HÖGSKOLAN
DALARNA

Examensarbete

Kandidatuppsats

Continuous Delivery

Utmaningar ett förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery

Challenges a maintenance management project may face during a transition to Continuous delivery

Författare: Erika Lööf och Colin Mack
Handledare: Ulrika Artursson Wissa
Examinator: Mathias Hatakka
Ämne/huvudområde: Informatik
Kurskod: IK2017
Poäng: 15 hp
Examinationsdatum: 160531

Abstract

Continuous delivery (CD) is a software engineering approach where the focus lays on creating a short delivery cycle by automating parts of the deployment pipeline which includes build, deploy-, test and release process. CD is based on that during development should be possible to always automatically generate a release based on the source code in its current state. One of CD's many advantages is that through continuous releases it allows you to get a quick feedback loop leading to faster and more efficient implementation of new functions, at the same time fixing errors. Although CD has many advantages, there are also several challenges a maintenance management project must manage in the transition to CD. These challenges may differ depending on the maturity level for a maintenance management project and what strengths and weaknesses the project has.

Our research question was:
"What challenges can a maintenance management project face in transition to Continuous delivery?"

The purpose of this study is to describe Continuous delivery and the challenges a maintenance management project may face during a transition to Continuous delivery.

A descriptive case study has been carried out with the data collection methods of interviews and documents. A situation analysis was created based on the collected data in a shape of a process model that represent the maintenance management projects release process. The processmodel was used as the basis of SWOT analysis and analysis by Rehn et al's Maturity Model. From these analyzes we found challenges of a maintenance management project may face in the transition to CD. The challenges are about customers and the management's attitude towards a transition to CD. But the biggest challenge is about automation of the deployment pipeline steps.

Keywords: Continuous Delivery, challenges, maintenance management project

Sammanfattning

Continuous Delivery (CD) är en metod inom systemutveckling där fokus ligger på att skapa en kort leveranscykel genom att automatisera delar av deployment pipeline vilket innehåller bygg-, deploy-, test- och releaseprocessen. CD bygger på att det alltid under pågående utveckling skall vara möjligt att automatiskt generera en release utifrån källkoden i dess aktuella tillstånd. En av CD's många fördelar är att man genom kontinuerliga releaser får en snabb feedbackloop vilket leder till snabbare och effektivare införande av nya funktioner, samtidigt som man åtgärdar fel. Även om CD har många fördelar finns det också flera utmaningar ett förvaltningsuppdrag måste hantera vid en övergång till CD. Dessa utmaningar kan skilja sig beroende på vilken mognadsnivå ett förvaltningsuppdrag befinner sig i och vilka styrkor och svagheter uppdraget har.

Vår frågeställning har varit:
"Vilka utmaningar kan ett förvaltningsuppdrag stå inför vid en övergång till Continuous Delivery?"

Syftet med denna studie är att få en ökad förståelse för Continuous Delivery samt vilka utmaningar förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery.

En beskrivande fallstudie har genomförts med datainsamlingsmetoderna intervjuer och dokument. Av insamlad data skapades en nulägesanalys i form av en processmodell över förvaltningsuppdragens releaseprocess. Den har använts som grund för en SWOT- analys och en analys med hjälp Rehn mfl's Maturity model. I från dessa analyser kom vi fram till utmaningar ett förvaltningsuppdrag kan stå inför vid en övergång till CD. Utmaningarna handlar dels om kunder och ledningens inställningar inför en övergång till CD. Men den största utmaningen gäller automatisering av delarna i deployment pipeline.

Nyckelord: Continuous Delivery, utmaningar, förvaltningsuppdrag

Förord

Vi vill tacka vår samarbetspartner Triona som har gjort detta examensarbete möjligt. Ett särskilt tack till våra handledare Anna Mårsell och Tomas Yvell som stöttat oss i vårt arbete.

Vi vill också rikta ett stort tack till vår handledare Ulrika Artursson Wissa på Högskolan Dalarna för stöd och hjälp under studiens gång.

Borlänge, 26 maj 2016

Erika Lööf
Colin Mack

Innehållsförteckning

Definitioner 1
1 Inledning 5
1.1 Bakgrund 5
1.2 Problemformulering 6
1.3 Syfte 7
1.4 Avgränsning 7
2 Teori 8
2.1 Begreppsgraf 8
2.2 Systemutvecklingsmodell 9
2.3 Continuous Integration 9
2.4 Continuous Delivery 9
2.4.1 Deployment pipeline 12
2.4.2 Automatiska tester 13
2.4.3 Verktyg för utveckling 13
2.4.4 Konceptuella modeller 15
2.6 Continuous Deployment 20
3 Metod 21
3.1 Forskningsprocessen 21
3.2 Litteraturstudie 23
3.3 Strategi 24
3.4 Datainsamling 25
3.4.1 Intervjuer 25
3.4.2 Dokument 27
3.5 Dataanalys 27
3.6 Forskningsetik 29
4 Sammanställning av förvaltningsuppdragen och nulägesanalys 30
4.1 Förvaltningsuppdrag A 30
4.2 Förvaltningsuppdrag B 33
5 SWOT Analys och Mognadsmodell 36
5.1 SWOT 36
5.1.1 SWOT tabeller 36
5.1.2 SWOT beskrivning av uppdrag A och B 38

5.2 Uppdragens mognad för Continuous Delivery 42
5.2.1 Uppdrag A mognad för Continuous Delivery 43
5.2.2 Uppdrag B mognad för Continuous Delivery 46
6. Diskussion och slutsats 49
6.1 Diskussion 49
6.2 Metodkritik 53
6.3 Slutsats 53
6.4 Kunskapsbidrag 53
6.5 Förslag på framtida studier 54
Källor .....
Bilaga 1 - Litteraturstudie, sökord .....
Bilaga 2 – Sökloggen .....
Bilaga 3 – Intervjufrågor .....

Tabell och Figurförteckning

Tabell 1 - Definitioner 1
Tabell 2 - SWOT tabell Uppdrag A 36
Tabell 3 - SWOT tabell Uppdrag B 37

Figur 1 - Begreppsgraf 8
Figur 2 - Deployment pipeline exempel (Humble & Farley, 2010, s.4) 12
Figur 3 - Versionshantering bild lånad från (Versionshantering, 2016, 2 maj) 14
Figur 4 - Stairway to Heaven, bild lånad från (Holmström Olsson, Alahyari, Bosch, 2012) 15
Figur 5 - Maturity Model (Rehn, Palmborg, & Boström, 2013) 19
Figur 6 - Modell över forskningssprocess. (Oates, 2006, s.33) 21
Figur 7 - Genomförande 22
Figur 8 - Processmodell, beskrivning av uppdrag A's releaseprocess 32
Figur 9- Processmodell, beskrivning av uppdrag B's releaseprocess 35
Figur 10 - Mognadsanalys, Uppdrag A 43
Figur 11 - Mognadsanalys, Uppdrag B 46

Definitioner

Tabell 1 - Definitioner

Acceptanstest	Dess uppgift är att se till att systemet möter kundens krav. (Holmström Olsson, Alahyari, Bosch, 2012)
Agila metoder	Lättrörligt arbetssätt, som kan hantera förändringar på ett bra sätt (Agile Sweden, 2016).
ASP.NET	”Används för att skapa dynamiska webbsidor och är utvecklat av Microsoft.”(ASP.NET, 2016)
Branch	el. gren som innehåller en modifierad kopia av projektet. (Versionshantering, 2016, 2 maj)
Build	En exekverbar artefakt kompilerad från källkod. (Pulkkinen, 2013)
C#	”ett objektorienterat programspråk utvecklat av Microsoft som en del av .NET-plattformen.”(C-sharp, 2016)
Continuous Delivery	Systemutvecklingsprocess där utvecklingsteamet producerar mjukvara i korta cykler och säkerhetsställer tillförlitliga releaser som kan släppas ut i produktion när som helst. Stegen i deploymentprocessen sker mestadels automatiskt från att utvecklaren checkar in kod. Slutet av processen innehåller manuella tester innan den distribueras ut i produktion. (Chen, 2015)
Continuous Deployment	Continuous Deployment (CDP) - Hanterar releaser av programkod där stegen mellan incheckning och distribution till

	produktion sker automatiskt. (Anderson, Kenyon, Hollis, Edwards, & Reid, 2014)
Continuous Integration	CI ses som en del av CD där fokus ligger på första steget i deploymentprocessen. Utvecklingsteamet checkar in sin kod kontinuerligt och integrerar den med mainbranchen. Utvecklarna får kontinuerligt feedback på om koden är körbar i mainbranchen vilket underlättar felsökning och rättning.(Meyer, 2014)
Commit	En commit lägger till den senaste ändringen till kälkoden. (Commit (version control), 2016, 24 maj)
Deploy	"The act of pushing a release into a given environment" (Swartout, 2012, s. 38)
Deployment pipeline	En delvis eller helt automatiserad implementation av en applikations build, deploy, test och releaseprocess. (Humble & Farley, 2010, s. 3)
Enhetstest	"Testnivå då systemets minsta beståndsdelar testas. Kallas även unittest, programtest och modultest" (Eriksson, 2008, s 377)
Feedbackloop	Channel or pathway formed by an 'effect' returning to its 'cause', and generating either more or less of the same effect. A dialogue is an example of a feedbackloop. (BusinessDictionary, 2016)
Förvaltning/Underhåll	"Aktiviteter för att hantera ett system efter leverans" Omfattar bl.a. kravhantering, testning, utveckling. (Eriksson, 2008, s. 383)

Incheckning	Se Commit
Integrationstest	“Testnivå för att visa att systemets komponenter fungerar ihop. Syftar till att hitta problem med gränssnitt och kommunikation mellan komponenterna”(Eriksson, 2008, s 376)
Källkod	”Instruktioner till dator, skrivna i ett programspråk” (IDG:s it-ord, 2016)
Leveranscykel	Tiden från att man bestämmer sig för att göra en förändring till att man har det tillgängligt för användarna. (Humble & Farley, 2010, s. 11)
Mainbranch	el. Trunk är ett begrepp för projektets huvudspår som innehåller all källkod.(Versionshantering, 2016, 2 maj)
Release	En release är en eller flera konfigurationsenheter som ska implementeras i produktmiljön. En konfigurationsenhet kan vara hårdvara, mjukvara eller dokumentation (Haverblad, 2007).
Releaseprocess	I detta arbete syftar releaseprocessen till stegen som en release tar, från en commit till en deploy hos kund.
Repository	En on- disk datastruktur som lagrar metadata för en uppsättning filer och / eller katalogstruktur(Repository, 2016,25 maj)
Regressionstest	”En metod inom programvarutestning som går ut på att testa hela eller delar av ett system, sedan ny funktionalitet har införts”. (Regressionstestning, 2016,25 maj)

Skript	“Ett som utför uppgifter i ett annat program”, “ Skript automatiserar arbetsuppgifter som annars skulle utföras steg för steg av användaren.” (IDG:s it-ord, 2016)
Systemtest	“Testnivå då det kompletta systemet testas på en övergripande detaljnivå. Här utförs både funktionella och icke funktionella tester”(Eriksson, 2008, s.380)
Testfall	“Strukturerat testunderlag som beskriver hur en funktion eller en egenskap ska testas. Innehåller bland annat teststeg och förväntat resultat”(Eriksson, 2008, s 381.)
Vattenfallsmodellen	En av de mest kända varianterna av sekventiell utveckling. Består av ett antal faser, dessa är oftast idé, analys, design, implementering, test. Vattenfallsmodellens grundtanke är att det som produceras i varje fas ska vara så bra att man inte ska behöva backa bak. (Eriksson, 2008, s 32)
WPF	” Windows Presentation Foundation (or WPF) is a graphical subsystem for rendering user interfaces in Windows-based applications by Microsoft.” (Windows Presentation Foundation, 2016)

1 Inledning

I det inledande kapitlet kommer vi behandla bakgrunden till vår studie som sedan efterföljs av en problemformulering som mynnar ut i vår frågeställning. Sedan presenteras syftet med studien och de avgränsningar vi gjort.

1.1 Bakgrund

Systemutveckling kan ske genom olika modeller. En av de äldsta och mest kända är vattenfallsmodellen där utvecklingen sker i sekventiella steg likt ett vattenfall (Balaji, Murugaiyan, 2012). Olsson m.fl. (2012) skriver att vid vattenfallsutveckling görs release till kunden först i slutet av projektet och feedback från kunden kommer därför inte förrän då. Många projekt har gått över till ett agilt arbetssätt, där projektet är lättrörligt och kan hantera förändringar under utvecklingens gång. Utvecklingen sker i iterativa steg där man lätt kan backa tillbaka och göra förändringar under projektets gång. (Balaji, Murugaiyan, 2012)

En del i det agila manifestet är att kontinuerligt kunna leverera funktionalitet till kund. Detta för att uppnå en snabb feedback och möjlighet till att förbättra utveckling och leverans av systemet. Men få företag har lyckats med en sådan implementering (Holmström Olsson, Alahyari, & Bosch, 2012).

Continuous Delivery (CD) är en metod inom systemutveckling där fokus ligger på att skapa en kort leveranscykel genom att automatisera delar av deployment pipeline vilket innehåller bygg-, deploy-, test- och releaseprocessen. CD bygger på att det alltid under pågående utveckling skall vara möjligt att automatiskt generera en release utifrån källkoden i dess aktuella tillstånd.(Humble & Farley, 2010) *“For a moment, imagine that your upcoming release could be performed with the push of a button”(Humble & Farley, 2010, s.21)*

En av CD:s många fördelar är att man genom kontinuerliga releaser får en snabb feedbackloop vilket leder till snabbare och effektivare införande av nya funktioner, samtidigt som man åtgärdar fel (Humble & Farley, 2010). En snabb feedbackloop gör att utvecklarna kan lägga tid på de funktioner som är användbara för verksamheten. Tidigare kom feedbacken på funktioner sent och bidrog till att onödigt mycket lagts på fel saker. Genom den snabba feedbacken vet du vilka funktioner som är värda all lägga mer tid på och hjälper utvecklarna att bygga rätt produkt. (Chen, 2015)

Även om CD har många fördelar finns det också flera utmaningar att hantera vid en övergång till CD.

> *“But it is not free; it is not painless. It is important to have the right level of sponsorship before you begin. A champion trumpeting the the horn for continuous delivery is a key starting point but you must garner buy-in from the executive and senior management teams.” (Neely & Stolt, 2013)*

En utmaning som finns är att det krävs resurser för implementering av CD. Det är därför viktigt att ha med sig ledningen så man får det stöd som krävs. Leppänen m.fl. (2015) skriver att det kan vara en utmaning att få med sig ledningen om stora förändringar i verksamheten krävs.

En konceptuell modell har skapats för att identifiera vart en organisation befinner sig mognadsmässigt för en övergång till Continuous Delivery. Den kallas Maturity model. Modellens avsikt är att ge struktur och förståelse till viktiga aspekter som måste övervägas vid övergången till CD. Den innehåller fem olika kategorier som representerar viktiga aspekter som behöver övervägas vid implementation av CD. Dessa kategorier är: Kultur och Organisation, Design och Arkitektur, Byggen och Deploy, Test och Verifikation, Information och Rapportering.(Rehn, Palmborg, & Boström, 2013)

Samarbetspartner

Vår samarbetspartner Triona är ett It-företag som kombinerar stort verksamhetskunnande inom framför allt transportinfrastruktur, trafik, transporter, skogsindustri och energi-/fordonsindustri med spetskompetens inom systemutveckling och systemförvaltning.
De är etablerade i Norge och Sverige, omsätter cirka 140 MSEK och har ca 130 medarbetare. Några exempel på kunder är ABB, NCC, MaserFrakt, NWP, Ramböll, SDC, SCA, Sveaskog, Statens vegvesen i Norge, Trafikverket och Örnfrakt. Triona har många uppdrag och är nyfikna på vad Continuous Delivery är och om det är intressant för dem och deras förvaltningsuppdrag.

När en första release har släppts i produktion räknas uppdraget till ett förvaltningsuppdrag. Då arbetar man enligt Nordström och Welander(2007) med:

Vidmakthållande
- Felrättningar, användarstöd och daglig drift och underhåll.

Vidareutveckling
- Anpassningar och förbättringar.

1.2 Problemformulering

Utifrån vår litteraturstudie har det framkommit att det finns olika utmaningar när det gäller en övergång till Continuous Delivery. Dessa utmaningar kan skilja sig beroende på vilken mognadsnivå förvaltningsuppdraget befinner sig i och vilka styrkor och svagheter uppdraget har. Om man vill gå över till CD i ett förvaltningsuppdrag kan det vara bra att veta vilka utmaningar man kan stå inför.

Detta leder till vår frågeställning med tillhörande delfrågor:

“Vilka utmaningar kan ett förvaltningsuppdrag stå inför vid en övergång till Continuous Delivery?”

Delfrågor:

1. Vilka styrkor och svagheter finns i de studerade förvaltningsuppdragen vid en övergång till Continuous Delivery?
2. Vilken mognadsnivå befinner sig de studerade förvaltningsuppdragen vid en övergång till Continuous Delivery?

1.3 Syfte

Syftet med denna studie är att få en ökad förståelse för Continuous Delivery samt vilka utmaningar förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery.

1.4 Avgränsning

Då detta är en fallstudie där vi studerar två fall på djupet kommer vi avgränsa oss till utmaningar som berör de specifika fallen. De utmaningarna som kommer ingå i vår studie är de som rör releaseprocessen och kategorierna “Byggen och Deploy” och “Test och Verifikation” i Maturity model.

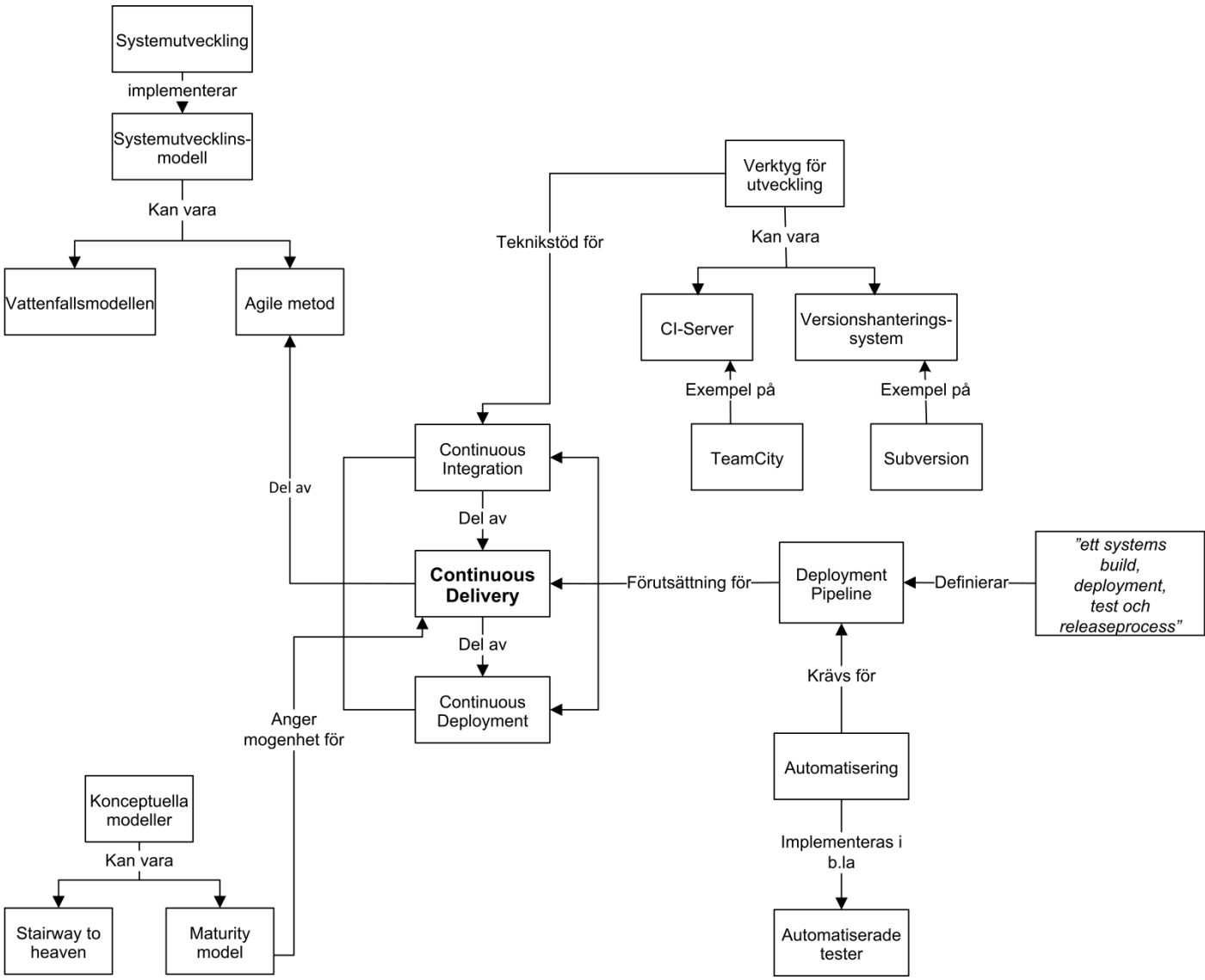
I denna studie ingår inte att presentera lösningar på de utmaningar som hittas.

2 Teori

I det här kapitlet tar vi upp teori som beskriver Continuous Delivery och begrepp i kring ämnet, samt några konceptuella modeller som använts i studien. Detta för att få en teoretisk referensram som grund för att kunna svara på frågeställningen ” Vilka utmaningar kan ett förvaltningsuppdrag stå inför vid en övergång till Continuous Delivery?”. Kapitlet inleds med en överblick av teorin i form av en begreppsgraf (se figur 1).

2.1 Begreppsgraf

Denna modell är sammanställd av oss och är baserad på den teori vi presentetar i teorikapitlet. Detta för att tydliggöra hur begreppen hänger ihop och är länkade till varandra.



Figur 1 - Begreppsgraf

2.2 Systemutvecklingsmodell

Systemutveckling kan ske genom olika modeller. En av de äldsta och mest kända är vattenfallsmodellen där utvecklingen sker i sekventiella steg likt ett vattenfall. (Balaji, Murugaiyan, 2012).

Dagens systemutveckling bedrivs i en turbulent omvärld. Snabbt föränderliga och oförutsägbara marknader, komplexa kundkrav som kan ändras under tiden samt en ökad takt på utveckling av nya teknologier. För att hantera detta är en agil utvecklingsmodell passande. (Holmström Olsson, Alahyari, & Bosch, 2012).

I en agil utvecklingsmodell är utvecklingen är lättrörlig och kan hantera förändringar under utvecklingens gång. Utvecklingen sker i iterativa steg där man lätt kan backa tillbaka och göra förändringar under uppdragets gång. (Balaji, Murugaiyan, 2012)

2.3 Continuous Integration

Continuous Integration (CI) är en del av agil systemutveckling. Teamets utvecklare checkar in sin kod i projektets centrala repository när en ändring eller en ny funktion är färdig. På så sätt integreras allas arbete på samma ställe och man kan upptäcka fel eller integrationsproblem i ett tidigt skede.(Abrantes & Travassos, 2011)

Efter incheckning körs ett CI system som försöker kompilera och bygga koden. Om bygget blir lyckat går det vidare till enhets- och integrationstest. Skulle bygget gå fel meddelas utvecklaren. Hela idén med CI är att upptäcka problem som uppstår vid integration av kod i ett så tidigt skede som möjligt och ge feedback tillbaka till utvecklarna. (Pulkkinen, 2013)

2.4 Continuous Delivery

Continuous Delivery (CD) är en metod inom systemutveckling där fokus ligger på att skapa en kort leveranscykel genom att automatisera delar av deployment pipeline vilket innehåller bygg-, deploy-, test- och releaseprocessen. CD bygger på att det alltid under pågående utveckling skall vara möjligt att automatiskt generera en release utifrån källkoden i dess aktuella tillstånd.(Humble & Farley, 2010)

“For a moment, imagine that your upcoming release could be performed with the push of a button”(Humble & Farley, 2010, s.21)

Består av stegen Continuous Integration men inkluderar också automatiska deployment till testmiljöer för automatiserade acceptans och kapacitets tester(Humble & Farley, 2010). Med Continuous Delivery (CD) så är det möjligt att till exempel utföra manuella utforskande tester eller acceptanstester med användare innan en release till produktionsmiljön.

Genom att automatisera återupprepbara delar i releaseprocessen kan du undvika problem och fel som orsakas av den mänskliga faktorn.(Humble & Farley, 2010)

CD handlar om att när som helst under utvecklingen kunna släppa en release som är redo för produktion genom ett knapptryck (Humble & Farley, 2010). Neely och Stolt (2013) skriver att CD handlar om att kunna släppa en release när vi vill. Det är inte hur ofta som är den avgörande faktorn. Det är förmågan att släppa när man vill.

Nyckelord för Continuous Delivery
Humble och Farley (2010) tar upp två viktiga nyckelord som är viktiga för CD, *automatiskt* och *frekvent*.

Automatiskt
Om deployment pipeline inte är automatiserad är den inte återupprepbar. Varje körning blir olika. Eftersom stegen är manuella är det stor risk för fel och det är svårt att utvärdera vad som gjorts. Man menar att man inte har någon kontroll över releaseprocessen och därför också svårt att säkerställa kvalitet. “Releasing software is too often an art; it should be an engineering discipline” (Humble & Farley, 2010, s.12).

Frekvent
Om releaser är frekventa kommer skillnaderna mellan releaserna vara små. Detta leder till att risken minskar betydligt med att släppa releaser och gör det mycket lättare att dra tillbaka en release. Frekventa releaser leder också till snabbare feedback som är en stor del i CD (Humble & Farley, 2010, s.12).

Humble och Farley (2010) säger att det är viktigt att man checkar in sin kod ofta till mainbranchen för att uppnå Continuous Deliverys fördelar. De säger också att genom frekventa incheckningar blir kodändringarna mindre och du minskar risken för konflikter med andra utvecklares kod.

> *”The key to fast feedback is automation” (Humble & Farley, 2010, s.14)*

Humble och Farley (2010) skriver att nyckeln till snabb feedback är automation. Vid manuella processer är man beroende av människor. Människor ska inte utföra manuella byggen, tester och deploymentprocesser. Människor är dyra och värdefulla, de ska koncentrera sig på att utveckla ett system som är till nöje för användaren och kunna leverera dessa ”nöjen” så fort som möjligt. Många steg i processen kan göras automatiska och utföras av en maskin. Men att implementera en deployment pipeline är något som kräver resurser och speciellt när det gäller de automatiska testerna. Ett av målen är att du frigör människor att göra någonting annat.

> *”We want to free peolpe to do the interesting work an leave repetition to machines” (Humble & Farley, 2010, s. 14)*

Utmaningar
Även om CD har många fördelar finns det också flera utmaningar att hantera vid en övergång till CD.

> *“But it is not free; it is not painless. It is important to have the right level of sponsorship before you begin. A champion trumpeting the the horn for continuous delivery is a key starting point but you must garner buy-in from the executive and senior management teams.” (Neely & Stolt, 2013)*

Chen (2015) skriver i en artikel om en verksamhet där en övergång till Continuous Delivery skett. I artikeln tar han upp de utmaningar som de stött på. Dessa delar han upp i kategorierna organisatoriska utmaningar, processutmaningar och tekniska utmaningar.

Chen (2015) skriver att de vid implementering av CD upplevde en organisatorisk utmaning. Att det kan skapas målkonflikter mellan avdelningar. *“Release activities involve many divisions of the company. Each has its own interests,ways of working, and perceived territories of control” (Chen, 2015).* För att lösa den utmaningen gjorde ledningen en omstrukturering av hela organisationen för att bryta barriärer som fanns mellan avdelningarna och jobbade för ett samarbete mellan avdelningarna istället.

Utmaningar gällande processer i verksamheten togs också upp. Chen (2015) skriver att traditionella processer hindrar CD. Han ger exempel på att en funktion som är redo för release normalt måste gå igenom en ändringshanteringsprocess. Detta kan fördröja releasen upp till fyra dagar. Om en funktion endast tar ett par dagar från utformning tills att den är redo för release är de extra fyra dagarna för mycket tid som drar ner hela CD processen. ”...this four-day period accounts for too much of the feature’s total cycle time” *(Chen, 2015)*

En teknisk utmaning är att det i nuläget inte finns någon robust standardlösning för att stödja CD. Chen (2015) skriver att de utvecklade en egen lösning vilket var kostsamt. Han skriver även att alla applikationer inte är passande för CD. Han syftar då på stora monolitiska applikationer.

En annan utmaning som finns är att det krävs resurser för implementering av CD. Det är därför viktigt att ha med sig ledningen så man får det stöd som krävs. Leppänen m.fl. (2015) skriver att det kan vara en utmaning att få med sig ledningen om stora förändringar i verksamheten krävs.

Leppänen m.fl. (2015) nämner att en av de primära utmaningarna är att kunden måste vilja och dessutom ha möjlighet att hantera fler releaser än vad de har i nuläget.

Utmaningar som vi studerar i detta arbete är de som berör releaseprocessen och kategorierna “Byggen och Deploy” och “Test och Verifikation” i Maturity model. Utmaningar inom detta tas inte upp i tidigare forskning. Även om Chen tar upp processutmaningar så syftar han till större processer i verksamheten där även ändringshanteringsprocessen är med.

2.4.1 Deployment pipeline

Deployment pipeline är en delvis eller helt automatiserad implementation av ett systems build, deployment, test och releaseprocess. Denna pipeline kan skilja sig mellan olika organisationer men principen för de olika stegen är densamma. När en incheckning sker av en ny funktionalitet triggar den igång en ny instans av pipelinen. Pipelinen består av olika steg av tester. Om den nya funktionaliteten klarar sig igenom alla steg är den redo för release. (Humble & Farley, 2010, s. 3)

Enligt Humble & Farley(2010) finns det tre huvudsyften med en deployment pipeline.

1. Genom att göra processen synlig för alla inblandade förbättras samarbetet.
2. Skapar bättre förutsättningar för feedback, så att problem kan identifieras och lösas tidigt i processen.
3. Skapar möjlighet att släppa releaser när man vill genom en automatiserad process.

Ett exempel (figur 2) på hur deployment pipeline i Continuous Delivery kan se ut

Figur 2 - Deployment pipeline exempel (Humble & Farley, 2010, s.4)

Commitsteget
Är det första steget i processen när utvecklaren checkar in sin kod. Systemet kontrolleras så att det fungerar utifrån en teknisk nivå. Koden kompileras, enhetstestas och analyseras. (Humble & Farley, 2010, s.110)

Acceptanstest
Efter commitsteget utförs acceptanstest där man kontrollerar att systemet fungerar på en funktionell och icke funktionell nivå.(Humble & Farley, 2010, s.110) Testets uppgift är att se till att systemet möter kundens krav. (Holmström Olsson, Alahyari, Bosch, 2012)

Kapacitetstest
Nästa steg i processen är att kontrollera systemets prestanda. Detta kan göras parallellt med de manuella testerna. Resultatet av testerna bedöms baserat på ett mänskligt beslut utifrån om det är acceptabelt. (Oktaba, 2015)

Manuella test
Detta steg kontrollerar att systemet uppfyller krav, upptäcker fel som automatiska testerna inte fångat, och verifierar att systemet ger ett värde till användarna. Manuella testerna inkluderar utforskande tester, integrations tester och acceptanstester där användarna är delaktiga. (Humble & Farley, 2010, s.110)

Release
Sista steget i deployment processen levereras system till användarna, antingen som en förpackad mjukvara eller att den installeras direkt in i produktion eller till en testmiljö som är identiskt med produktionsmiljön. (Humble & Farley, 2010, s.110)

2.4.2 Automatiska tester

Eriksson beskriver automatisering av test som "Processen att skriva testprogram som utför teststeg och kontrollerar resultatet"(2008). Det vill säga att man gör om de manuella testfallen till en automatisk process. Automatiserade tester ger störst nytta när test kan återanvändas och kan upprepas flera gånger. (Dustin, Rashka, & Paul, 1999)

Det är ofta billigare att använda sig av automatiska tester i samband med förvaltning jämfört med nyutveckling. Systemet och kraven är mer stabila och andelen ny kod är liten i varje release. (Eriksson, 2008)

Automatisering är en förutsättning för deployment pipeline. För det är bara genom automatisering du kan garantera en ändamålsenlig leverans genom ett knapptryck. Humble och Farley (2010) nämner att du inte behöver automatisera allt på en gång. Börja med dem delar i processen som är trögast och automatisera delarna stegvis över tid.

2.4.3 Verktyg för utveckling

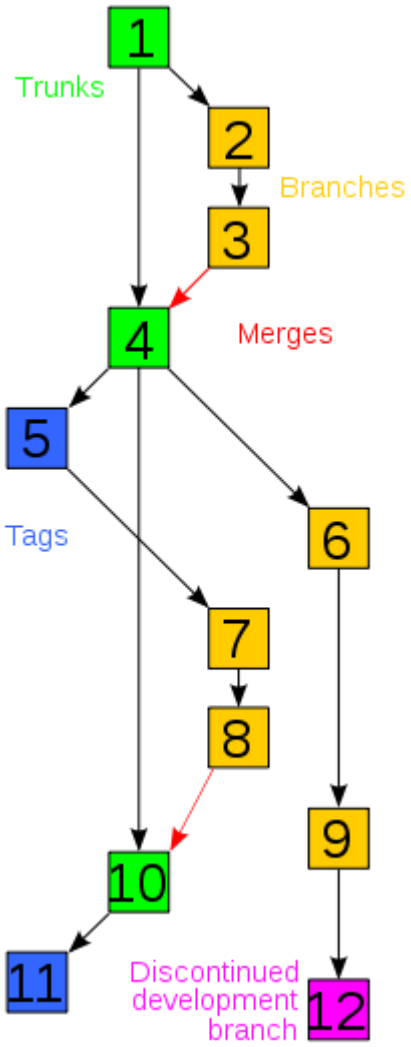
Continuous integration server (CI-server)

En continuous integration server är ett verktyg som kan användas av utvecklingsteam. Dess syfte är att övervaka kodbasen, och se om förändringar skett och sedan trigga ett bygge. (Meyer, 2014) Ett exempel på en CI-server är Teamcity som också har stöd för automatiska test och deployer till flera miljöer. Jet Brains skriver själva på sin webbsida -" Initially a Continuous Integration server, TeamCity has encompassed all the features you expect from a mature Continuous Deployment platform" (Jet Brains, 2016).

Versionshanteringssystem

Ett versionshanteringssystem hanterar källkoden och håller reda på flera versioner av dina filer. Om du ändar en fil har du fortfarande tillgång till den föregående versionen av den. Versionshanteringssystemet gör det även möjligt för fler utvecklare att arbeta på samma uppdrag utan att "förstöra" för varandra. Om du skulle vilja gå tillbaka till en tidigare version av koden är det möjligt tack vare versionshanteringen. Det är även möjligt att se historik i vad som gjorts, av vem och varför vilket förenklar felsökning i koden. Ett exempel på ett versionshanteringssystem är Subversion.(Humble & Farley, 2010, s.32)

Figur 3 ger ett exempel på hur en struktur kan se ut i ett versionshanteringssystem.



Figur 3 - Versionshantering
bild lånad från (Versionshantering, 2016, 2 maj)

Trunk – (eller master branchen) är projektets huvudspår.

Branch – (eller gren) är en modifierad kopia av projektet som kan föras tillbaka in i huvudprojektet. Detta kan användas för att arbeta separat utan att branch och trunk påverkar varandra.

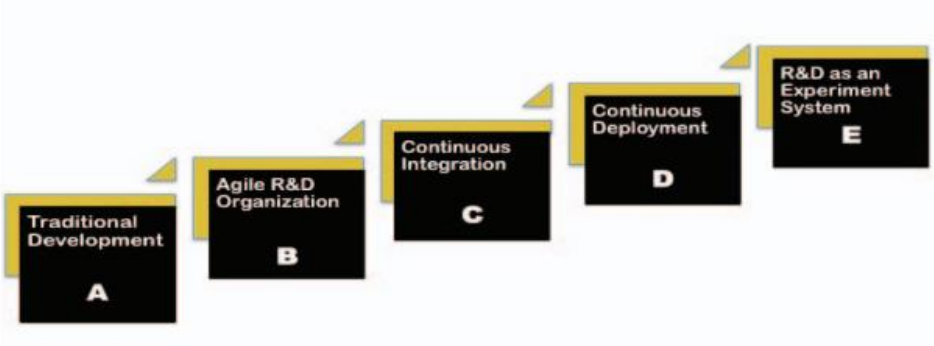
Merge- En punkt där en branch förs ihop med trunken.

Tag- Används för att strukturera förändringar i projektet och markerar upp ett särskilt tillstånd i koden vi vid en viss tidpunkt.(Versionshantering, 2016, 2 maj)

2.4.4 Konceptuella modeller

Under denna rubrik kommer två konceptuella modeller presenteras. Stairway to heaven (se 2.9.1) som tagits fram av Helena Holmström Olsson, Hiva Alahyari och Jan Bosch 2012. Den visar de steg som måste tas för att uppnå Continuous Deployment. Även fast Stairway to heaven har Continuous Deployment som mål är denna modell användbar för vår studie, då Continuous Delivery är ett steg på vägen till Continuous Deployment. Maturity model (se 2.9.2) är en modell som används för att se i vilken mognadsnivå uppdraget befinner sig för en övergång till Continuous Delivery.

Stairway to heaven
I en studie som gjorts för att belysa de steg som måste tas för att övergå till Continuous Deployment har en konceptuell modell tagits fram. Den visar de stegen i form av en trappa. De kallar den "stairway to heaven"(Holmström Olsson, Alahyari, Bosch, 2012).



Figur 4 - Stairway to Heaven, bild lånad från (Holmström Olsson, Alahyari, Bosch, 2012)

De beskriver "trappan" (figur 4) som den typiska utvecklingsvägen för företag som ska röra sig mot Continuous Deployment.

A. Traditionell Utveckling
Utveckling karaktäriserad av långsamma utvecklingscykler som till exempel Vattenfall. Vanligtvis är projekt teamen stora och kompetensen uppdelad i olika delar så som system arkitektur, design och test. Utvecklingen sker i sekventiella steg med en start av en omfattande planeringsfas inför varje projekt, Typiskt för denna approach är att leverans till kunden görs först i slutet av projektet och feedback från kunden kommer därför inte fören då.

B. Agile R&D (Research & development)
Nästa steg i utvecklingen är när systemutvecklingsteamet arbetar agilt men att produktstyrningen och test- och kvalitetssäkringen jobbar enligt traditionella utvecklingsmetoder. Även om det finns stora fördelar med att arbeta agilt är det inte säkert att feedback från kunden sker tillräckligt snabbt.

C. Continuous Integration
Ett företag som använder continuous integration har lyckas etablera rutiner som möjliggör frekvent integration av arbete, dagliga byggen och snabba commits till exempel automatiska byggen och automatiserade tester. Automatiska testfall, byggen och kompilering tillåter

utvecklingsteam att testa och integrera koden på en daglig basis vilket minimerar tiden det tar från att en idé tillkommer till att idén implementeras i programvaran. Vid det här "steget" arbetar både produktstyrningen och test- och kvalitetssäkringen enligt agila metoder.

D. Continuous Deployment
I detta steg sker distributionen av programfunktioner mot kunden kontinuerligt eller åtminstone mer frekvent. Detta ger kontinuerlig feedback från kunden och kan bidra till att man kan lägga mer tid på att utveckla funktioner som kunden har användning av och mindre tid på det som inte producerar värde för kunden.

E. R&D som ett "experiment system"
Sista steget på trappan (figur 4) är när hela R&D systemet svarar och agerar på direkt feedback från användarna och där den aktuella distributionen av programfunktioner ses som en typ av experiment och test av användarnas behov. På detta "steg" ses distributionen av programfunktioner mer som startpunkt för framtida finjusteringar av funktionalitet snarare än leverans av den slutliga produkten.

Maturity model
Maturity model är en mognadsmodell som används för att identifiera vart en organisation befinner sig mognadsmässigt för en övergång till Continuous Delivery. Modellens avsikt är att ge struktur och förståelse till viktiga aspekter som måste övervägas vid övergången till CD (Rehn, Palmborg, & Boström, 2013). Nedan beskrivs modellens fem nivåer och kategorier. Modellen finns i sin helhet på sida 19 (figur 5).

Fem nivåer
Modellen innehåller fem mognadsnivåer: Bas(base), Nybörjare (beginner), Medel (intermediate), Avancerad (advanced) och expert (expert). Det typiska medelvärdet är bas-nivån, där de flesta organisationer ligger idag. Vissa organisationer faller in under nybörjar-nivån i vissa kategorier och andra ligger till och med under bas-nivån (utanför modellen). Medel-nivån är den mognadsnivå man ser som en mogen nivå för Continuous Delivery där det anses att man kan börja dra nytta av de större effekterna. I de två sista nivåerna avancerad och expert kan man dra ännu större av effekterna (Rehn, Palmborg, & Boström, 2013). De två sistnämnda nivåerna beskrivs inte i teorikapitlet eftersom dessa nivåer inte är relevanta för studien.

Dessa nivåer är inte strikta och de måste inte följas stegvis, de fungerar mer som en bas för utvärdering och planering. Det är viktigt att hålla den övergripande mognadsnivån jämn över kategorierna och även vara beredd på att stora förändringar kan skapa misstro och ovillighet i organisationen, så man rekommenderas därför att ta ett steg i taget.(Rehn, Palmborg, & Boström, 2013)

Fem kategorier
Modellen definierar fem kategorier som representerar viktiga aspekter som behöver övervägas vid implementation av CD. Varje kategori har sin egen mognadsprogression, men typiskt för en organisation är att de mognar i flera kategorier samtidigt då de dessa kategorier är kopplade till varandra i organisationen.

Kultur & Organisation

Organisationen och dess kultur är en av de viktigaste aspekterna att överväga när man siktar på att skapa en hållbar Continuous Delivery miljö som drar nytta av de resulterade effekterna. Denna kategori har sitt fokus på arbetssättet och processerna i organisationen och hur dessa utförs påverkar en övergång till CD. En grund för att uppnå basnivån i denna kategori är att utvecklingsteamet måste prioritera arbete i backlog samt göra frekventa commits. För att uppnå en högre nivå krävs det dels att organisationen använder sig av ett agilt arbetssätt samt att det endast finns en backlog per utvecklingsteam.

Design & Arkitektur

Designen och arkitekturen av systemen har en stor påverkan på förmågan att övergå till Continuous Delivery. Ett system som är byggt med Continuous Delivery principer och ett snabbt ”releasetänk” från början, bidrar till en mycket smidigare resa. Att helt designa om hela system är inget attraktivt val för de flesta organisationer. För att uppnå basnivån så krävs det att organisationen har enskilda system för utveckling, bygge av system och release men att de börjat sträva efter ett gemensamt. För de högre nivåerna ska system först organiseras till moduler och sedan till komponenter som kan deployas individuellt. Det ska även finnas möjlighet att dölja funktioner som inte är färdigutvecklade samt att ingen eller minimal branchning ska användas.

Byggen & Deploy

Denna kategori är en viktig del för CD och det är här verktyg och automatisering träder in i “pipelinen”. Ett första intryck av en typisk “mature delivery pipeline” kan vara överväldigande beroende på hur mogen organisationen är i den nuvarande build och deployment processen. Denna kategori har sitt fokus på hanteringen av byggen och hur dessa deployas. En grund för att uppnå första nivån är att utvecklingsteamet använder sig av en versionshanterad kodbas och att det sker automatiska byggen. För de högre nivåerna krävs det dels att frekventa byggen görs samt att dessa arkiveras. För att uppnå medelnivån som anses vara den nivå man anses mogen för CD krävs de att varje commit ska trigga igång ett bygge, att byggda artefakter kan deployas i alla typer av miljöer samt att deployment pipeline har alla steg för att släppa en release till produktion.

Test & Verifikation

Test är en livsviktig del när implementation av CD ska ske. I denna kategori likt Build & Deploy involveras verktyg och automatisering. Det är också viktigt att konstant öka omfattningen på test av systemet för att kunna släppa frekventa releaser med gott samvete. Test involverar att verifiera förväntad funktionalitet utifrån krav men också att de nya funktionerna uppfyller de förväntade effekterna för verksamheten. För att uppnå basnivån krävs de automatiska enhetstester samt att organisationen använder sig av separata testmiljöer. För de högre nivåerna krävs de automatisering av integrationstester och komponenttester samt att delar av acceptanstestesterna körs automatiskt.

Information & Rapportering

Alla verksamhetsprocesser inkluderar specifik information som behöver hanteras och delges. Processen med utveckling av ett system är inget undantag. Information som typiskt hanteras vid denna typ av process är koncept så som komponenter, krav, versioner,

utvecklare, releaser och miljö. Denna kategori visar att hantering av information måste ske på ett korrekt sätt när en övergång till CD ska ske. Informationen måste vara koncis, relevant och åtkomlig vid rätt tidpunkt till rätt person för att uppnå den maximala effekten som är möjlig med CD. Bortsett från att information ska användas för att uppfylla affärskrav vid utveckling av nya funktioner är de också viktig att ha åtkomst till information som behövs för att mäta processen och ständigt förbättra den. En grund för att uppnå basnivå är att organisationen använder sig av grundläggande mätetal för processen samt manuell rapportering. För de högre nivåerna krävs de dels schemalagda kvalitetsrapporter, automatiskt rapportering och feedback samt att rapporteringshistorik är tillgänglig.

Figur 5 - Maturity Model (Rehn, Palmborg, & Boström, 2013)

2.6 Continuous Deployment

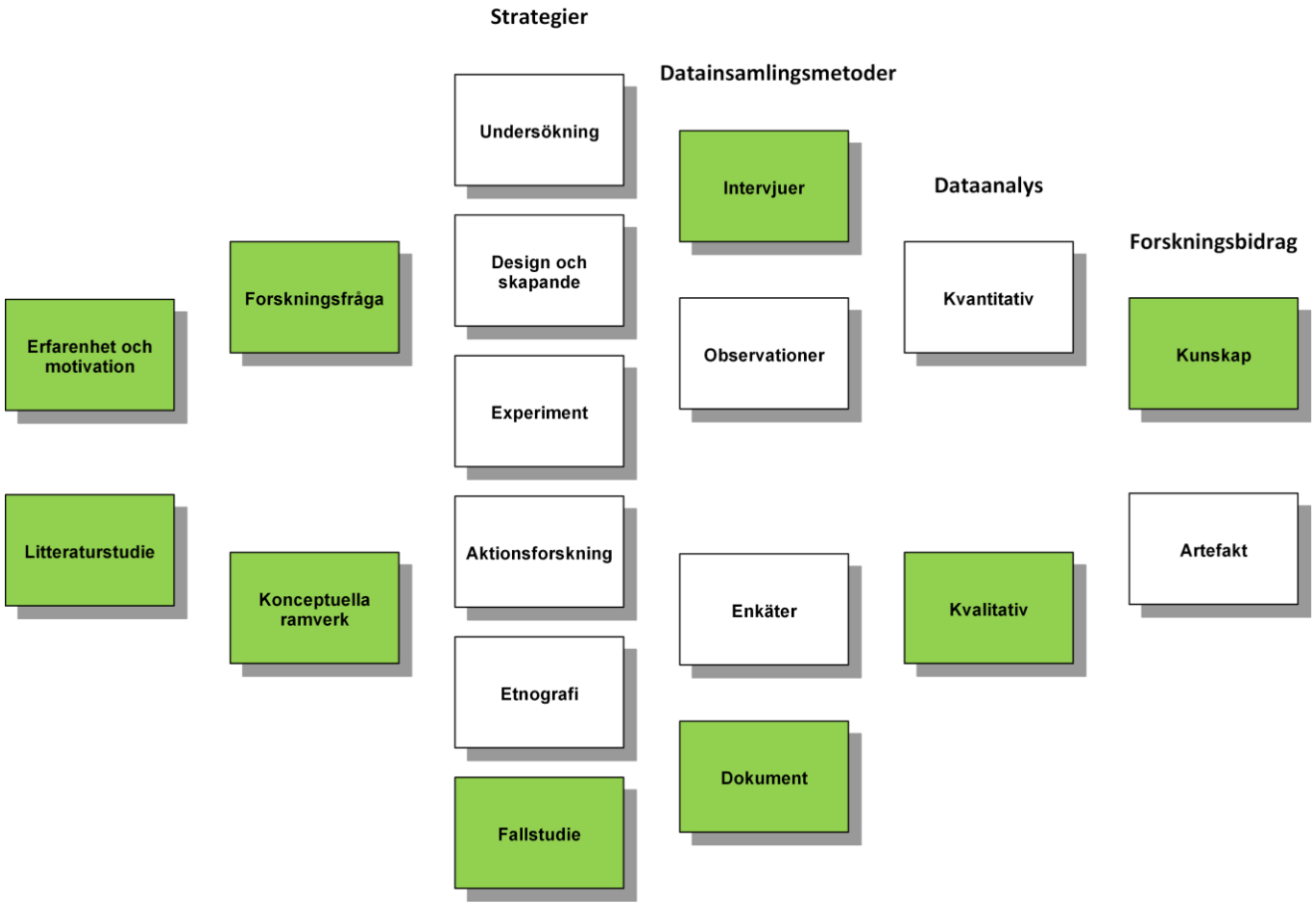
Continuous Deployment är något som ibland förväxlas med Continuous Delivery. För att uppnå till Continuous Deployment måste hela dess deployment pipeline vara automatisk, från att utvecklaren gör en commit till att en release går ut i produktion. Varje incheckning som kommer igenom de automatiska testerna genererar en release ut i produktionsmiljö. Alla steg sker automatisk inklusive acceptanstester som kontrollerar funktionella och icke-funktionella krav. För att detta ska fungera krävs det att de automatiska testerna håller en hög kvalitet. (Humble & Farley, 2010)

3 Metod

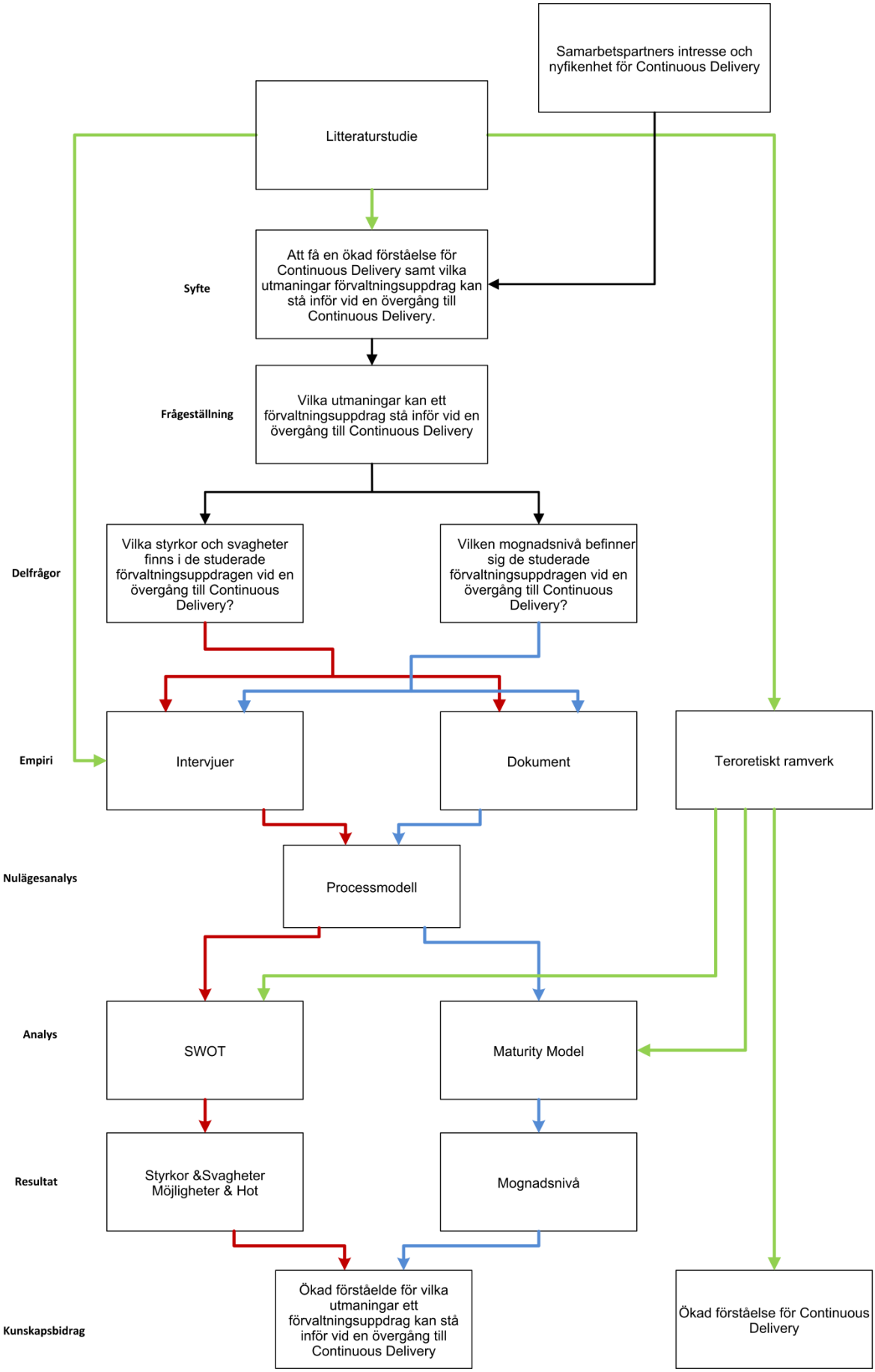
I det här kapitlet kommer vi beskriva vår forskningsprocess, litteraturstudie, strategi, datainsamling och dataanalys. Vi har använt oss av strategin fallstudie och datainsamlingsmetoderna intervjuer och dokument. Metodkapitlet beskriver även hur litteraturstudie och dataanalys genomförts.

3.1 Forskningsprocessen

Figuren nedan (figur 6) illustrerar forskningsprocessen och de steg vi genomfört under vårt examensarbete. Ett första steg i vår forskningsprocess var att utifrån vår egen erfarenhet och motivation göra en litteraturstudie för att få bättre förståelse för ämnet. Litteraturstudien finns beskriven i avsnitt 3.2. Utifrån litteraturstudien kunde vi formulera vår forskningsfråga med delfrågor (se avsnitt 1.2)och fick grunden till ett konceptuellt ramverk för studien. För att kunna besvara vår forskningsfråga använde vi forskningsstrategin fallstudie vilket beskrivs i avsnitt 3.3. För att få de empiriska data vi behövde användes intervjuer och dokument som datainsamlingsmetoder (se avsnitt 3.4). Vidare följde en kvalitativ analys av det data vi samlat in, hur detta genomfördes beskrivs i avsnitt 3.5. Forskningsbidraget för detta examensarbete är kunskap om Continuous Delivery samt vilka utmaningar ett förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery.



Figur 6 - Modell över forskningsprocess. (Oates, 2006, s.33)



Figur 7 - Genomförande

3.2 Litteraturstudie

Litteraturstudien sätter grunden för hela forskningsarbetet. Den delas upp i två delar. Första delen handlar om att söka i litteratur efter passande forskningsidéer och för att upptäcka relevant material inom ett specifikt ämne. Detta hjälper forskaren att sätta sig in i ämnet och definiera en forskningsfråga. När forskningsfrågan är definierad börjar andra delen av litteraturstudien. Där granskar man de källor man valt. Den delen pågår sedan under hela forskningsarbetet. (Oates, 2006)

Vi fick en forskningsidé presenterad till oss av vår samarbetspartner Triona. De har många uppdrag och var nyfikna på vad Continuous Delivery är och om det är intressant för dem och deras förvaltningsuppdrag.

Vår första del av litteraturstudien syftade till att sätta in oss i ämnet och formulera en specifik forskningsfråga som syftade till ett vetenskapligt bidrag. Vidare har litteraturstudien fortgått och hjälpt oss skapa ett konceptuellt ramverk till vår studie. Det har bidragit till vår bakgrund, teori och intervjufrågor. (figur 7)

Vår litteraturstudie startade med att leta efter tidigare studier och relevanta artiklar kring ämnet. Oates (2006) säger att internet och digitala databaser är en bra resurs för forskare att hitta information. Vi använde oss av Googles vetenskapliga sökmotor Google Scholar, Summon (Högskolan Dalarnas söktjänst) och Libris databas för examensarbeten för sökningar. Sökorden som användes var framförallt "continuous delivery", "continuous delivery" + implement och "continuous delivery" + development. Se bilaga 1 för samtliga sökningar och träffar.

Vi har även hittat äldre artiklar genom att utgå från en annan artikels referenser. Det kallas för backward search. (Avdic, 2016) Eftersom de flest artiklar vi studerat varit relativt nya har en backward search varit givande.

För att göra urval i de källor vi hittat började vi med att kolla på titel. Sedan lästes sammanfattning igenom för att få en uppfattning om innehållet i källan. Om detta var intressant gick vi vidare och läste resterande delar. Vi tittade på vilka referenser den aktuella källan använt sig av och försökte göra en bedömning utifrån vår egen erfarenhet om källan var tillförlitlig. De källor vi valt ut som relevanta för vårt examensarbete sammanställde vi i en söklogg där det framgår vart och hur källan hittades. Det framgår också en kort sammanfattning rörande innehållet i källan. Detta för att på ett systematiskt sätt få en övergripande bild av källorna. Sökloggen som helhet finns som bilaga 2.

Vi har bedömt att de artiklar vi använt i vårt examensarbete är tillförlitliga eftersom de varit antingen peer-reviewed, vetenskapligt framtagna eller på annat sätt relevanta för studien som att de blivit citerat många gånger från andra forskare. Vi har genom vår litteraturstudie sett flera av våra källor återkomma i andras arbeten inom ämnet, vilket stärker våra källor ytterligare.

3.3 Strategi

Vi har valt att använda oss av forskningsstrategin fallstudie för att genomföra vårt examensarbete. I en fallstudie studerar man en eller flera instanser av det fenomen man vill undersöka, till exempel en organisation, en avdelning eller ett projekt. Denscombe (2016) skriver att fallstudier fokuserar på en eller några få förekomster av ett särskilt fenomen för att få en djup redogörelse för händelser, förhållanden, erfarenheter eller processer som förekommer i detta särskilda fall.

Vår fallstudie innefattade att studera två förvaltningsuppdrag och undersöka vilka utmaningar de har för en övergång till Continuous Delivery. Vår samarbetspartner Triona (se avsnitt 1.1 Samarbetspartner) är ett It-företag som framförallt fokuserar på transportinfrastruktur, trafik, transporter, skogsindustri och energi-/fordonsindustri med spetskompetens inom systemutveckling och systemförvaltning. Vi har studerat två av samarbetspartnens uppdrag inom systemförvaltning.

- Förvaltningsuppdrag A förvaltar en molntjänst. Tjänsten är framtagen först och främst för sågverk i Sverige. Den hanterar upphandlingar och beställningar av transporter. Allt detta sker genom webben och tjänsten som uppdraget förvaltar. Tjänsten består bland annat av ett grafiskt gränssnitt skrivet i ASP.NET Web Forms, C# för web och tillhandahåller funktionalitet för lasshantering. Tjänsten har även ett grafiskt gränssnittför mobil. Även den är skriven i ASP.NET Web Forms, C#. Den tillhandahåller funktionalitet för lasshantering såsom boka och tacka nej för leverantörer.

- Förvaltningsuppdrag B är ett uppdrag som är på väg in i förvaltning av ett system som stödjer verksamhetsprocesser i transport- och anläggningsbranschen från avtalshantering, orderhantering, planering, operativ ledning och uppföljning. Systemet innehåller flera moduler varav en kontorsklient som är riktad till transportledare. Kontorsklienten kräver fast installation och innehåller 80 % av hela systemets funktionalitet. Den är skriven i .NET C# med WPF, för att möjliggöra administration av order. Består av en skalapplikation och ett för kunden antal valbara moduler för olika funktionalitet.

Dessa uppdrag valdes ut bland flera som tänkbara kandidater för en övergång till Continuous Delivery hos samarbetspartnern. Samarbetspartner var nyfikna på att ta reda på mer om vad Continuous Delivery är och vilka utmaningar uppdragen kan stå inför.

Fallstudie är en passande strategi för vårt examensarbete då vi studerat två förvaltningsuppdrag på djupet för att få en ökad förståelse för vilka utmaningar förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery. Enligt Oates (2006) finns tre olika typer av fallstudier. Utforskande, beskrivande, och förklarande fallstudier. Vår fallstudie är av typen beskrivande fallstudie. Fallen kan ses som en del av en beställd undersökning(Denscombe, 2016), då vi blev tillfrågade att utföra dessa av vår samarbetspartner. Våra möjligheter att själv välja fall har varit begränsade.

Vi fick tre uppdrag tilldelade av vår samarbetspartner. Ett av uppdragen valdes bort då det var under nyutveckling och många delar av releaseprocess som vi ville studera ännu inte var fastställd. Kvarvarande fall var förvaltningsuppdrag med två olika typer av system. Det vi studerat i fallen är förvaltningsuppdragens releaseprocess. Detta för att vår studie inriktar sig på utmaningar rörande kategorierna ”Byggen och Deploy” och ”Test och Verifikation” vilket är delar som är kopplade till deployment pipeline.

Det kan ibland vara svårt att göra generaliseringar utifrån en fallstudie då fallet ofta är unikt, men det är fortfarande möjligt. Några typer av generaliseringar som kan göras är: Att man skapar begrepp, teorier eller ramverk som även kan användas i andra studier(Oates, 2006). Denscombe (2016) skriver att man utifrån en fallstudie kan utveckla och generalisera teoretiska påståenden. Men att de inte ska betraktas som slutgiltiga eller absoluta. De teoretiska påståenden man kommer fram till blir en del i en pågående process och kan användas som en utgångspunkt i vidare forskning.

Vi har genom vår studie av två förvaltningsuppdrag fått ut analysresultat från fler analyser (se avsnitt 3.5) som varit specifikt för respektive fall. Genom dessa analyser har vi generaliserat det som vi upptäckt varit gemensamma utmaningar i båda uppdragen och presenterat dessa i vår slutsats. Vi anser att vår slutsats kan generaliseras till andra fall med förvaltningsuppdrag i liknande sits.

Vi har i vår studie använt oss av intervjuer och dokument. En fallstudie tillåter forskaren att använda en kombination av forskningsmetoder (observationer, intervjuer, dokument, frågeformulär) i undersökningen. Fallstudien inbjuder och uppmuntrar att forskaren gör det för att få en helhetssyn (Denscombe, 2016). Vi kommer i nästa avsnitt beskriva vår datainsamling närmare.

3.4 Datainsamling

För att kunna få en ökad förståelse för vilka utmaningar förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery, har vi använt oss av datainsamlingsmetoderna intervjuer och dokument(figur 7). Vi kommer i detta kapitel beskriva datainsamlingsmetoderna och hur vi gått tillväga.

3.4.1 Intervjuer

En av våra datainsamlingsmetoder har varit intervjuer. Oates (2006) säger att intervjuer är bra för att behandla ämnen på djupet och i detalj vilket passar vår fallstudie bra.
Oates (2006) delar in intervjuer i tre typer: strukturerade intervjuer, semistrukturerade intervjuer och ostrukturerade intervjuer. I en strukturerad intervju använder man fördefinierade och identiska frågor för varje intervju. Oates liknar en strukturerad intervju med en enkätundersökning men att forskaren i det här fallet antecknar svaren åt sin respondent(2006). En ostrukturerad intervju är motsatsen till den strukturerade. Där introducerar man ett ämne och låter respondenten styra och prata fritt och uttrycka sina idéer och tankar medan man som intervjuare försöker att påverka så lite som möjligt(Oates, 2006).

Vi valde att använda oss av semistrukturerade intervjuer.
I en semistrukturerad intervju finns det ett antal frågor eller teman som ska täckas in under intervjun, men det är inte spikat i vilken ordning frågorna ska ställas. Det är även möjligt att ställa fler frågor som uppstår under intervjun. En semistrukturerad intervju öppnar upp för att respondenten ska kunna prata fritt om ämnet. Vi valde att använda denna form av intervju för att ha möjlighet att ställa följdfrågor och på så vis få en så detaljerad bild av förvaltningsuppdragen som möjligt.

Vi utformade våra intervjufrågor(se bilaga 3) utifrån vår litteraturstudie(figur 7). De inledande frågorna som ställdes var både för att värma upp respondenten och för att ge oss en bakgrund om förvaltningsuppdragen. Därefter följde frågor som var mer specifikt om hur de deras releaseprocess ser ut. Dessa frågor användes som grund i vår nulägesanalys. För att respondenterna skulle vara förberedda på vilka typer av frågor som intervjun skulle handla om skickade vi intervjufrågorna till dem via mail. Oates(2006) skriver att detta kan vara bra för att respondenten får möjlighet att tänka igenom deras åsikter och vad de ska svara på frågorna.

Två av totalt tre intervjuer utfördes ansikte mot ansikte i lokaler på Triona och båda spelades in efter godkännande från respondenterna. Vi valde att spela in för att få ut så mycket som möjligt av intervjuerna och inte missa något. Oates (2006) skriver att det finns fördelar med att spela in intervjuer men att man måste få godkännande av respondenten. “Audio tape recording provides you with a complete record of everything that is, allowing you to concentrate on the process of the interview” (Oates, 2006, s. 191)

Intervjuerna var vardera en timme långa och styrdes av de semistrukturerade frågorna men mynnade också ut i diskussioner och mer detaljerade frågor inom releaserocessen och hur de arbetar. Respondenterna som intervjuades från förvaltningsuppdragen hade båda rollen förvaltningsledare. De valdes ut för att det hade störst överblick av förvaltningsuppdragen. Under en av intervjuerna kunde vissa frågor inte svaras på. Förvaltningsledaren hänvisade då till en utvecklingsledare som var mer lämpad att svara frågor som rörde mer detaljer i releaseprocessen och teknisk implementation. Då utvecklingsledaren inte var placerad på samma ort valde vi att göra denna tredje intervju via mail. I ett senare skede hade vi även telefonkontakt med utvecklingsledaren.

Vi är medvetna om att antalet intervjuer kan tyckas vara få för en fallstudie då man vill uppnå en djup beskrivning. Men vi har under arbetets gång haft löpande kontakt med våra respondenter för följdfrågor och validerat de resultat vi kommit fram till med dem. De har även varit delaktiga under analysfasen för validering (se avsnitt 3.5 under nulägesanalys). Intervjuer vi haft ansikte mot ansikte varit en timme långa och har varit mycket informativa och detaljerade. Därför anser vi att antalet intervjuer ändå är tillräckligt för att besvara vår frågeställning.

Genom att använda oss av intervjuer fick vi en detaljerad beskrivning av de två förvaltningsuppdragen vi studerat och det i sin tur har hjälpt oss att svara på vår frågeställning *”Vilka utmaningar kan ett förvaltningsuppdrag stå inför vid en övergång till Continuous Delivery?”*.

3.4.2 Dokument

Oates (2006) skriver att dokumentstudier är en datainsamling och kan ses som ett alternativ till intervjuer, observationer och enkäter. Dokument kan delas in i två typer, hittade dokument och genererade dokument. Hittade dokument är dokument som existerat innan studien gjorts. Den andra typen är genererade dokument som är dokument som tagits fram enbart i syfte för studien och som inte annars skulle existera. (Oates, 2006)

Vi har i vår studie använt oss av hittade dokument. I vår dokumentstudie har vi granskat dokument som har med de olika förvaltningsuppdragen att göra. Dokument som studerats är förvaltningsspecifikationer och förvaltningsprocesser. Dessa studerades för att styrka det som respondenterna sagt i intervjuerna och för att ge oss möjlighet att studera mer detaljer i förvaltningsuppdragen(figur 7). Då dokumenten är interna dokument som inte får visas offentligt är dessa inte med som bilagor.

3.5 Dataanalys

Efter datainsamling behöver man analysera insamlade data genom att leta efter mönster och teman (Oates, 2006). Det finns två olika sätt att analysera data: kvalitativ och kvantitativ analys. En kvantitativ dataanalys baseras på data som består av nummer eller tal som ofta förekommer i statistiska undersökningar eller experiment. En kvalitativ analys används vid analysering av icke-numerisk data- så som bokstäver, bilder och ljud. Kvalitativ data är huvudtypen av data som genereras av fallstudier, aktionsforskning och etnografi (Oates, 2006). I vår fallstudie har vi använt en kvalitativ dataanalys då data vi fått in från intervjuer och dokument var kvalitativ.

Innan en analys sker måste data struktureras och göras redo för analysering. Data som samlats in bör sparas under samma struktur och format. Detta kan underlätta när data ska genomsökas (Oates, 2006)

Vi inledde med att transkribera våra utförda intervjuer för att skapa struktur. Efter att transkribering skett delades det data vi samlat in i olika kategorier och sammanställdes i vårt empirikapitel. Genom att samla in data i kategorier skapade vi struktur som underlättade när vi utförde vår nulägesanalys, SWOT-analys och mognadsmodell. Vi skapade själva kategorier som vi tyckte passande utifrån insamlat data vi fått in. De kategorier som valdes var agilt arbetssätt, release, releaseprocess och kodhantering. Svar på frågor som uppstod under arbetets gång samt kompletterande data från dokumentstudier fördes också in under rätt kategori.

Nulägesanalys
En nulägesanalys i form av en processmodell över förvaltningsuppdragens releaseprocesser skapades utifrån intervjuerna och dokumentstudierna(figur 7). Den första versionen av processmodellen visades upp för utvecklingsledare i respektive uppdrag. Vi förde diskussioner kring modellen och kom fram till vilka revideringar som skulle behövas för att modellen skulle beskriva uppdragens releaseprocess på ett bra sätt. Detta gjorde att vi på så vis fick modellen validerad. Denscombe (2016) skriver att forskaren kan göra

respondentvalidering genom att visa upp data eller fynd för respondenten. Han skriver att man då får möjlighet att kontrollera träffsäkerheten, samt att forskarens förståelse kan bekräftas och eventuellt förbättras. Efter validering hade vi en processmodell som visar en korrekt bild av förvaltningsuppdragens releaseprocesser. Processmodellen har sedan använts som grund när vi analyserat uppdragens styrkor och svagheter och mognadsnivåer.

SWOT- analys
Eftersom vi ville undersöka styrkor och svagheter i förvaltningsuppdragen för en övergång till Continuous Delivery var en SWOT-analys en lämplig metod för att sammanställa dessa på ett strukturerat sätt.

SWOT-analysen har fått sitt namn ifrån engelskans Strengths, Weaknesses, Opportunities och Threats (Styrkor, svagheter, möjligheter och hot). Styrkor och svagheter är saker som den egna organisationen kan påverka. Möjligheter och hot är saker som organisationen själv inte kan påverka, utan styrs av till exempel andra organisationers beslut. Analysen går ut på att man på ett strukturerat sätt bedömer styrkor, svagheter, möjligheter och hot i en organisation eller inom en del av en organisation (KAMP företagsutveckling, 2016).

Vi har utifrån teori, insamlat data och nulägesanalysen i form av en processmodell identifierat styrkor, svagheter, möjligheter och hot för förvaltningsuppdragen för en övergång till Continuous Delivery. Dessa har grundats på den teori vi studerat och har placerats in i tabeller för att på ett strukturerat sätt visa de upptäckter som gjorts. Som komplement till SWOT-analyserna har upptäckterna beskrivits i löpande text där hänvisning till insamlat data och teori redovisats. Vi har placerat de styrkor och svagheter som hörde ihop under samma rubriker. Under analysen fann vi att många styrkor och svagheter återkom i båda förvaltningsuppdragen. För att undvika upprepning slog vi ihop beskrivningen av dessa.

SWOT-analysen svarar på vår delfråga ”Vilka styrkor och svagheter finns i de studerade förvaltningsuppdragen vid en övergång till Continuous Delivery?” (figur 7)

Mognadmodell (Maturity Model)
Utifrån Maturity model(Se avsnitt 2.9.2), insamlat data och en nulägesanalys i form av en processmodell har en analys av förvaltningsuppdragens mognadsnivå för en övergång till Continuous Delivery gjorts. De kategorier som analyserades i mognadsmodellen var ”Byggen och Deploy” och ”Test och verifikation”. Dessa kategorier valdes för att de berör releaseprocessen, samt att studiens omfattning inte gjorde de möjligt att analysera alla kategorier i Maturity Model. Analysen innefattade kriterierna inom nivåerna bas, nybörjare och medel. Med insamlade data och nulägesanalys som grund gick vi igenom modellen kriterier för kriterie och bockade av de som uppfylldes. Analysen resulterade i en modell med vilka kriterier som förvaltningsuppdragen uppnått i respektive kategori. Som komplement till modellen beskrivs det i löpande text vilken mognadsnivå förvaltningsuppdragen uppnått. För att skapa struktur valde vi att presentera varje kategori för sig och sedan de tre nivåer vi studerat. Vi har i den löpande texten hänvisat till insamlad data och teori.

Mognadsmodellen validerades likt processmodellen med utvecklingsledare i de båda förvaltningsuppdragen. Korrigeringar gjordes och resulterade i en mognadsmodell som visade korrekt bild av uppdragens uppfyllda kriterier.

De kriterier som förvaltningsuppdragen ej uppnått visar på vilka utmaningar de ställs inför om de vill implementera CD.

Vår analys av mognadsmodellen svarar på vår delfråga ” Vilken mognadsnivå befinner sig de studerade förvaltningsuppdragen vid en övergång till Continuous Delivery? (figur 7)

3.6 Forskningsetik

När man genomför en studie som involverar människor är det viktigt att den genomförs under etiskt acceptabla former. Vid Högskolan Dalarna finns det en Forskningsetisk nämnd som ser till att uppsatser på både grundnivå och avancerad nivå sker i enlighet med grundläggande forskningsetiska principer och krav. Nämnden har sammanfattat dessa krav och principer i punkter. Dessa skall man som student ha i åtanke innan, under och efter studien. (Högskolan Dalarna, 2013).

Vi har i vår studie följt Högskolan Dalarnas Forskningsetiska anvisningar för examens- och uppsatsarbeten. I kontakt med respondenter har det tydligt framgått vilken syfte vi haft med studien och att deltagande varit frivilligt. Ingen av respondenterna eller de uppdrag vi studerat har nämns vid namn i rapporten. Vi har vid intervjutillfället frågat respondenten om lov för ljudupptagning vilket i samtliga fall godkänts. Den empiri vi samlat in har delgivits respondenterna.

4 Sammanställning av förvaltningsuppdragen och nulägesanalys

I det här kapitlet kommer vi beskriva och sammanfatta de svar vi fått från våra intervjuer och dokumentstudier. Samt presentera en nulägesanalys i form av en processmodell för att beskriva förvaltningsuppdragens releaseprocess. Detta gör vi för att kunna besvara vår frågeställning ”Vilka utmaningar kan ett förvaltningsuppdrag stå inför vid en övergång till Continuous Delivery?”

4.1 Förvaltningsuppdrag A

Förvaltningsuppdrag A förvaltar en molntjänst. Tjänsten är framtagen först och främst för sågverk i Sverige. Den hanterar upphandlingar och beställningar av transporter. Allt detta sker genom webben och tjänsten som uppdraget förvaltar.

Agilt arbetssätt
Förvaltningsuppdrag A arbetar på ett agilt iterativt sätt. Förvaltningsledaren säger *“Vi jobbar inte renodlat agilt bara utan vi försöker ju å ena sidan fånga ärenden från alla de olika perspektiven vi måste täcka in. Å andra sidan att vi gör en bra prioritering då så att vi jobbar med rätt ärenden…”(citat från förvaltningsledare)*

Releaser
Tjänsten har 3-4 planerade releaser varje år. Inför varje år görs de en förvaltningsplan. Den baseras på ett övergripande behov, vad som behövs göras och var behovet finns. Utöver de planerade releaserna släpps det extra releaser på grund av buggar eller andra prioriterade ändringar. Antalet extra releaser varierar, förvaltningsledaren säger att det kan bli upp emot 10-15 extra releaser mellan de planerade. Anledningen till det stora antalet är att det är så lätt att driftsätta. Dessa releaser ges ut för att svara upp mot den vardag som är, vilket skulle kunna vara att en ny kund ställer nya krav, förvaltningsledaren säger “Det är en balansgång med det där att man inte får bli för händelsestyrd och springa på allt som rör sig” Förvaltningsledaren menar att det är viktigt och följa den förvaltningsplan som finns, men samtidigt kunna vara lättrörlig genom att hantera det som kommer in från support.

Förvaltningsledaren säger att de hela tiden har en dialog med kunderna inför en release. Om kunderna befinner sig i ett läge då de inte kan ta emot en release väntar man med det. De planerade releaserna sprids ut under året och de är noga med att inte driftsätta för nära inpå semestrar.

Det finnas bara en version av tjänsten i drift och samma version går ut till alla kunder. Förvaltningsledaren säger att de teoretiskt sett skulle kunna driftsätta mycket oftare. Han säger att det är en balansgång då det finns kunder som är måna att veta vad varje release innehåller.

Releaseprocess
Utvecklingsledaren berättar att de använder sig av TeamCity (Byggserver med stöd för Continuous Integration (Jet Brains, 2016)), där varje incheckning resulterar i en automatisk byggning av en artefakt(figur 8). Artefakterna sparas lokalt hos utvecklingsledaren. Enhetstester sätts igång manuellt. Därefter utför utvecklingsledaren manuella funktionstester av ad hoc karaktär. Om byggfel uppstår meddelas utvecklingsledaren genom mail samt att det blir synligt i TeamCitys gränssnitt. Utvecklingsledare säger att TeamCity endast används för byggfunktionen. Detta körs iterativt fram till att det närmar sig en planerad release. Utvecklingsledaren säger *"Vid ett release-arbete finns alltid ett angivet kodstopp för oss utvecklare. Strax före kodstopp kontrolleras alla ärenden av mig, d.v.s. enhetstestas och med viss kodrevision."* Vid godkännande sätts en DOD-stämpel(Definition of Done) på ärendet och systemet läggs in i testmiljön.

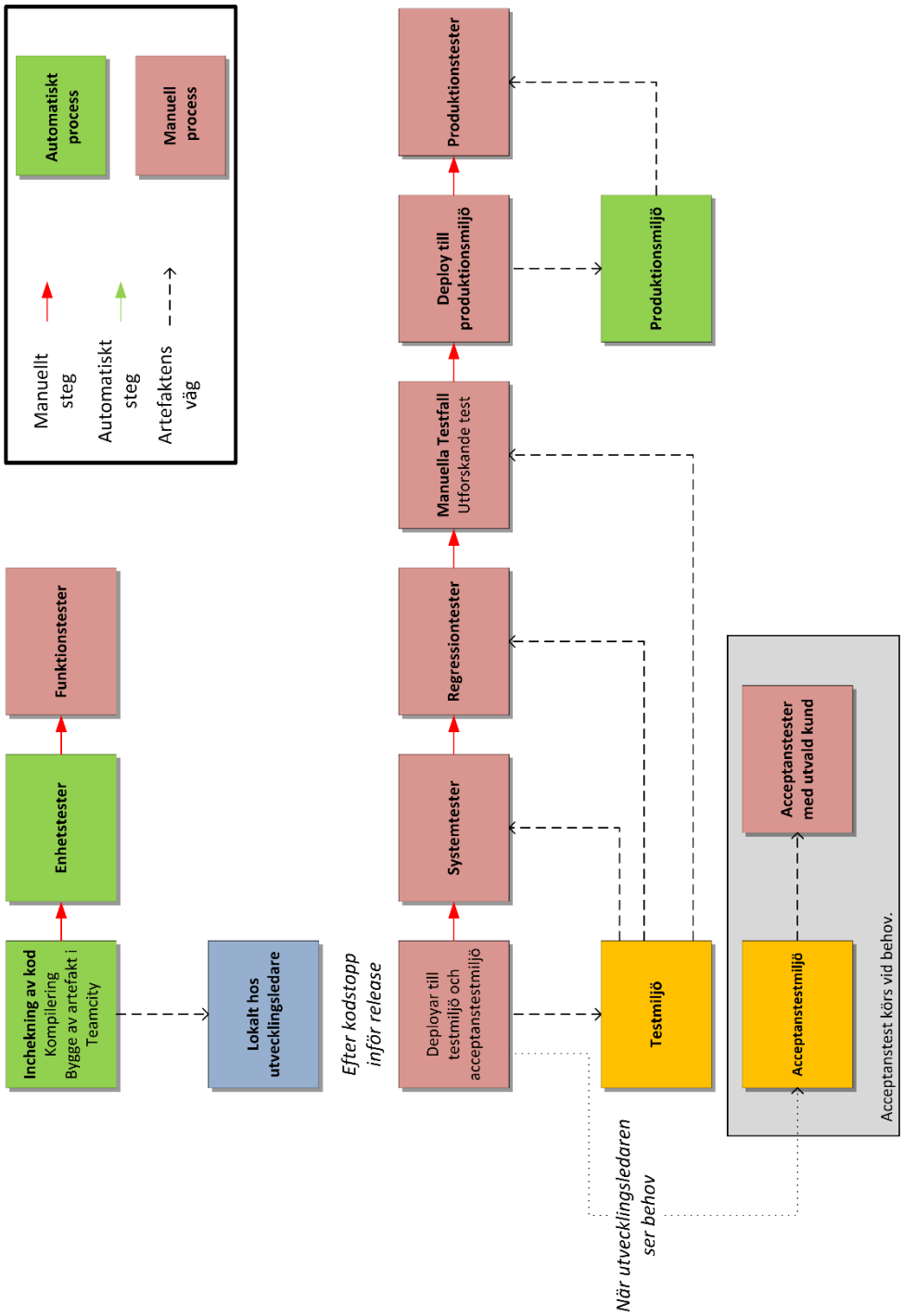
Testledare tilldelas ärenden för test och säkerställer att samtliga tester genomförs, såsom systemtester och regressionstester. Vid större förändringar av integrationsgränssnittet krävs att acceptanstester görs med utvald kund i AcceptanceTest-miljö med testledare.

Om det uppstår fel vid de manuella testerna tilldelas felet till utvecklingsledaren för åtgärd och så fortsätter det tills hela leveransen är godkänd.
När leveransen är godkänd att produktionssättas av förvaltningsledaren sker en manuell deploy nattetid till produktionsmiljön av utvecklingsledaren. För att säkerställa att de mest vitala funktionerna fungerar gör kvalitetsansvarig eller utvecklingsledare produktionstester. Det enda som sker automatisk i processen är automatiska byggen vid incheckning av kod. Utvecklingsledaren poängterar dock att det finns automatisk funktionalitet i Teamcity som de i dagsläget inte använder. Deployprocessen är standardiserad med hjälp av dokumentation och script finns för bland annat databasändringar. Utvecklingsledaren säger att de byggda artefakterna kan deployas i alla typer av miljöer.

Kodhantering
För versionshantering av kod används Subversion. Vid varje release skapas en ny branch, och utveckling av nästa release sker alltid i mainbranchen. Skulle det dyka upp buggar eller felaktigheter i den version som befinner sig i produktion åtgärdas det i korrekt branch. Efter åtgärd är det rutin på att merga till mainbranchen då ändringen annars försvinner i nästkommande version. När utveckling av funktionaliteter sker över längre tid skapas det en ny branch. Denna branch mergas ihop med mainbranchen när det är dags att leverera. Utvecklingsledaren poängterar att det är ovanligt att utvecklingen av ärenden sker över flera releaser. Byggen versionshanteras manuellt med koppling till tag i kodbas.

Nulägesanalys

Nulägesanalysen är en grafisk beskrivning av hur releaseprocessen ser ut i dagsläget i uppdrag A.



Figur 8 - Processmodell, beskrivning av uppdrag A's releaseprocess

4.2 Förvaltningsuppdrag B

Uppdrag B är ett uppdrag som är på väg in i förvaltning av ett system som innehåller flera moduler varav en kontorsklient. Kontorsklienten kräver fast installation och innehåller 80 % av hela systemets funktionalitet

Agilt Arbetssätt
Förvaltningsledaren i uppdraget berättar att de använder delar av den agila metoden SCRUM(En agil utvecklingsmetod(Scrum, 2015, 13 maj)).

Releaser
De planerade releaserna sker 2-3 gånger per år. Extra releaser släpps mellan de planerade och antalet varierar beroende på hur mycket fel som uppstår. Det finns kunder som har äldre releaser hos sig. Förvaltningsledaren säger följande när det gäller versioner ute hos kund ”Vi vill ju hela tiden att man har den nyaste, sen står de i avtalet att börjar de hitta fel i en gammal release så måste de uppdatera först.” Men han tillägger att de trots det ofta frångår från detta om det står still i produktionen hos en kund. Det leder till att de får arbeta i äldre versioner. Förvaltningsledaren uttrycker en önskan om ett krav på hur många releaser en kund får släpa efter med.

Releaseprocess
Uppdrag B använder sig likt uppdrag A av byggservern TeamCity.
Alla utvecklare checkar in sin kod i teamcity där den kontinuerligt kompileras och enhetstestas. Två gånger per dygn bygger TeamCity ett bygge som hamnar i “Nightly miljön” (figur 9). Ett bygge på natten och ett vid lunchtid. I Nightly sker manuella funktionstester där flera olika typer av tester utförs. Dessa tester görs utifrån testfall, där ingår delar av systemtest, regressionstest och integrationstest. Denna process sker iterativt tills det är dags för release. Vid den tidpunkten sätts ett kodstopp och man bygger ett bygge som sedan manuellt deployas till en testmiljö som kallas sprinttest. Sprinttest ses som en miljö som ska simulera produktionsmiljön. Deployprocessen till sprinttest sker manuellt för att efterlikna den riktiga deployprocessen till produktionsmiljön. Utvecklingsledaren poängterar att detta är ett medvetet val för att de ser denna deploy som ett test för den skarpa deployen till produktionsmiljön. Vid deploy till testmiljön testas det endast att gå över från den senaste versionen till den nya. Detta ser förvaltningsledaren som ett problem då de inte vet hur det fungerar vid en övergång från en äldre version. Det här är också en anledning till att man inte vill att kunder ska släpa efter för många releaser.

I sprinttestmiljön sker en iterativ process där man går igenom flera steg av manuella tester. När man stöter på ett fel fixas felen och det byggs om. När alla test körs igenom och allt är enligt krav är systemet redo för paketering.

Ute hos kund finns två miljöer. En testmiljö där kunden kan kontrollera grundläggande funktionalitet och en produktionsmiljö som används skarpt i verksamheten. Det finns kunder i uppdrag B som inte tycker att det är någon idé att installera det i kundens testmiljö. De tycker att det räcker med att se så att de stora flödena fungerar och sedan driftsätta. Kunden räknar med att det uppstår fel men också att Triona är snabba på att rätta felen.

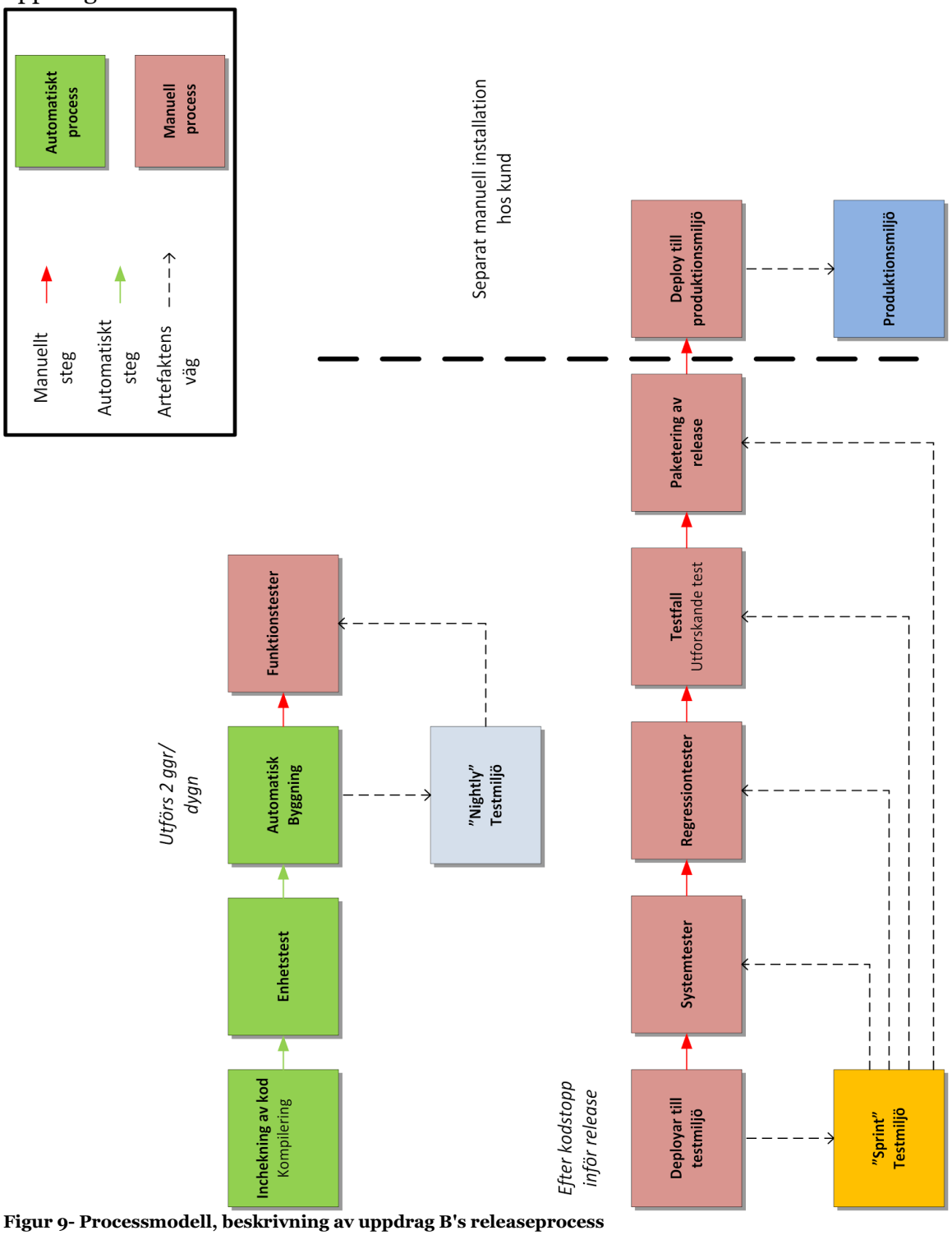
Förvaltningsledaren berättar att ett krönt bygge kan liknas en CD-skiva. När de krönt ett bygge så sker kommunikation med kunder om att en leverans är tillgänglig. Även om en leverans är tillgänglig sker inte driftsättning omgående, det beror på en rad faktorer. En av dessa är att vissa kunder inte vill ta emot en release. Det är också endast två personer som besitter kunskapen att driftsätta hos kund. Uppdrag B har en försiktighetsprincip att de vill driftsätta hos max två kunder i produktion samtidigt. När det gäller installationsprocessen så är den helt manuell.

Innan en release släpps har redan arbetet av nästa release påbörjats.

Kodhantering
För versionshantering av kod används Subversion. Vid varje release sker en avgrening till en ny branch och sedan går mainbranchen vidare mot nästa release. Upptäcks fel i en release så synkroniseras detta ner i mainbranchen. Utvecklarna checkar in sin kod mot samma branch. Byggen lagras i TeamCity med manuell versionshantering med koppling till tags i kodbas.

Nulägesanalys

Nulägesanalysen är en grafisk beskrivning av hur releaseprocessen ser ut i dagsläget i uppdrag B.



Figur 9- Processmodell, beskrivning av uppdrag B's releaseprocess

5 SWOT Analys och Mognadsmodell

Syftet med denna studie är att få en ökad förståelse för Continuous Delivery samt vilka utmaningar förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery. Vi kommer i detta kapitel presentera våra analyser i form av texter, tabeller och bilder.

5.1 SWOT

SWOT-analysen är baserad på insamlat data och nulägesanalys i form av en processmodell. Analysen är till för att identifiera styrkor, svagheter, möjligheter och hot som uppdraget har för en övergång till CD. Styrkor och möjligheter är aspekter som visar vart uppdraget står i nuläget för en övergång till CD. Svagheter och hot är aspekter som leder till utmaningar för en övergång till CD. SWOT-analysen presenteras först i tabellform. Sedan följer en beskrivning i löpande text med koppling teori och insamlat data. Denna analys svarar på vår delfråga *"Vilka styrkor och svagheter finns i de studerade förvaltningsuppdragen vid en övergång till Continuous Delivery?"*

5.1.1 SWOT tabeller

Uppdrag A

Styrkor 1. Arbetar agilt 2. Gör frekventa commits 3. Automatiskt bygge vid incheckning 4. Kodintegrering görs 5. Tekniskt stöd för automatiska deployer finns 6. Relativt enkel deploy till produktion 7. Möjlighet för mer frekventa deployer till produktion finns.	**Svagheter** 1. Releaser görs sällan 2. Många automatiska tester saknas. 3. Många tester görs manuellt. 4. Utnyttjar inte den teknik som finns för automatiska deployer 5. Saknas automatisk deploy till miljöer
Möjligheter 1. Att utnyttja tekniken fullt ut 2. Automatisera tester 3. Automatisera deploy	**Hot** 1. Kunden vill eller kan inte hantera fler/täta releaser 2. Att man inte får de resurser som krävs från ledningen för att implementera CD

Tabell 2 - SWOT tabell Uppdrag A

Uppdrag B

Styrkor	**Svagheter**
1. Arbetar agilt 2. Gör frekventa commits 3. Automatiska byggen 4. Kodintegrering görs. 5. Tekniskt stöd för automatiska deployer finns	1. Releaser görs sällan 2. Många automatiska tester saknas 3. Många tester görs manuellt 4. Utnyttjar inte den teknik som finns för automatiska deployer 5. Saknar automatiska byggen vid incheckning 6. Omfattande deployprocess med manuell fast installation hos kund
Möjligheter	**Hot**
1. Att utnyttja tekniken fullt ut 2. Automatisera tester 3. Automatisera deploy	1. Kunden vill eller kan inte hantera fler/täta releaser 2. Att man inte får de resurser som krävs från ledningen för att implementera CD

Tabell 3 - SWOT tabell Uppdrag B

5.1.2 SWOT beskrivning av uppdrag A och B

Styrka 1

- Uppdrag A
  Uppdraget arbetar agilt på ett iterativt sätt och detta är en grundförutsättning för en övergång till Continuous Delivery. (Se avsnitt 4.1)

- Uppdrag B
  Uppdrag B arbetar agilt med metoder från det agila arbetssättet SCRUM. (Se avsnitt 4.2)

Att arbeta agilt är en grundförutsättning för en övergång till Continuous Delivery. I den konceptuella modellen "Stairway to heaven" tas det agila upp som ett av grundstegen i utvecklingsvägen mot Continuous Deployment där Continuous Delivery ingår som en del (Holmström Olsson, Alahyari, & Bosch, 2012).(Se avsnitt 2.4.4)

Styrka 2, 3 och 4

- Uppdrag A
  Genom att uppdrag de kontinuerligt checkar in sin kod till en gemensam byggserver integreras koden till en och samma kodbas. Automatiska byggen triggas igång vid varje incheckning och man får en tidig feedback på ev byggfel. (Se avsnitt 4.1)

- Uppdrag B
  De använder sig av byggservern TeamCity. Alla utvecklare checkar kontinuerligt in sin kod i TeamCity där den kompileras och enhetstestas löpande. Sedan sker automatiska byggen två gånger per dygn. (Se avsnitt 4.2)

Humble och Farley (2010, s 59) säger att det är viktigt att man checkar in sin kod ofta till mainbranchen för att uppnå Continuous Deliverys fördelar. De säger också att genom frekventa incheckningar blir kodändringarna mindre och du minskar risken för konflikter med andra utvecklares kod. (Se avsnitt 2.4)
Enligt Abantes och Travassos (2011) bidrar integrering av kod till att man kan upptäcka fel och integrationsproblem i ett tidigt skede. (Se avsnitt 2.3)

Styrka 5

- Uppdrag A & B
  En styrka som båda uppdragen har är att de använder sig av Teamcity där det finns tekniskt stöd för automatiska deployer. (Se avsnitt 4.1)

Teamcity är en CI-server som har stöd för automatiska test och deployer till flera miljöer.(se avsnitt 2.4.3)

Styrka 6 och 7

- Uppdrag A
  Förvaltningsledaren säger att det är lätt att göra en driftsättning och att de teoretiskt sätt skulle kunna göra mer frekventa driftsättningar till produktion. Han säger även att de kan släppa mellan 10-15 extra releaser mellan de planerade vilket visar på att det är möjligt även praktiskt att göra mer frekventa releaser. (Se avsnitt 4.1)

Detta tar Humble och Farley (2010) upp som en fördel med CD. De säger att genom kontinuerliga releaser får en snabb feedbackloop vilket leder till snabbare och effektivare införande av nya funktioner, samtidigt som man åtgärdar fel. (Se avsnitt 2.4)

Svaghet 1

- Uppdrag A
  Releaser görs sällan. Uppdraget har endast 3-4 planerade releaser per år. (Se avsnitt 4.1)

- Uppdrag B
  De planerade releaserna sker endast 2-3 gånger per år. (Se avsnitt 4.2)

Detta är svagheter för en övergång till Continuous Delivery då de planerade releaserna kommer ut sällan och det dröjer ytterligare innan de är hos kund på grund av den omfattande deploymentprocessen. Enligt Humble och Farley (2010) ska släppa releaser frekvent för att uppnå en snabb feedbackloop som är en central del i CD. (Se avsnitt 2.4)

Svaghet 2 och 3

- Uppdrag A
  I releaseprocessen används många manuella tester och väldigt få automatiska. (Se avsnitt 4.1)

- Uppdrag B
  I uppdrag B görs de flesta tester manuellt. Det enda test som är automatiskt är enhetstest (se avsnitt 4.2).

Detta är en svaghet då en Continuous Delivery pipeline kräver flera automatiska teststeg. Humble och Farley (2010) skriver, genom att automatisera återupprepbara delar i releaseprocessen(pipelinen) kan du undvika problem och fel som orsakas av den mänskliga faktorn. (Se avsnitt 2.4)

Svaghet 4

- Uppdrag A
  En svaghet är att de inte utnyttjar den teknik som finns för automatiska deployer som de har tillgång till i Teamcity. (Se avsnitt 4.1)

- Uppdrag B
  En svaghet som uppdrag B har är att de inte använder det teknikstöd för automatiska deployer som de har tillgång till i Teamcity. (Se avsnitt 4.2).

Teamcity är en CI-server som har stöd för automatiska test och deployer till flera miljöer.(se avsnitt 2.4.3)

Svaghet 5 (Uppdrag A)

- Det saknas automatiska deployer till testmiljöer och produktionsmiljö. (Se avsnitt 4.1)

Detta är en svaghet då en Continuous Delivery kräver automatisering. Återigen hänvisar vi till Humble och Farley (2010) som skriver att genom att automatisera återupprepbara delar i releaseprocessen(pipelinen) kan du undvika problem och fel som orsakas av den mänskliga faktorn. (Se avsnitt 2.4)

Svaghet 5 (Uppdrag B)

- Saknar automatiska byggen vid incheckning.
  Uppdraget automatiska byggen triggas inte av varje incheckning.(Se avsnitt 4.2)

Detta är en punkt för medelnivå i build och deploy som enligt Maturity modellen (Rehn m.fl., 2013) ska uppfyllas för att vara mogen för Continuous Delivery.(Se avsnitt 2.4.4)

Svaghet 6 (Uppdrag B)

- Uppdrag B har i dagsläget en omfattande deployprocess med fast manuell installation hos kund. Det tar lång tid att få ut en release till alla kunder då den manuella installationen är tidskrävande och få besitter kunskap om driftsättning. Uppdragets försiktighetsprincip (max två installationer samtidigt) bidrar till att processen blir utdragen. (Se avsnitt 4.2)

Detta är svagheter för en övergång till Continuous Delivery då de planerade releaserna kommer ut sällan och det dröjer ytterligare innan de är hos kund på grund av den omfattande deploymentprocessen. Enligt Humble och Farley (2010) ska släppa releaser frekvent för att uppnå en snabb feedbackloop som är en central del i CD. (Se avsnitt 2.4)

Möjlighet 1

- Uppdrag A och B
  Om uppdragen skulle utnyttja tekniken fullt ut i Teamcity skulle det öka möjligheten för en övergång till Continuous Delivery. Teamcity är en CI-server som har stöd för automatiska test och deployer till flera miljöer.(se avsnitt 2.4.3)

Möjlighet 2

- Uppdrag A och B
  Genom att automatisera tester som är manuella skulle det öka möjligheten för en övergång till Continuous Delivery. Humble och Farley (2010) skriver att genom att automatisera återupprepbara delar i releaseprocessen(pipelinen) kan du undvika problem och fel som orsakas av den mänskliga faktorn. .(Se avsnitt 2.4)

Möjlighet 3

- Uppdrag A och B
  Genom att automatisera deployer skulle det öka möjligheten för en övergång till Continuous Delivery. Humble och Farley (2010) skriver att genom att automatisera återupprepbara delar i releaseprocessen(pipelinen) kan du undvika problem och fel som orsakas av den mänskliga faktorn.(Se avsnitt 2.4)

Hot 1

- Uppdrag A
  Förvaltningsledaren säger *"Om kunden befinner sig i ett läge då de inte kan ta emot en release väntar man med det"*. (Se avsnitt 4.1)

- Uppdrag B
  Förvaltningsledaren säger att vissa kunder inte vill ta emot en release (Se avsnitt 4.2).

Om detta är återkommande kan det bli ett hot mot en övergång till Continuous Delivery. Enligt Humble och Farley (2010) ska man släppa releaser frekvent för att uppnå en snabb feedbackloop som är en central del i CD. (Se avsnitt 2.4)

Hot 2

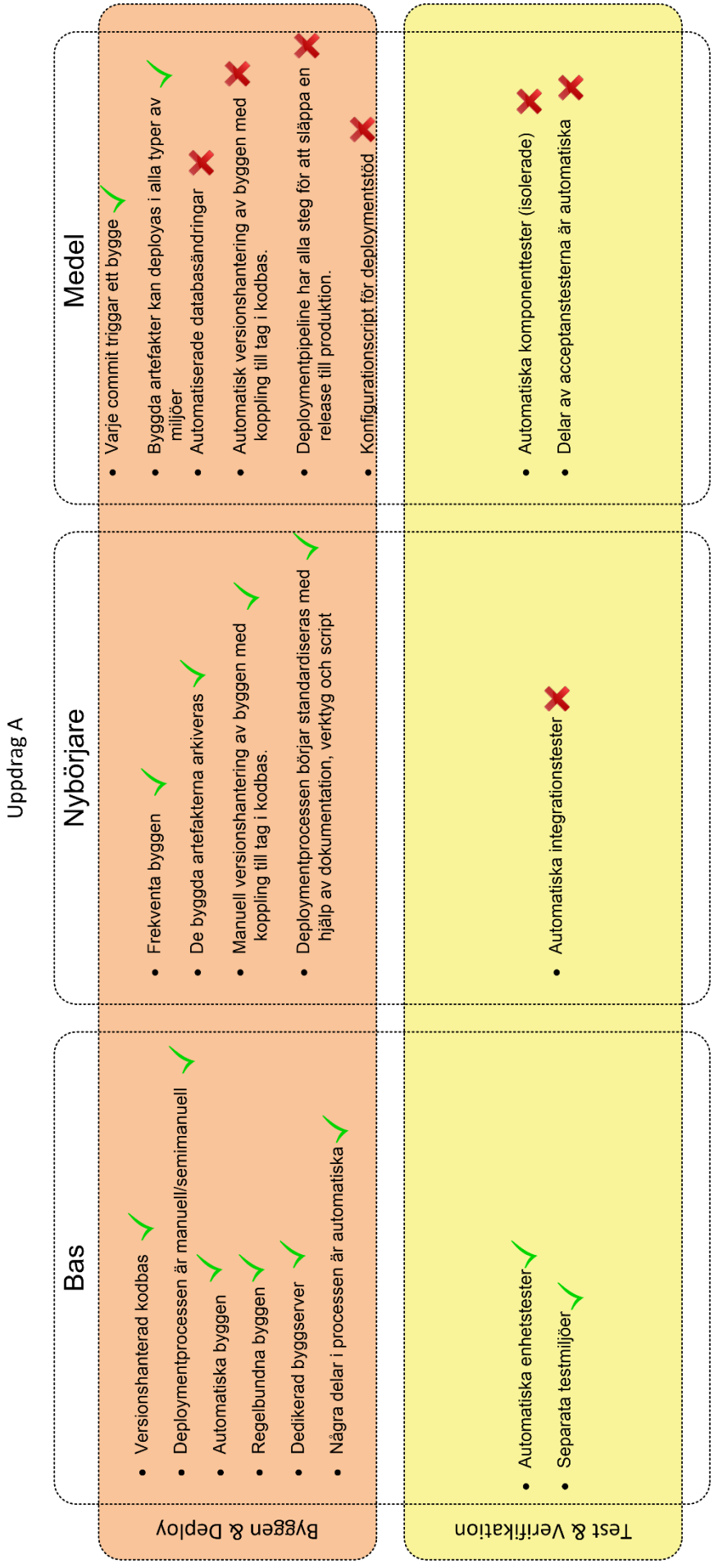
- Uppdrag A och B
  Att man inte får de resurser som krävs från ledningen för att implementera CD

Om man inte skulle få de resurser som krävs för att implementera CD uppstår ett hot. Neely och Stolt (2013) säger att det finns många fördelar med Continuous Delivery men poängterar "*But it is not free; it is not painless. It is important to have the right level of sponsorship before you begin" (Se avsnitt 2.4).*

5.2 Uppdragens mognad för Continuous Delivery

Mognadsmodellen visar vilken mognadsnivå uppdragen ligger på för en övergång till Continuous Delivery. Modellerna i avsnitt nedan visar vilka kriterier som uppfylls i nivåerna bas, nybörjare och medel av uppdragen i kategorierna Byggen och Deploy och Test och Verifikation. Denna analys svarar på vår delfråga *" Vilken mognadsnivå befinner sig de studerade förvaltningsuppdragen vid en övergång till Continuous Delivery?"*

5.2.1 Uppdrag A mognad för Continuous Delivery



Figur 10 - Mognadsanalys, Uppdrag A

Uppdrag A beskrivning mognadsmodell

Byggen och Deploy

Bas
Uppdrag A uppfyller alla kriterier för basnivån för Byggen och Deploy.
Basnivån innefattar versionshantering av kod, manuell eller semimanuell deploymentprocess automatiska byggen, regelbundna byggen, dedikerad byggserver samt att några delar i processen är automatiska.(Se avsnitt 2.4.4 Maturity model)
Uppdrag A använder sig av Teamcity som byggserver. Det sker en automatisk byggning vid varje incheckning av kod. Vid kodstopp sker deploy av artefakten manuellt till testmiljö och vid behov till acceptanstestmiljö.(Se avsnitt 4.1)

Nybörjare
Uppdrag A uppfyller alla kriterier för nybörjarnivån.
Nybörjarnivån innefattar frekventa byggen, byggda artefakter arkiveras, manuell versionshantering av byggen med koppling till tag i kodbas och sista kriteriet är att deploymentprocessen börjar standardiseras med hjälp av dokumentation, verktyg och script. (Se avsnitt 2.4.4 Maturity model)
Byggen sker frekvent, automatiskt vid varje incheckning av kod. De byggda artefakterna arkiveras lokalt hos utvecklingsledaren. Byggen versionshanteras manuellt med koppling till tag i kodbas. Deploymentprocessen är standardiserad med hjälp av dokumentation och script.(Se avsnitt 4.1)

Medel

Uppdrag A uppfyller inte alla kriterier på medelnivån.
Medelnivån innefattar varje commit triggar ett bygge, byggda artefakter kan deployas i alla typer av miljöer, automatiserade databasändringar, automatisk versionshantering av byggen med koppling till tag i kodbas, deployment pipeline har alla steg för att släppa en release till produktion samt konfigurationsscript för deploymentstöd. (Se avsnitt 2.4.4 Maturity model)
Det två kriterier som uppfylls är automatisk byggning vid varje incheckning av kod samt att de byggda artefakterna kan deployas i alla typer av miljöer.
(Se avsnitt 4.1)

De uppfyller inte kriterierna automatiserade databasändringar, automatisk versionshantering av byggen med koppling till tag i kodbas, en deployment pipeline med alla steg för att släppa en release till produktion samt konfigurationsstöd för deploymentstöd.

Test och verifikation

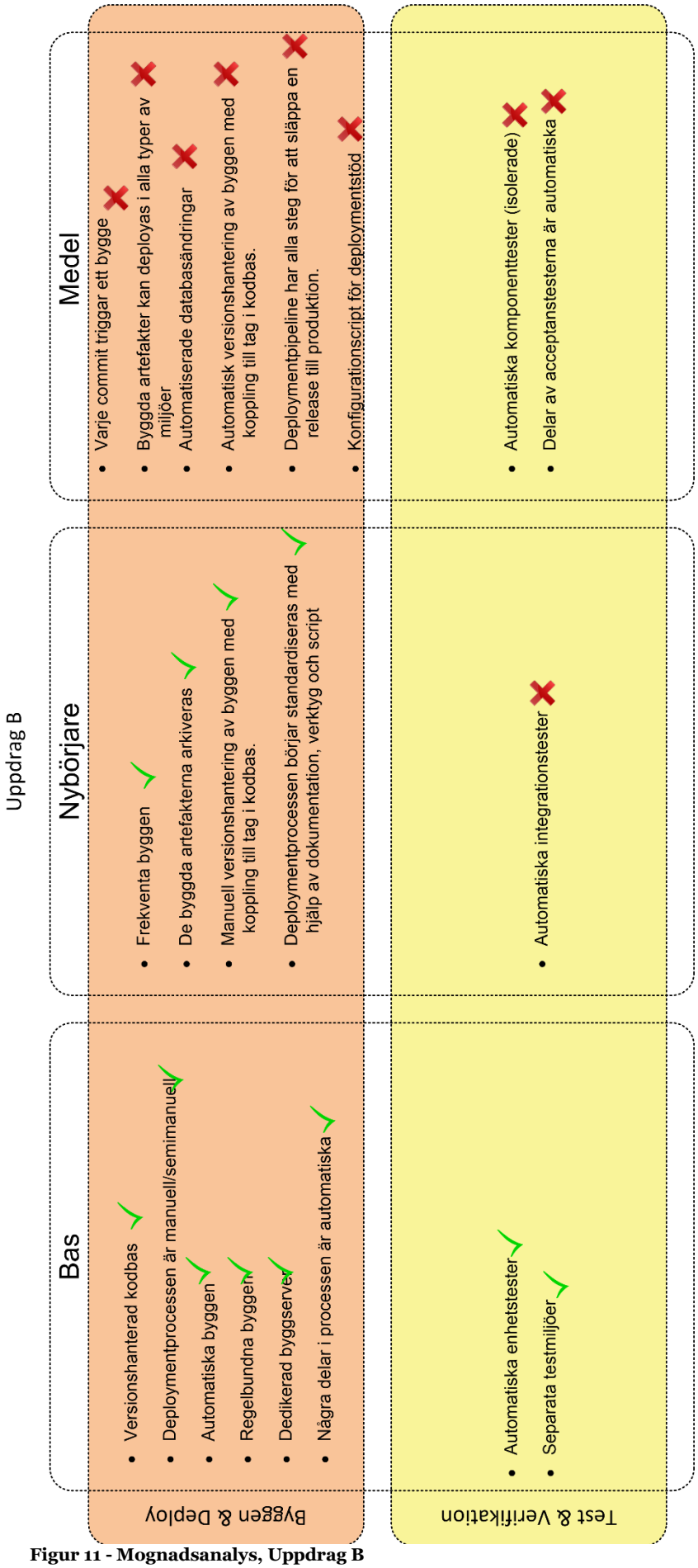
Bas
Uppdrag A uppfyller alla kriterier för basnivån.
Basnivån innefattar automatiska enhetstester och att separata testmiljöer finns. (Se avsnitt 2.4.4 Maturity model)

De uppfyller båda kriterierna genom att utvecklingsledaren kör automatiska enhetstester och de har två separata testmiljöer där den ena används tillsammans med kund för acceptanstest.(Se avsnitt 4.1)

Nybörjare
Uppdrag A uppfyller inte kriteriet för nybörjarnivån.
Nybörjarnivån innefattar automatiska integrationstester.(Se avsnitt 2.4.4 Maturity model)
De utför integrationstester men dessa är inte automatiska.(Se avsnitt 4.1)

Medel
Uppdrag A uppfyller inte kriterierna som krävs för medelnivån.
Medelnivån innefattar automatiska komponenttester samt delvis automatiserade acceptanstester.(Se avsnitt 2.4.4 Maturity model)
Uppdragets komponenttester är manuella och ingen automation finns för acceptanstester.(Se avsnitt 4.1)

5.2.2 Uppdrag B mognad för Continuous Delivery



Figur 11 - Mognadsanalys, Uppdrag B

Uppdrag B beskrivning mognadsmodell

Byggen och Deploy

Bas
Uppdrag B uppfyller alla kriterier för basnivån.
Basnivån innefattar versionshantering av kod, manuell eller semimanuell deploymentprocess automatiska byggen, regelbundna byggen, dedikerad byggserver samt att några delar i processen är automatiska.(Se avsnitt 2.4.4 Maturity model)
Uppdrag B använder TeamCity som byggserver där det finns funktionalitet för automatiska byggen. Systemet byggs automatiskt 2 gånger per dygn. När det sedan sätts kodstopp sker deploy av bygget manuellt till testmiljön. De använder sig av ett versionshanteringssystem för sin kod. (Se avsnitt 4.2)

Nybörjare
Uppdrag B alla kriterier för nybörjarnivån.
Nybörjarnivån innefattar frekventa byggen, byggda artefakter arkiveras, manuell versionshantering av byggen med koppling till tag i kodbas och sista kriteriet är att deploymentprocessen börjar standardiseras med hjälp av dokumentation, verktyg och script. (Se avsnitt 2.4.4 Maturity model)
Automatiska byggen utförs 2 gånger per dygn vilket är frekvent och de byggda artefakterna arkiveras i teamcity. Artefakterna(byggena) versionshanteras manuellt och koppling till tag i kodbas finns. Dokumentation och verktyg finns för standardisering av deploymentprocessen. (se avsnitt 4.2)

Medel
Uppdrag B uppfyller inte något av medelnivåns kriterier.
Medelnivån innefattar varje commit triggar ett bygge, byggda artefakter kan deployas i alla typer av miljöer, automatiserade databasändringar, automatisk versionshantering av byggen med koppling till tag i kodbas, deployment pipeline har alla steg för att släppa en release till produktion samt konfigurationsscript för deploymentstöd. (Se avsnitt 2.4.4 Maturity model)

Test och verifikation

Bas
Uppdrag B uppfyller alla kriterier för basnivån.
Basnivån innefattar automatiska enhetstester och att separata testmiljöer finns. (Se avsnitt 2.4.4 Maturity model)
När utvecklarna checkar in kod kompileras den och enhetstester körs automatiskt. Uppdrag B använder sig av två stycken olika testmiljöer. När ett bygge är klart hamnar det i första testmiljön “Nightly”, där det sker manuella funktionstester.
Vid kodstopp sker en manuell deploy till den andra testmiljön som ska fungera som en simulering av produktionsmiljön.(Se avsnitt 4.2)

Nybörjare
Uppdrag B uppfyller inte kriteriet för nybörjarnivån.
Nybörjarnivån innefattar automatiska integrationstester.(Se avsnitt 2.4.4 Maturity model)
Uppdraget utför integrationstester men dessa är inte automatiska.(Se avsnitt 4.2)

Medel
Uppdrag B uppfyller inte kriterierna som krävs för medelnivån.
Medelnivån innefattar automatiska komponenttester samt delvis automatiserade acceptanstester.(Se avsnitt 2.4.4 Maturity model)
Komponentester görs manuellt och acceptanstester utförs inte över huvudtaget.

6. Diskussion och slutsats

I detta kapitel diskuteras resultatet. Sedan följer en metodkritik följt av vår slutsats samt förslag på framtida studier.

6.1 Diskussion

Syftet med denna studie är att få en ökad förståelse för Continuous Delivery samt vilka utmaningar förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery.

För att besvara syftet behöver vi få svar på frågeställningen och delfrågor. Vi börjar vår diskussion med delfrågorna där vi hänvisar till resultat och för diskussion kring det. Slutligen tar vi upp frågeställningen och svarar på denna.

”Vilka styrkor och svagheter finns i de studerade förvaltningsuppdragen vid en övergång till Continuous Delivery?”

Svar på detta finns under avsnitt 5.1 SWOT
Här nedan följer en diskussion av SWOT-analysen.

Uppdelningen som följer är baserad på att styrkor och svagheter överlappar varandra och därför presenteras diskussionen kring aspekter som framkommit i analysen

Continuous Integration främjar övergång
Tydliga styrkor som framkommit i förvaltningsuppdragen är att de kontinuerligt checkar in kod till en gemensam byggserver där koden integreras. Genom detta finns egenskaper för Continuous Integration vilket är en stor styrka för CD genom att CI är ett steg på vägen.
Vi tror att denna styrka återfinns bland många förvaltningsuppdrag då detta verkar vara ett standardiserat arbetssätt. Genom att versionshanteringssystemet är kopplat till byggservern motiveras detta arbetssätt bland utvecklare då versionshantering är en förutsättning för att samarbeta med fler utvecklare.

Agilt arbete är en styrka
Båda förvaltningsuppdragen arbetar agilt i nuläget vilket är en styrka för en övergång till CD. Att vara agil innebär att vara lättrörlig, vilket krävs för att införa CD. Där man genom en kort feedbackloop måste kunna anpassa sig för att kunna effektivisera och införa nya funktioner. Att arbeta agilt tror vi är vanligt bland många förvaltningsuppdrag, hur man arbetar mer konkret ur det agila perspektivet är kanske mindre viktigt. Huvudsaken är att du har ett lättrörligt tänk och kan göra förändringar efter vägen.

Tekniskt stöd för automatiska deployer finns
Båda förvaltningsuppdragen använder sig av CI-servern TeamCity där det finns tekniskt stöd för att utföra automatiska deployer. Detta är en styrka hos båda uppdragen. Dock använder de inte den teknik som finns fullt ut. I ett av uppdragen påstår förvaltningsledaren att det skulle vara möjligt att göra mer frekventa releaser till produktion. VI tror också att detta är en möjlighet då man redan i nuläget släpper 10-15 extra releaser mellan de

planerade vilket innebär i snitt en release i veckan. Att kontinuerligt släppa releaser skulle då inte bli så stort steg.

Releaser görs sällan
I förvaltningsuppdragen görs det få planerade releaser varje år. Detta ser vi som en svaghet då de talar emot CD, där det krävs frekventa releaser. Att implementera frekventa releaser kan vara en utmaning då vissa kunder kanske inte vill eller har möjlighet att hantera dessa. Kundernas oförmåga eller ovilja till frekventa releaser kan därför vara ett hot. Det är viktigt att kunderna är med på CD då hela processen bygger på den feedback man får in från kund.

Automatiska tester saknas
I förvaltningsuppdragen utförs det många manuella tester. Saknaden av automatiska tester är stor. Vi ser detta som en svaghet då flera automatiska teststeg krävs i en deployment pipeline i Continuous Delivery.

Omfattande manuell deploymentprocess
I ett av förvaltningsuppdragen kom det fram att deploymentprocessen var väldigt omfattande då den krävde manuell installation hos kund. Detta är svaghet som man borde ta med i beaktning vid en övergång till CD.

Resurser krävs
Något vi ser som ett ev. hot är om ledningen inte skulle stödja en övergång till CD. En implementering av CD kräver stora resurser i både tid och pengar vilket en ledning måste vara med på.

Vilken mognadsnivå befinner sig de studerade förvaltningsuppdragen vid en övergång till Continuous Delivery?

Ett utförligt svar på detta finns under avsnitt 5.2 Uppdragens mognad för Continuous Delivery.
Här nedan följer en sammanfattning av svaret samt diskussion av mognadsanalysen.

Bas i “Byggen och Deploy”
Båda förvaltningsuppdragen uppfyller samtliga kriterier på bas nivån i kategorin “Byggen och Deploy”. När det kommer till nybörjarnivån uppfylls även där alla kriterier för båda förvaltningsuppdragen. När det kommer till Medelnivån uppfylls inte alla kriterier för kategorin “Byggen och Deploy”. Ett förvaltningsuppdrag uppfyller två av sex kriterier och det andra förvaltningsuppdraget inga. Detta innebär att de ligger på en mognadsnivå som är nybörjare i kategorin “Byggen och Deploy”.

Nybörjare i “Test och Verifikation”
Förvaltningsuppdragen uppfyller samtliga kriterier på bas nivån i kategorin “Test och Verifikation”. När det kommer till nivån nybörjare uppfylls inte kriteriet. På medelnivån uppfylls inga kriterier. Detta innebär att de ligger på en mognadsnivå som är bas i kategorin “Test och Verifikation”.

När ett uppdrag ligger på mognadsnivå nybörjare innebär det att uppdraget har egenskaper för CI och att man i vissa fall redan använder det. Detta är ett tydligt steg på vägen till CD. Uppdragen befinner sig på mognadsnivån bas i "Test och Verifikation" på grund av att de flesta automatiska tester saknas. Vi tror att automatiska tester är en förutsättning för att lyckas implementera CD i ett uppdrag.

Kriterier som krävs för medel i "Byggen och deploy"
De kriterier nedan beskriver vad som krävs för ett uppdrag som befinner sig på mognadsnivån nybörjare i kategorin "Byggen och Deploy" att ta sig till mognadsnivå medel. Några av dessa kriterier kan ses som utmaningar för uppdragen. Som vi nämnt tidigare i teorin i Maturity Model (Se avsnitt 2.9 Maturity model) är Medel den mognadsnivå som man ser som en mogen nivå för Continuous Delivery.

- Varje commit triggrar ett bygge(Uppfylls av förvaltningsuppdrag a)
  - Detta kriterium ser vi inte som en utmaning. Då vi inte tror att det är särskilt svårt att uppnå. Detta är något som uppdrag a redan uppfyller och handlar helt enkelt om en inställning som görs i hur byggservern ska trigga byggen. Anledningen till att uppdrag inte gör detta tror vi har att göra med att man tycker det är onödigt att bygga fler gånger än vad de gör i nuläget.
- Byggda artefakter kan deployas i alla typer av miljöer
  (Uppfylls av förvaltningsuppdrag a)
  - Detta kriterium kan ses som en utmaning. När artefakten byggs måste den vara körbar i alla typer av miljöer. Den ska inte behöva byggas om.
- Automatiserade databasändringar
  - Detta kriterium är även det en utmaning då det krävs att man skapar automatiska script för databasändringar.
- Automatisk versionshantering av byggen med koppling till tag i kodbas
  - Detta kriterium ser inte vi som en utmaning. Detta hör till en inställning av byggservern. I TeamCity är detta möjligt.
- Deployment pipeline har alla steg för att släppa en release till produktion
  - Detta kriterium ser vi som en väldigt stor utmaning då det innebär att en fullfjädrad deployment pipeline är implementerad. Med alla automatiska steg som krävs.
- Konfigurationsscript för deploymentstöd
  - Detta kriterium är en utmaning som innebär att man måste skapa script för konfiguration som behövs vid deployment till olika miljöer.

Kriterier som krävs för nybörjare och medel i "Test och Verifikation"
De kriterier nedan beskriver vad som krävs för ett uppdrag som befinner sig på mognadsnivån bas kategorin "Test och Verifikation" för att ta sig till mognadsnivå nivå Nybörjare och sedan Medel. Dessa kriterier kan ses som utmaningar för uppdragen.

- Automatiska integrationstester
- Automatiska komponenttester
- Delar av acceptanstester är automatiska

Samtliga kriterier i ”Test och Verifikation” som inte uppfylls handlar om automatisering. Detta är enligt vår mening en av de största utmaningarna med en övergång till CD. Att automatisera tester kräver resurser i form av tid och pengar.

För att ta reda på hur moget ett uppdrag är räcker det inte med att endast studera två kategorier. Därför kan vi endast svara på vilken mognadsnivå de befinner sig på i dessa kategorier. För en fullständig analys av mognadsnivå krävs en analys av alla kategorier.

“Vilka utmaningar kan ett förvaltningsuppdrag stå inför vid en övergång till Continuous Delivery?”

Ur SWOT-analysens resultat plockades svagheter och hot ut. Vidare plockades ej uppnådda kriterier ut ur mognadsanalysen. Tillsammans bildade svagheter, hot och ej uppnådda kriterier grunden för de huvudutmaningar vi har kommit fram till. Dessa utmaningar presenteras nedan och relateras till den teori vi studerat.

Stöd från ledning
Ur SWOT-analysen framkom ett hot, att man inte får de resurser som krävs för att implementera CD av ledningen. Neely och Stolt (2013) skriver att det är viktigt att få stöd ifrån ledningen innan du börjar med en implementering av CD för att få de resurser som krävs. Detta ser vi som en av utmaningarna som ett förvaltningsuppdrag kan stå inför.

Viktigt att få med kund
Ett möjligt hot som framkom i SWOT-analysen var att kunder inte kan eller har möjlighet att hantera en högre frekvens av releaser. Detta ser vi som en utmaning ett förvaltningsuppdrag kan stå inför då det är viktigt att få med sig kunderna i övergången till CD. Leppänen m.fl. (2015) skriver att en av de primära utmaningarna är att kunden måste vilja och dessutom ha möjlighet att hantera fler releaser än vad de har i nuläget. Detta tror vi har en stor påverkan om en övergång till CD ska vara möjlig, då CD bygger på att få tidig feedback från kunder.

Automatisering
Den största möjliga utmaningen vi ser som ett förvaltningsuppdrag kan stå inför är att automatisera delar i deployment pipeline. Det innefattar automatiska tester, databasändringar, deployment och konfigurationsscript. Automatiskt är ett av nyckelorden för Continuous Delivery (Humble & Farley, 2010, s 12). Med det vill vi säga att det är en viktig utmaning att klara av. Vi tror att resursaspekten är en del i utmaningen men att det säkert finns fler aspekter inom en automatisk implementering av delarna. Humble och Farley (2010) säger att implementera en deployment pipeline är något som kräver resurser och speciellt när det gäller de automatiska testerna.

Även om dessa utmaningar hittades specifikt för dessa två förvaltningsuppdrag tror vi att det kan vara till hjälp för andra förvaltningsuppdrag i liknande sits.

6.2 Metodkritik

Oates (2006) skriver att det kan vara svårt att göra generaliseringar utifrån en fallstudie då fallet ofta är unikt men det är fortfarande möjligt. Detta är något vi är medvetna om, men vi tror att studien kan vara till hjälp för förvaltningsuppdrag i liknande sits.

Mycket litteratur finns tillgängligt om Continuous Delivery. Detta gjorde att vi hade svårt att välja inriktning på vår studie och ledde till att vår avgränsning gjordes sent. Hade avgränsning gjort tidigare hade vi fått en mer detaljrik studie.

Vi inser nu att vår studie hade varit tillräcklig om vi studerat ett fall. Men det visste vi inte i begynnelsen av studien då vi trodde att vi skulle identifiera skillnader mellan de olika fallen. Vi tror att man med endast ett fall skulle fått en djupare bild av det enskilda fallet och sett fler detaljer.

För att få en helhetsbild på hur moget ett förvaltningsuppdrag är för en övergång krävs det att man studerar alla kategorier i Maturity Model. I denna studie har endast två kategorier studeras ”Byggen och Deploy” och ”Test och Verifikation”. Detta på grund av tidsaspekten.

Då vi inte har någon erfarenhet inom releaser och deployer och hur man ”vanligtvis” jobbar. Har det tagit lång tid för oss att sätta in oss i ämnet och detaljer i releaseprocessen. Hade vi varit mer erfarna inom ämnet kunde denna tid lagts på mer ingående analyser och insamling av mer data.

6.3 Slutsats

Vi har kommit fram till tre huvudutmaningar som förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery.

- En utmaning är att det kan vara svårt att få med sig ledningen vid en övergång till CD. För att få de resurser som krävs för en övergång till CD är det viktigt att få detta stöd.
- Att kunden måste vilja och dessutom ha möjlighet att hantera fler releaser än vad de har i nuläget är en utmaning.
- Automatisering av delarna i deployment pipeline är den största utmananigen och den viktigaste att klara av.

6.4 Kunskapsbidrag

Vårt kunskapsbidrag till forskningen är en ökad förståelse för vilka utmaningar som ett förvaltningsuppdrag kan stå inför vid en övergång till Continuous Delivery. Det praktiska kunskapsbidraget är en beskrivning av Continuous Delivery och vilka utmaningar som ett förvaltningsuppdrag bör ta i beaktning om man vill genomföra en övergång till Continuous Delivery.

6.5 Förslag på framtida studier

Något som vi såg tendenser på under studien gång men som vi inte hann undersöka var hur olika system påverkar en övergång till Continuous Delivery. Vi menar exempelvis ett uppdrag som förvaltar en molntjänst mot ett uppdrag som förvaltar ett system med fast installation.

Källor

Abrantes, J. F., & Travassos, G. H. (2011). Common Agile Practices in Software Processes. I 2011 International Symposium on Empirical Software Engineering and Measurement (s. 355–358).

Agile Sweden - nätverket för flexibla systemutvecklingsmetoder. (2016, april 19). Hämtad 19 april 2016, från http://www.agilesweden.com/

Anderson, K. H., Kenyon, J. L., Hollis, B. R., Edwards, J., & Reid, B. (2014). Continuous deployment system for software development [Patent]. Nevada: Amazon Technologies, Inc.

ASP:NET (2016, juni 14). ASP.NET I Wikipedia. Hämtad 14 juni 2016, från https://sv.wikipedia.org/wiki/ASP.NET

Avdic, A. (2016, mars). Litteratursökning och referenser, Högskolan Dalarna

Balaji, S., & Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. International Journal of Information Technology and Business Management, 2(1), 26–30.

BusinessDictionary. (2016, maj 25). Feedback loop. Hämtad 25 maj 2016, från http://www.businessdictionary.com/definition/feedback-loop.html

C-sharp (2016, juni 14). C-sharp I Wikipedia. Hämtad 14 juni 2016. från https://sv.wikipedia.org/wiki/C-sharp

Chen, L. (2015). Continuous Delivery: Huge Benefits, but Challenges Too. IEEE Software, 32(2), 50–54. http://doi.org/10.1109/MS.2015.27

Commit (version control). (2016, maj 24). Commit (version control). I Wikipedia, the free encyclopedia. Hämtad 24 maj 2016, från https://en.wikipedia.org/w/index.php?title=Commit_(version_control)&oldid=680279001

Computer Sweden, S. (2016, maj 12). Skript. Hämtad 12 maj 2016, från http://it-ord.idg.se/ord/skript/

Denscombe, M. (2016). Forskningshandboken: för småskaliga forskningsprojekt inom samhällsvetenskaperna (3., och uppdaterade uppl). Lund: Studentlitteratur.

Dustin, E., Rashka, J., & Paul, J. (1999). Automated software testing: introduction, management, and performance. Harlow: Addison-Wesley.

Eriksson, U. (2008). Test och kvalitetssäkring av IT-system (2. uppl). Lund: Studentlitteratur.

Haverblad, A. (2007). IT service management i praktiken (2., uppdaterade och utök. uppl). Lund: Studentlitteratur.

Holmström Olsson, H., Alahyari, H., & Bosch, J. (2012). Climbing the Stairway to Heaven – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software, IEEE Xplore (s. 392–399). Presenterad vid 2012 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), IEEE Xplore. http://doi.org/10.1109/SEAA.2012.54

Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston: Pearson Education.

Högskolan Dalarna. (2013, december 20). Forskningsetiska anvisningar för examens- och uppsatsarbeten.pdf. Hämtad 08 juni 2016, från http://www.du.se/PageFiles/120320/Forskningsetiska%20anvisningar%20f%C3%B6r%20examens-%20och%20uppsatsarbeten.pdf

IDG:s it-ord. (2016, maj 25). Källkod | IDG:s it-ord. Hämtad 25 maj 2016, från http://it-ord.idg.se/ord/kallkod/

Jet Brains. (2016, maj 10). TeamCity. Hämtad 10 maj 2016, från https://confluence.jetbrains.com/display/TCD9/TeamCity+Documentation

KAMP företagsutveckling. (2016, maj 9). SWOT analys. Hämtad 9 maj 2016, från http://www.kamp.se/pdf/SWOT_analys_030203.pdf

Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V. P., Itkonen, J., Mäntylä, M. V., & Männistö, T. (2015). The Highways and Country Roads to Continuous Deployment. IEEE Software, 32(2), 64–72. http://doi.org/10.1109/MS.2015.50

Meyer, M. (2014). Continuous Integration and Its Tools. IEEE Software, 31(3), 14–16. http://doi.org/10.1109/MS.2014.58

Neely, S., & Stolt, S. (2013). Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). I Agile Conference (AGILE), 2013 (s. 121–128). http://doi.org/10.1109/AGILE.2013.17

Nordström, M., & Welander, T. (2007). Mera affärsmässig förvaltningsstyrning: en bok om (system-)förvaltning. Stockholm ; Lund: Dataföreningen : Studentlitteratur[distributör].

Oates, B. J. (2006). Researching information systems and computing. London: SAGE.

Oktaba, P. (2015, september 16). Deployment pipeline for dummies | Continuous Dev. Hämtad 27 april 2016, från http://continuousdev.com/2015/09/deployment-pipeline-for-dummies/

Pulkkinen, V. (2013). Continuous Deployment of Software. Cloud-Based Software Engineering, 46–52.

Regressionstestning. (2016, maj 25). Regressionstestning. I Wikipedia. Hämtad 25 maj, från https://sv.wikipedia.orghttps://sv.wikipedia.org/w/index.php?title=Regressionstestning&oldid=29168299

Rehn, A., Palmborg, T., & Boström, P. (2013). The Continuous Delivery Maturity Model. Stockholm:Diabol.

Repository. (2016, maj 25). Repository (version control). I Wikipedia, the free encyclopedia. Hämtad 25 maj,från https://en.wikipedia.org/w/index.php?title=Repository_(version_control)&oldid=680088968

Versionshantering. (2016, januari 8). Versionshantering. I Wikipedia. Hämtad 8 jan, från https://sv.wikipedia.orghttps://sv.wikipedia.org/w/index.php?title=Versionshantering&oldid=32354406

Windows Presentation Foundation. (2016, juni 14). Windows Presentation Foundation. I Wikipedia. Hämtad 14 juni från https://en.wikipedia.org/wiki/Windows_Presentation_Foundation

Bilaga 1 - Litteraturstudie, sökord

SÖKORD	DATABAS	ANTAL TRÄFFAR	I TITEL	FULL TEXT	PEER-REVIEW	ALLT	DATUM
continuous delivery	Summon	243 019		X	X		2015-11-23
"continuous delivery"	Summon	2490		X	X		2016-03-23
"continuous delivery" + implement	Summon	166		X	X		2016-03-23
"continuous delivery" + implement	Summon	1	X				2016-04-05
"continuous delivery" + development	Summon	1659		X	X		2016-03-23
continuous delivery	Google scholar	2 990 000				X	2016-03-23
"continuous delivery"	Google scholar	20 700				X	2016-03-23
"continuous delivery" + implement	Google scholar	6 220				X	2016-03-23
"continuous delivery" + implement	Google scholar	2	X				2016-04-05
continuous delivery	Libris Examensarbeten	58		X			2015-11-23
"continuous delivery" + development	Google scholar	16400				X	2016-03-23
"continuous delivery" + development		9	X				
continuous delivery vs waterafall	Google scholar	26500				X	2016-03-29

Bilaga 2 – Sökloggen

Författare	Titel	År	KällTyp	Syfte/slutsats/kunskapsbidrag	Nyckelord	Abstract/ sammanfattning	Hittad var?
Humble, J Farley, D	Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation	2010	Bok	Aim to make the dleivery of software from the hands of developers into production a reliable, predictable, visible and largely automated process with well-uunderstood, quantifiable risks. Using the approach that we describe in this book, it is possible to go from having an idea to delivering working code that implements it it production in a matter of minutes or hours, while at the same time improving the quality of the software thus delivered.	Reliable software releases	Coverage includes • Automating all facets of building, integrating, testing, and deploying software • Implementing deployment pipelines at team and organizational levels • Improving collaboration between developers, testers, and operations • Developing features incrementally on large and distributed teams • Implementing an effective configuration management strategy • Automating acceptance testing, from analysis to implementation • Testing capacity and other non-functional requirements • Implementing continuous deployment and zero-downtime releases • Managing infrastructure, data, components and dependencies • Navigating risk management, compliance, and auditing	Hittad genom en första google sökning.
Mathias Meyer	Continuous Integration and Its Tools	2014	artikel	Continuous Integration-is meant to be the unbi- ased judge of whether a change works or not, thereby preventing the “it works on my machine” syndrome before the code hits production.	Continuous Integration	Feature fl ips and monitoring aren’t practices originally consid- ered with continuous integration, but they’re invaluable additions to the process of shipping early and often. With a responsible team fo- cused on shipping and increasing customer value, continuous inte- gration can be a catalyst for long- lasting change, just by adding a few simple practices to your team’s workfl ow. When you add these hab- its together along with feature fl ips, fully automated and fast deploys, and monitoring, you’re ready to ship value to your customers anytime.	Hittad i summon med sökordet "Continuous Integration" Peer rewied
Neely S, Stolt S	Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy)	2013	Konf. Artikel	This experience report describes the journey to continuous delivery with the aim that others can learn from our mistakes and get their teams deploying more frequently. We will describe and contrast this transition from the business (product management) and engineering perspectives.	Continuous Delivery, challenges, mistakes	Rally Software transitioned from shipping code every eight-weeks, with time-boxed Scrum sprints, to a model of continuous delivery with Kanban. The team encountered complex challenges with their build systems, automated test suites, customer enablement, and internal communication. But there was light at the end of the tunnel — greater control and flexibility over feature releases, incremental delivery of value, lower risks, fewer defects, easier on-boarding of new developers, less off-hours work, and a considerable uptick in confidence. This experience report describes the journey to continuous delivery with the aim that others can learn from our mistakes and get their teams deploying more frequently. We will describe and contrast this transition from the business (product management) and engineering perspectives.	Forward search på "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation"

Forts söklogg

Författare	Titel	År	KällTyp	Syfte/slutsats/kunskapsbidrag	Nyckelord	Abstract/ sammanfattning	Hittad var?
Andreas Rehn, Tobias Palmborg and Patrik Boström	The Continuous Delivery Maturity Model	2013	White Paper	In this paper Maturity Model aims to give structure and understanding to some of the key aspects you need to consider when adopting Continuous Delivery in your organization	Maturity, organization	The principles and methods of Continuous Delivery are rapidly gaining recognition as a successful strategy for true business agility. For many organizations the question is no longer “why?”, but rather “how?” How do you start with Continuous Delivery, and how do you transform your organization to ensure sustainable results. This Maturity Model aims to give structure and understanding to some of the key aspects you need to consider when adopting Continuous Delivery in your organization.	
Ville Pulkkinen	Continuous Deployment of Software	2013	Konf Artikel (proceeding)	Goal of this article is to describe the current stateof-practice and to find out what is the current status of the continuous deployment in the software engineering research. And also point out issues that continuous deployment introduces to the software development.	Software Engineering,Software engineering for Internet projects,Programming Environments/Construction Tools,Programmer workbench,Configuration Management,Software release management and delivery	Continuous integration practice has gained popularity in the industry in the last decade. In the continuous integration practice all changes the developer commits to VCS are automatically and continuously tested for integration problems.Somehavetakenthispracticeevenfurthertoinclude also automated acceptance testing. The most extreme practice is to automate the whole process so that the deployment to the production environment is done automatically if tests pass. This is called the continuous deployment. Goal of this article is to i) describe the current stateof-practice and to ii) find out what is the current status of the continuous deployment in the software engineering research. I also will iii) point out issues that continuous deployment introduces to the software development. Lastly I briefly iiii) present how do the current cloud-providers support the continuous deployment strategy. The concept of the continuous deployment as well as difference to the contionuous delivery and the continuous integration strategy is provided. Some views of the continuous deployment strategyfromtheindustrywillbepresented.Lastlywewillhave short review of how current cloud-providers are supporting the continuous deployment strategy.	Hittade genom forward search på Maturity model (hade citerat maturity model)
Helena Holmström Olsson, Hiva Alahyari and Jan Bosch	Climbing the Stairway to Heaven – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software	2012	Konf. Artikel	This paper presents a multiple-case study in which we explore barriers associated with the transition towards continuous deployment. Based on interviews at four different software development companies we present key barriers in this transition as well as actions that need to be taken to address these	Agile software development; customer collaboration; continuous integration; continuous deployment	Agile software development is well-known for its focus on close customer collaboration and customer feedback. In emphasizing flexibility, efficiency and speed, agile practices have lead to a paradigm shift in how software is developed. However, while agile practices have succeeded in involving the customer in the development cycle, there is an urgent need to learn from customer usage of software also after delivering and deployment of the software product. The concept of continuous deployment, i.e. the ability to deliver software functionality frequently to customers and subsequently, the ability to continuously learn from real-time customer usage of software, has become attractive to companies realizing the potential in having even shorter feedback loops. However, the transition towards continuous deployment involves a number of barriers. This paper presents a multiple-case study in which we explore barriers associated with the transition towards continuous deployment. Based on interviews at four different software development companies we present key barriers in this transition as well as actions that need to be taken to address these.	Hittade genom backward search på "Continuous Delivery- Huge Benefits but Challenges too"

Forts söklogg

Författare	Titel	År	KällTyp	Syfte/slutsats/kunskapsbidrag	Nyckelord	Abstract/ sammanfattning	Hittad var?
Lianping Chen, Paddy Power	Continuous Delivery- Huge Benefits but Challenges too	2015	Artikel	This paper describes the resulting CD capability, and reports the huge benefits and challenges involved.	Continuous Delivery, benefits, challenges	Here, I explain how we adopted CD at Paddy Power, a large book- making company. I describe the resulting CD capability and report the huge benefits and challenges involved. These experiences can provide fellow practitioners with insights for their adoption of CD, and the identified challenges can provide researchers with valuable input for developing their research agendas.	Hittade på summon Peer rewied och fulltext sökord: "continuous delivery" + implement
S.Balaji , Dr.M.Sundararajan Murugaiyan	Wateerfall vs V-Model vs Agile: A comparative Study On SDLC	2012	Artikel	This comparative summarizes the steps an organization would have to go through in order to make the best possible choice.	SDLC, Waterfall, V-Model, Agile	Abstract: Organizations that are developing software solution are faced with the difficult choice of picking the right software development life cycle (SDLC). The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The V-model represents a software development process which may be considered an extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape Agile Modeling is a practice-based methodology for modelling and documentation of software-based systems. It is intended to be a collection of values, principles, and practices for modelling software that can be applied on a software development project in a more flexible manner than traditional Modelling methods.	Hittade på google scholar Continuous Delivery vs waterfall
M. Leppänen; S. Mäkinen ; M. Pagels ; V. P. Eloranta ; J. Itkonen ; M. V. Mäntylä ; T. Männistö	The Highways and Country Roads to Continuous Deployment	2015	Artikel	Despite understanding the benefits, none of the companies adopted a fully automatic deployment pipeline. The companies also had higher continuous-deployment capability than what they practiced. In many cases, they consciously chose to not aim for full continuous deployment. Obstacles to full adoption included domain-imposed restrictions, resistance to change, customer desires, and developers' skill and confidence.	continuous delivery continuous deployment continuous integration software development software engineering thematic analysis	As part of a Finnish research program, researchers interviewed 15 information and communications technology companies to determine the extent to which the companies adopted continuous deployment. They also aimed to find out why continuous deployment is considered beneficial and what the obstacles are to its full adoption. The benefits mentioned the most often were the ability to get faster feedback, the ability to deploy more often to keep customers satisfied, and improved quality and productivity. Despite understanding the benefits, none of the companies adopted a fully automatic deployment pipeline. The companies also had higher continuous-deployment capability than what they practiced. In many cases, they consciously chose to not aim for full continuous deployment. Obstacles to full adoption included domain-imposed restrictions, resistance to change, customer desires, and developers' skill and confidence.	Hittad genom backward search på "Understandings and Implementations of Continuous Delivery"

Bilaga 3 – Intervjufrågor

Vilken roll har du i projektet?

Hur många år har du arbetet på Triona?

Hur många år har du arbetat i branschen?

Vad jobbar ni med för produkt?

Vilket typ av projekt är det? Nyutveckling, förvaltning…?

Hur många kunder använder den produkten?

Vad använder ni för utvecklingsmodell?

Var kommer utvecklingsidéerna från? Är det mest från kunders krav?

Hur ser release-processen ut? (Deployment pipeline. Från commit till release)

Hur går det till när man tar beslut om när en release skall göras?

Hur går det till när man tar beslut som vad en release ska innehålla?

Hur ofta släpps en ny release?

Hur ofta deployas nya versioner till kunden?

Hur väljs releasedatum ut?

Vad i processen sker manuellt och vad sker automatiskt?

Är kunden delaktig i deploymentprocessen?

Har kunden någon inverkan på när en version ska deployas?

Hur samlas feedback in från kund?

Hur används feedback från kunder?

Hur många slutanvändare har produkten?

Hur används kodhantering idag?

Hur sker testningen?

Vad sker manuellt/automatiskt?

Vilka tester görs?

Vilken förvaltningsmodell används?