Projekt - Skatteverket

Automatisk programvaruuppdatering

Slutrapport, *Programvaruprojekt*, 15 HP, VT-2022

Författare: Emil Nettby, Bo Splitgren

Handledare: Maritha Dahlin

Sammanfattning

I en föränderlig värld där programvaror behöver vara tillgängliga och säkra ifrån dolda problem samtidigt som berörda programvaror måste komma snabbt ut bland användarna. Krävs det tekniker för att kunna förändra eller uppdatera berörd programvara hos användaren. I denna rapport beskrivs en möjlig lösning för en automatiserad programvaruuppdaterings funktion, utan någon användare inblandad och det flöde som sker vid uppdatering behandlas som en atomicity, d.v.s. uppdateringens samtliga ingående moment måste genomföras korrekt annars återställs programmets status till samma som den hade innan uppdateringen påbörjades. Berört arbete är framtagen i samarbete med Skatteverket som en examensuppgift inom inriktningen Programvaruutveckling för programmet Systemvetenskap.

Nyckelord: Automatisk Uppdatering, Atomicity, Prototyp.

Abstract
In a changing world where software needs to be accessible and secure from hidden problems while affected software needs to come out quickly among users. Technicians are required to be able to change or update the relevant software of the user. This report describes a possible solution for the function of an automated software update, without any user intervention and the flow that occurs during the update is treated as an atomicity, i.e. all the steps of the update must be carried out correctly. Otherwise the status of the program will be restored to the same as it had before the update began. The relevant work has been developed in collaboration with the Swedish Tax Agency as a degree assignment within the specialization Software Development for the Programme in Systems Science.

Keywords: Automatic software update, Atomicity, prototyping.

Förord

Vi tackar alla bidragande parterna för hjälpen i framtagandet av rapport både vad gäller konstruktiv kritik och tillrop när motgångarna har känts övermäktigt. Stort tack tillägnas vår uppdragsgivare Skatteverket samt våra examinatorer.

Innehållsförteckning

1. Introduktion

I det här projektet kommer vi arbeta åt Skatteverket för att utveckla en prototyp som påvisar hur en automatisk program uppdateringsfunktion kan utformas. Tanken med prototypen är att den ska göra det möjligt för programmet som behöver uppdateras att själv söka efter tillgängliga uppdateringar och uppdatera om det är möjligt.

1.1. Uppdragsgivare

Skatteverket är en fristående myndighet som lyder under regeringen. De har i uppgift att hantera folkbokföring, äktenskapsregister, fastighetstaxering samt samla in skatter. Dessa register hjälper skatteverket att utfärda identitets instrument till personer som begär och kan verifiera berörda uppgifter i folkbokföringen. Skatteverket har även andra viktiga uppdrag såsom att utreda skattebrott samt agera borgenär åt staten (Skatteverket, 2022).

Skatteverket behöver en stor organisation för att upprätthålla riktigheten i lagrade register men även underlätta att skatteinsamlingen sker på ett säkert sätt. Berörd avdelning har både kravet att lagra och underlätta användandet av informationen internt samt till valda delar externt. För att utföra detta har Skatteverket en större egen mjukvaruutvecklings avdelning.

1.2. Praktiskt uppdrag

Uppdraget är alldagligt hållen och beskrivs enligt följande citat ur uppdragsbeskrivning för automatisk uppdateringar av applikationer är:

> *Användaren ska om möjligt enbart få information om att applikationen uppdateras och ska inte behöva installera något mer än så. Eftersom din tjänst har stora binärer och liknande så måste du också kunna hantera transporten av dessa till miljontals användare inom ett par dygn från att du släpper en uppdatering.*

Såsom beskrivningen visar ligger tyngdpunkten mot automatik med så liten inblandning som möjligt från användaren av applikationen. Till detta så kan applikationen innehålla en eller flera stora binärer. Dessa binärer kan vara vitala för applikationens funktion och måste kunna spridas med största möjliga hastighet ut till samtliga användare, utan att strypa källan av utsända uppdateringar.

Med dessa stora målande beskrivningar behöver problemet specificeras mer ingående och till vår hjälp fick vi följande hållpunkter som styrande delar i realiseringen av prototypen.

- Applikation som använder uppdateringstjänsten ska vara av desktop typ. Med andra ord program som körs på en lokal dator i sin helhet
- Kunna uppdatera hela eller delar av applikationen.
- Säkerställa att uppdateringen inte förstör applikationen.
- Skapa så stor dela av uppdateringsfunktionen som möjligt via egen kod.

Hållpunkterna ger en mer handfast utgångspunkt på hur prototypen bör utformas för att uppfylla funktion önskemålet. Gällande automatiserad uppdateringsprocess med stora binärer och snabb spridning.

1.3. Avgränsningar

Berörd projektbeskrivning belyser två stora delar som interagerar med varandra, en del för spridningen av de större uppdaterings binärer och en del som hanterar appliceringen av dessa stora uppdaterings binärer i berörd applikation. Att realisera båda dessa invecklade delarna under denna begränsade tidsperiod som är till förfogande gör att vi i samförstånd med vår uppdragsgivare. Koncentrerar arbetet på den senare delen d.v.s. uppdaterings prototypen och använda en färdig lösning för spridnings funktionen.

Till detta så kommer uppdaterings prototypen interagera med användaren via ett enkelt gränssnitt av något slag som för typen av valt programspråk ger användaren den information eller valmöjlighet som behövs för att agenten ska kunna utföra en applicering innan berörd applikation aktiveras.

Under arbetes gång hoppas och förväntas vi att det finns en dialog mellan oss som utförare och uppdragsgivaren, för att med återkoppling och samtal runt andra detaljer identifiera uppslag och eventuella lösningar på delproblem som driver projektet i önskad riktning till ett slutresultat. Våra förväntningar inför projektet är att vi kommer fram till ett hållbart koncept med ett möjligt förslag till prototypen. Förhoppningarna är att vi båda tar vårt ansvar och arbetar tillsammans för att samla material och information för att kunna applicera det i ett programmeringsspråk som vi utförare behärskar. Slutligen hoppas vi att våran examinator av projektet anser att arbetet är väl genomfört samt vår uppdragsgivare finner våran realisering användbar i berörd verksamhet eller hos tredje part.

1.4. Författarens eller delförfattarnas bidrag

Vi som är författarna bakom denna avhandling har lagt upp vårt arbete på så sätt att vi söker efter användbart material i form av forskningsartiklar eller dylikt och sedan delar vi upp punkterna mellan oss i kapitlet så vi är tidseffektiva i vår arbetsprocess.

2. Problematisering och forskningsanknytning

Går det att undvika att uppfinna hjulet på nytt är det såklart att föredra men frågan är om det finns en färdig lösning som går att applicera i våran prototyp eller om den behöver anpassas för att fungera. Ett exempel på ett program som använder en automatisk uppdateringstjänst är Mozilla firefox som fungerar på det sättet att när programmet startas upp börjar den genast leta efter nya uppdateringar som kan finnas tillgängliga. Uppgiften är alltså ett exempel på hur en lösning ska fungera för att skicka ut uppdateringar till en applikation per automatik som sedan appen kan uppdatera sin mjukvara med. Syftet med detta är att helt enkelt låta programvaran själv söka efter tillgängliga uppdateringar för att på så sätt hela tiden vara försedd med den senaste versionen av mjukvaran.

2.1. Problematisering

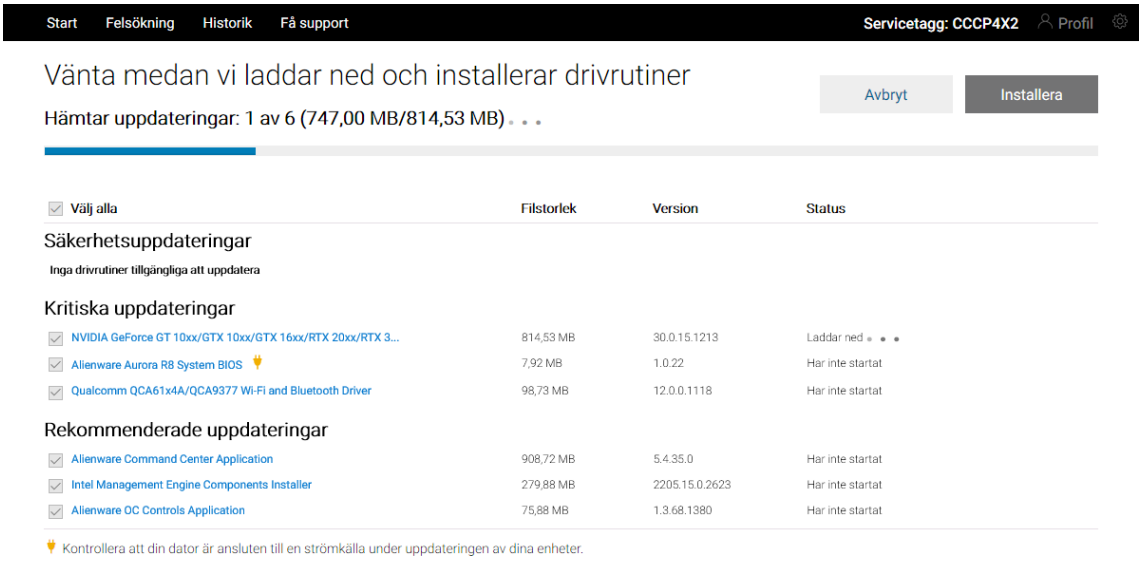
Vid utvecklingen av ett program finns det ofta ett problem som programmerarna måste sätta sig ner tillsammans och titta på för att komma över fund med vad som kan vara lösningen till problemet. I detta projekt kommer den största utmaningen/problematiseringen att vara hur vi ska få förbindelsen att fungera mellan våran uppdateringstjänst och applikationen som ska uppdateras. En annan stor utmaning ,som vi också högst troligt kommer stöta på i vårt arbete, är hur applikationen ska uppdateras på datorer som varit offline under en längre tidsperiod. Ett exempel på ett annat fall med samma problematisering är Updated as a service, UaaS, (Liu et al 2015, 483) artikel om hur VMs(Virtual Machine) ska hållas konstant uppdaterade även när de är offline. Deras lösning på problemet var att ägaren för respektive VM måste manuellt uppdatera programvaran periodvis med förutsättningarna att tidsintervallet blir så kort som det bara går (UaaS, 2015). Skulle en användare önska uppdatera sina VMs under tiden de är offline krävs det en betalningssumma för att starta upp dem i offline-läge och sedan stänga av dem igen. Denna metod är inte hållbar eller optimal sett ur ett ekonomiskt perspektiv, den är också tidskrävande vilket företag inte har råd med. Det finns ett flertal varianter av lösningar för problemet där principen är att de installerar uppdateringar genom att antingen ändra på det nedre lagret i molntjänsten (IaaS) eller att helt enkelt byta ut filerna. Oavsett val av tillvägagångssätt finns det inga garantier för att uppdateringen kan implementeras under kontrollerade omständigheter. Detta beror på att den föredrar Linux framför windows vilket gör det svårt för windows användare att implementera den på sina enheter samt att VMs har olika säkerhetskrav som försvårar uppdateringsprocessen samtidigt som packages inom VMs har varierande uppdaterings krav.

Det finns självklart fler än ett sätt att utveckla en uppdateringstjänst på och ett annat sådant sätt är genom outsourcing. Fördelen med att använda sig av outsourcing vid utveckling av en tjänst är att det går snabbt att genomföra då alla utvecklingsprocesser sker samtidigt. Detta är särskilt vanligt vid utvecklingen av ett mjukvarubaserat komponentsystem där det krävs mycket tid och manskap för att skapa ett större system. Den stora nackdelen med outsourcing är att kostnaderna vanligtvis brukar bli betydligt högre än om företaget hade valt att hålla utvecklingen inom organisationen. För att underlätta utvecklingsarbetet använde sig (Nguyen 2008, 444) av ett formalism vilket medförde en modell av processen i mjukvaruutveckling i en komponentbaserat miljö. Denna formalism kan användas för att få till en sundhet i arbetsprocessen vid utvecklandet av en sådan komponentbaserad mjukvaruuppdatering. Modellen bidrar också till att en implementation av ett uppdateringsverktyg för främst komponentbaserade system men kan även användas till andra system. Verktyget gör det möjligt för utvecklare att enklare följa händelseutvecklingen och uppdateringsprocessen för att på så sätt försäkra sig om att utvecklingsprocessen går rätt till.

2.2. Liknande system / utredningar / uppdrag

I sökandet efter likvärdiga lösningar inom det aktuella området med framställning av mjukvaruuppdaterings program, framkom hos flera forum för program framställningen att det inte behövs en speciallösning utan att använda någon av de öppna källkods lösningarna som redan finns.

Det finns flertalet lösningar som liknar den prototyp som vi ska ta fram där vissa har mer gemensamt än andra. Det finns en variant av en mjukvaruuppdaterings process från Dells Alienware som fungerar på liknande sätt rent utseendemässigt som vår lösning ska se ut och där är det främst själva uppdateringsprocessen som vi är särskilt intresserade av. Här nedan kommer en bild på hur det kan se ut när drivrutiner installeras bland flera:



2.3. Relaterad forskning / litteratur

Inom berört problemområde finns flertalet undersökningar som behandlar mjukvaruuppdatering och behovet av att kunna genomföra uppdateringar av mjukvaran. Tyngdpunkten med en uppdatering är öka kvaliten i mjukvaran och addera eller förbättra funktioner. Till det så finns möjligheten att upprätthålla och minska säkerhetsrisker. De senare har Liu och Richardson (2000) tagit fasta på gällande automatiserade uppdateringar av tjänster inom operativsystemet Linux och via skriptspråket TestTalk, vid rapportens skapande var kunskapsnivån hos användarna låg gällande uppdatering och framförallt installation. En normal installation i Linux vid denna tidpunkt var ofta av typen kompilera programvarans källkod, Liu och Richardson 2000 påtalar att detta arbetssätt försvårar underhåll och uppdateringar av Linux, och välkomnar de pakethanterare programtjänster som skapas. Detta tillsammans med Liu och Richardson (2000) framtagna lösning underlättar automatiserad uppdatering och underhållsarbete i operativsystemet Linux. Som motpunkt till den beskrivna modellen av uppdaterings problematiken framför Orso el. al (2002) en gällande uppdatering av kritiska program eller tjänster med kravet på att dessa under inga omständigheter får stoppas. Orso el.al. (2002) påvisar en teknik som har förmågan att uppdatera ett programs inre delar under drift utan att förstöra det tillstånd som den förändrande delen befinner sig inom, Orso et. al (2002) benämner detta som dynamisk uppdatering. Vidare beskriver Orso et. al. (2002) hur de löser detta genom

kodanalys och skapa inkapslingar som en väg att förändra logiken utan att förändra logikens tillstånd.

I vår problembeskrivning ligger inte tyngdpunkten mot ett operativsystem i stort utan mer mot en individuell applikation, även om deras tillvägagångssätt att beskriva om det finns uppdateringar är applicerbar i vårt fall. Detta genom att Liu och Richardson (2000) använder en manifest-fil skapad i XML som innehåller både tjänstens namn, versionsnummer, samt hur kritisk berörd uppdatering är baserad på version av operativsystemet. En större skillnad återfinns hos Orso et. al (2002), via deras lösning gällande förändra logiken under användande, detta försvårar realiseringen i detta stadium av författarnas kunskapsnivå men tillför förståelse av behovet med uppdateringar.

2.4. Implikationer för uppdraget

Planerade källor att använda för att utveckla tjänsten	Källornas roll i projektet	Hur källorna bidrar med information till tjänsten	Användning av litteraturen för utvärdering av arbetet	Relevant kunskap för uppdragsgivaren
UaaS - Software Update as a Service for the IaaS Cloud.	Syftet är att skapa ideér och tankar hos oss för hur vi ska lösa tjänstens utskick av uppdateringar.	Tanken är att vi inte ska behöva uppfinna hjulet på nytt utan ta inspiration av befintliga verk.	Eftersom källan är en bidragande faktor till lösningen på uppdateringstjänsten kommer den ta plats i utvärderingen.	Lösningarna som vi tagit fram kan vara något han inte tänkt på.
IEEE - Component-based Software Update Process	Ett ytterligare exempel på ett tillvägagångssätt för att skapa en uppdateringstjänst.	Meningen är att ge oss flera olika perspektiv på hur vi ska utveckla vår tjänst.		Lösningen kommer innefatta data som är av intresse för uppdragsgivaren.
Automatic updating method based on Maven	En variant på hur en uppdaterings metod kan se ut.	Artikeln presenterar ett automatiskt uppdateringsverktyg som kommer vara användbar vid utvecklingen av tjänsten.		Metoden är ett bra sätt att skapa en automatisk uppdatering stjänst på vilket är gynnsamt för alla involverade i projektet.

Automated security checking and patching using TestTalk	Förklarar hur ett tillvägagångssätt av skapandet av ett uppdateringsprogram kan se ut.	Innehåller en manifest-fil från XML som innehåller viktig info om uppsättningen av tjänsten ska se ut.		Finns en möjlighet att information som nämns i artikeln är användbart för uppdragsgivaren i sin användning av vår prototyp.

2.5. Förväntade kunskapsbidrag

Vår problemklass har vi definierat som en variant av atomicity som kan beskrivas som en funktion av databassystem där haken är att en transaktion måste vara allt eller inget. Detta menas att när en funktion verkställs ska allting som funktionen är menat att utföra antingen ske eller så händer ingenting alls. Det finns inget mellanting i atomicity utan det är ett så kallat allt-eller-inget förhållande precis som i vår uppdateringstjänst där en uppdatering ska antingen genomföras eller inte alls. Den här prototypen av en uppdateringstjänst kommer vara ett koncepttest (Proof of concept) som ska demonstrera hur den kan användas på bästa sätt. Prototypen kommer inte vara av försäljnings kvalité när deadline infaller men den kommer däremot vara leveransklar till uppdragsgivaren för vidare validering och testning gentemot deras system som finns hos Skatteverket. Den funktionalitet som kommer finnas med uppfyller de krav som ställts men den kommer förhoppningsvis bidra med en hållbar lösning till användningsområdet.

Eftersom vi ska utveckla en modell av typen mjukvaruuppdaterings program kan man klassa den som en variant av en innovativ programvara. Denna programvara ska bidra till att alla som är beroende av skatteverkets e-tjänster ska kunna ta del av denna modell och få sin applikation uppdaterad när som helst under dygnet oavsett var du befinner dig. Målet med denna modell är att ge skatteverket en hållbar och tillförlitlig modell av en uppdateringstjänst som ska bidra till att applikationen i fråga alltid har den senaste versionen när en uppdatering körs oavsett nuvarande version och enhet som används. Detta ska också på sikt bidra till Skatteverkets fortsatta framgång inom mjukvaruuppdatering.

I texten “Insufficient theoretical contribution” pratar Ågerfalk (2014) om relationen mellan teori och empiriska fakta som observerbar fakta till teoretiska påståenden. Han tar även upp bland annat att teori och empirisk fakta går hand i hand där teori bidrar till att klargöra tankar och fenomen samt förbereda oss för vad våra förutsägelser kan leda till. Empirisk data däremot är ett krav för att validera och utveckla den teori som valts ut och kunna utvärdera design. Det finns likheter mellan vårt projekt och det Ågerfalk (2014) säger i sin artikel, det är dels upplägget och planeringen av projektet dels det praktiska utförandet alltså när teorin appliceras på utvecklandet av uppdateringstjänsten.

I Gregor och Hevners verk från 2013 förklarar de relaterade problem med kunskapsbidrag i designforskning inom IT-branschen. Syftet med artikeln är att hjälpa forskare att omfamna och producera kunskap under ett forskningsprojekt eller liknande. Det ska också underlätta för läsare och kritiker att enklare urskilja vad forskningsartiklar faktiskt bidrar med i ett forskningssammanhang. Detta har vi dragit nytta av i vårt projekt när vi samlar kunskap från olika källor som sedan sammanställs i vår rapport för att dels beskriva för läsaren vad det är vårt uppdrag handlar om och dels utöka vår egen kunskapsbas om uppdateringstjänster inom IT-branschen. Baskerville (2011) identifierar nio guidelines som kan användas för att reflektera över design och kunskapsutveckling, vidare tar Baskerville (2011) upp bland annat att man ska nå det primära kunskapsmålet genom att producera kunskap inom forsknings domänen, vilket kan förklaras som att forskare använder sig av kortfattad och tydlig kunskap såväl generellt som inom designprocessen för att förklara design domänen.

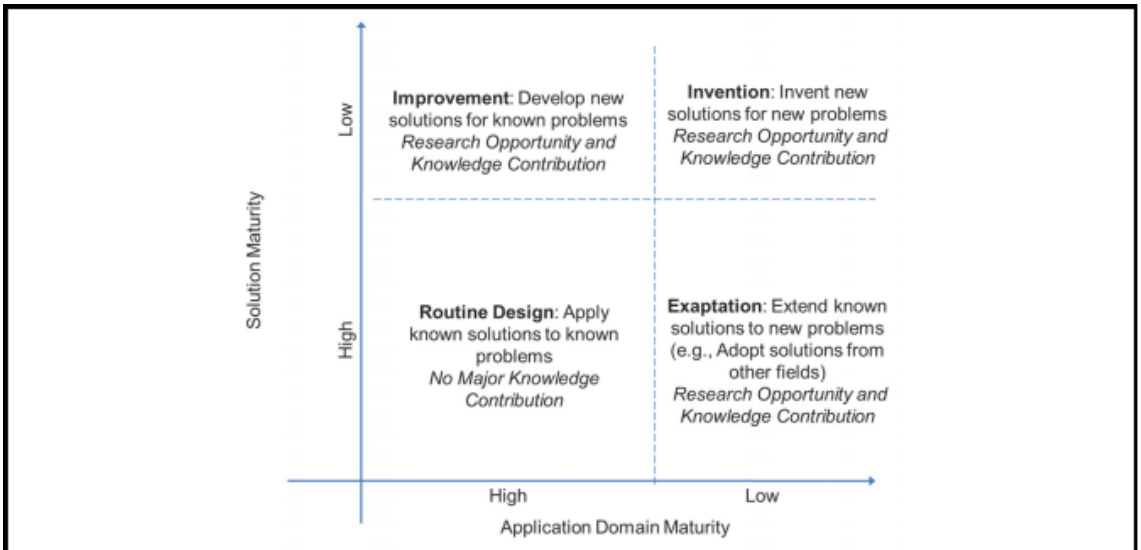


Figure 3. DSR Knowledge Contribution Framework

Figuren ovan representerar kunskapsbidrag av forskningsprojekt och DSR(Design Science Research). På x-axeln syns problemets två faser från high to low och på y-axeln syns den nuvarande fasen som kan agera som en utgångspunkt för lösningar till forskningsfrågor vilket också går från high to low.

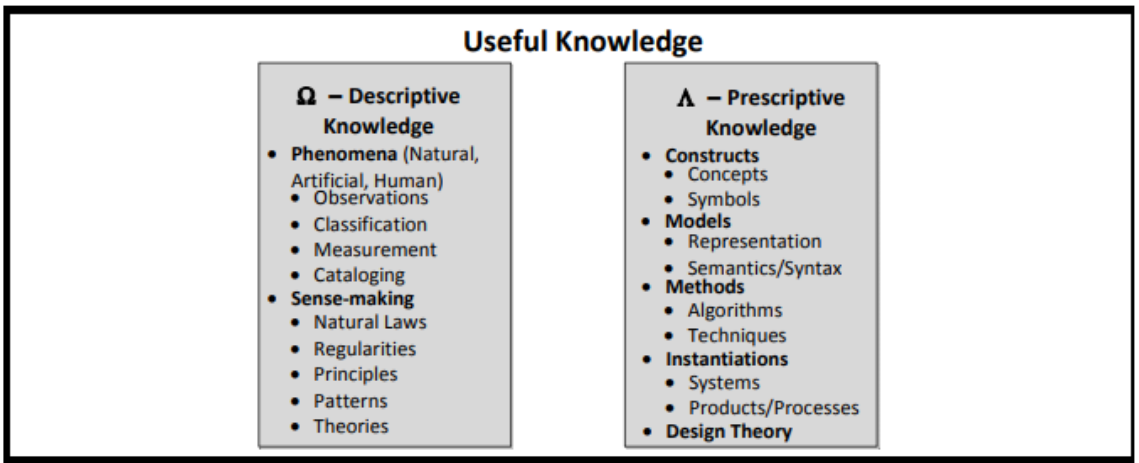


Figure 1. The Design Science Research Knowledge Base

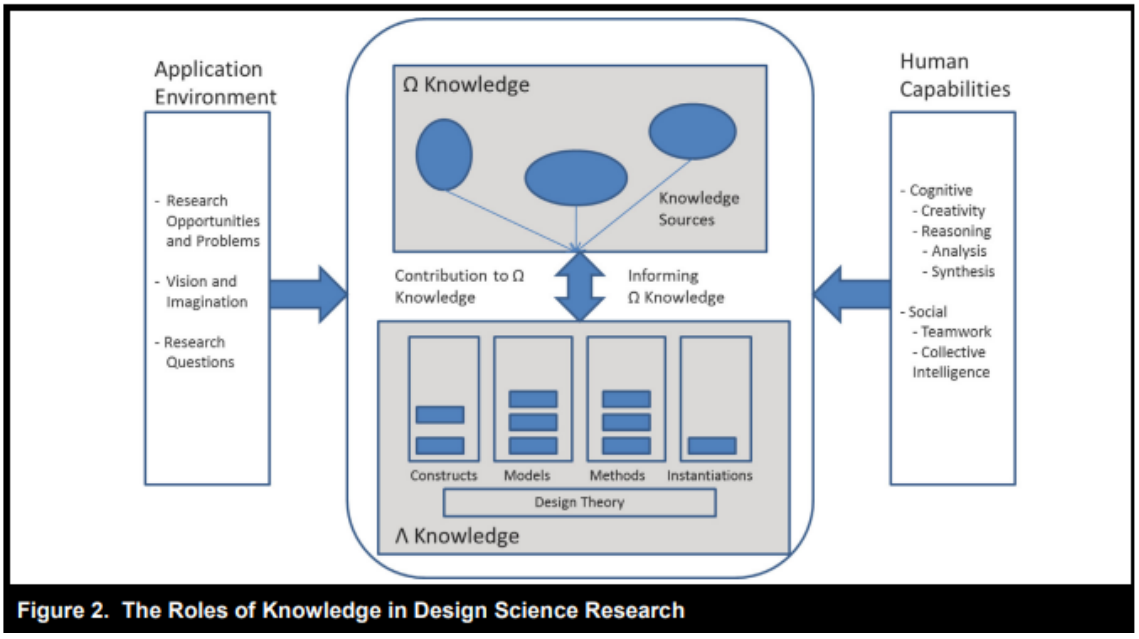


Figure 2. The Roles of Knowledge in Design Science Research

Modellen ovan är ett exempel på att hur det går till när forskning bedrivs ur ett vetenskapligt perspektiv. All forskning som någonsin bedrivits är alltid baserat på en redan befintlig uppfinning som är utgångspunkten för den nya forskningen (Gregor och Hevner, 2013).

3. Forsknings- och utvecklingsprocessen

3.1. Projektöversikt

När projektarbetet började hade vi en plan att lägga upp alla uppgifter som behövde göras i trello för att få en bättre struktur på vår arbetsprocess och tydligare se vad som behövde göras just nu. Tanken med detta var god och fungerade fint till en början men ju mer arbetet flöt på desto mer började vi arbeta på varsitt håll vilket resulterade i att ingen längre uppdaterade innehållet i trello. Vi gjorde istället så att vi delade upp kapitlena i mellan oss och fokuserade på de punkter som vi blev tilldelade att fylla på med information.

Planeringen överlag hade kunnat vara bättre i arbetet men tidsplanen har framförallt varit svår att förhålla sig till med tanke på att det är flera kapitel som ska fyllas med information och det blir lätt att man fastnar på vissa punkter i ett kapitel som tar mer tid än nödvändigt. Planen var att få klart kapitel 1 och 2 tills deadline för kapitel ett men kapitel 1 drog mer ut på tiden än väntat så det fördröjde hela arbetet med nån vecka.

Händelseförloppet av arbetsprocessen ser ut på följande vis:

Vecka	Vad gjordes	Deadlines
1	Första veckan snackade vi ihop oss i gruppen hur vi skulle påbörja arbetet och vi kom fram till att det bästa var att boka ett möte med uppdragsgivaren som gjordes.	
2	Efter vi hållit ett möte med uppdragsgivaren satte vi igång med att börja leta efter lämplig litteratur för att kunna fylla på med information i rapporten. Vi började även brainstorma ideér till innehållet för kapitel 1 och påbörjade skrivandet.	
3	När vi nu hade fått material både från uppdragsgivaren och via insamlad litteratur kunde vi skriva in den information som krävdes för att få klart kapitel 1. Arbetet på kapitel 2 påbörjades.	Inlämningsdatum för kapitel 1 var 13:e april.
4	Kapitel 1 är klart så vi jobbar på med kapitel 2-3 inför deadline men hinner inte riktigt klart i tid. Vi sa på seminariet den 21:a april att	Inlämningsdatum för kapitel 2-3 20:e april som ska innehålla problematisering och forskningsanknytning samt forsknings- och

	vi kommer lägga mer fokus på den praktiska delen dvs själva utvecklandet av prototypen.	utvecklingsprocessen.
5	Vi startar igång med kodningen och gör upp en plan på vilka delar vi ska ha med i vårt program. Det togs fram förslag på olika programmeringsspråk och filhanteringsprogram som behövs finnas med.	
6	Arbetet rullar på bra men ligger efter i tidsplaneringen. Kod-delen börjar ta form och vi använder oss av GIT för att dela kod sinsemellan.	
7	Koden närmar sig funktionskontroll och delar av rapporten byggs ut med mer information även om det finns delar som behöver upparbetas.	Inlämningsdatum för kapitel 4 11:e maj som ska innehålla designresultatet.
8	Rapporten fylls på med mer information. och delar omarbetas	
9	Rapporten fylls på med mer information. och delar omarbetas	
10	Rapporten fylls på med mer information. och delar omarbetas	Inlämning slutversion 31 Maj, redovisning 3 juni.

3.2. Metodstöd för forskning och utveckling

I första skedet av projektet hade vi en plan att köra utefter Scrum och ha ett fysiskt möte/träff varje fredag för att utvärdera arbetet som gjorts under veckan. Denna plan höll inte mer än några veckor sen blev det att vi istället skriver till varandra dagligen vad som behövs göras och om något är absolut nödvändigt att diskutera bestämmer vi en tid att träffas. Vi använde oss av trello under de första veckorna för att få någon typ av struktur på planeringen och lättare kunna följa arbetsprocessen. Den idéen rann ut i sanden precis som Scrum men det var främst för att vi kände att det går minst lika bra att vi skriver upp på discord vad som behövs göras och sen gör man var sin grej. Detta har fungerat relativt bra förutom att det har garanterat påverkat tidsplaneringen på ett negativt sätt. Vi har även varit konsekventa med att få in en bokad tid med handledaren en gång i veckan för att ta upp svårigheter och problem som vi stött på i arbetet.

Det går att hitta likheter i vårt projektarbete och det som Hevner tar upp i sin artikel “A Three Cycle View of Design Science Research” som berör designforskningsmetoder (Hevner, 2007). I artikeln ställer Hevner frågan huruvida ett forskningsprojekt kan balansera input från forskning och de oönskade error som medföljer. Det vi har märkt när vi samlar information från litteratur är att det är i princip omöjligt att implementera data utan att den ändrar den befintliga mjukvaran till något sämre eller att någon del inte fungerar som den är tänkt. Det gäller således att alltid försöka matcha det befintliga projektet med input från litteraturen utan att viktig information går förlorad i processen.

3.3. Datainsamling

För att generera data till vårt arbete har vi först tagit reda på vad det är för delar vi ska ha med i modellen och sen utgått ifrån varje del när vi samlar in data. För att veta att det är rätt typ av data man hittat krävs det att du är insatt i vad det är du ska använda den data till som du söker efter. I artikeln skriven av Goldkuhl (2019) tar han upp olika sätt att samla in/generera data på som kallas för empiriska forskningsmetoder. En av dessa är “Document study” som innebär att man helt enkelt söker efter relevant information i form av dokument som kan antingen ske genom undersökningar av fysisk litteratur eller genom att söka på internet. Av alla dessa empiriska forskningsmetoder anser vi att document study stämmer bäst in på hur vi har samlat in data under vårt arbete vilket skett nästintill uteslutande via sökningar på webben. Eftersom vi har använt oss av denna typen av forskningsmetod kan man se tydliga liknelser med vad Goldkuhl skriver i sin artikel och hur vi har jobbat för att samla in data. I artikeln står det om hur man genererar data genom att hitta dokument med relevant data innehåll som man sedan implementerar i sitt projekt, vilket är mer eller mindre det vi gjort under hela arbetsprocessen.

3.4. Dataanalys

Det som gjorts under arbetet är att vi haft regelbunden kontakt med både uppdragsgivaren och handledaren under hela arbetsprocessen och bett om råd när det behövs och gör avstämningar om det som åstadkommits är korrekt.

3.5. Forskningsetik och sekretess

I framtaget av underlag till prototypen har vi efterlevt de rekommendationer som är applicerbara för berört arbete i enlighet med Vetenskapsrådets riktlinjer för god forskningssed (Vetenskapsrådet, 2017).

3.6. IP-rättigheter

Den framtagna prototypen och den programkod som är skapad är enligt uppdragsgivaren, författarnas egendom. I delar av koden används öppen APIer som går under Apache License 2.0, det ger oss stora möjligheter att använda kodbasen som vi själva anser passa oss.

3.7. Antaganden

Ett antagande vi gjort under projektets gång är att prototypen ska uppfylla alla de krav som ställts av uppdragsgivaren. Ett annat antagande var att vårt projekt ska se professionellt ut med tydliga beskrivningar i rapporten om vad det är vi har gjort i detta arbete och uttrycka oss på ett vetenskapligt sätt när tillfället är rätt. Övriga antaganden hittas under bilaga 1 i vår product backlog, samt i bilaga 2.

4. Designresultat

4.1. Bakgrund och introduktion

Uppdraget gick ut på att skapa en automatiserad uppdateringsfunktion eller prototyp där av med så liten inblandning från användare som möjligt. I beskrivningen nämns det att binärer som förmedlades till uppdateringsfunktionen kan vara stora och spridningen bör ske relativt fort. Samt att uppdateringsfunktionen inte leder till ett icke fungerande program, dvs en uppdatering ska genomföras i sin helhet. Önskemålet från uppdragsgivaren är att den kod som presenteras bör vara vår egen i största möjliga mån.

Beskrivningen av uppdragen ger att någon typ av förprogram eller del av det program som användaren startar, har någon typ av detektion som kan särskilja på en uppdatering som redan är applicerad eller ej. För att inte förhindra användandet av det program som uppdateringsfunktionen ska hantera, bör detektion inte använda en större mängd system resurser. Men även innehålla någon typ av felkontroll om berörd uppdaterings binär inte är tillgänglig av någon okänd anledning.

Därtill bör denna uppdateringsfunktion innehålla kontroller för att säkerställa att den publicerade och hämtade uppdaterings binär har rätt innehåll. Detta första steg skapade det flöde som återses i Bilaga 3 Huvudflödet. Berört förprogram kontrollerar om det finns en uppdatering samt verifierar med den lokal statusen.

Förprogrammet genomför detta, genom att överföra en mindre binär som innehåller berörd information gällande publicerad uppdatering. Lokalt har förprogrammet sparat senaste informations binär och via dessa informations binärer kan statusen bestämmas. Är den den lokala statusen äldre än publicerad börjar förprogrammet överföra berörd uppdaterings binär. I slutfasen av detta flöde kontrolleras att hemtagen uppdaterings binär är korrekt utan förändring i överföringen.

Inom denna del fick följande tekniska avgöranden framställas för att kunna realisera berörd funktionalitet.

4.2. Programspråk

Valet av programspråk för att realisera önskad funktionalitet är baserad på förkunskaper och behovet att ta fram en prototyp relativt snabb med minsta möjliga tidsåtgång. I enlighet med uppgiftens utformning finns inga direkta kodkonventioner men däremot vissa önskemål, det medför att vi har utgått ifrån de regler som SUN Microsystems och senare ORACLE har satt upp för skapande av programvaror i språket JAVA, som är det språk vi båda behärskar med godtagbar teknisk nivå. I förarbetet med prototypes förverkligande ett större klassdiagram som kan åter ses bilaga 4

4.3. Publicering och spridning

Med fokus inriktad mot den del av uppdateringsfunktionen som tar emot och utför uppdateringen, behövs en förenklad och universell lösning som uppfyller avgränsningen. Vilket pekar på att inte fokusera djupare på spridnings delen i underlaget från uppdragsgivaren. Med denna avgränsning föll valet på att använda den universallösning som fungerar bra för att realisera den mottagande parten. Men berörd val är nog inte optimal om uppdaterings binärerna och mängden mottagare är av kritisk storlek och mängd för vald teknik. Men för att skapa en prototyp som uppfyller kraven för uppdatering automatiken är en HTTP-baserad publicerings

och spridnings baserad lösning en utmärkt medelväg att använda. Det kräver inga större insatser att aktiver en lokal tjänst som hanterar berörd funktionalitet, gällande spridning och publicering. Samma enkelhet gäller för de funktion som hanterar hämtande av binärer från den publicerade platserna i koden, valt programspråk hanterar detta likvärdigt med lokala filer om än sökvägen är URL-baserad.

Efter att uppdaterings binären är hemtagen och kontrollerad korrekt, kan uppdaterings fasen påbörjas. Fasens alla steg ska ses i Bilaga 3 - Uppdaterings Flödet. I denna del av förprogrammet hanteras uppdaterinsbinärens innehåll, samt utför några filförlyttningar som behövs göras utföras inför uppdateringen ska utföras.

I framtagen lösning är uppdateringen binären en samlingsfil som behöver splittras i individuella filer. För att uppdaterings flödet ska ha någon uppfattning om vilka filer och vart i programstrukturen dessa finns för berört program som ska uppdateras. Behövs någon typ av informationsbärare ingå i uppdaterings binären, som innehåller berörd information. Genom att använda denna lösning kan samlings filens struktur hållas enkel och via den inbäddade informationens filen, så kan förprogrammet veta vilka filer och vart dessa befinner sig.

Indirekt blir denna informationsfil även en utförande lista, på hur uppdateringen ska ske. I uppgiftsbeskrivningen står det att en uppdatering inte får förhindra användaren från att använda det program som uppdaterings prototypen övervakar. För att lösa detta önskemål infördes en återställnings möjlighet,se bilaga 3 - Återställnings flödet, men även som beskrivs i den senare delen av Bilaga 3 - uppdaterings flödet, utförs en säkerhetskopia på berörd programdel innan en uppdatering sker. Genom att förprogrammet har en arbetslista på vilka filer som har blivit uppdaterade och om något fel uppstår kan den berörda listan användas för att återställa de programdelar som har blivit uppdaterat korrekt fram till det att felet detekterades.

Inom denna del fick följande tekniska avgöranden framställas för att kunna realisera berörd funktionalitet.

4.4. Uppdateringsmodell och detektering

I den kritiska valet gällande hur uppdateringarna ska utföras belyser Orso et. al. 2002 problematiken med dynamisk uppdatering i aktiva program. Ett steg för att undvika denna mer program tekniska inriktning som Orso et. al. 2002 har tagit valdes att utföra uppdateringen i icke aktivt program, via en launcher teknik, dvs ett program som är ansvarig för uppdatering flödet och därefter startar det egentliga programmet (huvud-prog). Via den tekniken kan problematiken med uppdatera ett aktivt program undvikas i vår prototyp av automatiserad uppdatering. Samtidigt har vi en programdel som måste fungera optimalt och utföra nya uppdateringar, men hur kan launchern detektera ny uppdateringar.

Vi hade inte någon direkt uppfattning hur det borde ske men Liu och Richardson (2000) beskriv en möjlighet med att använda textbaserade informations filer med ett specifikt format, i deras fall XML. Detta togs fasta på, men i motsats till Liu och Richardson (2000), valdes att förenkla text filens format samt förnya den något. Formatet som valdes är JSON. Detta främst för Lin et. al. (2012) beskriver hur dessa två format behöver olika mängder tid för att bli processade och Lin et. al. (2012) påvisar att JSON är det bättre formatet om processtid är en viktig parameter. Detta i åtanke samt med en utgångspunkt att launcher skulle använda så få resurser som vårt tekniska kunnande medgav, för att kunna fungera enligt de uppställda önskemålen.

När det kommer till själva resultatet av uppdateringsfunktionen inleds hela processen med att programmet gör en sökning(loggning) efter tillgängliga uppdateringar och packar innehållet i uppdateringsfilen om den nu hittade en. Sedan läses en rad in från manifestet med sökväg och filnamn. Tillslut säkerhetskopieras allt och uppdateringen utförs enligt våra instruktiuoner. Resultatet som vi kommit fram till är helt enkelt det som redovisas i kapitel 4 samt under bilaga

3, utöver vår uppdateringsfunktion som är den största delen av uppgiften finns också återställningsfunktionen som har i uppgift att återskapa föregående fil innan uppdateringen om den skulle gå snett under uppdateringsprocessen. Den fungerar på liknande sätt som själva uppdateringsfunktionen med undantag att den har ett annat syfte och funktion. Även om prototypen i sig inte blev riktigt som vi tänkt oss från början var det ändå "tillräckligt" för en prototyp enligt Skatteverket. Lösningen i sig uppfyller kravspecifikationen och går att köra utan några större problem. Mer information gällande uppdateringsmodellen hittas under bilaga 3 "Flödeschema: Uppdatering".

4.5. Atomicity

Problematiken med att en uppdatering kan få huvud-prog slutar fungera medförde att vi behövde se uppdateringsflödet som atomärt (atomicity), med liknande tankesätt som finns inom databaser gällande sammansatta kritiska operationer, Orso el. al (2002) beskriver att de använder samma tankesätt i deras arbete. Genom att atomärisera vår lösning gällande uppdatering flödet kan huvud-prog skyddas mot felaktigheter i uppdaterings flödet. Det skyddar dock inte mot felaktigheter som kan komma via korrekta uppdatering binärer, som blivit felaktigt vid överföring eller manipulerade. Det motdrag som används för att skydda mot problem i hämtning och manipulerade uppdaterings binärer, är checksummor.

En matematisk algoritm som Nguyen (2005) beskriver mer ingående. Huvudsyftet är snabb feldetektering vid överföring eller för att försäkra att inget blivit förändrat. Nguyen (2005) beskriver att dessa checksummor är beroende av binären som checksumman beräknas på, d.v.s. summan blir lika varje gång och ändras binären kan detta detekteras i checksumman. Vår lösning använder en checksumma baserad på SHA1 algoritmen, en något förbättrad algoritm av MD5. Nguyen (2005) förklarar att MD5 kan i vissa unika fall ge samma checksumma för två helt olika binärer. Detta är bar en del i att skapa en atomär funktion, för att kunna återställa vid detekterade problem införde vi den enklaste formen av säkerhet d.v.s. kopiera den binär som kommer att behandlas till en säker plats för att kunna genomföra en återställning om checksumman detekterade en förändring.

Denna del gällande uppdatering flödet och dess atomära natur återfinns i Bilaga 3 - Uppdaterings flödet och Bilaga 3 - Återställnings flödet.

4.6. Updateringsbinärerns sammansättning

Uppdaterings binären kan ses som en samling av fristående mindre binärer som ingår inom det program som ska uppdateras. Dessa del binärer har en specifik plats inom strukturen för programmet som ska uppdateras. För att vår prototyp ska kunna veta vilka binärer som ska ersättas och vart dessa återfinns i programmets som ska uppdateras substruktur. Behövs en arbetsorder liknande informationsbärare. I den lösning som skapades utgör denna informationsbärare av en tecken separerad fil, CSV enligt RFC 4180 (rfc-wiki 2020). Det är en äldre modell av informationsutväxling format mellan olika system, formatet kräver dock att både mottagare och avsändare använder samma typ av separationstecken och datatyp. Denna universella uppbyggnad samt extrema enkelhet, medför att användandet inte behöver stora resurser för att behandlas i prototypen.

CSV-filen har fått funktionen att fungera som arbetsorder med vilket prototypen kan arbeta sig igenom dess innehåll och efter hand utföra uppdaterings momentet, samt om något fel detekteras används den för återställnings flödet.

4.7. Kontroll och förbättringspotential

Realiseringen av prototypen med berörda funktioner sker mot ett "hello world" program som efter varje uppdatering förändrar sin utformning av ingående filer och utskrift till konsolen. I nuvarande utformning av prototypen är långtids loggning en icke fråga utan lösningen lägger ut

mer text än nödvändigt i konsolen. Att välja en större mängd utmatnings information har använts för att felsöka men även underlättat vart i flödesdiagrammet realiseringen befinner sig, samt via dessa utskrifter, felsöka logiska och program tekniska missar.

I denna testande del av realiseringen städades många logiska samt tekniska missar bort, samtidigt har möjliga förbättringar och logiska tankevurpor framkommit som behöver hanteras snarast. För att säkra en bättre funktionalitet i den realiserade prototypen gällande uppdaterings flödet och förstärka dess atomära natur.

Nuvarande utformning har svårt att hantera en djupare relativ mappstruktur, vidare behövs en bättre kontroll på vilka delar som blivit säkerhetskopierade. Återställnings flödet behöver en egen log eller arbetsorder att använda för att garantera en mer korrekt återställning utan att rest binärer uppstår. Till detta kan även hanteringen av uppdaterings detekteringen förbättras för att undvika problem om en uppdaterings binär blir förbigången i ordningsföljden.

4.8. Kontrollresultat

Prototypen kan i dagsläget få en enklare “Hello World” program att exekveras. Finns en web-server aktiv på förbestämd URL där uppdaterings binär med tillhörande beskrivande text-fil i JSON-format finns. Kan prototypen detektera en uppdatering och applicera den i berört “Hello World” program. Under utprovningen kontrollerades att uppdaterings flödes fungerande enligt plan. I detta uppdagades ett tankefel som generade att prototypen inte kunde hantera uppdaterings binärens innehåll utan fel detekterade och flödet avbröts. Ett liknande fel kunde skönjas i återställnings flödet men i nuvarande testportfölj är detta problem intet.

Berört “Hello world” program blir uppdaterat i tre steg från det enkla som skriver ut en text till konsolen i main-klassen till den slutgiltiga som har en tillhörande java-klass som skriver ut till konsolen. Detta är mer för att påvisa att prototypen kan utföra det som uppdraget gav men inte mer än så.

5. Utvärdering

5.1. Demonstrativ utvärdering

Lösningen har ännu inte demonstrerats för någon person som är utanför projektet med undantag för uppdragsgivaren som följt arbetsprocessen. Planen är att låta några personer från vår klass testa prototypen och ge deras åsikter på lösningen. Det är viktigt för oss som utvecklare att få feedback på vårt arbete för att kunna bygga vidare och förbättra det som är mindre bra.

5.2. Experimentell utvärdering

De tester som utförts har enbart varit manuella tester där vi kör igenom hela koden för att se så att alla delar fungerar som de ska. De olika metoderna och klasserna i koden har utvecklats separat i en variant av microservices där vi sedan slog ihop allt och testade koden. En del fick bytas ut efterhand som inte fungerade med övriga delar samt att vi testat olika metoder som exempelvis “maven” för att spara filer online istället för lokalt. Än så länge är det bara vi författare och uppdragsgivaren som testkört koden så vi har enbart våra synpunkter att utgå ifrån, men tanken är att någon klasskamrat ska få testa programmet och ge sina synpunkter på det hela. Får vi fler inputs från olika håll kommer det högst troligt höja kvalitéten på lösningen.

5.3. Tolkande utvärdering

När vi påbörjade arbetet att skapa en lösning åt vår uppdragsgivare var kraven kortfattade och simpla vilket har lett våra tankar mot att genomföra denna uppgiften enligt metoden KISS (Keep It Simple, Stupid). Vi har således arbetat efter att göra lösningen så pass enkel så att alla som är det minsta insatt i programmering ska förstå vad koden gör samtidigt som den uppfyller alla krav och fungerar optimalt för den påtänkta syftet. I den här avhandlingen har vi använt oss av ett relativt brett spann av litteratur som kan medföra positiva respektive negativa effekter på vårt arbete. Det är å ena sidan en fördel att ta information från flera olika källor för att säkerställa att informationen stämmer och att få flera infallsvinklar på saken, å andra sidan kan det leda till att texten blir överarbetad och man upprepar samma sak flera gånger i ett stycke. Efter att vi samlat på oss en måttlig mängd olika litterära verk sållade vi bort de som var mindre givande och fokuserade på de mer tyngre verken som vi anser har bidragit med mer trovärdig input till projektet samtidigt som det höjer kvalitéten på hela arbetet.

5.4. Formativ utvärdering genom designprocessen

Arbetsprocessen kan delas upp i tre faser där vi utförde faserna systematiskt för att försäkra oss om att vi får med den information vi behöver för att genomföra arbetet. Vi började med i första fasen att brainstorma fram litteratur i form av aktivt sökande på internet för att få en stadig ståndpunkt i vårt arbete. I den första fasen, som innefattar till stor del förarbete, gjordes även en noggrann granskning av flödesprocessen dvs. en överblick av uppdragsbeskrivningen för att veta vilka delar som behöver vara med i prototypen för att alla krav ska uppfyllas. Här valde vi också ut JAVA som programmeringsspråk som vi ville använda oss av för att utveckla prototypen, utöver det bestämdes valet av Gson för att konvertera JAVA-objekt till JSON. När första fasen var avklarad och förarbetet gjort började vi titta på liknande program/applikationer som vi kunde låta oss inspireras av för att skapa vår egna lösning. Tanken med detta var att vi kunde utgå från befintliga lösningar för att slippa uppfinna hjulet på nytt och snabbare komma igång med utvecklingen. Den andra fasen gick helt enkelt ut på att genomföra det som planerades i föregående steg och skriva på rapporten samt utveckla prototypen. I den tredje och sista fasen gällde det att fixa klart de sista delarna på rapporten respektive lösningen och sammanställa hela projektet inför slutinlämning och opponering. Om man tittar på vårt tillvägagångssätt som vi använt oss av i denna uppgift kan vi konstatera att det blev ett fulländat projekt som uppfyller de krav som ställts och på så sätt säkerställer en god kvalité av resultatet.

5.5. Pragmatisk utvärdering

Vår tanke är att prototypen ska vara tillräckligt tillfredsställande för uppdragsgivaren att den således skall kunna utföra de uppdrag som lösningen är tänkt att göra. Eftersom det inte har specificerats av uppdragsgivaren vad det är för typ av program vår lösning är till för eller hur prototypen ska fungera i praktiken kan vi bara utgå ifrån att den ska uppfylla de krav som ställts. Vi hoppas att vår lösning kommer till användning för skatteverket och kan användas av alla inom organisationen som behöver få sin mjukvara uppdaterad, men med avseende på hur liten roll vår prototyp verkar få i deras verksamhet ska man inte räkna med för mycket.

5.6. Sammanfattande karaktärisering och reflektion

Vi har karaktäriserat våra utväderingsutsatser i projektet utifrån Cronholm & Goldkuhls utvärderingsmodell (2003). De övriga utvärderingstyperna utgick på grund av att de antingen innefattade användare eller induktiv ansats som inte stämde överens med vårt projekt. Vår utvärdering kan karaktäriseras med typologi 3 “Type 3 - Criteria-based evaluation of IT-systems as such”. Huvudperspektivet för typologi 3 stämmer överens med vårt projekt med anledning av att den följer kriterier som sitt huvudfokus då uppdragsgivaren försåg oss med en kravlista på specifikationer för vår lösning.

Under vårt arbete med insamling av litteratur för lösningen kom vi över flera informationsrika källor som gav oss värdefull input för utvecklingen av prototypen. Ett exempel på en användbar källa är Orso et. al. (2002) där det finns en teknik för att uppdatera en mjukvara i java på ett dynamiskt sätt som har visat sig vara givande i vårt arbete. Vi har även använt oss av Github Google (2022) som källa för att få reda på hur man konverterar java objekt till JSON och gson var lösningen på problemet. Sedan använda vi också Nguyen (2005) för att hitta ett sätt att hantera eventuella fel i koden på ett snabbt och effektivt sätt. Vår lösning är av deduktiv natur eftersom vi har utgått från det allmänna resonemanget till det specifika i arbetsprocessen. Deduktiv slutledning innebär att man utifrån allmänna slutsatser kan dra specifika antaganden vilket är precis det som skett i vårt arbete. I denna avhandling har vi använt oss av en heuristisk utvärdering vilket innebär att vi författare agerar som användbarhetsexperter och utvärderar vårt eget arbete för att se användbarheten med det. Vår uppdragsgivare kan klassas som användaren i det här fallet som kommer testa lösningen och leta efter brister som vi har missat i våra tester.

Vårt huvudmål har hela tiden varit att uppfylla de krav som vår uppdragsgivare gett till oss men också att sätta upp delmål på vägen för att vara mer effektiv i arbetet mot huvudmålet. Går man lite djupare in på vad målet med hela projektet är kan man dra paralleller till det som Cronholm och Goldkuhl(2003) säger i sin artikel om att målet är att skaffa sig en djupare förståelse för hur IT-systemen fungerar och används.

Valet av vår design med typologi 3 som fokuserar på kriterier som målsättning kommer med både för och nackdelar. Eftersom vårt projekt följer en kravlista som kriterier anser vi är en fördel för oss som underlättar arbetet då det är tydligt med vad som måste göras. En annan fördel är att användarna inte är inblandade i arbetsprocessen utan vi har kunnat jobba på i lugn och ro med vår lösning och sen låta uppdragsgivaren testa om prototypen faller honom i smaken. Å ena sidan hade det varit en fördel att köra utan användare men å andra sidan har det varit en nackdel då vi inte kunnat genomföra regelbundna tester genom användarna för att se vad som funkar bra/mindre bra med lösningen. Det brukar vara svårt för en själv när man gjort ett arbete att direkt hitta fel vilket är därför man kan låta andra utomstående läsa igenom arbetet som ser på det med andra ögon än vad vi gör och kan hitta saker som vi missat.

6. Diskussion och abstraktion

Det som gjort att vår arbetsprocess blev effektiv var att vi helt enkelt delade upp arbetet mellan oss så man visste exakt vad som skulle göras just då. Det går att argumentera för att vårt viktigaste beslut som togs under processen gällande resultatet var att följa kravlistan till punkt och sätta prio ett på att få funktionaliteten på plats med minimal ansträngning. Sedan om det fanns tid och ork förfina lösningen och göra den mer användarvänlig. Som nämnts tidigare i rapporten har vi valt typologi 3 från Cronholm & Goldkuhls utvärderingsmodell (2003) för vår designprocess, vilket stämmer överens med de beslut vi tagit under arbetets gång. Det är framförallt kravlistan som vi följt slaviskt som kan kopplas till kriterierna som Cronholm & Goldkuhl tar upp i typologi 3 där vi ser tydliga likheter med vår arbetsprocess.

I detta projekt bygger hela arbetsprocessen på den kravlista vi fick av uppdragsgivaren i början av arbetet. Det som varit nyckeln till framgång i projektet var att vi hade ett möte med uppdragsgivaren i början och gjorde upp en plan för hur vi skulle ta oss an uppgiften, det var också vid detta tillfälle som vi fick kravlistan. Att vi spånade och samlade mängder av litteratur under de första veckorna har varit en annan orsak till vår framgång i att hitta olika lösningar att implementera i vår lösning samtidigt som vi fyllde på rapporten med information gällande prototypen. Skulle vi göra ett liknande projekt idag hade vi gjort på liknande sätt som i detta fallet men kanske med en något tydligare plan för tidsfördelningen på varje uppgift. I övrigt har det funkat bra med det upplägg vi använt oss av så det tar vi med oss till nästkommande projekt.

Det som varit mest användbart i projektet har varit att börja med att spåna fram användbar litteratur för skrivandet och utvecklingen av prototypen samt mötet med uppdragsgivaren. En av de mest givande av de litterära verken vi samlat ihop har varit Nguyen (2008) för att skapa en lösning med möjlighet att skicka ut mjukvaruuppdateringar per automatik i en komponent baserad miljö. En källa som har varit mycket användbara och som nämnts i föregående kapitel är Orso, Rao, Harold (2002) som tar upp en teknik för att uppdatera en mjukvara i java som varit särskilt gynnsamt för oss i det här arbetet då det är nästintill enbart java som vi använt oss av. En annan mycket användbar källa vi använt oss av är Github Google (2022) där vi fick lära oss hur man konverterar objekt i java till json genom att använda java-biblioteket gson. En sista källa som visat sig vara värdefull i vårt arbete är Nguyen (2005) som förespråkar en algoritm som används för att hitta eventuella fel som uppstått i koden när programmet körs. Det går att argumentera för att vi skulle använt oss av en annan approach där vi försökte upprätthålla en tajtare kontakt med uppdragsgivaren och be om råd med litteratur som han kunde rekommendera istället för att söka enbart på egen hand. Man kan å ena sidan argumentera för att vi hade snabbare kommit fram till ett resultat om vi gjort på detta sättet istället, men å andra sidan var uppdragsgivaren konstant stressad över sig egna arbete och hade svårt att ta sig tid för oss vilket innebar att vi tog hjälp från andra håll och förlitade oss på våra kunskaper vi fått under utbildningen. Visst det går nästan alltid att hitta plats för förbättringar på ett arbete som genomförts men det är heller inte alltid som en annan approach nödvändigtvis behöver resultera i att kvalitéten på arbetet blir förbättrad. Vi är nöjda med vår insats och de val vi tagit under detta projekt och är stolta över vad i åstadkommit.

I denna avhandling har vi lärt oss om hur kontakt med en statlig myndighet inom IT-branschen fungerar i praktiken och hur det är att få arbeta gentemot en större organisation. Vi har också lärt oss använda en modell i ett arbete och arbeta utefter den när man både skriver en rapport och utvecklar en mjukvara. Denna modellen vi jobbat efter är den kriteriebaserade listan som skickades ut av uppdragsgivaren under första mötet vi som ägde rum. En sak som vi både har lärt oss och upptäckt under arbetets gång är att det är ett krav att kunna grundläggande programmering även om nivån på uppdraget kan vara på en mer avancerad nivå för att lyckas med uppgiften.

Avslutningsvis kan vi konstatera att vi har uppfyllt hela kravlistan i detta projekt samt fått ett godkännande av uppdragsgivaren och således lyckats med uppgiften i helhet.

7. Slutsatser

Här sammanfattas viktiga resultat och rekommendationer för fortsatt arbete.

7.1. Slutförande av uppdraget

I enlighet med de uppställda kriterierna i kapitel 1 och uppdragsgivarens önskemål har prototypen fått den funktionalitet som efterfrågas om än med minsta möjliga marginal. Prototypen kan detektera och överföra från extern plats till lokal plats en uppdaterings binär och processa dess innehåll för att uppdatera ett program, utan att medföra problem med programmets funktion. Inom denna del kan arbetet anses som en succe.

7.2. Fortsatt praktiskt arbete

För fortsatt förbättrad funktionalitet i prototypen, behöver delar av de tidigare redovisade problemen från Kapitel 4, kontroll och förbättringspotential beaktas. Prototypen kan i dagsläget inte hantera program med en lokal mappstruktur djupare än en undermapp, samtidigt så behöver prototypen även kunna arbeta med förhöjd behörighet, i dagsläget körs prototypen med samma behörighet som exekverande användare. Vidare bör atomäriseringen förbättras med omarbetat lösning för kontroll mot felaktigheter vid fil-operationer. samt förbättra hur prototypen utför återställning efter problem detektering. En mer beständig loggning som loggar rätt information för berörd funktion och utförd operation. Alla dessa delar ökar funktionell kvaliteten på prototypen, vidare bör någon typ av paketerings funktion skapas för att på ett enkelt sätt generera uppdaterings binärer inför publicering. Listan med förbättringspotential kan utökas med fler identifierade funktioner från underlaget.

7.3. Forskningsbidrag

I enlighet med den uppställda problem klassen i kapitel 2.5 — atomicity, framkommer att berörd klass är en icke omnämnd men väsentlig del inom programvaruuppdateringar. Det omnämns hastigt i Orso et. al. (2002) rapport men inte av Liu och Richardson (2000). Via vårt arbete belyses detta med atomicity inom programvaruuppdateringar och bidrar med en belysning av berört koncept inom ämnesområdet än att tillföra något mer konkret.

Denna något minimala bidragande till forskningsvärlden gällande programvaruuppdatering och atomicity ger i vårt arbete möjligen ger fler frågor än svar samtidigt som bidraget är av tillämpande art via den prototyp som är framtagen. I enlighet med de teorier som Ågerfalk (2014) lägger fram.

7.4. Fortsatt forskning

Med de som framkommit gällande forsknings bidragande delen framstår det att det finns områden för mer forskning.

RQ1: Kan atomicity som underliggande teknik vara en lösning gällande kritiska programvaruuppdateringar och på vilket sätt kan atomicity bidra till resultatet.

Vidare så finns följande fråga som är mer riktad mot slutresultatet av en programvaruuppdatering och då främst av typen automatiserade.

RQ2: Hur påverkar atomicity användarupplevelsen inom automatiserad uppdatering av programvara samt bidrar detta till en djupare förtroende av automatiserade uppdateringar av programvara.

7.5. Metodreflektion

Vår arbetsprocess har kommit att präglas av den kravlista som varit vår utgångspunkt i detta projektet och som kommit att forma vårt arbetet. Under arbetets gång har vi läst in oss mer och mer på berörda begrepp inom ämnet såsom atomicity, XML, gson etc. som alla har påverkat och drivit på arbetsprocessen mot slutresultatet. Det har å ena sidan varit en stundtals lång och utdragen process att gå igenom all litteratur som krävdes för underlaget i arbetsprocessen, men å andra sidan har det varit en lärorik resa som har gett oss mycket ny kunskap om saker vi inte hade en aning om innan.

Sammanfattningsvis kan vi dra slutsatsen att arbetsprocessen har lett till ett resultat som präglas av våra innovativa idéer och forskning som bidragit till ett slutresultat.

Källförteckning

A. Orso, A. Rao and M. J. Harrold, "A technique for dynamic updating of Java software," *International Conference on Software Maintenance, 2002. Proceedings.*, 2002, pp. 649-658, doi: 10.1109/ICSM.2002.1167829.

Baskerville, R. Kaul, M. Storey, V. 2011. Unpacking the duality of design science.In ICIS 2011 Proceedings.

B. Lin, Y. Chen, X. Chen and Y. Yu, "Comparison between JSON and XML in Applications Based on AJAX," *2012 International Conference on Computer Science and Service System*, 2012, pp. 1174-1177, doi: 10.1109/CSSS.2012.297.

Cronholm, Stefan. Goldkuhl, Göran. 2003. Strategies for information systems evaluation-six generic types. *Electronic Journal of Information Systems Evaluation.* 6(2), 65-74.

C. Liu and D. J. Richardson, "Automated security checking and patching using TestTalk," *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*, 2000, pp. 261-264, doi: 10.1109/ASE.2000.873673.

G. D. Nguyen, "Error-detection codes: algorithms and fast implementation," in IEEE Transactions on Computers, vol. 54, no. 1, pp. 1-11, Jan. 2005, doi: 10.1109/TC.2005.7.

Github Google (2022), “A Java serialization/deserialization library to convert Java Objects into JSON and back “, URL https://github.com/google/gson accessed 20220504

Gregor, S. Hevner, A.R. 2013. Positioning and Presenting Design Science Research for Maximum Impact. *Mis Quarterly.*

Hevner, A. R. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems.* Hämtad: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1017&context=sjis

RFC-Wiki (2020), ”RFC4180”, URL https://www.rfc-wiki.org/wiki/RFC4180 access 20220509

Skatteverket. (2022). *Det här gör skatteverket.*

Uaas: Software Update as a Service for the IaaS Cloud.

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7207390&casa_token=06FiiipFFNUAAAAA:FxQY2tRIQzxRV9wGbrk0uh0lVES0C8GnVprW1OiO4Wo10r2mf-yRYuXpE4kBH9i-D2Di7SjHXw

Tien N. Nguyen. 2008. *Electrical and Computer Engineering Department.* Diss., Iowa State University. Component-based Software Update Process in Collaborative Software Development. https://ieeexplore.ieee.org/document/4724576

Vetenskapsrådet (2017). God forskningssed. Stockholm: Vetenskapsrådet

Bilaga 1 – Product Backlog

Vision: Skapa en prototyp som påvisar funktionaliteten för automatisk uppdatering som hanterar stora binärer.

#	**Beskrivning**	**Prioritet**	**Kommentar**	**Källa**
1	Bestämma överförings- protokoll av binärer		Ha god överföringskapacitet.	
2	Bestämma hur detektering av ny uppdatering sker		Använda sekvensnummer eller något annat som enkelt kan jämföras mot lokalt tillstånd	
3	Bestämma format på berörd databärare för detektering		Någon typ av filformat som använda för utväxling av data mellan olika system	
4	Typ av information, data bäraren innehåller		Bör finnas med särskillnader information. Eventuellt namn på uppdaterings binären samt möjlighet till innehållskontroll av uppdaterings binären, såsom checksumma.	
5	Bestämma format på uppdaterings binären		Binären är någon typ av samlings fil med flera ingående filer	
6	Bestämma format på berörd databärare för innehålls filen av uppdaterings binären		Behöver innehålla information om vart och namn på filen som ska	

			uppdateras finns samt innehållskontroll, såsom checksumma.	
7	Realisera Agentens förenklade UX			
8	Implementera överförings protokollet i Agenten			
9	Agenten detekterar ny uppdatering			
10	Agenten tar ner uppdaterings binären, till systemets temporär mapp		Bör använda en överföringsteknik som inte förhindrar användande av datorn	
11	Agenten, verifiera mottagen uppdaterings binären		Behöver ha god data genomströmning, utan att förhindra användares användande av datorn	
12	Agenten kopierar innehållet i uppdaterings binären till en undermapp i Systemets temporär mapp		Berörd mappnamn bör bestå av delar av eller hela uppdaterings binärens namn	
13	Agenten läser in innehållsförteckningen som ingår i uppdaterings binären.		Filen bör innehålla vad som ska uppdateras samt någon typ av checksumma för att kontrollera korrekthet	
14	Agenten utför en säkerhetskopiering		använder innehålls filen,	

			för att kopiera rätt fil.	
15	Agenten utför uppdateringen enligt metoden ersätt fil		Använder innehålls filen för att uppdatera rätt fil	
	Agenten verifiera att uppdatering utförts korrekt		Behöver ha god data genomströmning, utan att förhindra användares användande av datorn	
	Agenten återställer felaktig uppdatering		Använder innehålls filen samt någon typ av minne för att återställa berörda filer.	
	Agenten aktiverar programmet			
	Antaganden/Eget initiativ		Vi vill få vår prototyp att se professionell ut samtidigt som den uppfyller alla kraven från uppdragsgivaren. Det ska vara enkelt för en oinsatt person att följa arbetet i rapporten.	

Bilaga 2 – Funktionalitet

Bilaga 2

Ett förslag på projekt för en student intresserad av systemutveckling

Uppdaterings service för desktop applikation.

När man väl har en mjukvara som man är nöjd med och man tänker "nu vill jag lägga ut den här så att folk kan ladda ner och installera på sin dator" så brukar det genast komma in lite "felrapporter" och önskemål som kan och måste åtgärdas, nu har du gjort en ny version av din mjukvara men hur levererar du den till dina användare om de började med att tanka ner en installer på din hemsida, hur vet dom om att det ens finns en uppdatering?

Projektet är som sådan:

För att underlätta för användaren ska du skapa en tjänst som uppdaterar din applikation som är installerad på datorn.

Användaren ska om möjligt enbart få information om att applikationen uppdateras och ska inte behöva installera något mer än så. Eftersom din tjänst har stora binärer och liknande så måste du också kunna hantera transporten av dessa till miljontals användare inom ett par dygn från att du släpper en uppdatering.

· En Applikation ska kunna uppdateras löpande, applikationen kan vara vad som helst egentligen som ni har liggandes, kanske en annan kursuppgift eller hobbyprojekt, problemet kanske är att utforma en applikation som går att uppdatera?

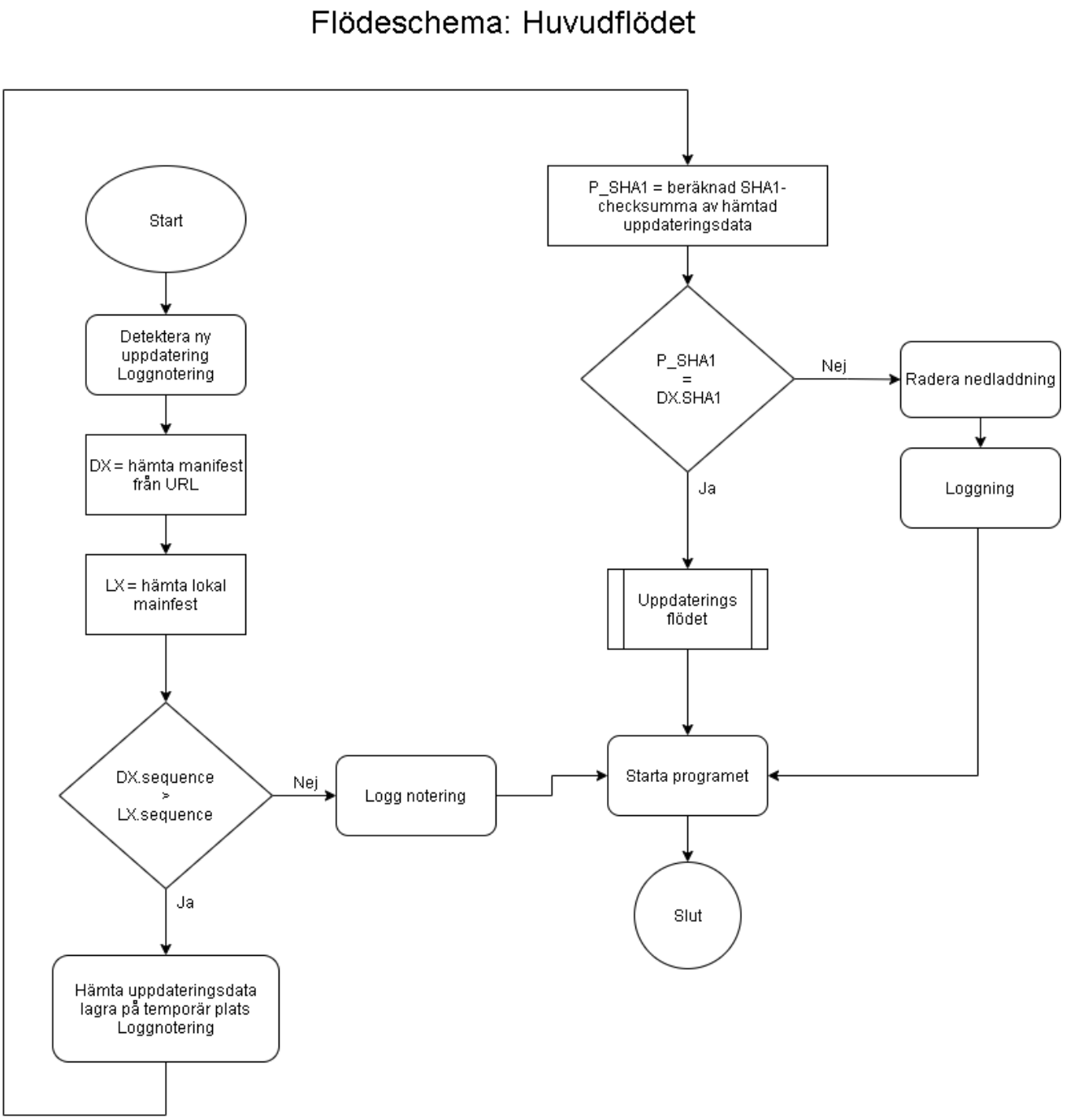
· Själva uppdateringen ska kunna hantera delar av applikation och inte hela applikationen för att spara på trafik.

· en lösning för att kunna leverera stora filer till större mängd användare samtidigt, en direkt filöverföring från din server är inte önskvärd för uppdateringsdata.

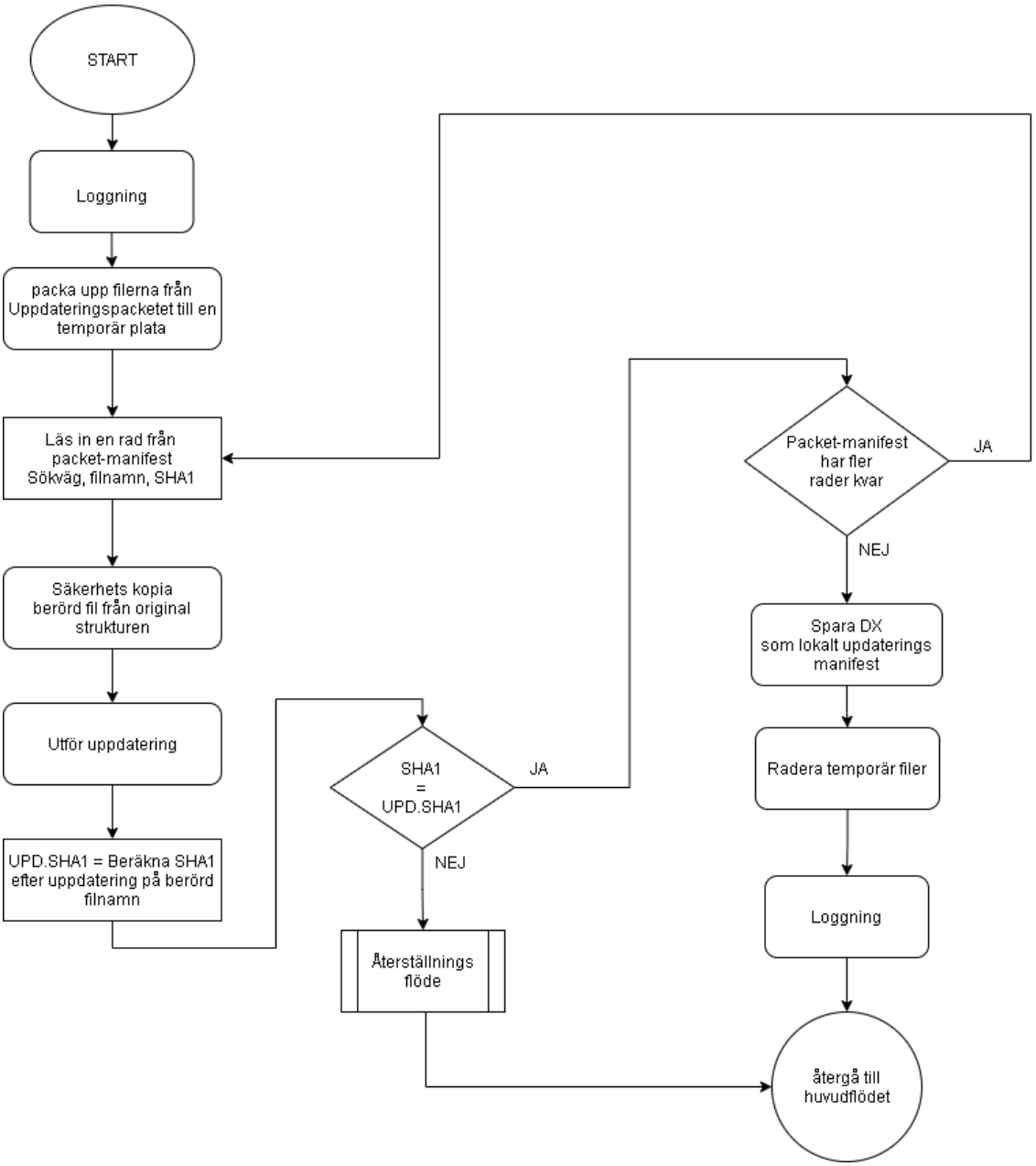
· Uppdateringen ska garanteras att inte förstöra applikationen.

· Inga färdiga lösningar för uppdatering är tillåtet, nätverksbibliotek och standard fönster bibliotek (windowsforms, qt, SDL eller liknande) är OK.

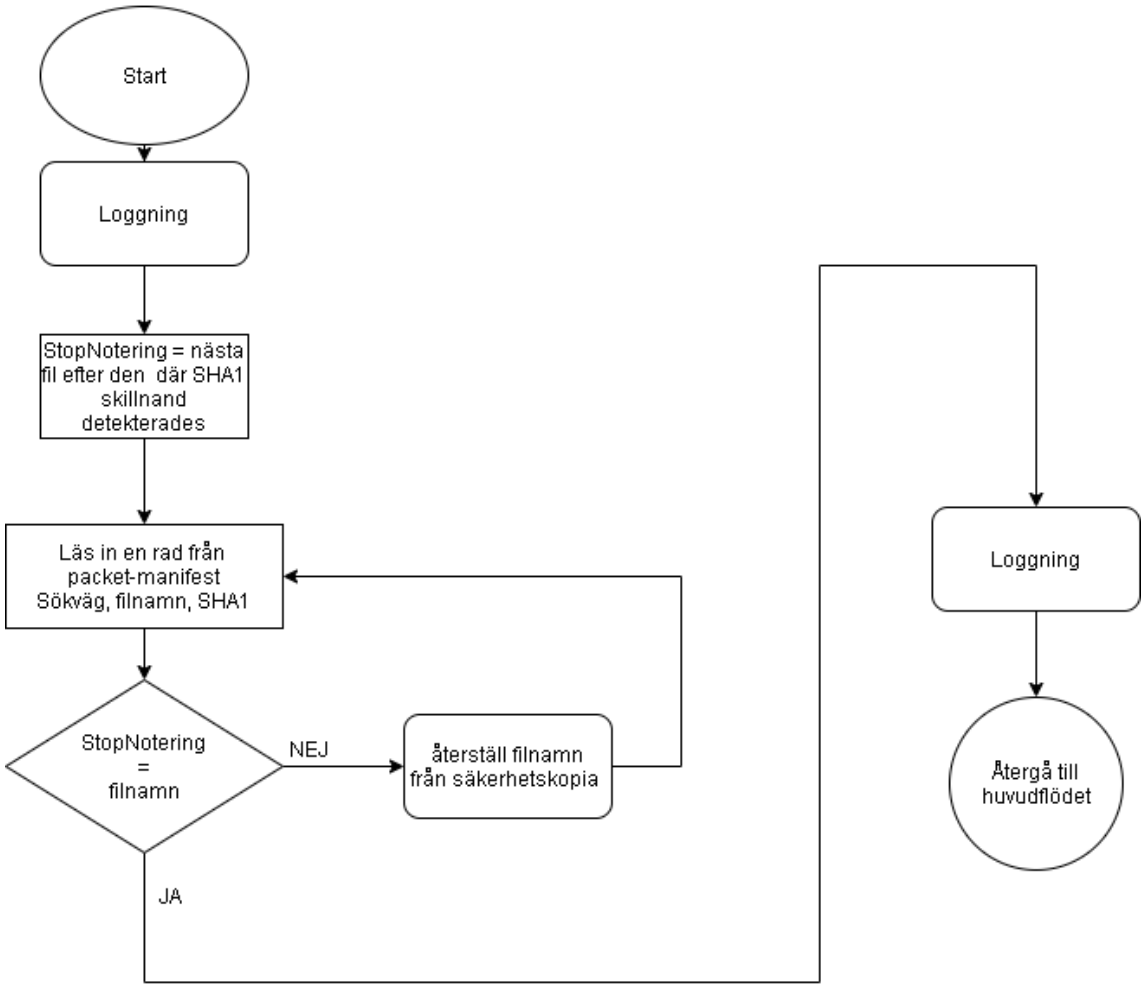
Bilaga 3 – Flödesscheman



Flödeschema: Uppdatering



Flödeschema: Återställning



Bilaga 4 - Klassdiagram