Automatisering av mjukvaruleverans med kontinuerlig leverans

Jarl-Otto Siikasmaa
Kandidatavhandling i Datateknik
Handledare: Ivan Porres
Fakulteten för naturvetenskaper och teknik
Åbo Akademi
2019

Abstrakt

Varje lyckad produkt i ett lyckat mjukvaruutvecklingsprojekt levereras till användarna. Utvecklingen slutar dock inte i första leveransen, utan produkten fortsätter att formas enligt användarnas behov. Under de senaste åren har takten mellan leveranserna och behovet för snabbare processer ökat. Automation spelar en stor roll i dessa processer och arbetet för bättre leveranskedjor utförs av allt flera företag.

I den här kandidatavhandlingen tas först automationen i mjukvaruutvecklingsprojekt kort upp. Automatisering har skett i olika takt för faserna i ett projekt, men metodologierna har senare förenats till sammanfattande koncept.

Efter presentationen av bakgrunden för ämnet ligger fokusen på implementering av automation i mjukvaruleverans. Detta diskuteras genom exempel på befinnande lösningar och verktyg som används inom industrin. Senare vägs upp nyttan och nackdelarna av automatiserad leverans i ett mjukvaruutvecklingsprojekt.

Nyckelord: mjukvaruutveckling, mjukvaruleverans, kontinuerlig leverans

Innehållsförteckning

1. Inledning

Leverans av mjukvaran i ett mjukvaruutvecklingsprojekt är en väsentlig del av skapandet av värde i projektet. Det är fråga om processen i projektet då mjukvaran har sammanställts och är färdig för distribution. Kontinuiteten för processen kommer från snabba inkrementella leveranser och automationen som kan utsträckas över hela utvecklingsprocessen; allt från integration av nya egenskaper till samlande av feedback. Inom denna avhandling kommer jag dock att fokusera på kontinuiteten i leveransen.

Automatisering av mjukvaruleverans användes för första gången i stora mjukvaruutvecklingsprojekt [1]. Metoderna för automatisering av leverans fungerar väl ihop med lean-tänkande, där enbart processer som ökar värdet i utvecklingsarbetet behålls [2]. I denna avhandling kommer jag att utforska tillämpningar av olika slag som utvecklare kan implementera i företaget för att spara arbetstid och undvika repetitivt manuellt arbete. Vid leverans av mjukvara uppstår särskilt ofta repetitiva arbetsuppgifter, då konfigurationerna i den levererade programmiljön måste vara förutsägbara och pålitliga.

Under det gångna året har jag jobbat i ett uppstartsföretag som mjukvaruutvecklare. Dessa frågor kring pålitliga processer, särskilt i leveransen, har uppstått med jämn takt. Mitt intresse för implementering av effektiva och automatiserade processer härstammar från beslut vi har fattat inom företaget. Ekonomiska och personalresurserna i uppstartsföretag är ofta väldigt begränsade och styr i hög grad beslutsfattandet. Metoderna som jag kommer att presentera har löst dessa frågor inom vårt företag och har bevisats fungera även i större koncerner.

Syftet med denna avhandling är att introducera läsaren till de möjligheter som kontinuerlig leverans av mjukvara medför i ett utvecklingsprojekt samt bakgrunden till och kraven som ställs på implementationen av metoderna i ett programvaruutvecklingsprojekt.

2. Bakgrund

2.1. DevOps

DevOps har sitt ursprung i frustration på problematiska kommunikationen mellan arbetsuppgifterna i ett programvaruutvecklingsprojekt [3]. DevOps är en metodologi som försöker binda mjukvaruutvecklare med it ledningen [3] [4]. Meningen är att öka flexibiliteten och kunskaper i grupper som består av dessa två arbetsuppgifter [3]. Varje iteration av metodologin har ökat samarbetet mellan arbetsuppgifterna i ett projekt. Första versionen hade grundtanken att utnyttja agila utvecklingsmetoder i varje utvecklingsprojektets fas. Senaste versionen av DevOps är DevSecOps, där ansvariga för mjukvarans säkerhet räknas med i gruppen. Grunderna för den nyare versionen har varit problemet med flaskhalsen då säkerhetsansvariga granskar en leveransversion första gången då programmet ska levereras. Detta ökar leveranstiden märkvärdigt, ifall mjukvaran har brister i säkerheten.

Huvudpunkterna i DevOps metodologin är att fördela ansvaret jämt mellan alla parter som är med i mjukvaruutvecklingsprojektet [1]. Därför anses metodologierna inom DevOps vara ett grundbehov för en väl implementerad kontinuerlig leveranskedja. Detta märks på arbetsmarknaden då efterfrågan på DevOps kunskaper har ökat enligt undersökningen utfört av The Linux Foundation och Dice [4].

Kontinuerlig integration och leverans fungerar naturligt ihop med DevOps metodologin. Kontinuerliga processerna strävar efter förutsägbara resultat, vilket kräver kommunikation mellan olika grupper i ett mjukvaruutvecklingsprojekt. En lyckad integration av nya egenskaper till källkoden skapar grunden för leveransen.

2.2. Kontinuerlig integration

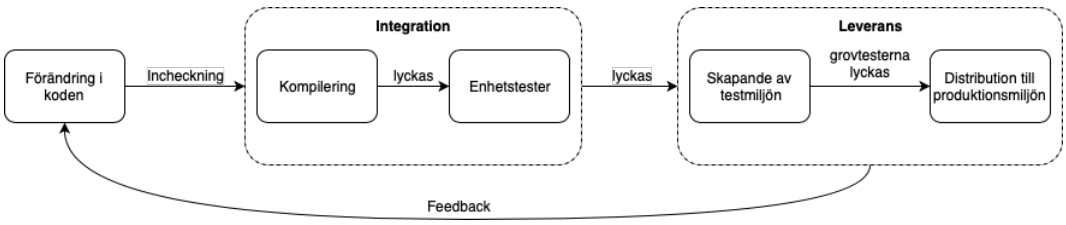
Varje kandidat för programmets leveransversion genomgår testning och kompilering före leveransen. Detta kallas för integration. Kontinuitet i processen

strävar efter genomkörning av kompilering och tester vid varje förändring i koden, istället för en körning före leveransen [1]. Metoden följer testdriven systemutveckling, som ingår i arbetssätten för extremprogrammering [1]. Detta gäller även för kontinuerlig leverans, då koden måste klara av alla tester innan den kan användas.

Kontinuerlig integration räknas vara som grundbehov för implementering av kontinuerlig leverans [4]. Inkrementella förändringar i koden ska testas och kompileras automatiskt för att uppnå snabbhet och hög effektivitet i processen.

2.3. Mjukvaruleverans

Programvaruutvecklingsprojekt strävar efter leverans av programvara och modifikationer på koden enligt användarnas feedback. Långa tider mellan leveransen och feedbacken ökar risken som projektet innebär. Föråldrad källkod för programvaran kräver alltmer förnyelser med tiden. Detta kallas för programmets tekniska skuld. Nyttan av feedbacken förelägger även nya risker, då användarnas åsikter om programvarans värde avviker från utvecklarnas visioner. Således ska varje implementerad egenskap i mjukvaran härstamma från en aktiv återkopplingsprocess, i vilken användarna kan påverka mjukvarans utveckling. På engelska talar man ofta om *pipelines* i sammanhang med kontinuerliga processer i mjukvaruutveckling. Dessa pipor syftar till hopkopplade processer, vilket i kontinuerlig leverans är en leveranspipa. Exempel på dessa piplinjer illustreras i figur 1.



Figur 1 – Modell för mjukvaruutveckling

Programvarans distribution är en del av hela leveranspipan. Processen har länge bestått av manuellt arbete. Arbetet som krävs av distributörerna vid leverans består främst av skapande av en produktionsmiljö för programvaran. Denna miljö kan vara i en fysisk dator eller en virtuell maskin som ställer vissa krav på hårdvara, infrastruktur och externa program. Enbart en miljö för programvaran räcker oftast inte i större projekt, där programmet kommunicerar med olika delar så som databaser, vilka i sin tur behöver annorlunda konfigurationer. Varje beståndsdel ökar däremot leveranstiden.

Tiden det tar att leverera programvaran kallas för leveranscykeln, på engelska *release cycle*. Processen börjar med feedback från användaren eller kvalitetskontrollen. Företag med flera mjukvaruutvecklare har uppdelat ansvaret för leveransprocessen i mindre faser så som utveckling, testning och leverans. Enligt undersökningen publicerad i IEEE varierar tiden beroende på industrin där programvaran används [5]. Det går dock inte att definiera en längd för leveranscykeln på basen av programmets användningsområde. Variationen har däremot varit mellan några veckor till flera månader. Leveransen i sin helhet blir en viktig del av programvaruutvecklingsprojektet, eftersom projektets värde skapas först då programvaran är i användning.

Det manuella arbetet i mjukvaruleveranser har sina grunder i säkerställande av programvarans funktionalitet under riktiga omständigheter. Det handlar om att testa programvaran i miljön där programmet kommer att användas och varje kandidat för programmets leveransversion genomgår samma omfattande testning. I företaget Paddy Power har inställandet av testmiljön före automatiseringen tagit upp till 3 veckor, varefter testandet har börjat [6]. Alla system kan däremot inte implementera automation i leveransen av programmet. Program som används i inbyggda och säkerhetskritiska system har bevisats minska möjligheten för automation [5].

Nya leveransversioner distribueras med många nya egenskaper, istället för enstaka förbättringar. Långa leveranscykler möjliggör detta och ifall leveransen består för det mesta av manuellt arbete så ökar större förbättringar effektiviteten på distributionen. Däremot, de första implementerade förändringarna väntar på resten av planerade implementationerna i pågående leveranscykeln. Under tiden kan det

komma duplikat av feedbacken då användarna inte är medvetna om befinnande förbättringar som väntar på resten av schemalagda förändringar.

2.4. Automatisering av leveransen

Automation i mjukvaruutveckling är inte ett nytt fenomen. Det har tillämpats automatiska tester för mjukvaran redan från början av 1990-talet [1]. Kontinuerlig integration är ett exempel på automatisering i mjukvaruutveckling. En förändring i koden som genomgått integrationspipan blir en levererbar version av programmet. Leveransen borde även automatiseras för att dra nyttan av andra automatiseringar [1]. Så som i kontinuerlig integration är pipan en central del av kontinuerlig leverans.

Början till leveranspipan räknas vara integration pipans slut. Kompilerade versionen av programmet har således skapats och säkerställts att fungera som sådant, utan förbindelser till andra beståndsdelar av programmet. Automationen i leveransen siktar på att lösa problemen med skapandet av omgivningen där programmet kommer användas i verkligheten. Detta kan indelas i faserna *acceptance test stage, manual test stage* och *release stage*, vilka på svenska motsvarar godkännande testfasen, manuella testfasen och utsläppningsfasen.

Godkännande testfasen siktar på att uppfylla funktionella och icke funktionella kraven som ställs av projektägaren och användarna. Programmet testas i den omgivning som den kommer att användas i med att upprätta automatiskt till exempel infrastrukturen [1]. Ifall det uppstår problem med installationen av kompilerade versionen från integrationsfasen avbryts piplinjen, i enlighet med principen om testdriven systemutveckling.

Manuella testfasen innehåller inte automation, men är däremot en efterföljd av automatiska processer och följs av automatiska utsläppningsfasen [1]. Behovet för manuella tester kan ha sitt ursprung i svårigheterna att testa användargränssnitt i webbläsare och mobilspel [5]. Fasen kan även vara ett tecken på opålitliga tester

och utvecklare, även om DevOps metodologierna strävar efter öppenhet inom projektet. Däremot kan fasen säkerställa värdeskapandet av förändringen och öka öppenhet, ifall fasen utförs på ett fungerande sätt.

Utsläppningsfasen, vilket är den sista fasen i leveranskedjan, innehåller åtgärder för automatiserad distribution av mjukvaran. Avsikten med fasen är att nya versionen ska uppnå användarna snabbt och smidigt för att samla in feedback före följande leveranscykeln. För mera diversifierad feedback på nya egenskaper används det så kallade *canary releases*, där en mindre andel av användarna grovtestar nya versionen före den tas allmänt i bruk [1]. I denna fasen har automation utnyttjats till att tilldela användarna olika leveransversioner och utföra rullande leverans av programmet, så att leveransen inte orsakar driftsavbrott.

Även om det är frågan om en process som är automatiserad, tyder det inte på att ingen människa skulle vara i kontakt med processen [1]. Tvärtom, ökar det pålitligheten av processen, då någon kan se med sina egna ögon att processen fungerar. Då DevOps metodologin används inom projektet, kan processen egentligen övervakas av alla deltagare i projektet.

3. Implementering av kontinuerlig leverans

I detta kapitel fokuserar vi på befinnande implementationer av kontinuerlig leverans i mjukvaruutvecklingsprojekt och kraven som ställs för mjukvaran, infrastrukturen och övervakning.

3.1. Utveckling av verktyg för kontinuerlig leverans

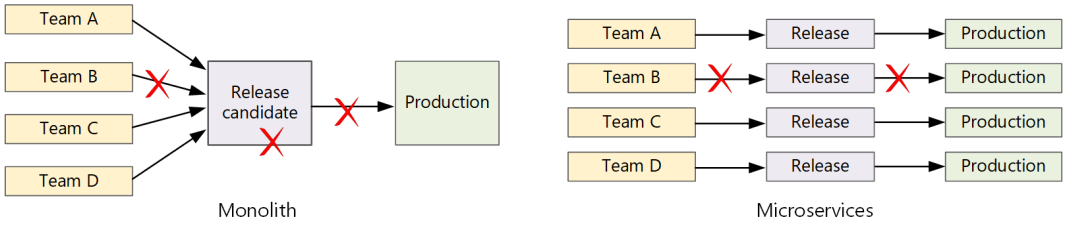
Det finns inga standarder inom industrin för implementering av kontinuerlig leverans. Metoderna för kontinuerlig leverans har implementerats på flera olika sätt i företag. Det finns färdiga verktyg som kan ta hand om både integrationen och leveransen, så som Gitlab och Jenkins, men sedan finns det verktyg som företag har själv utvecklat för eget bruk. Amerikanska videotjänsten Netflix började förflyttningen från lokala servrar till molntjänster år 2009 och redan 7 år senare lyckades de överföra alla applikationer till molnen [7] [8]. Detta krävde utveckling av verktyget Asgard, vars syfte är att underlätta resurshanteringen av molntjänster i organisationer med flera utvecklare [8]. Molntjänsterna har även introducerat nya metoder för distribution av applikationer i klungor av mindre tjänster. Kraven för snabb mjukvaruleverans har dock förändrats sedan första öppna versionen som publicerades 2012. Ersättande verktyget Spinnaker utvecklades sedan för att uppfylla dessa krav för en fullständigt kontinuerlig leveranskedja.

Google har utvecklat motsvarande verktyg, av vilka Kubernetes har snabbt tagits i bruk inom industrin. Kubernetes följer idéer så som snabb distribution, fördelning av program till tjänster och ökat samarbete mellan mjukvaruutvecklare [8]. Motsvarande orkestreringsverktyget till Kubernetes är Nomad utvecklat av Hashicorp Inc. Båda möjliggör infrastrukturen för rullande leverans av applikationer, vilket undviker driftsavbrott vid utsläppningsfasen. Således uppfylls en av idéerna med kontinuerlig leverans. Rullande leveransen i verktygen har förverkligats med virtuella instanser av applikationer, så som virtuella maskiner och programbehållare. Exempel på verktyg för dessa är Docker och Vagrant, som har blivit närapå standarder för virtuella omgivningar för applikationer.

Senaste lösningen till standardisering av kontinuerlig leverans har kommit från samarbetet mellan företagen som implementerat kontinuerlig leverans och stiftelsen *The Linux Foundation*. Samarbetsresultatet blev en stiftelse, *The Continuous Delivery Foundation*, som utvecklar och upprätthåller projekt som Spinnaker och Jenkins [9].

3.2. Krav på implementering av kontinuerlig leverans

Ett program kan bestå av flera olika tjänster. En applikation består oftast av en framände som syns för användaren och en bakände som växlar data till framändan. Dessa kan spjälkas till även mindre tjänster så som databaser och lastbalanserare. En helhet av svagt kopplade tjänster kallas för mikrotjänster. Kontinuerlig leverans underlättar implementeringen av mikrotjänster, eftersom varje del av applikationen kan utvecklas individuellt och samtidigt vilket ökar minskar tiden för leveranscykeln. Detta illustreras i figur 2 När automationen från kontinuerlig leverans är kombinerad med mikrotjänster är utvecklingen väldigt effektivt och målet för metodologin uppnås.



Figur 2 - Visualisering på effektiv utveckling med mikrotjänster och kontinuerlig leverans.

Källa: https://docs.microsoft.com/en-us/azure/architecture/microservices/ci-cd

För att testning av mikrotjänsten skulle inte vara begränsad av en annan mikrotjänst, måste även infrastrukturen vara tillräckligt flexibel. Nuförtiden kan integrationstestandet utföras på applikationsnivå med virtuella maskiner eller programbehållare, *containers* på engelska. Applikationer som utnyttjar sig av virtualiserad omgivning och sparar data utanför exekveringsmiljön, så som

databaser, räknas till tillståndslösa applikationer. Detta betyder att då applikationen startas skapas det nya inställningar av applikationen själv. Distributören kan således välja inställningar i farten, vilket underlättar reservprocedurerna för distribution av äldre versioner.

Som grundläggande krav för automatisering av leveransen kan räknas vara versionskontroll [1]. Detta beror på möjligheten att styra nyutvecklade egenskaper i programmet direkt till granskning innan leveransen. Versionskontroll möjliggör även automatiskt återladdning till tidigare versioner, ifall det sker fel i leveransen av nya versionen [10]. För implementationen antas dock att applikationen levereras fullständigt automatiskt.

3.2.1. Mjukvara

I kontinuerlig leverans förändras applikationen inkrementellt och förändringarna blir snabbt tillgängliga. Detta ställer baskraven att koden är återanvändbart, modulärt och lätt att upprätthålla. Brister i dessa krav leder till förlängda leveranscykler och starkt kopplade tjänster, vilket minskar i sin tur nyttan av automatiserad leverans.

Mjukvaran ska anpassa sig till leveransprocessen, inte tvärtemot. Kompilerade versionen av programmet borde användas från början till slut för att minimera risken att det sker förändringar i kompilerade versionen vid kompilation på en annan maskin.

Eftersom leveransprocessen kräver en lyckad integration av nya egenskaper, ska testerna även täcka de nya egenskaperna. Utan tester är processen inte pålitligt och strider mot metodologierna i kontinuerlig leverans. Automationen i processen är därmed beroende av väl täckande tester på mjukvaran.

3.2.2. Infrastruktur

Leverans av flera mikrotjänster som kopplas an och av vid leverans och fel ställer även stora krav för infrastrukturen. Modulariteten som mikrotjänsterna inbringar ett projekt ökar behovet för abstraktion av tjänsterna [11]. Eftersom det kan finnas flera instanser av en applikation, ökar komplexiteten för manuell hantering av värd i nätverket mellan mikrotjänsterna.

Orkestreringsverktyg för programbehållare, så som Kubernetes och Nomad, har löst problemet med så kallade tjänster, som gömmer varje instans av en applikation bakom ett värdnamn. Resurserna, så som hårdskivor, kan därmed delas mellan alla instanser som tillhör identifierade tjänsten. Tjänsterna kan även fungera som lastbalanserare, som fördelar trafiken jämt mellan instanserna eller enligt resurserna. Detta möjliggör rullande leveransen för tjänsterna, då vissa instanser kan ta emot trafik medan andra uppdateras.

Abstraktionerna har dock nackdelen med migrering av data. Nyare versioner av applikationer kan använda sig av annorlunda datastrukturer än de gamla, vilket minskar kompatibiliteten mellan versionerna. I databaser syns migration som förändrade tabellstrukturer, vilket påverkar tjänsterna kopplade till databasen. Problem kan uppstå då nya versioner av applikationen försöker hämta data från en obefintlig kolumn i databasen som håller på att uppdateras.

Eftersom det applikationen ska testas före användning i produktionsomgivningen, skapas det ofta separat miljö för att grovtesta applikationen. Miljön motsvarar oftast produktionsmiljön, vilket minimerar riskerna för miljö specifika problem. Det rekommenderas dock högst 2 miljön för testning, eftersom vid mer komplexa applikationer blir konfigurationerna besvärliga.

3.2.3. Övervakning

Övervakning av utvecklingsprocessen sker på flera nivåer med kontinuerliga metoder. Versionskontroll underlättar övervakandet en stor del, ifall det implementeras rätt med kontinuerlig leverans. Processerna kräver förvaltning av förändringarna i källkoden och utöver dessa även av konfigurationerna [1].

Till leveransprocessen tillhör även övervakning av leveransversionen funktionalitet i produktion. Det kan vara fråga om insamlande av feedback, belastningsmonitorering och loggning. Datamängden blir särskilt stor då applikationen är indelad i flera mikrotjänster och det uppstår problem i produktionsmiljön. Lösning på detta problemet är användningen av A/B-tester, där användarna får olika versioner av programmet och användningen analyseras noggrant. Ett exempel på A/B-testande är tidigare nämnda distributionsstrategin *canary releases*. För att implementera möjligheten för A/B-testande, ska det finnas flera versioner applikationen i användning. Detta kräver däremot att varje version stöds och fungerar normalt.

4. Nyttan av kontinuerlig leverans i projekt

Implementering av kontinuerlig leverans ställer krav på flera olika nivåer av projektet. Exemplen på företag som implementerat automatiserade leveransprocesser är globala företag som har märkt nyttan över kraven. Finns det något nyttan av kontinuerlig leverans?

Av de grundläggande kraven för automatisering av leveransen är versionskontroll redan till stort nytta inom utvecklingsprojekt. Det möjliggör reservprocedurer för återställning till äldre versioner ifall senaste leveransversionen uppges som icke-funktionell. Nuförtiden implementerar flesta it-företag versionshantering, inte enbart för källkod men för allt projektrelaterat [1].

För att hålla lätt reda på vad som har ändrats i källkoden vid leverans, hjälper versionskontroll även till. Större förändringar med flera nya egenskaper blir dock svårt att genomsöka. Dessutom kan de första implementerade egenskaperna vid utvecklingsfasen vänta länge på feedback då funktionaliteten inte testas före allt annat blir färdigt. Mindre inkrementella förändringar i källkoden har visats öka projektets värde, eftersom egentliga leveranscykeln blir kortare och feedbacken snabbare.

Största fördelen som automationen inbringar ett projekt är konsistens i processerna. Manuella konfigurationerna och testerna av programmets leveransversion kan leda till fel som loggas inte. Detta leder till en utredning, vilket förlänger leveranstiden och kan minska kundens tillförlitlighet på projektets lyckande. Därför har flera företag implementerat även automatisering av konfigurationer som stöd, även om leveransen i sig skulle ske manuellt.

4.1. Projektägare

I långsiktiga projekt skapas det ofta iterativt prototyper för produkten. Mjukvaruutvecklingsprojekt skiljer sig inte från dessa. Redan vid början av

projektet kan leveransen automatiseras, även om produktutvecklandet har inte blivit färdigt. Detta lönar sig för skapandet av funktionella prototyper. I undersökningen utfört av Leppänen [5] har kollaborationen mellan projektdeltagarna ökat och mjukvaruutvecklandet har i princip börjats då egenskaperna har blivit överenskomna. Återkopplingsprocessen är även snabbare redan från början, ifall projektägaren blir introducerad med automatisk leverans.

Under projektets gång blir det billigare att fixa fel, ju tidigare de upptäcks [1].

4.2. Projektarbetare

Då leveransen är automatiserad minskar behovet för manuellt arbete. Projektarbetare kan hålla sin fokus på utvecklingsarbetet och oftast utses det någon person inom projektet att konfigurera och övervaka automatiserade leveransprocessen. Ingående automatiserade testerna kräver mera pålitliga tester av utvecklarna, vilket i sin tur styr mot goda praxis i utvecklingsarbetet. Detta gäller även för föråldrande kod, som upprätthålls i långsammare takt än resten av applikationen, och fördelning av applikationen till mindre tjänster för parallell utveckling.

4.3. Användare

Applikationen kan snabbare modifieras för att motsvara användarens behov. Detta beror på mängden feedback som användarna ger och leveranscykeln som används i projektet.

5. Framtid

WIP

För att förstärka grunderna för användning av kontinuerlig leverans i organisationerna, har The Linux Foundation startat en stiftelse Continuous Delivery Foundation. Föreningens mål är att öka samarbetet mellan organisationer som använder kontinuerlig leverans, utan att anknyta sig till något tjänst för mjukvaruleverans.

6. Diskussion

DevOps i samband med kontinuerlig integration och leverans har visat vara ett sätt för att öka konkurrenskraft i produktutveckling inom mjukvaruutvecklingsprojekt. Satsningarna för implementationen kan vara stora med tanke på personalens kunskaper och tekniska resurskravet, men vidare utveckling av verktygen har löst behoven både internt och externt i organisationer. Metodologierna har redan funnits för några år och under denna tid har leveransens målplattformar skiftat från lokala lösningar till mera molnbaserade system.

För att vidare utveckla denna avhandling borde funktionaliteten och nyttan av en kontinuerlig leveranskedja uppföljas under en längre tidsperiod. Första standarderna för kontinuerlig leverans börjar forma sig med uppväxandet av stiftelsen The Continuous Delivery Foundation, var stora it-företag deltar i formandet av modeller och goda praxis för kontinuerlig leverans. Bristerna i standarderna försämrar möjligheten att jämföra nyttan av implementationerna, då varje system inte kan tillämpa automatiska processer på grund av säkerheten. Inom den närmaste framtiden kan standardiseringen dock utsträckas även till de säkerhetskritiska systemen, ifall det finns tillräckligt med studier och bevis på fullständigt automatiserade leveranskedjor.

Tack vare de befinnande verktygen för kontinuerlig leverans, kan metoderna tas i bruk utan extra kostnad och stora satsningar. Varje projekt har möjligheten att prova processerna redan i början för att undvika senare krångligheter vid övergång från en utvecklingsmodell till en annan.

7. Bilagor

8. Referenser

[1] J. Humble och D. Farley, Continuous delivery : reliable software releases through build, test, and deployment automation, Boston, MA: Addison-Wesley Professional, 2010.

[2] E. Ries, The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, 2011.

[3] M. Loukides, ”The evolution of DevOps,” O'Reilly Media, 31 August 2017. [Online]. Available: https://www.oreilly.com/ideas/the-evolution-of-devops. [Använd 14 March 2019].

[4] The Linux Foundation, Dice, ”The 2018 Open Source Jobs Report,” The Linux Foundation and Dice, 2018.

[5] M. Leppänen, S. Mäkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mäntylä and T. Männistö, "The highways and country roads to continuous deployment," *IEEE Software,* vol. 32, no. 2, 2015.

[6] L. Chen, ”Towards Architecting for Continuous Delivery,” i *2015 12th Working IEEE/IFIP Conference on Software Architecture*, Montreal, QC, Canada, 2015.

[7] E. Burns, A. Feldman, R. Fletcher, T. Lin, J. Reynolds, C. Sanden, L. Wander och R. Zienert, Continuous Delivery with Spinnaker, Sebastopol, CA: O'Reilly Media Inc, 2018.

[8] K. Hightower, B. Burns och J. Beda, Kubernetes: Up and Running, Sebastopol: O'Reilly Media Inc., 2017.

[9] The Linux Foundation, ”The Continuous Delivery Foundation,” The Linux Foundation, [Online]. Available: https://cd.foundation. [Använd 19 3 2019].

[10] L. Chen, ”Continuous Delivery: Huge Benefits, but Challenges Too,” *IEEE Software,* vol. 32, nr 2, 2015.

[11] P. Swartout, Continuous Delivery and DevOps: A Quickstart Guide” Excerpt From: “Continuous Delivery and DevOps: A Quickstart Guide, Birmingham: Packt Publishing, 2012, pp. 20-60.