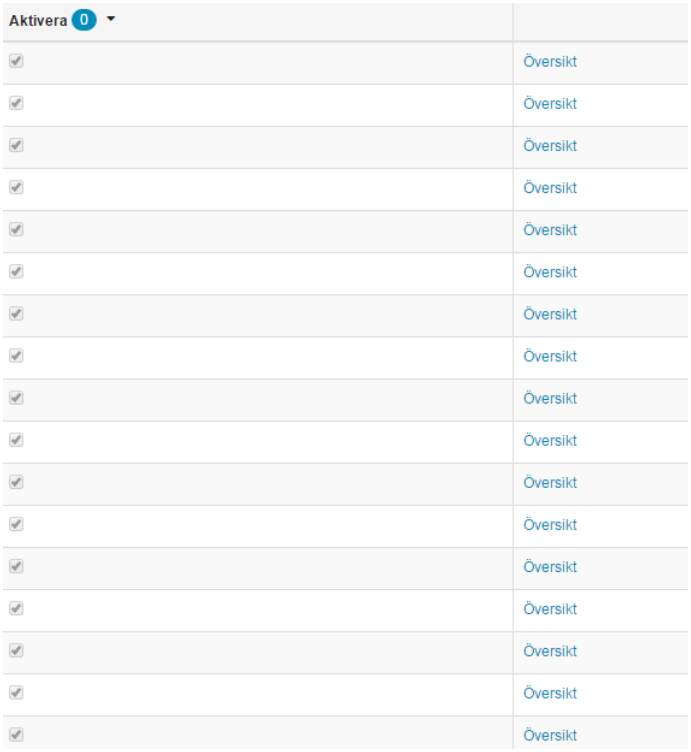


Figur B.7: Översiktssidan för admin-kontot

För att lista alla användare trycker man på ”Användaradministration” och där man kan söka efter en viss användare.

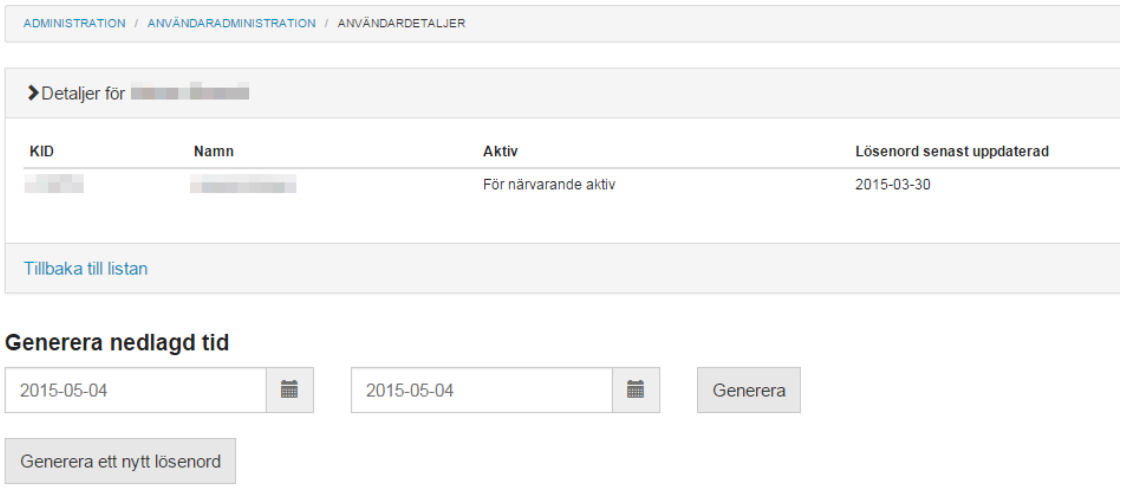


Figur B.8: Användaradministration, del 1

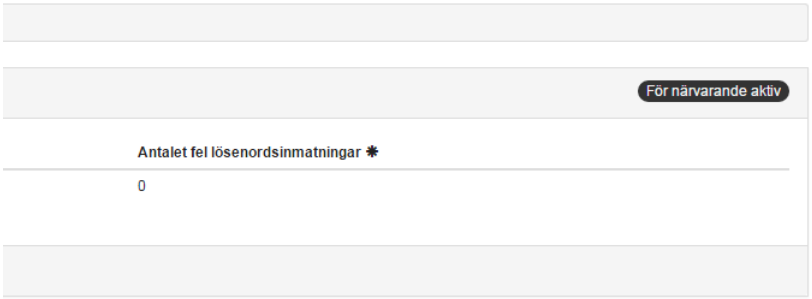


Figur B.9: Användaradministration, del 2

För att få en mer detaljerad översikt över en användare trycker man på ”Översikt” och där kan man till exempel generera ett nytt lösenord för användaren eller se hur mycket denna har rapporterad under en viss period.

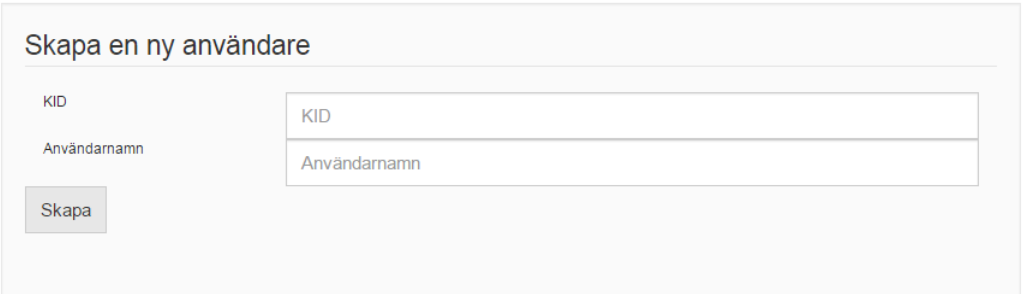


Figur B.10: Översiktssidan för en användare, del 1



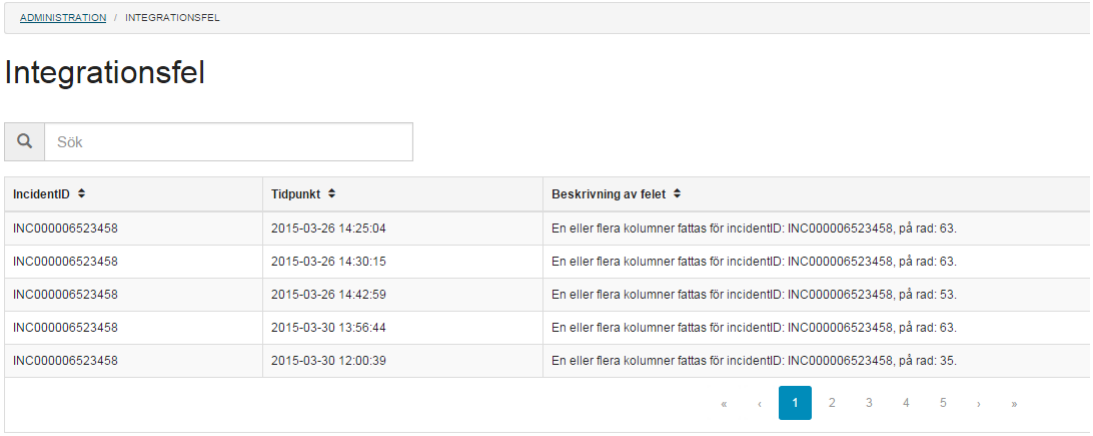
Figur B.11: Översiktssidan för en användare, del 2

En administratör kan lägga till nya användare i systemet genom att specifiera KID (som fungerar som ID för inloggning) och namn.

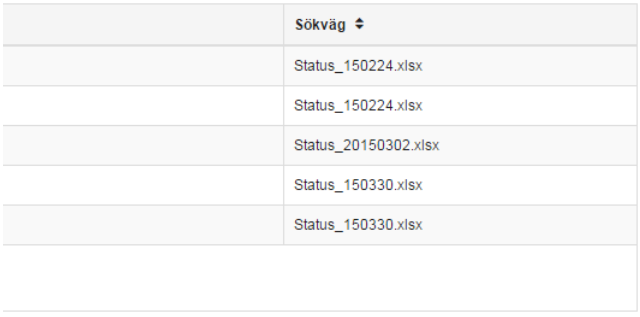


Figur B.12: Dialog för att skapa en ny användaren, kräver administrationsrättigheter

En administratör kan även se integrationshistoriken eftersom detta identifieras och lagras av systemet.



Figur B.13: Integrationshistorik av den automatiserade integrationen (kräver administrationsrättigheter för att se), del 1



Figur B.14: Integrationshistorik av den automatiserade integrationen (kräver administrationsrättigheter för att se), del 2

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2015-08-28

EXAMENSARBETE Vad karaktäriserar komplexa ärenden i mjukvaruprojekt?
STUDENTER Mehmet Fatih Cicek, Soheil Afghani Khorasgani
HANDLEDARE Markus Borg (LTH), Anders Malmberg (CGI)
EXAMINATOR Per Runeson

En studie kring komplexa ärenden i mjukvaruprojekt

POPULÄRVETENSKAPLIG SAMMANFATTNING **Mehmet Fatih Cicek, Soheil Afghani Khorasgani**

Studier visar att en stor andel av mjukvaruprojekt skjuter över den planerade budgeten och missar uppsatta deadlines. Vårt examensarbete fokuserar på att ta fram kriterier för komplexa ärenden, i syfte att uppnå bättre kontroll över kostnader.

I de avancerade mjukvarusystem som utvecklas idag krävs det en hel del processer och arbetsmetodik för att säkerställa att produkten håller en bra kvalité. Företag som är marknadsdrivna måste ta snabba beslut för att kunna konkurrera på marknaden. Svårigheten uppstår i att organisationen oftast är tvungen att hantera väldigt många ändringsförfrågningar som kommer från kunder och dessa kan vara av varierande komplexitet, dvs. kräva olika mycket resurser att åtgärda. Om man inte reagerar tillräckligt snabbt kan det ge upphov till allvarliga konsekvenser för företag och dess kundrelationer. CGI är bland världens största IT-företag och jobbar mot många kunder. Supportorganisationen i Malmö klassificerar ett ärende som komplext när ett fördefinierat antal timmar har tidrapporterats. Genom att implementera ett webbaserat tidrapporteringssystem identifierades de mest tidskrävande ärendena, vilka därefter användes som underlag för djupstudier. Tillsammans med intervjuer av supportpersonal på företaget identifierade vi potentiella faktorer som kan förklara varför komplexa ärenden blir större och mer tidskrävande än övriga incidenter.

Arbetet resulterade i bland annat två listor med potentiella komplexitetsfaktorer som tagits fram från intervjuerna och de analyserade ärendena, och har kategoriserats i följande grupper: **”Kodegenskaper”**, **”Process och verktygsmognad”**, **”Kunskapsfördelning”**, och **”Kundens prioriteringar”**. Elva av de 24 faktorer som tagits fram genom analysen rapporterades även av supportpersonalen under intervjuerna. Dessa faktorer kan användas som beslutsstöd av företag som har förvaltningsarbeten för att tidigt i arbetet identifiera ärenden som kan vara tidskrävande, och på så sätt omfördela tid och resurser på ett gynnsamt sätt. Om ett ärende påvisar flera av de faktorer som presenterats kan det indikera på att ärendet kommer att bli tidskrävande. Detta har främst CGI användning av eftersom studien baseras på de ärenden som hanteras av företaget. Några exempel på de faktorer som vi tagit fram är:

- Felet går inte att återskapa.
- Arbetet påverkade gammal funktionalitet negativt.
- En ändring i en modul leder till att andra moduler måste modifieras.

Ytterligare har vi möjliggjort för företaget att ha större insikt i hur mycket tid (och därmed resurser) som läggs ned på varje ärende genom att utveckla ett tidrapporteringsystem. Systemet kan dessutom användas för att förbättra företagets ärendehanteringskostnader eftersom det ger en tydligare översikt över arbetet, vilket var en saknad funktionalitet, och vi har även föreslagit metoder för att prediktera vilka ärenden som kommer att bli komplexa. En pilotversion av tidrapporteringssystemet är satt i drift sedan april och har använts av ett 30-tal supportingenjörer på CGI.

Med vår rapport bidrar vi med större insikt inom ärendehantering och komplexa ärenden. Våra resultat kan användas som en bas för vidare forskning, och kan kompletteras med empiriska studier som undersöker huruvida dessa faktorer även kan identifieras i andra supportorganisationer.