LUNDS UNIVERSITET
Ekonomihögskolan

Institutionen för informatik

QA i organisationer som tillämpar DevOps

- Ett skifte från kvalitetssäkring till kvalitetsassistans

Kandidatuppsats 15 hp, kurs SYSK02 i informationssystem

Författare: Richard Johansson
Linus Karlsson

Handledare: Odd Steen

Examinatorer: Magnus Wärja
Paul Pierce

Framlagd: Maj 2016

QA i organisationer som tillämpar DevOps: Ett skifte från kvalitetssäkring till kvalitetsassistans

Slutseminarium: 2016-05-26

Författare: Richard Johansson och Linus Karlsson

Utgivare: Inst. för informatik, Ekonomihögskolan, Lund universitet

Dokumenttyp: Kandidatuppsats

Antal sidor: 78

Nyckelord: DevOps, Quality Assurance, Quality Assistance, QA, kvalitetssäkring, testning

Abstrakt

En effektiv mjukvarulivscykel och korta iterationer är idag förutsättningar för mjukvaruutvecklande organisationer. DevOps är en uppsättning koncept som ämnar optimera mjukvarulivscykeln genom att strömlinjeforma både utvecklings- och leveransprocessen och således korta ner tiden till marknaden för mjukvaruförbättringar. I ett sådant klimat, där snabba och frekventa releaser prioriteras högt, kan det tänkas att kvalitetssäkring nedprioriteras eller genomförs i mindre utsträckning, vilket kan ha en negativ inverkan på produktens kvalitet och användarnas upplevelser. DevOps-litteratur framhåller kvalitetssäkring som viktigt, men fokuserar lite på kvalitetssäkringens förhållande till DevOps och hur kvalitetssäkringsaktiviteter bedrivs i en DevOps-kontext. Denna studie avser identifiera detta. En kvalitativ studie där djupintervjuer med fyra personer som representerar två DevOps-präglade mjukvaruutvecklande organisationer genomfördes för insamling av empirisk data. Båda organisationerna bedriver kvalitetssäkring på liknande sätt; genom kontinuitet och automation, mätning, övervakning och återkoppling, som möjliggörs av en samarbetspräglad kultur och organisationsstruktur. Studien påvisar ett tydligt skifte från kvalitetssäkring mot kvalitetsassistans. Kvalitetssäkring är i en DevOps-kontext en kollektiv insats, vilket ställer krav på att alla som är involverade i produktutvecklingen kontinuerligt arbetar med, samt tar ansvar för, kvalitet. QA-funktionen understödjer och coachar individer och grupper i detta arbete, genom att tillhandahålla kunskap, rådgivning och verktyg. Således assisteras kvalitet proaktivt, snarare än säkras reaktivt.

Innehåll

Tabeller

Figurer

1 Introduktion

1.1 Bakgrund

Mjukvaruutvecklande företag står inför en gemensam utmaning. Kunder förväntar sig snabba svar på deras ständigt föränderliga krav, både vad gäller funktionella och icke-funktionella egenskaper hos mjukvaruprodukter (Humble & Farley, 2010). Frekventa releaser är avgörande för att mjukvaruutvecklande organisationer ska kunna tillfredsställa marknadens höga förväntningar. Att ha en kort mjukvarulivscykel och därmed snabbare tid till marknaden för produktförbättringar betyder för mjukvaruutvecklande organisationer ett övertag gentemot konkurrenter och möjligheten att ligga i fas med nya marknadstrender (Wettinger, Breitenbücher & Leymann, 2014; Armenise, 2015).

Företag som Facebook, Flickr och Netflix har anammat detta och driftsätter nya releaser på daglig basis (Feitelson, Frachtenberg & Beck, 2013; Bang et al., 2013). Denna utveckling mot kontinuerlig leverans av mjukvaruförbättringar ställer både krav på, och skapar möjligheter för, mjukvaruutvecklande organisationer (Claps, Berntsson Svensson & Aurum, 2015). För att hålla sig ikapp med marknadens höga releasetempo och kunders föränderliga behov måste företag förbättra kommunikationen mellan intressenter, automatisera processer och öka flexibiliteten i design, utveckling, leverans och drift av sina mjukvaruprodukter och tjänster (Lwakatare, Kuvaja & Oivo, 2015). Agil mjukvaruutveckling har blivit allt mer vanligt för att öka flexibiliteten, produktiviteten och tempot i utvecklingsprocessen. Agila metoder och dess tillämpningar automatiserar och ökar produktiviteten i mjukvarulivscykelns planerings-, utvecklings-, integrations- och enhetstestningsfas, men adresserar inte dess senare faser som integrations- och systemtestning, driftsättning, release, underhåll och övervakning (Virmani, 2015; Claps et al., 2015; Rodríguez et al., 2016).

DevOps (en ordsammansättning av Development och Operations) är en framväxande uppsättning koncept, som till skillnad från agil mjukvaruutveckling fokuserar på optimering av hela mjukvarulivscykeln, inklusive dess senare faser (Armenise, 2015). DevOps kan beskrivas som en evolution av agil mjukvaruutveckling då det till stor del bygger på samma principer, bland annat stärkt kommunikation och samarbete mellan organisatoriska funktioner, nyttjande av gemensamma processer och verktyg, så väl som automation av manuella aktiviteter. Skillnaden är att DevOps fokuserar på att eliminera flaskhalsar och hinder som förekommer i hela mjukvarulivscykeln, inte bara i dess inledande faser. Dess yttersta syfte är att effektivisera mjukvarulivscykeln och således göra det möjligt för mjukvaruutvecklande organisationer att snabbare kunna få ut mjukvaruförbättringar på marknaden.

1.2 Problemområde

I mjukvaruutvecklande företag som följer en sekventiell utvecklingsmodell är de organisatoriska funktionerna utveckling, QA (Quality Assurance) och systemadministration ofta separat uppdelade (Erich, Amrit & Daneva, 2014). Utvecklare arbetar avskärmat och anses vara färdiga när koden är skriven, och låter sedan QA testa produkten utifrån dess initiala kravspecifikationer, som därefter lämnar de resterande stegen i mjukvarulivscykeln till systemadministratörer att hantera på egen hand (Virmani, 2015; Roche, 2013). Ungefär samma situation gäller för företag som tillämpar agila metoder, skillnaden är att utvecklarna arbetar flexibelt i iterationer och integrerar samt enhetstestar sin kod oftare (Armenise, 2015), det förekommer emellertid ändå en slags "over-the-wall"-mentalitet. Detta uppdelade arbetsflöde kan leda till att de olika parterna hamnar i en situation där det är lätt att skylla på varandra när slutprodukten inte fungerar som tänkt, vilket kan leda till att den helhetliga mjukvarulivscykeln blir utdragen och tidskrävande (Hütterman, 2012). DevOps fokuserar på detta och utlovar kortare tid till marknaden genom en kombination av bland annat korta iterationer, automation, eliminering av flaskhalsar och funktionsöverskridande samarbete som sträcker sig över hela produktutvecklingsorganisationen.

Termen DevOps antyder att "Dev" och "Ops" bör föras samman. Ett av dess främsta åtaganden är just detta; att minska det gap som ofta förekommer mellan utveckling och systemadministration, genom att fostra tätare samarbete och integration mellan de båda funktionerna (Smeds, Nybom & Porres, 2015; Wettinger et al., 2014; Erich et al., 2014; Lwakatare et al., 2015). Virmani (2015) påpekar emellertid att DevOps inte enbart syftar till att integrera mjukvaruutveckling och systemadministration, utan att det i stort kretsar kring organisationsövergripande optimering av utvecklings- och leveransprocessen och produkten. Bang et al. (2013) beskriver att DevOps ämnar integrera samtliga funktioner som är involverade i en mjukvaruutvecklande organisations produktutveckling. Detta leder till att QA hamnar i en oviss position. I organisationer som präglas av korta iterationer, automation och tvärfunktionellt samarbete blir gränserna mellan funktionerna utveckling, QA och systemadministration allt mer flytande (Roche, 2013). Typiska arbetsuppgifter hos QA-funktionen som exempelvis testning av funktionalitet präglas till stor del av automation och integreras tidigare i utvecklingsprocessen. Ansvaret för många kvalitetssäkringsaktiviteter kan till och med komma att flyttas över helt och hållet till utvecklare eller systemadministratörer (Erich et al., 2014; Bhasin, 2012; Collins & de Lucena, 2012). QA-funktionens arbete blir allt svårare att förutse, och dess roll i organisationen allt mer otydlig (Roche, 2013).

Att aktivt arbeta mot säkerställande av mjukvarukvalitet är och förblir icke desto mindre en fundamental aktivitet inom mjukvaruutveckling, i synnerhet för organisationer där en snabb och effektiv mjukvarulivscykel är en prioritet (Rodríguez et al., 2016). Att produkten prioriteras att släppas på marknaden snabbt kan tänkas leda till kvalitetskompromisser vilket i sin tur kan leda till otillfredsställda användarupplevelser. Detta kan skada organisationens rykte och trovärdighet och därmed medföra dyra kostnader för att åtgärda detta problem (Edelen, 2014). Känslan av kvalitet, och kvalitetssäkring för att uppnå detta, har en betydande inverkan på

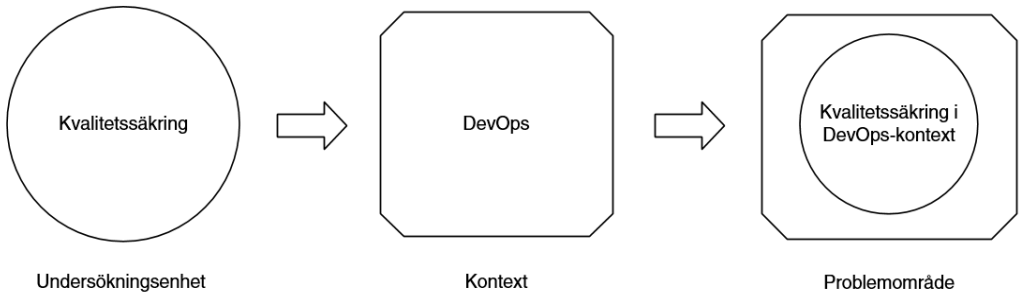
slutanvändares bedömning av produkten, och är därav en avgörande faktor för organisationens framgång. Litteratur inom DevOps framhåller kvalitetssäkring som essentiellt och pekar bland annat på att QA bör inta en mer proaktiv ställning, involveras tidigare, och samarbeta tätare med utvecklare och systemadministratörer för att kunna åtgärda problem så tidigt som möjligt (Roche, 2013; Erich et al., 2014; Armenise, 2015; Rodríguez et al., 2016). Däremot finns det begränsat med litteratur som mer ingående beskriver kvalitetssäkringens förhållande till DevOps och hur kvalitetssäkringsaktiviteter bedrivs i DevOps-präglade mjukvaruutvecklande organisationer.

1.3 Forskningsfråga

DevOps förespråkar samarbete och integration av mjukvaruutvecklande organisationers produktutvecklingsfunktioner. Detta ämnar optimera både utvecklings- och leveransprocessen och därmed effektivisera mjukvarulivscykeln och förkorta tiden till marknaden. DevOps-litteratur framhåller kvalitetssäkring som viktigt och nödvändigt, men utesluter i stor utsträckning hur det förhåller sig till DevOps och hur arbetet med kvalitetssäkring bedrivs i en DevOps-kontext. Ur detta härleds vår forskningsfråga som formulerats till: *Hur bedrivs kvalitetssäkring i organisationer som tillämpar DevOps?*

1.4 Syfte

Studiens problemområde indikerar att det är ovisst hur kvalitetssäkring bedrivs i DevOps-präglade organisationer. Alltså, undersökningsenheten i fokus är kvalitetssäkring och vi vill ta reda på hur den förefaller i en DevOps-kontext. Syftet med denna studie är därmed att identifiera hur mjukvaruutvecklande organisationer som tillämpar DevOps bedriver kvalitetssäkringsaktiviteter.



Figur 1.1: Förtydligande illustration av studiens syfte

1.5 Avgränsningar

Undersökningsenheten för denna studie är som tidigare beskrivet kvalitetssäkring, och kontexten är DevOps. Således kommer vi inte behandla eller utvärdera potentiella fördelar, risker eller utmaningar med DevOps i sig. Vi kommer heller inte att undersöka eller identifiera hur slutproduktens eller utvecklings- och leveransprocessens kvalitet påverkas som ett resultat av kvalitetssäkringen, utan istället fokusera på hur kvalitetssäkringsaktiviteterna i sig bedrivs i mjukvaruutvecklande organisationer som tillämpar DevOps.

1.6 Intressenter

Intressenter för denna studie inkluderar individer och grupper som arbetar med kvalitet och kvalitetssäkring, både direkt (till exempel QA-personal och testare) och indirekt (till exempel utvecklare, systemadministratörer, produktägare, business analysts, agila coacher och ledare). Forskare inom ämnesområdena DevOps, kontinuerlig integration, kontinuerlig leverans, kontinuerlig driftsättning, mjukvarukvalitetssäkring, mjukvarukvalitet, testning, testautomation och agil mjukvaruutveckling är också intressenter då dessa ämnesområden är relaterade.

1.7 Centrala begrepp

Nedan förklaras ett antal nyckelbegrepp som används återkommande i uppsatsen och är därav essentiella för läsaren att tolka i enlighet med vår intention.

Kvalitetssäkring - Med detta begrepp syftar vi till kvalitetssäkring av mjukvara, vilket på engelska kallas Software Quality Assurance (SQA). I och med att uppsatsen emellertid handlar uteslutande om kvalitetssäkring i ett mjukvarusammanhang används termen Quality Assurance eller QA. Viktigt är att skilja på begreppen *kvalitet* och *kvalitetssäkring*.

Mjukvarulivscykel - Kedjan av faser och aktiviteter inom produktutvecklingsprocessen, vilken på engelska kallas System Development Life Cycle (SDLC). Denna inkluderar två huvudprocesser: (1) utvecklingsprocessen vilken innefattar planering, utveckling, integration och till viss del testning; samt (2) leveransprocessen vilken innefattar testning, driftsättning, release, underhåll och övervakning.

Driftsättning - På engelska Deployment. Denna term syftar till fasen då en produkt eller funktion rent tekniskt försätts i en förproduktions- eller produktionsmiljö. Driftsättning i produktionsmiljö innebär nödvändigtvis inte att produkten är tillgänglig eller synlig för användare.

Release - Denna term ämnar beskriva ett medvetet produktsläpp eller en produktlansering vilken resulterar i att slutanvändaren har full tillgång till den släppta funktionen i produktionsmiljö.

2 Litteratur kring kvalitetssäkring och DevOps

Detta kapitel ämnar redovisa litteratur inom ämnesområdena kvalitetssäkring och DevOps. Litteraturgenomgången inleds genom att definiera vad kvalitetssäkring innebär och förankra dess syfte. Därefter förklaras bakgrunden till DevOps, för att sedan definiera dess syfte och innebörd, samt kategorisera dess grunddrag.

2.1 Kvalitetssäkring

Kvalitetssäkring (Quality Assurance, QA) är enligt IEEEs standard en uppsättning aktiviteter som definierar och utvärderar graden av ändamålsenlighet i mjukvaruprocesser för att tillhandahålla belägg som fastställer tillit i att mjukvaruprocesserna är lämpliga för, och producerar mjukvaruprodukter av relevant kvalitet för, deras avsedda syfte (IEEE, 2014). Kvalitetssäkring är alltså en uppsättning aktiviteter, och är inte att förväxla med den kvalitet som aktiviteterna bidrar till. QA-funktionens uppgift är att producera och samla in bevis som utgör grunden för att kunna ge ett motiverat förtroende för att mjukvaruprodukten uppfyller dess syfte. Kvalitetssäkringsaktiviteter ämnar kontrollera att mjukvaruprodukten uppfyller dess produktkrav, att enskilda aktiviteter uppfyller utvecklings- och leveransprocessens ursprungliga plan och krav, och att process- och produktkraven i sin tur uppfyller intressenternas initiala krav (IEEE, 2014). I praktiken innebär det bland annat kravställning, kodgranskning och olika typer av testning (Scharff, 2011). Just testning är enligt Nabulsi och Hierons (2015) en av de mest essentiella aktiviteterna inom kvalitetssäkring, vilket vidare bekräftas av Khalane och Tanner (2013) samt Sumrell (2007). Kvalitet definieras av IEEE (2014) som den fullständiga uppsättning attribut som ger en mjukvaruprodukt förmågan att tillfredsställa uttryckta eller underförstådda krav från intressenter. Krav från intressenter förfinas till produktkrav, vilket innefattar både funktionella krav samt kvalitetsattribut (icke-funktionella krav). Typiska kvalitetsattribut är användbarhet, säkerhet och prestanda (Wiegers & Beatty, 2013). Kvalitetsattribut specificerar hur väl mjukvaruprodukten utför de funktionella kraven.

Kvalitetssäkring genomförs på olika sätt beroende på vilken utvecklingsmetodik som används (Huo et al., 2004). I traditionella utvecklingsmetoder som till exempel vattenfallsmodellen fastställs tydliga projektfaser, vilka producerar distinkta leverabler som måste färdigställas innan nästa fas kan påbörjas (Huo et al., 2004). Därmed utförs kvalitetssäkring (till exempel testning) statiskt (sekventiellt) efter att utvecklingsprocessen är färdigställd, och utförs ofta manuellt. Agila metoder däremot bygger på korta iterationer, vilket medför i sin natur att QA-

personal arbetar mer dynamiskt och tätare tillsammans med utvecklarna, vilket leder till mer frekvent återkoppling (Huo et al., 2004). Inom agil mjukvaruutveckling sker vissa kvalitetssäkringsaktiviteter som enhetstestning mer frekvent med hjälp av automationsverktyg (Collins & de Lucena, 2012). DevOps är inte en utvecklingsmetod i sig, och kan således inte likställas vid agila metoder eller vattenfallsmodellen. Dess karaktärsdrag skapar emellertid både möjligheter för, och ställer krav på, hur kvalitetssäkring genomförs. För att skapa en grundförståelse för hur kvalitetssäkring bedrivs i en DevOps-kontext kommer begreppet DevOps att redas ut i kommande delkapitel.

2.2 DevOps

I detta delkapitel följer inledningsvis en beskrivning av härkomsten av DevOps och hur det relaterar till och bygger på agil mjukvaruutveckling. Vidare jämförs litteraturens ståndpunkter vad gäller definitionen av DevOps samt vilka grunddrag som karaktäriserar det. Slutligen tas en nyanserad definition av DevOps fram och kompletteras med en sammanställning av dess tre huvudsakliga kategorier av grunddrag som utmärkande framgår i genomgången litteratur.

2.2.1 DevOps är en evolution av agil mjukvaruutveckling

DevOps härstammar från idén om att korta ner tiden till marknaden och eliminera flaskhalsar i mjukvarulivscykelns kedja av faser och aktiviteter. Många av de principer och begrepp som DevOps bygger på, har rötter i de principer för agil mjukvaruutveckling som etablerades år 2001 i *The Agile Manifesto* (Beck et al., 2001). Agil mjukvaruutveckling fokuserar på att täta samarbetet mellan kunder och utvecklare genom iterativ utveckling, små och frekventa lösningsframtaganden och korta återkopplingscykler (Rodríguez et al., 2016; Cao et al., 2009). Agil mjukvaruutveckling har visat sig bidra till betydande effektivisering av mjukvarulivscykelns utvecklingsprocess och har därmed blivit allt mer vanlig bland mjukvaruutvecklande organisationer (Kaleshovska et al., 2015; Virmani, 2015; Armenise, 2015). Även om konceptet agil mjukvaruutveckling ökar produktiviteten i planerings-, utvecklings-, integrations- och enhetstestningsfasen, adresserar det inte mjukvarulivscykelns senare faser som integrations- och systemtestning, driftsättning, release, underhåll och övervakning (Virmani, 2015; Claps et al., 2015; Rodríguez et al., 2016). Att enbart ha en effektiv utvecklingsprocess räcker inte för att åstadkomma kortare tid till marknaden, eftersom leveransprocessen fortfarande agerar flaskhals. DevOps kan hävdas vara en evolution av agil mjukvaruutveckling som fokuserar på optimering av hela mjukvarulivscykeln, inklusive dess senare faser (Armenise, 2015).

DevOps varken är eller förespråkar emellertid någon specifik mjukvaruutvecklingsmetod, utan definieras bäst som en uppsättning koncept. En princip inom DevOps är att låta ett teams medlemmar själva bestämma vilka specifika metoder och verktyg de vill använda (Erich et al., 2014), vilket kan härledas från den agila grundprincipen “individer och interaktioner över verktyg och processer” (Beck et al., 2001). Det är därför svårt att peka på någon specifik me-

tod för mjukvaruutveckling som bör användas i en DevOps-organisation. Däremot kan det hävdas att sekventiella utvecklingsmetoder som vattenfallsmodellen inte är kompatibla med DevOps, eftersom att DevOps likt agil mjukvaruutveckling bygger på korta iterationer. Det finns vidare indikationer på att konkreta mjukvaruutvecklingsaktiviteter från agila metoder så som testdriven utveckling, beteendedriven utveckling, parprogrammering och refaktorisering ofta används i samband med DevOps (Erich et al., 2014; Rodríguez et al., 2016).

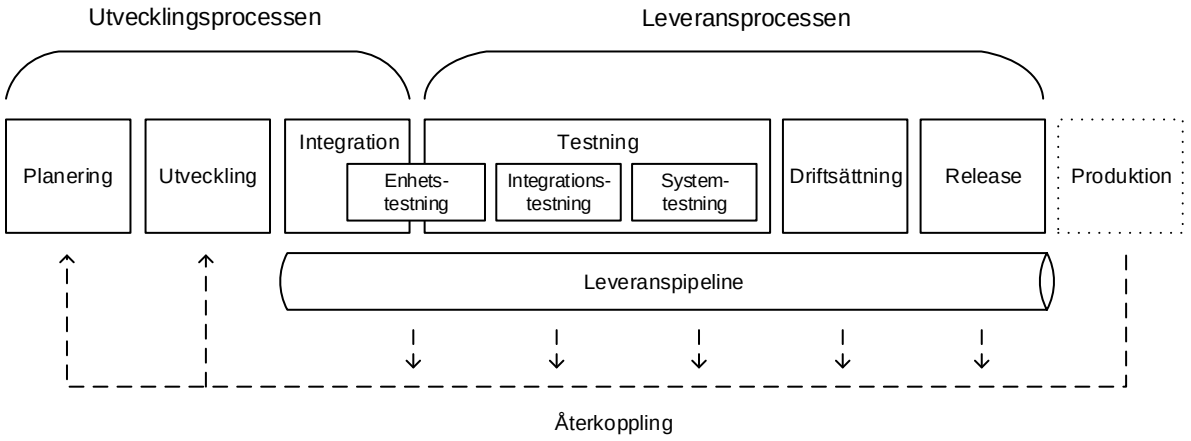
2.2.2 Leveranspipeline för kontinuerlig integration, leverans och driftsättning

En av DevOps grundprinciper är tvärfunktionellt samarbete. Genom att förena produktutvecklingsorganisationen kring samma verktygskedja kan mjukvaruutvecklande företag ta steget från agil mjukvaruutveckling och kontinuerlig integration, mot DevOps och kontinuerlig leverans eller kontinuerlig driftsättning. Dessa är grundläggande koncept inom DevOps som skapar kontinuitet i mjukvarulivscykeln med hjälp av tvärfunktionella processer och automationsverktyg (Rodríguez et al., 2016; Armenise, 2015; Smeds et al., 2015).

Kontinuerlig integration (Continuous Integration) introducerades ursprungligen av den agila metoden Extreme Programming (XP) och var ett av de första koncepten som gjorde det möjligt för mjukvaruutvecklande organisationer att signifikant effektivisera mjukvarulivscykeln (Armenise, 2015). Kontinuerlig integration är en dynamisk aktivitet vilken går ut på att utvecklare checkar in små kodändringar till en delad kodbas flera gånger per dag som sedan integreras till en enda mjukvaruartefakt. Automatiserade enhetstester som sedan triggas i samband med varje artefaktsbyggning ser till att defekter i koden upptäcks och flaggas omedelbart, vilket leder till att felsökning av integrationsproblem tar mindre tid och produktiviteten i utvecklingsprocessen ökas (Armenise, 2015).

Kontinuerlig leverans (Continuous Delivery) inkorporerar samt utökar konceptet kontinuerlig integration genom vidare automation av mjukvarulivscykelns leveransprocess, mer specifikt integrationstestning, systemtestning, driftsättning och releasehantering (Armenise, 2015). Kontinuerlig leverans är ett av de mest centrala koncepten inom DevOps och är en förutsättning för att åstadkomma en strömlinjeformad leveransprocess. *Kontinuerlig driftsättning* (Continuous Deployment) är en variation av kontinuerlig leverans, som går ut på att omedelbart produktionsdriftsätta nya kodändringar som passerar leveranspipelinens automatiserade steg och tester, utan några manuella kontroller (Rodríguez et al., 2016).

En *leveranspipeline* (Delivery Pipeline) är den kedja av tekniska verktyg som möjliggör kontinuerlig leverans och kontinuerlig driftsättning. Den etablerar stöd för att ha en fullt automatiserad leveransprocess från incheckning av kod, integration och artefaktsbyggning vidare till enhets-, integrations- och systemtestning, följt av driftsättning i en förproduktions- eller produktionsmiljö, och till sist release till marknaden (Rodríguez et al., 2016; Waller, Ehmke & Hasselbring, 2015). Figur 2.1 illustrerar sammanhanget mellan kontinuerlig leverans med en leveranspipeline och mjukvarulivscykelns processer och faser.



Figur 2.1: Leveranspipelinens förhållande till mjukvarulivscykelns processer och faser

2.2.3 Utredning av DevOps definition och grunddrag

Det finns olika definitioner av vad DevOps är och vad det innefattar. Roche (2013) fastställer att det finns två skilda ställningstaganden bland bloggare och branschvetare, där vissa menar att DevOps är en specifik arbetsroll som kräver blandade färdigheter inom både utveckling, kvalitetssäkring och systemadministration. Andra bloggare och branschvetare hävdar att DevOps är en filosofi, eller ett koncept, som inte kan härledas till en specifik arbetsroll. Roche (2013) pekar på att det senare av de två är det mest utbredda synsättet.

Hüttermann (2012) förstärker detta och hävdar att DevOps inte heller går att förlägga till en specifik DevOps-avdelning i en organisation. Han karaktäriserar de huvudsakliga grunddragen som kultur, automation, mätning samt delande och definierar DevOps som: tillämpningar som strömlinjeformar mjukvaruleveransprocessen, och fokuserar på lärande genom att strömma feedback från produktion till utveckling och därmed optimera mjukvarulivscykeln (Hütterman, 2012, s. 20). Enligt Smeds et al. (2015) är emellertid Hüttermans (2012) definition otillräcklig eftersom att de kulturella aspekterna av DevOps inte behandlas. De pekar istället mot Walls (2013) som beskriver att DevOps innebär en kulturell förändring som präglas av kulturella karaktärsdrag som samarbete, öppen kommunikation, sammanfogning av incitament och ansvar, respekt och tillit.

Smeds et al. (2015) fastställer att det inte råder några tvivel om att kultur och organisering verkar ha stor betydelse. Kultur kan emellertid inte definiera DevOps och dess grunddrag i sig, eftersom att begrepp som öppen kommunikation och tvärfunktionellt samarbete kan appliceras i vilken organisation som helst där människor med olika bakgrund eller färdigheter möts. DevOps bör således inte enbart definieras med utgångspunkt i dess kulturella aspekter. Smeds et al. (2015) beskriver istället DevOps med ett antal processmässiga kapabiliteter vilka är kontinuerlig planering, samarbetsmässig och kontinuerlig utveckling, kontinuerlig integration och testning, kontinuerliga releaser och driftsättningar, kontinuerlig övervakning och optimering av infrastruktur, kontinuerlig övervakning av användarbeteende och återkoppling,

samt återhämtning efter systemfel utan fördröjning. Dessa kapabiliteter understödjs av verktygskedjor för automation av artefaktsbyggning, testning, driftsättning, övervakning, återhämtning, infrastruktur, samt konfigurationshantering. Kulturella aspekter som möjliggör kapabiliteterna är delade mål, framgångsdefinitioner och incitament, delade arbetssätt, ansvar och kollektivt ägarskap, delade grundvärden, respekt och tillit, ansträngningslös kommunikation, samt kontinuerligt experimenterande och lärande.

Bang et al. (2013) hävdar att eftersom DevOps integrerar utveckling och systemadministration är ett flertal olika intressenter inblandade, bland andra business analysts, utvecklare, testare, QA-personal, system- databas- och nätverksadministratörer, web masters och säkerhetsansvariga. I och med denna diversifierade sammansättning av roller hävdar författarna att DevOps grunddrag är samarbetskultur mellan alla inblandade, automation av integration, driftsättning och testning, mätning av processer, kostnader, värde och teknisk metrik, samt delande av kunskap och verktyg. Lwakatare et al. (2015) konkretiserar DevOps i de fyra liknande grunddragen samarbete, automation, mätning, samt övervakning. Slutligen har Erich et al. (2014) utfört en systematisk litteraturgenomgång av ämnesområdet och konstaterar att DevOps har sju distinkta grunddrag, vilka alla kan deriveras från tidigare nämnda författares definitioner. Dessa är samarbetskultur, automation, mätning, delande av kunskap, datadriven kvalitetssäkring, molntjänster, samt strukturer och standarder.

Tabell 2.1: Sammanställning av olika författares syn på DevOps grunddrag

Författare	*Samarbetspräglad organisationsstruktur och kultur*	*Kontinuitet och automation*	*Mätning, övervakning och återkoppling*
Erich et al. (2014)	• Samarbetskultur • Delande av kunskap • Strukturer och standarder	• Automation • Molntjänster	• Mätning • Datadriven kvalitetssäkring
Hütterman (2012)	• Kultur • Delande	• Automation	• Mätning
Lwakatare et al. (2015)	• Samarbete	• Automation	• Mätning • Övervakning
Smeds et al. (2015)	• Delade mål, framgångsdefinitioner och incitament • Delade arbetssätt, ansvar och kollektivt ägarskap • Delade värderingar, tillit och respekt • Ansträngningslös kommunikation • Kontinuerligt experimenterande och lärande	• Kontinuerlig planering • Samarbetsmässig och kontinuerlig utveckling • Kontinuerlig integration och testning • Kontinuerliga releaser och driftsättningar • Tekniska verktyg för automation och konfigurationshantering	• Kontinuerlig övervakning och optimering av infrastruktur • Kontinuerlig övervakning av användarbeteende och återkoppling
Bang et al. (2014)	• Samarbetskultur • Delande av kunskap och verktyg	• Automation av integration, testning och driftsättning	• Mätning av processer, kostnader, värde och teknisk metrik
Walls (2013)	• Öppen kommunikation • Sammanfogning av ansvar och incitament • Respekt • Tillit		

2.2.4 Summering av DevOps definition och grunddrag

Med utgångspunkt i föregående sektions utredning av definitioner och perspektiv på DevOps har återkommande grunddrag identifierats (se tabell 2.1), vilka har formulerats som tre huvudkategorier av grunddrag som ligger till grund för denna uppsats: (1) *samarbetspräglad organisationsstruktur och kultur;* (2) *kontinuitet och automation;* samt (3) *mätning, övervakning och återkoppling.* Dessa tre huvudgrunddrag bidrar till vår definition av DevOps som har utformats till: *DevOps är en uppsättning koncept som ämnar optimera hela mjukvarulivscykeln och minimera tid till marknaden genom processer och verktyg för automation, kontinuitet, mätning, övervakning och återkoppling, vilket möjliggörs av en samarbetspräglad kultur och organisationsstruktur.*

Med mjukvarulivscykeln avses en produktförbättrings totala livstid, från början till slut. Dess faser innefattar planering, utveckling, integration, testning, driftsättning, release, underhåll och övervakning (Smeds et al., 2015; Waller et al., 2015; Rodríguez et al., 2016). Dessa faser och aktiviteter utförs kontinuerligt, vilket i denna kontext innebär i små portioner och utan

fördröjning (Smeds et al., 2015). Automation syftar till att strömlinjeforma utvecklings- och leveransprocessen med hjälp av tekniska automationsverktyg för att möjliggöra kontinuerlig integration, kontinuerlig testning, kontinuerlig leverans och kontinuerlig driftsättning. Automation frigör anställda från manuella repetitiva uppgifter och låter dem fokusera på innovation, kreativitet och produktivitet (Smeds et al., 2015). Övervakning, mätning och återkoppling innebär insamling av data kring användarbeteende och realtidsprestanda som sedan återkopplas och används primärt i utvecklingsprocessen (Roche, 2013; Liu, Li & Liu, 2014; Smeds et al., 2015). En samarbetspräglad kultur och organisationsstruktur innebär öppen och transparent kommunikation, delande av mål, värderingar, incitament, ansvar, kunskap och verktyg, gemensamma arbetssätt och kollektivt ägarskap, samt fokus på respekt och tillit (Walls, 2013; Smeds et al., 2015; Erich et al., 2014; Hütterman, 2012; Lwakatare et al., 2015).

2.3 Sammanfattning av litteratur

Kvalitetssäkringsaktiviteter ämnar fastställa tillit i att mjukvaruprodukten uppfyller dess produktkrav och syfte, att enskilda aktiviteter uppfyller utvecklings- och leveransprocessens ursprungliga plan och krav, samt att process- och produktkraven i sin tur uppfyller intressenters explicita eller implicita krav (IEEE, 2014). Dessa aktiviteter kan genomföras på olika sätt beroende på utvecklingsmetodik.

DevOps varken är eller förespråkar någon specifik mjukvaruutvecklingsmetod, utan är en uppsättning koncept som bygger på många av de grundläggande principerna från agil mjukvaruutveckling. DevOps förlänger och applicerar agila principer över hela både utvecklings- och leveransprocessen för att optimera mjukvarulivscykeln och åstadkomma en snabbare tid till marknaden. DevOps huvudsakliga grunddrag är samarbetspräglad organisationsstruktur och kultur, kontinuitet och automation, samt mätning, övervakning och återkoppling.

Traditionellt finns en strukturell uppdelning mellan utveckling, QA och systemadministration, där varje funktion är specialiserad på dess respektive fas(er) inom mjukvarulivscykeln. Att fösa produkten genom dessa traditionellt isolerade avdelningar leder till utdragna processer med många manuella steg och godkännanden (Armenise, 2015). Till skillnad från agil mjukvaruutveckling, som uppmuntrar ett tätare samarbete mellan kunder, affärsintressenter och utvecklare, förordar DevOps ett funktionsöverskridande samarbete som sträcker sig över hela produktutvecklingsorganisationen (Rodríguez et al., 2016). Kontinuerlig leverans och DevOps kräver att organisatoriska funktioner integreras, vilket uppnås genom verktyg och processer för automation och kontinuitet, mätning, övervakning och återkoppling, vilket i sin tur möjliggörs av en samarbetspräglad kultur och organisationsstruktur.

3 Kvalitetssäkring i en DevOps-kontext - Teoretiskt ramverk

I följande kapitel appliceras undersökningsenheten kvalitetssäkring i kontexten DevOps. Litteratur som förenar de båda ämnesområdena redovisas, och i dess helhet bildar kapitlet det teoretiska ramverk som ligger till grund för studiens fortfarande. Kapitlet är strukturerat i tre delkapitel vilka motsvarar DevOps tidigare fastställda huvudgrunddrag och i varje delkapitel presenteras i sin tur en rad kapabiliteter som karaktäriserar kvalitetssäkring i denna kontext. Slutligen presenteras en undersökningsmodell som sammanfattar det teoretiska ramverket.

3.1 Samarbetspräglad organisationsstruktur och kultur

Detta delkapitel redovisar ett antal kapabiliteter för kvalitetssäkring i DevOps-tillämpande organisationer, ur ett organisationsstrukturellt och kulturellt perspektiv.

K1 - Delande av arbetssätt, ansvar och kollektivt ägarskap för kvalitetssäkring

Kollektivt ägarskap är ett grundläggande grunddrag av DevOps, som syftar till att minska friktion som traditionellt kan uppstå vid missförstånd mellan avdelningar (Smeds et al., 2015). Följaktligen arbetar de anställda kollektivt på arbetsuppgifterna som ett "team av teams". Detta kollektiva ägarskap syftar till att gå ifrån det traditionella flödet där utvecklare anser sig vara klara när QA-personalen tar över och börjar testa, och systemadministratörernas arbete påbörjas först när det är dags för driftsättning. Istället fostras ett gemensamt ansvar för kvalitetssäkring där samtliga parter tar ansvar för att säkerställa hög kvalitet och användbarhet (Roche, 2013). Kontinuerlig optimering av utvecklings- och leveransprocessen och utökat samarbete mellan utvecklare, QA-personal och systemadministratörer förhöjer mjukvarulivscykelns effektivitet, såväl som produktens kvalitet (Bang et al., 2015; Erich et al., 2014).

K2 – QA har en teknisk profil och arbetar nära utvecklare och systemadministratörer

DevOps fordrar enligt Armenise (2015) att QA-personal har en mer teknisk profil än i traditionella organisationer, då de samarbetar tätare med både mjukvaruutvecklare och systemadministratörer genom mjukvarulivscykeln. Testaktiviteter integreras med utvecklings- och leveransprocesserna i så stor utsträckning som möjligt (Rodríguez et al., 2016), och ansvaret för vissa kvalitetssäkringsaktiviteter flyttas över helt och hållet till utvecklare och systemadministratörer (Sumrell, 2007; Erich et al., 2014). Detta ställer krav på att QA har en bredare för-

ståelse för systemets helhet, och inte minst teknisk kunskap inom centrala DevOps-förfaranden som till exempel testautomation. Roche (2013) påpekar att QAs roll i en DevOps-präglad mjukvarulivscykel påminner mer om en proaktiv kvalitetsförespråkare som förser utvecklare och systemadministratörer med datadrivna insikter om potentiella problem, än en reaktiv testare av funktionalitet. Detta kräver att organisationsstrukturen ser till så att QA-personal kan kommunicera och samarbeta ansträngningslöst med utvecklare och systemadministratörer.

K3 - Gemensamma mål och incitament för kvalitetssäkring

Delade arbetssätt och ansvar för kvalitetssäkring kräver delade mål och incitament (Smeds et al., 2015). Walls (2013) beskriver att eftersom alla delar ett och samma mål, att skapa en bra produkt för kunderna, bör den kollektiva insatsen belönas just när kundupplevelsen är bra. Utvecklare belönas således inte när de producerar mycket kod, QA-personal belönas inte för hur många tester de exekverar och systemadministratörer straffas inte när koden inte fungerar som tänkt i produktionsmiljön. Istället motiveras teamet att gemensamt driva kvalitetssäkringsarbetet och förbättra process- och produktkvaliteten (Walls, 2013). Bhasin (2012) formulerar detta som att kvalitetssäkring är en kontinuerligt integrerad del av alla involverades dagliga arbete.

K4 - Kontinuerligt experimenterande, lärande och delande av kunskap

En DevOps-präglad kultur genomsyras enligt Smeds et al. (2015) av innovation och kontinuerligt lärande. Detta stöds av en uppmuntran till en experimenterande och kunskapsdelande kultur där tidigare misslyckanden ses som möjligheter att dra lärdomar från. Detta utförs i praktiken genom att skapa tillfällen där anställda kan dela och diskutera sina erfarenheter. Scrum refererar till dessa tillfällen som retrospektiva möten (Cho, Huff & Olsen, 2011). Vidare fostras innovation genom att låta individer utföra småskaliga experiment med vetskapen om att de får misslyckas, vilket oavsett resultat bidrar till ytterligare kunskap som kan delas inom organisationen (Rodriguez et al., 2016). Organisationer som tillämpar hög grad av kunskapsdelande mellan individer, och därmed kontinuerligt lärande, har påvisats öka individernas prestationsförmåga vilket leder till ökad kvalitet i arbetsprocesserna (Ozer & Vogel, 2015). Att kontinuerligt dela kunskap inom en organisation minimerar därmed risken för återupprepning av misstag och bidrar till kvalitetssäkring av utvecklings- och leveransprocessen.

K5 - QA har inflytande över designbeslut

Enligt Roche (2013) krävs det vid beslut kring mjukvarudesign och arkitektur god insikt från olika typer av expertisområden kring hur systemet som helhet kan påverkas av beslutet. DevOps bygger på att det finns en samarbetspräglad kultur där tvärfunktionellt delande av ansvar, arbetssätt och kunskap är karaktärsdrag. Detta medför att olika personer med olika inriktningar besitter både exklusiv domänkunskap såväl som en överskådlig helhetsbild av produkten. Varje individ kan således bidra med olika typer av expertis vid diskussioner och beslut kring designfrågor. I detta sammanhang besitter QA ett unikt och värdefullt perspektiv på användares krav på kvalitet, precision och prestanda, vilket grundar sig i både erfarenhet

och domänkunskap såväl som insikter från analys av användar- och prestandadata (K13). Eftersom att DevOps förespråkar datadrivna återkopplingsslingor genom kontinuerlig mätning och övervakning finns det möjligheter att ta beslut baserat på faktagrundad information. Att QA-personal har inflytande över designbeslut är därför viktigt, då de har insikt i hur dessa typer av beslut kan påverka produktens kvalitetskänsla och användarnas upplevelser. Således kan team ta informerade designbeslut vilka grundas i prediktiv problemanalys och obestridlig information i form av mönster i användar- och prestandadata (K14) (Roche, 2013).

K6 - QA har inflytande över releasebeslut

Releasebeslut påverkas i regel alltid av olika drivkrafter från olika intressenter. Roche (2013) beskriver att beslut och diskussioner kring huruvida en defektfylld funktion kan släppas eller inte, bör ta hänsyn till på vilket sätt defekterna kan komma att påverka produktens helhetliga kvalitetskänsla. Med hjälp av information och återkoppling från kontinuerlig monitorering av användarbeteende och prestanda (K13) kan beslutsfattare proaktivt avgöra den inverkan som en defektfylld release kommer att ha på slutanvändarnas upplevelse. QA bör ha inflytande över dessa diskussioner och beslut, just för att de besitter datadriven och objektiv insikt i vilka produktområden som är kritiska för den övergripande användarupplevelsen. De kan identifiera och presentera vad slutanvändaren faktiskt kommer att se, och hur ofta (K14). Data bör användas som en ledstjärna vid dessa beslut (Roche, 2013). I de fall där det inte finns tillräcklig tillgång till datadrivna insikter eller erfarenheter kan principer som gradvis utrullning, A/B-testning (K15) och mörklagd produktlansering (K16) tillämpas. På så vis kan funktionen driftsättas för ett fåtal användare vilket medför att genomslagseffekt, såväl som drifts- och prestandaproblem proaktivt kan identifieras (Adams & McIntosh, 2016).

3.2 Kontinuitet och automation

I detta delkapitel redovisas kapabiliteter som är kopplade till kvalitetssäkringsaktiviteter med anknytning till kontinuitetsprinciper, utvecklingsmetoder, verktygskedjor och automation.

K7 - Kontinuerlig och automatiserad testning av funktionalitet och kvalitet

Kontinuerlig testning och automation av testaktiviteter är centrala grunddrag i DevOps (Waller et al., 2015; Smeds et al., 2015). Kontinuerlig testning ämnar föra samman testaktiviteter med utveckling av kod i så stor utsträckning som möjligt. På så vis kan defekter åtgärdas snabbare eftersom att kontexten i vilken de förekommer fortfarande är färsk i utvecklarens tankar. Således är det enklare att identifiera den underliggande orsaken till defektens uppkomst vilket kan förhindra en oönskad dominoeffekt. Oftast understöds dessa testaktiviteter av någon form av automation (Fitzgerald & Stol, 2015). Kontinuerlig och automatiserad testning kan användas för kvalitetssäkring av både funktionella krav såväl som kvalitetsattribut (Waller et al., 2015). Genom metoder som test- och beteendedriven utveckling (K8) kan automatiserade testsystem för verifikation och validering av funktionalitet byggas in i koden. Kvalitetsattribut, även kallade icke-funktionella krav, innefattar både de som är statiskt för-

kroppsligade i mjukvaruproduktens arkitektur (exempelvis modularitet och testbarhet), såväl som de som endast kan observeras under exekvering (exempelvis prestanda och säkerhet) (Wiegers & Beatty, 2013). De förstnämnda kan verifieras och valideras genom exempelvis referentgranskning av kod (K9). Den senare typen av kvalitetsattribut kan mätas och kvalitetssäkras genom exempelvis stress- och belastningstester i en förproduktionsmiljö, för att proaktivt kunna avgöra potentiella driftproblem innan de driftsätts i produktion (Waller et al., 2015).

K8 - Testdriven och beteendedriven utveckling samt refaktorisering

Testdriven utveckling (Test Driven Development, TDD) och beteendedriven utveckling (Behaviour Driven Development, BDD) är två agila mjukvaruutvecklingsmetoder som naturligt inkorporerar kontinuerlig testning (Erich et al., 2014). Vid testdriven utveckling beskriver utvecklaren en funktion som ska implementeras i form av ett testfall som kopplas till ett automatiserat test i koden, för att sedan utveckla precis så mycket kod som krävs för att passera testet och således verifiera funktionen. Koden refaktoriseras sedan för att minimera den tekniska skulden. Beteendedriven utveckling fungerar på ungefär samma vis, dock är en viktig skillnad att funktionsbeskrivningen skrivs i form av ett (för funktionen) önskat beteende, på ett naturligt eller domänspecifikt talspråk som alla intressenter kan förstå (Erich et al., 2014). Dessa typer av mjukvaruutvecklingsmetoder medför att både kvalitets- och funktionalitetskriterier kan testas i ett tidigare skede (Solis & Wang, 2011).

K9 - Kvalitetssäkring av källkod genom kontinuerlig referentgranskning

Agil mjukvaruutveckling och DevOps bygger på korta iterationer, små inkrement och frekventa releaser. Detta i sig reducerar per automatik defekters livslängd, eftersom att mjukvarulivscykeln och därmed tiden mellan injektion, upptäckt och borttagning av defekter är så pass kort. För att däremot förhindra att defekter uppstår från första början är det praxis att koden referentgranskas innan den checkas in i leveranspipelinen (Davis, 2013). Referentgranskning av kod ämnar upptäcka förbisedda programmeringsmisstag och därmed förbättra kodkvaliteten. Sådan granskning kan genomföras i form av exempelvis informella kodgenomgångar, formella inspektioner eller parprogrammering (Huizinga & Kolawa, 2007). Parprogrammering är en aktivitet som härstammar från XP och som går ut på att två utvecklare arbetar tillsammans vid en och samma dator. Den ena utvecklaren agerar förare och programmerar, samtidigt som den andra utvecklaren agerar navigatör och kontinuerligt referentgranskar koden, identifierar logiska och taktiska fel, tar hänsyn till den övergripande arkitekturen, brainstormar och föreslår bättre lösningar. Parprogrammering kommer med produktkvalitetsmässiga fördelar som färre defekter, bättre arkitektonisk design och bättre kodstruktur, såväl som ökad produktivitet, kunskapsdelning och kommunikation inom teamet (Di Bella et al., 2013). Oavsett om referentgranskningen sker i form av parprogrammering, kodgenomgång eller inspektion kan den med fördel baseras på fördefinierade kvalitetschecklistor, då det är mest effektfullt (Fitzgerald & Stol, 2015).

K10 - Infrastruktur som kod för automation av infrastruktur

Infrastruktur som kod (Infrastructure as Code, IaC) är ett sätt att hantera konfiguration av IT-system som om de vore mjukvarukomponenter. Det bygger på idén om att merparten av alla aktiviteter kring ett systems infrastruktur går att automatisera (Lwakatare et al., 2015; Spinellis, 2012). Genom att använda konfigurationshanteringsverktyg som styrs av automationsregler skrivna i form av kodscript kan driftsättning, konfiguration och uppgradering av infrastruktur hanteras och automatiseras på ett effektivt och konsekvent sätt. Genom att skriva konfigurationsregler som kodscript kan samarbetet mellan utvecklare och systemadministratörer förstärkas i och med att båda parter kan förstå dess innebörd (Spinellis, 2012). I en molnbaserad miljö medför detta att infrastruktur snabbt kan konfigureras och att mjukvarufunktionalitet kan driftsättas upprepningsbart och snabbt (Lwakatare et al., 2015). På så vis erbjuds användare en mer konsekvent upplevelse i och med att nya funktioner och förbättringar görs tillgängliga snabbare och mer frekvent (Schaefer, Reichenbach & Fey, 2013).

K11 - Beteendedriven systemadministration för gemensam förståelse för infrastruktur

Gohil, Alapati och Joglekar (2011) beskriver att effekterna av infrastruktur som kod ytterligare kan accelereras genom så kallad *beteendedriven systemadministration* (Behavior Driven Operations, BDOps). Det är ett koncept som bygger på principerna från beteendedriven utveckling (Gohil et al., 2011). Beteendedriven utveckling är emellertid en metod som enbart appliceras över själva utvecklingsprocessen i mjukvarulivscykeln (Solis & Wang, 2011). Beteendedriven systemadministration applicerar liknande principer på den systemoperativa delen av mjukvarulivscykeln. Genom att med scenariobeskrivningar redogöra för hur infrastruktur bör bete sig, på ett naturligt talspråk eller ett allmänt utbrett domänspecifikt språk, förs affärsintressenter, utvecklare, testare och systemadministratörer närmare samman i och med att alla kan förstå hur infrastrukturen sätts upp. Detta reducerar gap mellan krav och slutprodukt genom att minimera antalet överlämningar, kommunikationsmissar och misstolkningar mellan de inblandade (Gohil et al., 2011). De olika parter som omger kvalitetssäkring kan således arbeta tillsammans på att specificera hur systemet ska fungera och leverera den användarupplevelsen som önskas (Floris, Amrit & Daneva, 2014).

3.3 Mätning, övervakning och återkoppling

Detta delkapitel redovisar de kvalitetssäkringskapabiliteter som berör mätning av process- och produktkvalitet, övervakning och datainsamling, samt kontinuerlig återkoppling.

K12 - Kontinuerlig mätning och optimering av utvecklings- och leveransprocessen

Enligt Roche (2013) betonar DevOps vikten av att sätta perspektiv på effektivitet och processtänk, vilket ställer krav på mätning och optimering av utvecklings- och leveransprocessens prestanda genom att inkorporera metrik som kan bidra till att öka produktiviteten. I och med att DevOps bygger på gemensamt ansvarstagande och tvärfunktionellt samarbete krävs det nya sätt att mäta prestationer och kvalitet (DeGrandis, 2011). Traditionellt sett mäts utveckl-

ingspersonal individuellt baserat på antal levererade funktioner inom ett visst tidsspann, vilket uppmuntrar till en högre leveranstakt och ständiga förändringar. Samtidigt mäts systemadministratörers enskilda prestationer baserat på deras förmåga att vidhålla stabilitet i systemet, vilket uppmuntrar en lägre leveranstakt. Denna motsägelsefulla paradox leder många gånger till friktion mellan utvecklare och systemadministratörer (Erich et al., 2014). Detta påpekas även av Le-Quoc (2011) som skriver att införande av DevOps bör inkludera någon form av organisationsövergripande metrikdrivet ramverk som kan användas för att definiera delade och angripbara mätvärden. Detta knyter an till och lägger grund för kapabilitet K3 som berör delande av mål och incitament, vilken går ut på att motivera en kollektiv insats mot förbättrad process- och produktkvalitet.

K13 - Kontinuerlig övervakning och återkoppling av användarbeteende och realtidsprestanda

En viktig kapabilitet inom DevOps är kontinuerlig övervakning och återkoppling av realtidsprestanda och användarmönster i produktionsmiljön, något som traditionellt kanske inte tas i beaktning inom mjukvaruutveckling (Smeds et al., 2015; Liu et al., 2014). Återkoppling i denna kontext innebär omfattande insamling av data kring produktens infrastrukturprestanda, samt data kring hur och när användarna interagerar med produkten. Detta skapar återkopplingsslingor till utvecklingsprocessen, vars information kan användas till att planera och implementera förbättring och optimering av produkten. Detta möjliggör experimenterande och innovation i mjukvaruutvecklingen (Smeds et al., 2015). Claps et al. (2015) beskriver detta som en möjlighet att ta kvalitetssäkring till en ny nivå genom att med hjälp av prestanda- och användardata upptäcka nya insikter kring produktens kvalitet och användbarhet. Enligt Liu et al. (2014) är kontinuerlig övervakning och optimering nyckeln till framgång för att kunna uppnå ett långsiktigt åtagande vad gäller snabbt gensvar på ständigt föränderliga krav från användare. Genom kontinuerlig analys av prestanda- och användardata kan infrastrukturen och produkten kontinuerligt förädlas, vilket leder till en allt mer engagerande användarupplevelse (Liu et al., 2014).

K14 - Datadriven genomslags- och riskanalys baserad på återkoppling och användarscenarion

DevOps förespråkar som nämnt kontinuerlig analys av användarbeteende. Genom att samla in data och skapa återkopplingsslingor från produktion till planerings- och utvecklingsfasen kan en ny dimension av proaktiv kvalitetssäkring uppnås genom analys av typiska användarscenarion. Genom kartläggning och prioritering av de användarfall som en användare kan och sannolikt kommer att stöta på kan produktens motsvarande funktionalitetsområde bedömas efter hur kritiskt det är. Funktionalitetsområden som berörs av typiska användarfall bör således behandlas som högriskområden, inom vilka eventuella problem sannolikt kommer att ha stor inverkan på den slutliga användarupplevelsen. Den fundamentala skillnaden mellan traditionell riskanalys och en DevOps-präglad sådan är således tillgången till data. Återkoppling av användardata är en grundläggande kapabilitet som bör ligga till grund för proaktiv kvalitetssäkring i form av genomslags- och riskanalys (Roche, 2013).

K15 - Gradvis utrullning av releaser och mätning av genomslagskraft med A/B-testning

Gradvis utrullning av en release innebär att endast en delmängd av produktionsmiljöerna (det vill säga några få användare) får tillgång till den släppta funktionen. Denna övervakas och mäts sedan noggrant för att utvärdera om funktionen besitter den kvalitet och medför det genomslag som önskas. Detta återupprepas sedan gradvis tills alla produktionsmiljöer har tillgång till den nya funktionen (Adams & McIntosh, 2016). Skulle något kritiskt problem uppmärksammas så kan utrullningen reverseras, vilket innebär att produkten kvalitetssäkras genom tidiga indikationer till kvalitetsbrister. En närliggande testningsmetodik som bygger på gradvis utrullning kallas A/B-testning, där olika alternativ för ny funktionalitet rullas ut till olika produktionsmiljöer, men fortfarande under små kontrollerade former. Utifrån mätning och övervakning av användarbeteende kan dessa alternativ sedan utvärderas baserat på genomslagskraft, för att sedan fortsätta utrullningen med det mest framgångsrika alternativet (Adams & McIntosh, 2016). Detta ligger som grund till en datadriven kvalitetssäkringsprocess snarare än åsiktsdriven, vilket är en central kapabilitet inom DevOps (Roche, 2013).

K16 - Mörklagd produktlansering för proaktiv problemidentifikation

Generellt sett produktionsdriftsätts endast sådana funktioner som är färdigställda och redo för att göras tillgängliga för produktens slutanvändare. Driftsättning efterföljs alltså typiskt av omedelbar release. Vid så kallad *mörklagd produktlansering* driftsätts däremot även ofärdiga funktioner, men görs osynliga eller dolda för slutanvändaren för att kunna utföra olika typer av kvalitetssäkring direkt i produktionsmiljön. Då funktionen är driftsatt i faktisk produktionsmiljö kan till exempel integrations-, prestanda- och skalbarhetstester utföras genom att applikationen skickar olika tillrop till den nya driftsatta funktionen, utan synlighet för användaren (Adams & McIntosh, 2016). Mörklagd produktlansering är med andra ord ett proaktivt tillvägagångssätt för att testa och kvalitetssäkra ny funktionalitet i en skarp miljö, utan att användarupplevelsen drabbas.

3.4 Undersökningsmodell

De i det teoretiska ramverket redovisade kapabiliteter som indikerar hur kvalitetssäkring sker i en DevOps-organisation sammanfattas nedan i en undersökningsmodell. Denna agerar vidare som underlag för insamling av empirisk data.

Tabell 3.1: Det teoretiska ramverkets undersökningsmodell

Kategori	*Kapabilitet*	
Samarbetspräglad organisationsstruktur och kultur	K1	Delande av arbetssätt, ansvar och kollektivt ägarskap för kvalitetssäkring
	K2	QA har en teknisk profil och arbetar nära utvecklare och systemadministratörer
	K3	Gemensamma mål och incitament för kvalitetssäkring
	K4	Kontinuerligt experimenterande, lärande och delande av kunskap
	K5	QA har inflytande över designbeslut
	K6	QA har inflytande över releasebeslut
Kontinuitet och automation	K7	Kontinuerlig och automatiserad testning av funktionalitet och kvalitet
	K8	Testdriven och beteendedriven utveckling samt refaktorisering
	K9	Kvalitetssäkring av källkod genom kontinuerlig referentgranskning
	K10	Infrastruktur som kod för automation av infrastruktur
	K11	Beteendedriven systemadministration för gemensam förståelse för infrastruktur
Mätning, övervakning och återkoppling	K12	Kontinuerlig mätning och optimering av utvecklings- och leveransprocessen
	K13	Kontinuerlig övervakning och återkoppling av användarbeteende och realtidsprestanda
	K14	Datadriven genomslags- och riskanalys baserad på återkoppling och användarscenarion
	K15	Gradvis utrullning av releaser och genomslagsanalys genom A/B-testning
	K16	Mörklagd produktlansering för proaktiv problemidentifikation

4 Metod

I följande kapitel redogörs för uppsatsens metodval, vilket utförs som en kvalitativ studie med fokus på djup och nyanserad empiri insamlad genom öppna intervjuer. Intervjuernas syfte är att samla in empirisk data kring hur kvalitetssäkring bedrivs i mjukvaruutvecklande organisationer som tillämpar DevOps. Intervjuerna utgår ifrån det teoretiska ramverket och dess tillhörande undersökningsmodell och följer de kategorier och kapabiliteter som presenterats.

4.1 Behov för empirisk data

De empiriska data kring organisationsstruktur och kultur som måste samlas in bidrar till en mer verklighetsförankrad inblick i hur exempelvis kollektivt ägarskap, delande av arbetssätt, ansvar och mål tillämpas i praktiken och hur det påverkar kvalitetssäkringsaktiviteter. Vidare behövs empirisk data kring hur tekniska verktygskedjor och processer används för att få en djupare förståelse för hur kontinuitet och automation tillämpas och stödjer arbetet med kvalitetssäkring. Slutligen behöver vi samla in data kring hur mätning, övervakning och återkoppling sker i verkligheten och understödjer kvalitetssäkringsrelaterade aktiviteter.

4.2 Metodval

Då vårt problemområde utgår från en explorativ ansats väljer vi att utföra en kvalitativ och intensiv studie, vilket ger en djup och nyanserad helhetsbild av en specifik situation (Jacobsen, 2002). Vi vill få förståelse mellan undersökningsenheten (kvalitetssäkring) och kontexten genom återgivelser av personliga upplevelser, då individuella tolkningar av kontexten sannolikt varierar i detta sammanhang. Detta sker i praktiken lämpligt genom intervjuer med en öppen ansats och med ett fåtal relevanta intervjupersoner. En intervjuguide med omfattande och breda frågor utformas och används som grund för att genomföra intervjuerna, vilket ger oss möjligheten att ställa mer specifika följdfrågor, vilket är normativt i en kvalitativ forskningsstudie (Jacobsen, 2002). Vidare lämpar sig en kvalitativ undersökning i detta fall bättre än en kvantitativ sådan, i och med att fenomenet DevOps är relativt outforskat och vi vill få möjlighet till ökad klarhet i problemområdet (Jacobsen, 2002).

4.3 Urval av enheter

För att hitta lämpliga undersökningsenheter som befinner sig inom ramen för vårt problemområde och således kan bidra till att svara på vår forskningsfråga, sökte vi kontakt med mjukvaruutvecklande organisationer som utifrån vår definition ger indikationer på tillämpning av DevOps. I och med att dessa enheter anses befinna sig inom ramen för uppsatsens kontext är det sannolikt att de besitter relevant kunskap och erfarenheter, och kan återge dessa i form av personliga upplevelser. För att få insikt i organisationernas arbete kring kvalitetssäkring valde vi att efterfråga intervjupersoner med arbetsroller som både arbetar med kvalitetssäkring på djupet, till exempel QA Engineer, såväl som de som har ett mer övergripande perspektiv på hur kvalitetssäkring förhåller sig till organisationen som helhet, exempelvis QA Manager. Efter korrespondens med olika organisationer etablerades kontakt med företagen Spotify och Klarna, vilka båda är baserade i Stockholm.

Spotify verkar inom musiktjänstebranschen och är transparenta med sitt tillämpande av DevOps, vilket indikeras genom både externa publikationer (Hedström, 2014) och en serie av företaget publicerade videoklipp som beskriver dess "Engineering Culture" (Spotify, 2014). Klarna verkar inom branschen för utveckling av betaltjänster och har enligt oss en DevOps-präglad organisation. Det indikeras bland annat av att företaget har positioner som Continuous Delivery Engineer och Software Engineer in Test, där den sistnämnda innefattar arbetsuppgifter som exempelvis testautomation, rådgivning kring designbeslut samt ambassadörskap för kvalitet genom hela ingenjörsorganisationen (Klarna, 2016). Att Klarna tillämpar DevOps indikeras ytterligare av det faktum att de under en tid tagit hjälp av konsulter från företaget Diabol med att implementera en plattform för kontinuerlig leverans och automation av releaseprocessen (Diabol, 2016). Totalt fyra personer från de båda företagen med olika QA-relaterade arbetsroller användes som subjekt för denna studie. Dessa presenteras i tabell 4.1.

Tabell 4.1: Intervjupersoner för insamling av empiri

ID	*Arbetsroll*	*Företag*	*År på företaget*
P1	Team Lead Quality Assistance	Spotify	5
P2	Quality Assistance Engineer	Spotify	3
P3	Team Lead System Test	Klarna	5
P4	Software Engineer in Test	Klarna	2 ½

4.4 Genomförande av intervjuer

Intervjuerna valdes att genomföras över kommunikationsplattformarna Google Hangouts och Skype, då personliga besök skulle vara både tids- och kostnadskrävande. Intervjuerna valdes att hållas relativt öppna genom att utgå från tematiska ämnesområden, samt genom att ställa

intervjufrågor utifrån en halvt fastställd ordningsföljd. Detta möjliggör att de mest relevanta områdena för eftersökt empiri lyfts upp för att få en mer nyanserad tolkning av ett ämne, men utesluter inte intervjupersonernas möjlighet till breda och öppna svar. Intervjuguiden utformades utifrån undersökningsmodellen presenterad i delkapitel 3.4 för att få insikt i intervjupersonernas personliga upplevelser och tolkningar kring arbetet med kvalitetssäkring.

4.4.1 Intervjuguide

Samtidigt som vi ville hålla intervjuerna så öppna som möjligt konstruerades en guide för intervjufrågor som användes som tematisk grund inför intervjuerna vilket enligt Jacobsen (2002) benämns som en semistrukturerad intervju. Detta för att inte förbise eller utesluta viktiga aspekter som vi fann relevanta för kvalitetssäkringsarbetet på organisationerna, vilket ligger som grund till uppsatsens frågeställning. Guiden utgår från fyra tematiska områden där tre av dessa utgår från kategoriseringen av DevOps grunddrag som grundlades i kapitel 2.2.4, och som vidare användes för att beskriva det teoretiska ramverket i kapitel 3. Det fjärde och även inledande tematiska intervjuområdet utformades i syfte att ge en introducerande bild av intervjupersonerna samt att etablera en form av personlig relation. Då inte alla intervjupersoner talar svenska utformades två versioner av intervjuguiden; en på engelska och en på svenska. Intervjuguiderna i sin helhet återfinns i bilaga B1 och B2. Nedan följer en kort beskrivning av de tematiska intervjuområdena och en förklaring av varför de är relevanta för studien.

Bakgrund

Detta tematiska område agerar som en introduktion till intervjupersonernas arbetsroll för att få en grundläggande förståelse för vilka typer av arbetsuppgifter vederbörande utför. Det fyller även syftet att etablera en personlig relation till intervjupersonen.

Samarbetspräglad organisationsstruktur och kultur

Kvalitetssäkring genom en rad olika kapabiliteter baserade på en samarbetspräglad organisationsstruktur och kultur har identifierats och ligger därmed som grund inför dessa tematiska intervjufrågor. De ämnar ge en inblick i hur organisationen som helhet möjliggör eller begränsar de anställda till att dela ansvaret för kvalitetssäkring; graden av kollektivt ägarskap.

Kontinuitet och automation

Frågorna inom detta område avser att ge en inblick i organisationernas arbete angående kontinuitet och automation, till exempel vilka tekniska verktyg och metoder de använder för utveckling, kvalitetssäkring, och leverans, samt hur mjukvarulivscykeln är utformad.

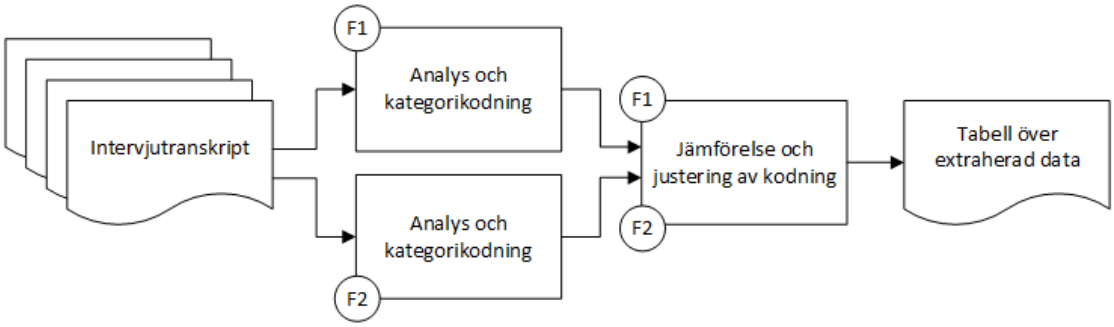
Mätning, övervakning och återkoppling

Dessa frågor är utformade för att ge en inblick i hur företaget arbetar med kvalitetssäkring utifrån mätning, övervakning och återkoppling; därmed även till vilken grad de utnyttjar data för att optimera både produktens och leveransprocessens kvalitet.

4.5 Analys och presentation av data

Vid utförande av kvalitativa intervjuer rekommenderar Jacobsen (2002) att reducera komplexiteten av insamlad data genom att systematiskt bearbeta och kategorisera den. Även då en djupintervju ämnar till att samla in detaljrik data måste det vara möjligt att få en överblick och kunna kombinera sammanhang för att kunna göra någon form av generalisering. Då intervjuerna renderar rådata i form av ljudinspelningar transkriberar vi dessa till text för att enklare kunna bearbeta den. Att kunna kommentera och stryka under särskilda passager i en transkriberad intervju menar Jacobsen (2002) effektiviserar arbetet med dataanalys. Dessa transkriberingsprotokoll tilldelas ett nummer, och återfinns i kronologisk utförd följd i uppsatsens bilaga. Transkriberingsprotokollen har även tilldelats radnumrering för att med enkelhet kunna hänvisa till insamlad data i studiens resultat.

Vidare så delas insamlad data in i kategorier för att kunna likställa ämnen som berör varandra. Detta reducerar både komplexiteten för enskilda datainsamlingar (enskilda intervjuer), samt för jämförbarheten datainsamlingar emellan. Kategorierna bör vara relevanta för de data vi samlar in (Jacobsen, 2002). Vi har därför valt att hänföra kategorierna till intervjuguidens tematiska områden: *Samarbetspräglad organisationsstruktur och kultur; Kontinuitet och automation;* samt *Mätning, övervakning och återkoppling*. Det tematiska intervjuområdet *Bakgrund* har medvetet exkluderats från att kategoriseras i transkriberingsprotokollen för att undvika förvirring.



Figur 4.1: Arbetsflöde för analys och kategorikodning av empirisk data

Både för läsbarhetens och spårbarhetens skull har vi under analysarbetet kodat transkriberingsprotokollen utifrån ovan nämnda kategorier, samt applicerat ett kategori-ID. Varje intervjutranskript analyserades och kodades individuellt av uppsatsens författare, därefter jämfördes och justerades kodningen för att uppnå konsensus i kategoriseringen av data, slutligen skapades en tabell som visar från vilken respondent och vilken rad relevant data extraherats. Figur 4.1 visar arbetsflödet för analys- och kodningsarbetet, där (F) indikerar författare.

Exempelvis är data inom kategorin Kontinuitet och automation markerad med blå färgkod och har ID-taggen (K). På samma sätt är data inom kategorin Samarbetspräglad kultur och organisationsstruktur markerad med grön färgkod och har ID-taggen (S). Exempel på kategorisering och kodning av data från intervjun med P2 presenteras nedan i tabell 4.2.

Tabell 4.2: Exempel på kategorisering och kodning av intervjudata

Rad	*Frågor och svar*	*Kategori*
35	**Vilka är det som bygger dessa tester?**	
36	Det är de som utvecklar funktionen. Det är också en del av code reviewen som blir öppen för hela teamet. I en sådan code review faller det sig naturligt att individer med olika specialiseringar tittar på olika saker, beroende på omfattning.	S, K

4.6 Undersökningskvalitet

Detta delkapitel presenterar åtgärder och avvägningar som vidtagits för att styrka studiens trovärdighet.

4.6.1 Etik

En tydlig dialog angående frivilligt deltagande mellan intervjupersonerna har förts, vilket har givit dem möjligheten att även avböja medverkan i studien. Intervjupersonerna tillfrågades inledningsvis under intervjutillfällena om samtycke till inspelning av samtalet, samt informerades om studiens fullständiga syfte, vilket är ett grundförhållande för en etisk utförd undersökning (Jacobsen, 2002). Vidare anonymiseras intervjupersonernas namn i presentationen av data för att inte kunna identifieras specifikt utifrån insamlad data, vilket ger intervjupersonerna rätt till privatliv, vilket är ytterligare ett etiskt grundförhållande. Slutligen, då intervjupersonerna har rätt till fullständig återgivning av information presenteras även det empiriskt insamlade materialet i ett så brett kontext som möjligt för att ge en så rättvis tolkning som möjligt. Alla intervjupersoner kommer även ha tillgång till den slutgiltiga presentationen av data vid studiens färdigställande för att kunna validera att våra tolkningar är korrekt återgivna.

4.6.2 Studiens reliabilitet

En studies påvisade resultat kan påverkas av en rad olika faktorer, vilket innebär att tillförlitligheten för utförandet av studien bör granskas (Jacobsen, 2002). Intervjupersonerna kan till exempel ge varierande information beroende på kontext eller intervjuarens uppsyn och beteende, även benämnt som kontexteffekten samt intervjuareffekten. För att resultaten skulle bli likvärdiga utgick intervjuerna utifrån likvärdiga förutsättningar; tid och plats var för intervjupersonerna känt sedan innan, vilket gör att intervjun inte blir ett överraskande moment. Vidare utfördes även intervjuerna på intervjupersonernas egna kontor över videolänk, vilket är en naturlig miljö där ovanlig stimuli inte bör förekomma. Underlag inför intervjuerna skickades även i förväg till intervjupersonerna för att reducera eventuell stress. För att reducera eventuell påverkan eller influens av intervjupersonens svar var inga andra personer än vederbörande närvarande under intervjuerna.

Vidare kan otillfredsställande registrering av data påverka en studies tillförlitlighet. Då vi ville försäkra oss om att denna registrering skulle vara av tillfredsställande kvalitet åtogs en rad åtgärder för att registreringen skulle ske standardiserat. Då intervjuerna utfördes över en kommunikationsplattform beroende av en nätverksuppkoppling var det av högsta prioritet att vi använde oss av en stabil uppkoppling med minimal risk för nätverksstörningar. Vidare för att kunna analysera och bearbeta våra insamlade data utan risk för att förlora någon information spelades intervjusamtalen in med två separata mobiltelefonsapplikationer för diktering. Mobiltelefonerna försattes i ljud- och vibrationslöst läge för att inte störa inspelningarna.

4.6.3 Studiens validitet

Studiens interna giltighet, det vill säga till vilken grad studiens resultat är trovärdiga, valideras inledningsvis utifrån en kritisk granskning av våra källor av information (Jacobsen, 2002). Vi anser att tillgången till våra önskade enheter som vi ansåg innefattade riktig information var god, då båda organisationerna som representerades i studien är framgångsrika och väletablerade inom sina respektive branscher. Enheterna tillfrågas om information kring förstahandsupplevelser och egna tolkningar, vilket ger en närhet och större tilltro till det fenomen vi avser att belysa (Jacobsen, 2002). Vidare har enheterna extensiv erfarenhet och kunskap av fenomenet, då majoriteten har arbetat inom området kvalitetssäkring på olika företag och projekt. Då intervjupersonerna har olika bakgrund och inte är beroende av varandra kan informationen även tillskrivas en högre grad av giltighet, än om de hade drivits av gemensamma motiv (Jacobsen, 2002).

5 Resultat

I detta kapitel presenteras studiens resultat i form av insamlad empirisk data genererad utifrån fyra genomförda djupintervjuer. Två mjukvaruutvecklande organisationer och fyra intervjupersoner användes till framställningen av empiri.

5.1 Presentation av empiriska resultat

Detta delkapitel innehåller tre sektioner som motsvarar det teoretiska ramverkets kategorier tillika tematiska ämnesområden som användes vid insamlings- och analysarbetet. Varje sektion inleds med en översikt av empirisk data i tabellform i förhållande till undersökningsmodellen. Inom varje sektion följer därefter en detaljerad löptextbeskrivning av resultaten.

5.1.1 Samarbetspräglad organisationsstruktur och kultur

Denna sektion redovisar empiriska resultat kopplade till samarbetspräglad organisationsstruktur och kultur. Tabell 5.1 visar från vilken intervjuperson och vilken rad i transkriberingsprotokollen resultaten är extraherade.

Tabell 5.1: Översikt av data inom samarbetspräglad organisationsstruktur och kultur

Kategori	*Kapabilitet*		*P1 (Rad)*	*P2 (Rad)*	*P3 (Rad)*	*P4 (Rad)*
Samarbetspräglad organisationsstruktur och kultur	K1	Delande av arbetssätt, ansvar och kollektivt ägarskap för kvalitetssäkring	8, 10, 12, 38, 40, 54	6, 12, 18, 20, 22, 24, 36, 56, 74, 76	2, 6, 8, 12, 14, 16	2, 12, 16, 18, 48
	K2	QA har en teknisk profil och arbetar nära utvecklare och systemadministratörer	8, 10, 12, 28, 36, 40	6, 14, 16, 22, 24, 32, 36	6, 8, 12, 22, 32	16, 24, 30
	K3	Gemensamma mål och incitament för kvalitetssäkring	16	68	18, 20	58, 60, 62
	K4	Kontinuerligt experimenterande, lärande och delande av kunskap	18, 38, 48	26, 28, 30, 38, 62, 66	8, 38, 42, 44	64, 66, 68, 72
	K5	QA har inflytande över designbeslut	24, 28	22, 24, 32	22, 30	
	K6	QA har inflytande över releasebeslut	30, 32	56, 78	24, 26, 34, 38	30, 44, 46, 50

Spotify organiserar produktutvecklingen som en matrisorganisation, där varje vertikalt team är ansvarigt för en viss produkt eller tjänst (P1: 8). Dessa team är multidisciplinärt sammansatta och utgörs av roller med olika inriktningar och discipliner (P2: 6, 22, 36). I varje team finns typiskt en produktägare, en agil coach, en QA-ingenjör, en varierande uppsättning av utvecklare, samt i vissa fall en UX-designer (P1: 14). Teammedlemmarna sitter fysiskt tillsammans och samarbetar integrerat och tvärfunktionellt (P1: 28).

Klarna har sedan ett år tillbaka en matrisorganisationsstruktur (P3: 12). Dessförinnan fanns en centraliserad QA-avdelning som efter behov placerade ut testare och QA-personal i de olika utvecklingsteamen (P4: 2). Numera är emellertid alla team tvärfunktionellt sammansatta av personer med olika inriktningar, där ett typiskt produktutvecklingsteam består av en produktägare, en team lead och en uppsättning utvecklare med olika specialiseringar (P3: 8), varav en specialisering är Software Engineer in Test (P3: 22). Varje team ingår i en viss domän som arbetar med ett övergripande produktområde, och på domännivå finns även en ingenjörschef, en arkitekt och en produktchef (P3: 6, 8).

Spotify lägger stor vikt vid kollektivt ägarskap och delande av ansvar. P1 beskriver kvalitetssäkring som en laginsats och understryker att varje team har helhetsansvar för dess producerade kvalitet: *"kvalitet är en holistisk sak som måste ägas av alla i teamet"* (P1: 12). P2 menar att kvalitet varken kan ägas av en enskild individ eller ett team (P2: 12) och påpekar att det istället är viktigt att definiera och fokusera på det värde teamet levererar för organisationen som helhet (P2: 6, 20, 56). För att kunna leverera värde krävs det att hela produktorganisationen i form av beställare, produktägare, businessägare, utvecklare och QA-ingenjörer spenderar tid tillsammans och har en grundläggande förståelse för varandras angelägenheter (P2: 6, 74, 76, 78). Kvalitet ägs kollektivt och är något som ingenjörsorganisationen gemensamt bygger upp (P2: 6). Det finns emellertid dedikerade QA-ingenjörer på Spotify som, förvisso inte ansvarar för men, fokuserar särskilt på kvalitetsaspekter (P1: 10, 12; P2: 24). De har inriktningen "Quality Assistance" istället för "Quality Assurance" (P1: 10; P2: 14) och agerar kvalitetscoacher inom teamen (P1: 10, 12; P2: 32). QA-ingenjörer har stora tekniska krav på sig då de utöver coachning till stor del arbetar med utveckling av testverktyg och testautomation. De behöver kunna gräva i, och därmed ha förståelse för, teamets gemensamma kodbas och system (P1: 12; P2: 16). Enhets- och integrationstester skrivs i regel av utvecklarna i samband med att de utvecklar funktioner, medan QA-ingenjörerna fokuserar mer på coachning, helhetsöversikt, systemtestning och regressionstestning inför release (P1: 12, 36, 40; P2: 36).

Teamen på Klarna har autonomt ansvar för den eller de tjänster de äger, vilket innebär att de ansvarar för produktens hela livscykel, från utveckling och testning till driftsättning, underhåll och on-call-support, såväl som konstruktion av deras egna verktyg och leveranspipeline (P3: 8, 14; P4: 2, 12). Varje team fattar sina egna beslut och står sina egna kast, de har emellertid krav på sig att alltid leverera vältestade produkter med hög kvalitet (P3: 14). Klarna har utöver utvecklingsteamen också en rad supporterande team, varav ett är System Test (P4: 16). I detta team ingår både intervjuperson P3 och P4 (P3: 2; P4: 2). Deras uppgift är att utföra end-

to-end-tester på systemnivå som sträcker sig över samtliga tjänster och produkter som är sammankopplade (P3: 2; P4: 2). Ibland är dessa tester inkorporerade i de andra teamens leveranspipelines, då sannolikt sist i exekveringsordningen (P4: 30). System Test stödjer produktutvecklingsteamen med bland annat testramverk, verktyg, strukturer, metoder och rådgivning för att gemensamt åstadkomma överlappande testtäckning, både på komponentnivå såväl som på systemnivå (P3: 2; P4: 18, 48). P4 berättar att alla team samarbetar och att mycket tid spenderas på kommunikation för att skapa en systemövergripande förståelse för hur alla tjänster är ihopkopplade (P4: 16). P3 understryker att även arkitekter, produktägare, produktchefer, team leads och utvecklare samarbetar ofta och nära (P3: 16). I stort sett all testning på Klarna är automatiserad (P3: 32; P4: 24) och de allra flesta testnings- och kvalitetssäkringsaktiviteter utförs i utvecklingsteamen (P4: 48). Personer som arbetar med kvalitetssäkring besitter således en teknisk profil eftersom att de praktiskt taget arbetar med mjukvaruutveckling (P4: 30).

Spotify har inga explicit definierade mål eller incitament vad gäller just kvalitetssäkring, då det upplevs som svårt att mäta kvalitet. De anställda drivs istället av möjligheten att kunna skapa nya saker som används och uppskattas av användarna (P1: 16) och de har stor frihet att belöna sig själva när de har åstadkommit något bra (P2: 68). Innovation och experimenterande uppmuntras till hög grad och företagskulturen genomsyras av att misslyckanden är helt i sin ordning, så länge man kan dra lärdomar från dem (P1: 18; P2: 26, 30). P1 beskriver att den typen av kulturell transparens är viktig för att konkurrensmässigt ligga i framkant och att den kontinuerligt eftersträvas, men att den blir allt mer svår att upprätthålla i takt med att företaget växer (P1: 18). De anställda delar kontinuerligt kunskap om arbetsmetoder och verktyg som visat sig fungera bra, men teamen bestämmer själva vilka verktyg och metoder som fungerar bäst för dem (P1: 38; P2: 38, 66). P1s team har exempelvis retrospektiva möten där de diskuterar vad som gått bra, vad som gått dåligt, och vad som går att göra bättre (P2: 62). P1 berättar att vissa team utövar parprogrammering och att det är ett bra sätt att dela kunskap och skapa förtroende för varandra: *“don’t fear to fail”* (P1: 48). Det ser emellertid olika ut i de olika teamen (P2: 66).

På Klarna har alla teammedlemmar personliga mål som utformas och uppdateras kontinuerligt i samråd med sina team leads, och dessa går ofta hand i hand med teamets mål (P3: 18, 20; P4: 58, 60). P3 ger ett exempel på att ett kvalitetsrelaterat mål kan vara att halvera antalet tester utan att tappa testtäckning, men understryker samtidigt att sådana mål inte hjälper företaget så mycket och att det är svårt att kvantifiera kvalitetssäkring (P3: 20). P4 beskriver att de anställda istället motiveras av positiv återkoppling och nya utmaningar och ansvarsområden (P4: 62). Både P3 och P4 anser att Klarna har en kultur som stödjer innovation och experimenterande (P3: 42; P4: 66). Team spenderar ofta tid på att utforska och prova nya ramverk och verktyg som de tror kan gynna företaget, och det är upp till teamen att bestämma vilka tekniker och metoder de vill använda (P3: 42; P4: 64, 68). P4 upplever att Klarna stödjer misslyckanden och att inget finger pekas om något går fel (P4: 64). Både P3 och P4 uppger att det däremot är viktigt att lära av misstagen och se till att det inte händer igen (P3: 44; P4: 64). P3 berättar att hans team arbetar mycket med att dela kunskap och försöka få övriga team att

förstå vad de gör, hur och varför de testar vissa saker. Det uppmuntras till att andra team tittar på systemtestverktygen och kommer med kritik och förslag till förbättringar (P3: 38).

Spotify har en decentraliserad systemdesignprocess och alla tekniska designbeslut fattas lokalt inom teamen (P1: 28). Både P1 och P2 förklarar att samtliga individer i teamet har gemensamt ansvar för, och lika stort inflytande över, beslut kring design och arkitektur, men att det faller sig naturligt att QA mer aktivt söker upp och coachar diskussioner kring kvalitetsegenskaper som exempelvis testbarhet (P1: 28; P2: 22, 24). Det fungerar just eftersom att QA-ingenjörer och utvecklare sitter bredvid varandra, kommunicerar aktivt och för informella diskussioner, vilket enligt P2 är oerhört värdefullt (P1: 28; P2: 24). En data- och statistikavdelning förser teamen med återkoppling som ligger till grund för sessioner och samtal där problemområden diskuteras och kvalitetsförbättringar prioriteras (P2: 32).

P3 beskriver att på Klarna finns det ingen dedikerad roll som bara är ansvarig för kvalitet i samband med beslut kring design och arkitektur. Vissa team har en Software Engineer in Test, och de försöker bygga produkten på ett sätt som bidrar till god testbarhet. Samtidigt ligger det ansvaret lika mycket hos övriga utvecklare (P3: 22). När ett nytt initiativ startas möts domänens produktchef med utvecklingsteamet och pratar om vilka förväntningar som finns på produkten. I samband med detta brukar teamet komma med förslag på hur funktionen kan konstrueras och testas på bästa sätt. Därefter implementeras funktionen. Ifall funktionen sträcker sig över flera tjänster koordinerar System Test-teamet med de andra teamen som äger dessa tjänster, för att säkerställa att fullständig testtäckning uppnås (P3: 30).

Spotify har ingen process som säger att QA måste ge klartecken till en release. Teamet äger kollektivt produktens kvalitet och därför ägs även alla releasebeslut gemensamt av teamet (P1: 30). P1 beskriver att företaget undviker att låta QA-ingenjören agera portvakt inför en release, då det hämmar resten av teamet från att lära sig att bedöma huruvida något är redo för release eller inte (P1: 30). Vid meningsskiljaktigheter inom teamet är det i slutänden produktägaren som fattar releasebeslut, eftersom att denne besitter både ett affärsmässigt och tekniskt perspektiv (P1: 30). Däremot är QAs möjlighet att påverka releasebeslut mycket stor (P1: 30; P2: 56) och folk inser generellt att det är bra att ha QA i ryggen (P1: 30). Ifall det finns kända defekter i en funktion men man överväger att släppa den ändå på grund av affärsmässiga eller andra värdeskapande orsaker, görs om möjligt en datadriven analys av hur stort problem det är för den övergripande kvaliteten samt vilken inverkan defekten kommer ha på slutanvändarna (P1: 32, P2: 78). P2 beskriver det som ett pragmatiskt förhållningssätt där det gäller att alltid titta på det värde som skapas för organisationen som helhet (P2: 56, 78).

Även om System Test-teamet på Klarna arbetar med end-to-end-testning och står för de slutgiltiga testerna i exekveringsordningen (P4: 30) understryker P4 att de inte är portvakter inför en release (P4: 44). Däremot bidrar de ofta med rådgivning och rekommendationer då de besitter en helhetsförståelse för systemet och enkelt kan avgöra om en defekt är kritisk eller inte (P4: 44). De fokuserar på att stödja övriga team med information, kunskap och självförtroende när de står inför releasebeslut (P4: 44, 48). System Test har stort inflytande över releasebeslut (P4: 46), men det är emellertid teamet som äger produkten eller tjänsten som i slutänden äger

beslutet (P4: 44). Inför större teamöverskridande produktsläpp brukar System Test bjuda in intressenter och team som har en insats i releasen till ett "go or no-go"-möte (P3: 34; P4: 50) där de går igenom och diskuterar områden som eventuellt bör kvalitetssäkras närmare (P3: 34). Ifall någon uttrycker att det saknas testtäckning över ett visst område är det produktägare och team lead som avgör hur det bör prioriteras (P3: 24). P3 beskriver att det emellertid är orimligt att försöka täcka alla testscenarion (P3: 24). I de fall genomförs en datadriven riskbedömning av hur stort problem det är, baserat på trender som går att utläsa från användares typiska beteende (P3: 24, 26). P3 beskriver att det handlar om att ta informerade beslut baserade på de data som finns och understryker att ju mer det går att backa upp argument med data, desto enklare är det att ta beslut (P3: 38). Det gäller att vara transparent mot produktägarna och föra en konstruktiv dialog (P3: 38).

5.1.2 Kontinuitet och automation

Denna sektion redovisar empiriska resultat kopplade till kontinuitet och automation. Tabell 5.2 visar från vilken intervjuperson och vilken rad i transkriberingsprotokollen resultaten är extraherade.

Tabell 5.2: Översikt av data inom kontinuitet och automation

Kategori	*Kapabilitet*		*P1 (Rad)*	*P2 (Rad)*	*P3 (Rad)*	*P4 (Rad)*
Kontinuitet och automation	K7	Kontinuerlig och automatiserad testning av funktionalitet och kvalitet	34, 36, 38, 40, 42	32, 34, 40, 42	2, 32, 34, 50, 54, 60	12, 14, 18, 22, 24, 26, 30
	K8	Testdriven och beteendedriven utveckling samt refaktorisering	34, 44	38, 78	42	28
	K9	Kvalitetssäkring av källkod genom kontinuerlig referentgranskning	46, 48, 52	36, 40, 42	28, 46, 48, 50	28, 30, 32
	K10	Infrastruktur som kod för automaton av infrastruktur	54	46	8, 10, 52	34, 36
	K11	Beteendedriven systemadministration för gemensam förståelse för infrastruktur		48	56	38

Spotify automatiserar sina testaktiviteter så långt det går och P2 beskriver att de följer en triangelformation som består av allra flest enhetstester, något färre integrationstester, samt ett fåtal end-to-end-tester. De sistnämnda körs emellertid inte kontinuerligt eftersom att de ofta tar stopp i de fall där tredjepartsintegrationer är involverade (P2: 34). När ny kod checkas in passerar den en rad automatiserade enhetstester, integrationstester och vissa driftsättningstester (P1: 36). Enhetstester skrivs av utvecklarna i samband med att vissa ägnar sig åt med testdriven utveckling (P1: 34, 36) och beskrivs av P1 som ett bra sätt att kontinuerligt kontrollera att koden uppfyller sitt syfte, såväl som att det är fördelaktigt ur en dokumentationssynpunkt (P1: 34). Prestandatester i livemiljö körs och övervakas kontinuerligt under utvecklingens

gång, vars resultat återkopplas omedelbart för att proaktivt säkerställa att det som byggs kommer fungera i produktionsmiljö (P1: 42). Att kontrollera testtäckning och diskutera designprinciper är en del av den kodgenomgång som genomförs innan något checkas in (P1: 46; P2: 40). QAs roll i kontexten är att utveckla testverktyg, utföra kodgenomgångar, visa värdet av kvalitet och rådgiva exempelvis kring när och hur kontinuerligt tester bör exekveras (P1: 40, 46; P2: 16). Både P1 och P2 understryker att automation emellertid inte kan ersätta domänkunskap och mänsklig intuition och beskriver att QA ofta engagerar sig i gruppbaserad explorativ testning (P1: 36; P2: 32, 42).

I stort sett all testning på Klarna är automatiserad (P3: 32; P4: 24). Varje team ansvarar för att skriva och bygga in åtminstone tester på enhetsnivå i källkoden, såväl som att skapa testdata och exekvera testerna (P4: 12, 14). När en kodändring checkas in i leveranspipelinen startas ett jobb som kör enhetstesterna och rapporterar tillbaka resultatet (P3: 50, 54). Om alla tester passeras fortsätter leveranspipelinen med andra nivåer av testning och sannolikt är de sista testerna i exekveringsordningen de end-to-end-tester som System Test-teamet bygger (P4: 30). System Test ansvarar för testning på systemnivå och utvecklar och äger det automatiserade testramverk som används för att säkerställa att alla individuella tjänster fungerar tillsammans (P4: 12, 22). Det finns även ett dedikerat Performance Test-team som bedriver icke-funktionell testning, både inom de olika teamens leveranspipelines såväl som i dedikerade prestandatestmiljöer (P3: 60). Prestanda- och systemtester körs kontinuerligt och dess resultat rapporteras via dashboards som alla har tillgång till, således kan de olika teamen se hur deras respektive tjänster mår (P4: 18). Ifall framtagandet av en ny funktion kräver att flera team är inblandade koordinerar System Test samarbetet mellan teamen vad gäller att uppnå överlappande testtäckning (P3: 2; P4: 26). De organiserar emellanåt sessioner där flera team samlas och utför explorativ testning (P3: 34, P4: 26).

Spotify har inga uttalade krav på specifika utvecklingsmetoder (P2: 38). P1 beskriver förvisso att vissa utvecklingsteam tillämpar testdriven utveckling (P1: 34) och nämner även beteendedriven utveckling, men förklarar att företaget undviker påtvingande av dessa utvecklingsmetoder, då de kan medföra friktion och kräver stora omställningar i tankesätt (P1: 44). P2 anser att strikt följande av utvecklingsmetoder ger förhållandevis lite värde och att motsvarande synergieffekter uppstår naturligt om samma grupp sitter tillsammans en längre tid (P2: 38). Refaktorisering av teknisk skuld prioriteras enligt vilken trade-off man står inför. Om snabb leverans i ett visst fall gynnar Spotify som helhet, trots att den interna produktkvalitetsnivån drabbas, måste affärssidan acceptera att utvecklingsteamet ägnar sig åt refaktorisering en tid därefter. Det handlar om ett pragmatiskt förhållningssätt och transparenta diskussioner (P2: 78).

På Klarna har alla team frihet att välja sina egna metoder (P4: 28). Många team utövar testdriven utveckling, parprogrammering och mobbprogrammering, men det varierar mellan de olika teamen (P4: 28). P4 beskriver att han utövar testdriven utveckling i vissa fall, men att det beror på hur mycket tid som finns och hur komplext det han bygger är (P4: 28). P3 beskriver att det är svårt att prioritera mellan att bygga nya funktioner och att ägna sig åt refaktorise-

ring av teknisk skuld, och att det handlar om att hitta en bra balans för att investera framåt (P3: 42).

På Spotify måste all kod som checkas in genomgå en kodgranskning av en eller flera personer i teamet (P1: 46; P2: 40). Granskningen sker informellt och utgår inte från några specificerade checklistor eller standarder (P1: 52). P2 förklarar det som att allt som går att skriva ned även går att automatisera, manuell kontroll av sådant som syntax och stavfel är därmed slöseri med tid (P2: 42). Kodgranskningen på Spotify är en öppen process där det faller sig naturligt att individer med olika specialiseringar tittar på olika saker (P2: 36). P1 beskriver att QA-ingenjörer ofta blir tillfrågade om att delta i granskningen, och att de gärna gör det, men understryker samtidigt att vem som helst i teamet kan göra det (P1: 46). Parprogrammering utförs sporadiskt på företaget och P1 önskar att fler gjorde det, däremot påpekar han att parprogrammering aldrig kan ersätta kodgranskningen helt och hållet, även om det löser liknande problem (P1: 48).

På Klarna måste en kodgranskning genomföras av två personer varje gång en kodförändring är redo att checkas in och integreras med den gemensamma kodbasen (P3: 46; P4: 30). Ingen formell checklista följs vid granskningen, däremot har de flesta team någon typ av Linter-verktyg som automatiskt kontrollerar kodens syntax och kvalitet (P3: 50). Ifall det rör sig om en väldigt komplex funktion tillsätts en person som både granskar, exekverar och testar koden för att uppnå en djupare granskning (P3: 46; P4: 32). På Klarna bedrivs både par- och mobbprogrammering (P3: 28). P3 beskriver att parprogrammering emellertid inte eliminerar behovet för kodgranskning, eftersom att det finns en risk för att paret blir entusiastiska och snöar in sig i en viss riktning. Han påpekar att kodgranskning som genomförs av personer som inte varit involverade i utvecklingen är bra, då de har färsk syn och enklare kan upptäcka problem och ifrågasätta designbeslut (P3: 48). De team som inte parar vid utvecklingen beskrivs emellertid som mer rigorösa när det kommer till kodgranskning (P3: 28).

Spotify hade för några år sedan en dedikerad Operations-avdelning som hanterade all uppsättning och konfiguration av infrastruktur. Idag är mycket av det arbetet flyttat till utvecklarna, eftersom att det har visat sig vara friktions- och kostnadskrävande att ha systemadministrations- och utvecklingsaktiviteter separerat uppdelade (P1: 54). P1 beskriver att ifall ett driftproblem skulle inträffa är det lättare för utvecklingsteamet att lösa det eftersom att det vet vilka förändringar som har driftsatts och således snabbare kan identifiera orsaken till problemet (P1: 54). Företaget använder sig av konceptet infrastruktur som kod med hjälp av konfigurationshanteringsverktyget Puppet, som ärver en uppsättning Ruby-klasser och skapar en definition för hur infrastrukturen ska se ut (P2: 46). P2 berättar att det inte finns något behov för talspråkskrivna beteendebeskrivningar för infrastruktur eftersom att Ruby är ett relativt lättläst programmeringsspråk (P2: 48).

På Klarna finns det dedikerade IT Ops-team som supporterar produktutvecklingsteamen med bland annat driftsättning, nätverk och infrastruktur (P3: 8; 10). Trots att de tillhör en annan avdelning är det ofta nära samarbete mellan teamen när det behövs (P3: 10). Klarna driftar sina produkter på en hybridmolnlösning och utnyttjar konfigurationsramverken Chef och An-

sible för konfigurering av infrastruktur med hjälp av kodscript (P3: 52; P4: 34, 36). Varje team behandlar sin infrastruktur som kod och varje gång en ändring görs i ett konfigureringsscript måste den referentgranskas på samma sätt som med källkod (P4: 36). Både P3 och P4 uppger att det inte finns något behov för att tillämpa beteendedriven systemadministration, eftersom att syntaxen i form av språken Ruby och Jamel är så pass lättförstådd (P3: 56; P4: 38). P3 beskriver att domänspråk för konfigurering av infrastruktur har både för- och nackdelar. Ett problem med det kan vara att saker abstraheras bort på grund av att en beteendebeskrivning kanske inte exakt matchar det som egentligen utförs (P3: 56).

5.1.3 Mätning, övervakning och återkoppling

Denna sektion redovisar empiriska resultat kopplade till mätning, övervakning och återkoppling. Tabell 5.3 visar från vilken intervjuperson och vilken rad i transkriberingsprotokollen resultaten är extraherade.

Tabell 5.3: Översikt av data inom mätning, övervakning och återkoppling

Kategori	*Kapabilitet*		*P1 (Rad)*	*P2 (Rad)*	*P3 (Rad)*	*P4 (Rad)*
Mätning, övervakning och återkoppling	K12	Kontinuerlig mätning och optimering av utvecklings- och leveransprocessen	20, 22	62, 64, 66	18, 70	56
	K13	Kontinuerlig övervakning och återkoppling av användarbeteende och realtidsprestanda	20, 56, 58, 60	50, 60, 74	26, 54, 58, 66	18, 20, 40
	K14	Datadriven genomslags- och riskanalys baserad på återkoppling och användarscenarion	20, 32, 60	50, 58, 60, 74	24, 38	
	K15	Gradvis utrullning av releaser och genomslagsanalys genom A/B-testning	62, 64	52, 60	64, 66	50, 52
	K16	Mörklagd produktlansering för proaktiv problemidentifikation	32, 66	54	64, 66	50, 52

Konkret mätning av utvecklings- och leveransprocessens kvalitet anser både P1 och P2 vara svårt, eftersom att det i det stora hela handlar om mjuka aspekter och en kollektiv laginsats (P1: 20; P2: 66). P1 beskriver att det är svårt att veta vilken inverkan ett enskilt team har på den övergripande användarupplevelsen (P1: 22), han lägger emellertid emfas vid att det är viktigt att ändå försöka hitta sätt att mäta processen (P1: 20). Utvecklingsteam mäter ledtider och tittar på hur lång tid det tar från att ha börjat skriva kod till att den är driftsatt och att resultatet av den syns i återkopplingsslingorna. Utifrån det kan teamet analysera om det går att korta ner tiden för exempelvis kompilering och exekvering av automatiserade tester (P1: 22). Mätning och optimering av processer sker på olika sätt och att varje team är ansvarigt för att hitta de verktyg som fungerar bäst (P2: 66). P2s team har retrospektiva möten där det förs

diskussioner kring vad som gick bra, dåligt och åtgärdspunkter (P2: 62). P2 upplever att saker som behöver komma upp till ytan vad gäller teamprestanda gör det naturligt, utan behov av mätvärden och estimeringar (P2: 64).

Mätning och optimering av utvecklingsprocessen handlar enligt P3 om samarbete och att gå igenom de olika teamens produkter och tester. På så sätt kan man identifiera vilka team som har flest fel och upptäcka om det finns diskrepanser mellan förväntningar och resultat (P3: 70). Typiska orsaker till problem är brist på information eller för lite tid allokerad till planering i kombination med en stor backlog (P3: 70). Varje team har egna KPIer i form av så kallade Objective Queue Results, som används för att mäta och optimera teamets resultat (P3: 18; P4: 56). System Test mäter exempelvis hur många av organisationens tester som fallerar på grund av deras end-to-end-tester, eftersom de för tillfället har ett mål om att reducera den siffran med 50 %. P4 beskriver också att de ständigt arbetar på att identifiera nya mätvärden och mätinstrument som kan användas (P4: 56). För att optimera leveransprocessen beskriver P3 att System Test brukar analysera de olika teamens leveranspipelines och sedan hjälpa dem att bli av med tidskrävande manuella steg. P3 berättar att manuella steg brukar leda till att fler fel uppstår (P3: 70).

Spotify övervakar kontinuerligt antal användare och deras ursprung och bakgrund, såväl som deras beteenden i form av exempelvis klickfrekvens, sessionstid och konverteringsgrad (P1: 20, 56; P2: 60). Det finns dashboards på kontoret som visar realtidsprestanda som exempelvis CPU- och databasbelastning (P1: 56). Ifall exempelvis antal köp drastiskt förändras i förhållande till de historiskt normala nivåerna kan teamet som äger betalningsfunktionen undersöka vad som driftsatts, och således identifiera vad orsaken är (P1: 56; P2: 50). Denna typ av data används som riktlinjer för explorativ testning och problemanalys (P2: 50). Det finns en intern avdelning som arbetar med att utveckla färdiga komponenter och verktyg för övervakning, dessa kan utvecklingsteamen utnyttja och bygga in i sina produkter (P1: 58). Nyttjandet av denna typ av data handlar både om att fasa ut funktioner som inte används, som att identifiera möjligheter för förbättring av produkten (P1: 60). P2 beskriver att det är viktigt att begära ut återkoppling från användare för att kunna bedöma vilken nivå av kvalitet som bör vidhållas (P2: 74).

Klarna övervakar och sparar information om vilka anrop användare gör och under vilka tidspunkter de gör dem, för att kunna göra statistik av det och analysera fram trender (P3: 26). Denna information används bland annat vid planering av systemtester, för att identifiera vad en användare förväntar sig av en viss funktion. System Test går igenom olika scenarion och säkerställer proaktivt vilka fel som kan inträffa och hur sannolikt det är. Utifrån detta konstrueras testerna (P3: 28). Användardata används också för att genomföra riskanalyser inför releaser (P3: 38). Detta brukar ligga till grund för de rekommendationer som System Test förser beslutsägarna med. P3 beskriver att tillgången till data gör att dessa beslut enklare kan tas (P3: 38).

Vid introduktion av nya funktioner genomför Spotify en analys av hur komplexa eller svårförstådda de är, genom gradvis utrullning och kontinuerlig övervakning (P1: 62). Spotify arbetar

aktivt med A/B-testning (P1: 64; P2: 52) och släpper en funktion först till 1 % av användarna, som därefter övervakas noggrant för att analysera deras beteenden och samla in eventuella klagomål. Graden av aktivering och retention undersöks noggrant och jämförs mot någon fördefinierad metrik, och baserat på genomslagskraft tas sedan ett vetenskapligt grundat beslut om att antingen rulla ut funktionen till 100 % eller gå tillbaka till ritbordet (P1: 62; P2: 60). Om en funktion visar sig vara framgångsrik görs en datadriven riskbedömning baserat på i vilket skede funktionen är och vad den riskerar att påverka (P1: 32; P2: 54). Därefter släpps funktionen gradvis till resten av användarna med hjälp av mörklagd produktlansering, vilket benämns som "feature toggles". Syftet är att minimera sprängradien av eventuella prestanda- och driftsproblem som kan uppstå (P1: 66).

Samtliga funktioner som Klarna driftsätter i produktionsmiljö är till en början inaktiverade som standard (P4: 50). Funktionerna aktiveras sedan för ett fåtal användare och A/B-testas för att säkerställa att de fungerar och för att mäta och övervaka hur användarna reagerar samt hur funktionen påverkar konverteringsgraden (P3: 64, 66; P4: 52). P3 beskriver att de har så kallade "feature flags" för mörklagd produktlansering (P3: 64), det är emellertid upp till varje enskilt team och dess produktägare att avgöra om funktionen ska rullas ut gradvis, helt och hållet, eller inte alls (P4: 54).

5.2 Tabellsammanfattning av resultat

I detta delkapitel följer tabell 5.4 som sammanfattar de empiriska resultaten i ett konkretiserat tabellformat.

Tabell 5.4: Sammanfattning av resultat i tabellform

Kategori	*Kapabilitet*		*P1*	*P2*	*P3*	*P4*
Samarbetspräglad organisations-struktur och kultur	K1	Delande av arbetssätt, ansvar och kollektivt ägarskap för kvalitetssäkring	Ja	Ja	Ja	Ja
	K2	QA har en teknisk profil och arbetar nära utvecklare och systemadministratörer	Ja	Ja	Ja	Ja
	K3	Gemensamma mål och incitament för kvalitetssäkring	Ja, implicit	Ja, implicit	Ja	Ja, implicit
	K4	Kontinuerligt experimenterande, lärande och delande av kunskap	Ja	Ja	Ja	Ja
	K5	QA har inflytande över designbeslut	Ja	Ja	Ja	Ja
	K6	QA har inflytande över releasebeslut	Ja	Ja	Ja	Ja
Kontinuitet och automation	K7	Kontinuerlig och automatiserad testning av funktionalitet och kvalitet	Ja	Ja	Ja	Ja
	K8	Testdriven och beteendedriven utveckling samt refaktorisering	Varierande	Varierande	Varierande	Varierande
	K9	Kvalitetssäkring av källkod genom kontinuerlig referentgranskning	Ja	Ja	Ja	Ja
	K10	Infrastruktur som kod för automation av infrastruktur	Vet ej	Ja	Ja	Ja
	K11	Beteendedriven systemadministration för gemensam förståelse för infrastruktur	Vet ej	Nej	Nej	Nej
Mätning, övervakning och återkoppling	K12	Kontinuerlig mätning och optimering av utvecklings- och leveransprocessen	Ja	Ja, implicit	Ja	Ja
	K13	Kontinuerlig övervakning och återkoppling av användarbeteende och realtidsprestanda	Ja	Ja	Ja	
	K14	Datadriven genomslags- och riskanalys baserad på återkoppling och användarscenarion	Ja	Ja	Ja	Ja
	K15	Gradvis utrullning av releaser och genomslagsanalys genom A/B-testning	Ja	Ja	Ja	Ja
	K16	Mörklagd produktlansering för proaktiv problemidentifikation	Ja	Ja	Ja	Ja

6 Diskussion

I detta kapitel diskuteras studiens empiriska resultat i relation till det teoretiska ramverket för att uppmärksamma överensstämmelser och avvikelser som påträffats. Vidare förs även resonemang kring vad dessa överensstämmelser och avvikelser kan bero på.

6.1 Samarbetspräglad organisationsstruktur och kultur

Båda organisationerna har en tvärdisciplinär organisationsstruktur där varje team består av personer med olika inriktningar, varav någon i teamet i regel är specialiserad på QA eller testning. Båda organisationerna benämner dessa ingenjörer, och P2, P3 samt P4 understryker att de praktiskt taget är utvecklare, då de till stor del arbetar med utveckling av verktyg, programmering och testautomation. Detta är i linje med hur Armenise (2015) och Rodríguez et al. (2016) uttrycker sig när de hävdar att QA-personal i en DevOps-organisation har en teknisk profil och arbetar nära utvecklare och systemadministratörer, och således behöver förståelse för systemets och teamets behov, såväl som teknisk kunskap. Sumrell (2007) och Erich et al. (2014) skriver vidare att vid iterativ utveckling likt DevOps flyttas många kvalitetssäkringsaktiviteter över helt till utvecklare, vilket styrks av samtliga respondenter som berättar att enhets- och integrationstestning automatiseras och utförs av utvecklarna; QA fokuserar istället på helhetsöversikt samt övergripande system- och regressionstestning. Det är här tydligt att de empiriska resultaten är i linje med det litteraturen beskriver. I en DevOps-organisation är team tvärfunktionellt sammansatta av personer med teknisk profil, och kvalitetssäkringsaktiviteterna är distribuerade över hela teamet.

Empirin visar att team anses vara autonoma och har kollektivt ansvar för att den produkt teamet äger är vältestad och håller hög kvalitet. P2 beskriver att kvalitet är något som ägs av organisationen som helhet, och att varje team och dess individer bidrar till att leverera en bra helhetlig användarupplevelse. Detta är i linje med Smeds et al. (2015) och Bhasin (2012) som beskriver att anställda arbetar kollektivt som ett “team av teams” och kvalitetssäkring är en kontinuerligt integrerad del av allas dagliga arbete, samt Roche (2013) som vidare hävdar att samtliga inom organisationens produktutveckling gemensamt ansvarar för att säkerställa hög kvalitet och användbarhet. QA försöker inte kontrollera eller säkra kvalitet, snarare göra det möjligt för alla i teamet att bygga och testa produkten på ett sätt som gör att den håller hög kvalitet. QAs fokus är att stödja sina kollegor med kunskap, testramverk och verktyg, samt rådgivning och coachning kring kvalitetsaspekter och testning. P1 beskriver att QAs roll är att

stötta teamet med kvalitetstänk, likt en agil coach stöttar processer. Det kan härledas till Roche (2013) som menar att QAs roll i ett tvärdisciplinärt team är mer av en proaktiv kvalitetsförespråkare som bidrar med insikter i kvalitetsfrågor, än en reaktiv testare. Empirin styrker litteraturen vad gäller delande av arbetssätt, ansvar och kollektivt ägarskap. Strävan mot hög kvalitet och bra användarupplevelser är en holistisk sak som ägs gemensamt av hela produktutvecklingsorganisationen. Organisationer som tillämpar DevOps har trots det till synes personer som är inriktade på QA, de har emellertid inte mer ansvar för kvalitet än någon annan, inte heller har de någon form av kontrollant- eller portvaktsroll. QAs roll i teamet är att fokusera extra på kvalitetsaspekter och att söka upp och coacha frågor kring kvalitet och testning. P1 och P2 föredrar termen Quality Assistance istället för Quality Assurance och menar att det inte går att säkra kvalitet - däremot går det att assistera individer och team med verktyg, metoder och rådgivning som hjälper dem att arbeta på ett sätt som leder till hög kvalitet.

Ingen av intervjupersonerna beskriver någon form av explicita målsättningar eller incitament för kvalitetssäkring och kvalitet, dels för att kvalitet är svårt att kvantifiera och mäta, och dels för att fokus inte bör ligga vid individuella prestationer. Detta står delvis i kontrast till Le-Quoc (2011) som skriver att DevOps bör komma med ett organisationsövergripande metrikdrivet ramverk som kan användas för att definiera delade och angripbara mätvärden. P1 uttrycker att målet är att gemensamt skapa en bra produkt som uppskattas av användare, vidare beskriver både P1 och P4 att de drivs av möjligheten till nya utmaningar, ansvarsområden, positiv återkoppling från organisationen och uppskattning från kunder. Walls (2013) och Smeds et al. (2015) uttrycker sig liknande och skriver att i en DevOps-organisation delar alla samma mål - att skapa bra upplevelser för användare. Således belönas organisationens kollektiva insats när kunderna är nöjda. De anställda på Klarna har emellertid personliga och teambaserade mål som *kan* vara relaterade till förbättring av testaktiviteter, P3 understryker dock att sådana mål sällan bidrar till högre kvalitet i sig. Mycket av litteraturen ligger här i linje med empirin. I en DevOps-organisation delar alla det gemensamma målet om att skapa en produkt med hög kvalitet. Däremot visar empirin att det är svårt att ha explicita incitament, eftersom att kvalitet i praktiken är lika med kundnöjdhet. Kunder bedömer produkten efter dess helhetsintryck och det går således inte att härleda god kvalitet till enskilda individers eller gruppers prestationer. Därför är det med DevOps, som bygger på en kollektiv insats, svårt att applicera konkreta incitament och belöningar för kvalitetssäkring, i synnerhet på en lägre nivå än organisationsnivå.

Samtliga intervjupersoner uppger att deras organisationer uppmuntrar innovation och experimenterande, och att det är helt i sin ordning att misslyckas så länge det leder till ny kunskap som kan delas. Smeds et al. (2015) och Rodríguez et al. (2016) lägger emfas vid att detta är ett viktigt grunddrag för DevOps och Ozer och Vogel (2015) skriver att kunskapsdelande mellan individer och kontinuerligt lärande leder till ökad kvalitet i arbetsprocesser. P2 uppger att en bra plattform för delande av kunskap om framgångar och misslyckanden är retrospektiva möten, vilket även Cho et al. (2011) tar upp. Empirin och litteraturen pekar på att kontinuerligt delande av kunskap inom en DevOps-organisation är en form av kvalitetssäkring eftersom att det minskar risken för återupprepning av misstag i utvecklings- och leveransprocessen. Empi-

rin visar också att experimenterande med, och delande av kunskap om, nya arbetsmetoder och verktyg är viktigt för att hitta smartare sätt att genomföra kvalitetssäkringsaktiviteter.

Vidare visar empirin att design- och releasebeslut tas gemensamt inom teamen, där QA har inflytande likställt alla andra. Som tidigare nämnt har QA inte en kontrollerande eller ansvarstagande roll för kvalitet, utan vid problematiska beslut är det produktägaren som avgör. Vid tvivelaktiga releaser som till exempel innefattar kända defekter utförs om möjligt en datadriven analys för att ta informerade beslut gällande den övergripande inverkan på kvaliteten och slutanvändarupplevelsen. Det handlar i slutändan om att avgöra vilket organisationsövergripande värde som kan skapas utifrån en eventuell release, vilket underlättas av tillgången till data. Empirin är i linje med hur Roche (2013) belyser det QAs och teamets gemensamma inflytande över design- och releasebeslut, samt utnyttjande av data vid beslutsfattande.

6.2 Kontinuitet och automation

Empirin visar att testaktiviteter i de båda organisationerna är automatiserade till en så hög grad som möjligt för att undvika tidskrävande manuell testning av funktionalitet, vilket ligger i linje med vad Waller et al. (2015) och Smeds et al. (2015) hävdar; att testning i en DevOps-präglad mjukvarulivscykel i stor utsträckning understöds av automation. I samband med incheckning och integration av kod triggas automatiserade enhetstester i leveranspipelinen, vars resultat sedan återkopplas. Om alla enhetstester passeras fortsätter leveranspipelinen med integrations- och end-to-end-tester, samt vissa prestanda- och driftsättningstester efter behov.

Dessa automatiserade tester byggs som regel in i källkoden av teamet under utvecklingens gång, vilket enligt Solis och Wang (2011) möjliggör att kvalitets- och funktionalitetskriterier kan kvalitetssäkras i ett tidigare skede. Erich et al. (2014) beskriver att detta med fördel kan genomföras med hjälp av metoder som testdriven och beteendedriven utveckling, och att dessa metoder lämpar sig väl för DevOps.

Erich et al. (2014) understryker samtidigt att en princip med DevOps är att låta team själva bestämma vilka specifika metoder och verktyg de vill använda. Detta styrks av empirin som visar att ett flertal olika metoder tillämpas. P1 beskriver det som att påtvingande av specifika metoder kan medföra friktion och kräva stora omställningar i tankesätt. P2 och P4 berättar att deras team ofta anpassar sina metoder efter vilken uppgift de står inför, för att kunna arbeta flexibelt utifrån ett pragmatiskt och synergistiskt tillvägagångssätt. Litteraturen och empirin verkar vara eniga i att med DevOps är val av utvecklingsmetoder frivilligt, så länge de involverar proaktiv kvalitetssäkring och kontinuerlig testning i form av inbyggda automatiserade testsystem och leder till leverans av vältestade produkter med hög kvalitet.

Empirin lyfter samtidigt att det inte går att automatisera kvalitetssäkring helt och hållet och att gruppbaserad manuell explorativ testning ofta utförs inför större releaser som sträcker sig över flera produktområden, där flera team har varit involverade i utvecklingen. Explorativ

testning är en typ av testning som inte har tagits upp i den DevOps-litteratur vi identifierat. Detta kan bero på att explorativ testning enligt empirin endast utförs sporadiskt och efter behov, till skillnad från automatiserad testning som enligt både empiri och litteratur är en nyckelaktivitet för en DevOps-organisation. Därmed uppmärksammar forskare och författare inom DevOps inte explorativ testning i någon större utsträckning och det tillägnas således inget betydande utrymme i litteraturen. Det faktum att de båda studerade organisationerna utför explorativ testning efter behov indikerar emellertid att det kan anses vara en viktig kvalitetssäkringsaktivitet i en DevOps-kontext.

Samtliga intervjupersoner uppger att referentgranskning alltid genomförs av minst en person innan ny kod checkas in för integration, vilket Davis (2013) beskriver som viktigt för att kunna upptäcka problem i ett tidigare skede. Fitzgerald och Stol (2015) menar att referentgranskning utifrån fördefinierade standarder och checklistor är mest effektfullt, detta förekommer däremot inte i praktiken. I förväg nedskrivna krav på basala ting som syntax och namngivningskonventioner kan kontrolleras automatiskt med hjälp av analysverktyg, referentgranskningen fokuseras istället kring designbeslut på en högre nivå och sådant som kräver mänsklig intuition. Vidare beskriver Di Bella et al. (2013) och Huizinga och Kolawa (2007) att parprogrammering är ett proaktivt sätt att kontinuerligt referentgranska kod. Empirin visar däremot att parprogrammering är en frivillig och sporadiskt tillämpad metod som aldrig kan *ersätta* en ordentlig referentgranskning, eftersom att utvecklare som parprogrammerar ofta kan snöa in sig i ett ensidigt perspektiv. Återigen tyder empirin på att kvalitetssäkring av okonstlade saker med fördel kan understödjas av automation och att val av metoder sker flexibelt efter behov. Det är tydligt att i DevOps är mänsklig och kritisk referentgranskning av källkod en viktig kvalitetssäkringsaktivitet som aldrig ersättas helt och hållet av varken automationsverktyg eller specifika utvecklingsmetoder.

Båda organisationerna beskriver att infrastrukturkonfigurering behandlas på samma sätt som mjukvarukomponenter med hjälp av konfigurationshanteringsverktyg som Chef och Puppet. Ändringar i konfigureringscript referentgranskas innan incheckning och genomgår automatiserade tester i leveranspipelinen. Detta har medfört att det nästan uteslutande är utvecklingsteamen som arbetar med uppsättning och konfiguration av infrastruktur, vilket en av respondenterna förklarar minskar kostnader, friktion mellan avdelningar, samt underlättar problemidentifikation vid driftsättning. Både Spinellis (2012) och Lwakatare et al. (2015) beskriver likt att infrastruktur som kod (IaC) förstärker samarbetet mellan utveckling och systemadministration, och därmed möjliggör upprepningsbar och frekvent driftsättning av nya funktioner och felrättelser. Schaefer et al. (2013) menar att detta leder till en mer konsekvent användarupplevelse, vilket utifrån empirin även kan härledas från att driftsättningsproblem kan åtgärdas snabbare. Gohil et al. (2011) menar att IaC ytterligare kan accelereras genom att beskriva konfigurationen med beteendebeskrivningar på ett naturligt domänspråk, vilket benämns som beteendedriven systemadministration (BDOps). Empirin tyder emellertid på att det inte finns något behov av att beskriva infrastrukturbeteende på något domänspråk, eftersom att syntaxen i konfigurationsverktygen anses vara så pass lättförstådd. Detta kan vara en effekt av att de

personer som har en roll i produktutvecklingen och kvalitetssäkringsarbetet besitter en teknisk profil, och därmed redan har en underliggande förståelse för enklare programmeringsspråk.

6.3 Mätning, övervakning och återkoppling

DevOps lägger enligt Roche (2013) vikt vid kontinuerlig mätning och optimering av utvecklings- och leveransprocessens prestanda och Le-Quoc (2011) menar att det är nödvändigt med ett organisationsövergripande metrikdrivet ramverk för att kunna följa upp målsättningar och identifiera möjligheter till förbättringar. Empirin visar att ingen av organisationerna arbetar explicit utifrån något metrikdrivet ramverk, men att kontinuerlig utvärdering och optimering av processer sker på andra sätt. Fokus ligger på bland annat mätning och optimering av ledtider, samt minimering av kompileringstider och antal fallerade tester. Även mer informell uppföljning och optimering av processer sker i form av retrospektiva möten och fortlöpande identifikation och strömlinjeformning av tidskrävande steg och aktiviteter. Litteraturen och empirin i kombination antyder att i organisationer som tillämpar DevOps kontinuerligt bedriver mätning, optimering och kvalitetssäkring av processer, men på ett naturligt, informellt och pragmatiskt sätt.

Beträffande produktkvalitet är det utifrån empirin tydligt att genomslags- och riskanalyser genomförs och baseras på återkoppling av användarbeteende och användarmönster. I linje med detta hävdar Liu et al. (2014) och Claps et al. (2015) att slutanvändarupplevelsen kan optimeras genom kontinuerlig övervakning av användardata för att proaktivt identifiera potentiella högriskområden och sannolika scenarion som användaren kommer att stöta på. Både realtidsprestanda och användarbeteenden övervakas kontinuerligt och används i återkopplingsslingor av de båda organisationerna för att snabbt kunna identifiera problem, samt för att göra prediktiva analyser av hur design- och releasebeslut kommer att påverka användarna och systemet i dess helhet. Detta är vidare i linje med Roche (2013) som menar att tillgången till användardata från produktionsmiljön ger utrymme för proaktiv kvalitetssäkring och är en grundpelare för beslutsfattande i DevOps-organisationer. Empirin och litteraturen stämmer väl överens gällande att DevOps-präglade organisationer tar beslut baserat på kartläggning och analys av olika prestanda- och användarscenarion, vilket möjliggörs av kontinuerlig mätning, övervakning och flödande återkopplingsslingor.

Empirin visar vidare att nya funktioner utvärderas genom att initialt rulla ut dessa till endast ett fåtal användare, för att sedan analysera genomslagskraften utifrån särskilda mätvärden, så som aktivering, retention och konverteringsgrad. A/B-testning utförs även i samband med detta, vilket beskrivs av Adams & McIntosh (2016) som ett tillvägagångssätt för att utvärdera olika alternativ till funktionen i fråga. Beslut om fullständig utrullning eller reversering tags sedan utifrån en datadriven bedömning av risk och genomslagskraft. Adams & McIntosh (2016) beskriver i linje med empirin hur gradvis utrullning av nya funktioner tidigt kan indikera kvalitetsbrister. De menar vidare att funktioner kan driftsättas utan synlighet för slutanvändarna, med så kallad mörklagd produktlansering, i syfte att utföra diverse prestandatester i

produktionsmiljö. Empirin tyder på att mörklagd produktlansering utförs när beslut om fullständig utrullning tagits, då i syfte för att minimera sprängradien gällande till exempel prestanda- och driftsproblem. Litteraturen och empirin är här i linje och indikerar att kvalitetssäkring i samband med driftsättning och release genomförs i en kombinerad form av gradvis utrullning av funktioner, A/B-testning och mörklagd produktlansering, dels för att kunna stödja releasebeslut med information om risker och genomslagskraft, samt för att kunna reducera antalet påverkade användare vid potentiella driftsättningsproblem.

7 Slutsats

DevOps är en uppsättning koncept som ämnar optimera hela mjukvarulivscykeln och förkorta tiden till marknaden. Snabbare respons på kunders ständigt föränderliga önskemål åstadkoms genom att eliminera flaskhalsar i både utvecklings- och leveransprocessen och särskilt fokus ligger på integration av traditionellt sett uppdelade organisatoriska funktioner. I en DevOps-kontext, där snabba och frekventa produktsläpp prioriteras högt, kan det tänkas att kvalitetssäkring nedprioriteras eller genomförs i mindre utsträckning - inte minst eftersom att QA-funktionens arbetsuppgifter är allt mer flytande, distribuerade eller automatiserade. Litteratur inom DevOps beskriver kvalitetssäkring som viktigt, men fokuserar lite på dess förhållande till DevOps och hur arbetet med kvalitetssäkring ser ut i en DevOps-kontext. Studien ämnade svara på frågan: *Hur bedrivs kvalitetssäkring i organisationer som tillämpar DevOps?*

Vi har med denna studie funnit att aktiviteter relaterade till optimering av både process- och produktkvalitet i stor utsträckning återfinns i organisationer som tillämpar DevOps. Dessa aktiviteter karaktäriseras av ett proaktivt tillvägagångssätt i ett kvalitetsassisterande syfte, snarare än reaktivt och kontrollerande.

I en DevOps-präglad organisation är team autonoma, multidisciplinära och tvärfunktionellt sammansatta av personer med tekniska profiler och kvalitetssäkringsaktiviteter ägs av och är distribuerade över hela teamet. Kvalitet är i slutändan samma sak som kundnöjdhet och är en holistisk sak som ägs av hela organisationen, då alla delar det gemensamma målet om att skapa en bra produkt med hög kvalitet. Det går således inte att härleda hög kvalitet till enskilda individers eller gruppers prestationer och det är därför även svårt att applicera konkreta individuella incitament eller belöningar för kvalitetssäkring. Ett fåtal mätvärden i form av ledtider och vissa instrument som retrospektiva möten används för att avgöra hur team presterar, men generellt sett används inga rigorösa metrikdrivna ramverk för att kvalitetssäkra utvecklings- och leveransprocessen.

Trots att kvalitetssäkring genomförs som en kollektiv insats återfinns personer som inriktar sig speciellt på QA och testning. Dessa personer varken ansvarar för eller kontrollerar kvalitetssäkring och kvalitet, utan fokuserar på att söka upp och coacha kvalitetsfrågor. Det är i en DevOps-organisation, med snabba och frekventa produktsläpp, lönlöst att försöka *säkra* kvalitet genom manuella kontroller i dedikerade kvalitetssäkringsfaser. Det bildar flaskhalsar i mjukvarulivscykeln, hämmar enskilda individers självstyre och förstör idén om kollektivt ägarskap för kvalitet. Istället *assisteras* kvalitet proaktivt genom att kontinuerligt coacha team och individer med kunskap, rådgivning och testverktyg som stödjer det kollektiva arbetet mot hög kvalitet och bra användarupplevelser. Experimenterande med, och delande av kunskap

om, nya metoder och verktyg understödjer arbetet med kvalitetssäkring genom att minska risken för återupprepning av misstag.

Arbetet med själva mjukvaruutvecklingen sker i en DevOps-organisation på ett pragmatiskt sätt, där metoder som testdriven och beteendedriven utveckling, parprogrammering och mobbprogrammering är vanligt förekommande, men högst frivilliga. Produktkvalitet i form av exempelvis testbarhet, modifierbarhet och användbarhet beaktas och optimeras kontinuerligt av samtliga i teamet, och coachas av personer med inriktning på QA och testning. Verifikation av elementär funktionalitet sker i ett tidigt stadie genom att bygga in automatiserade testsystem i koden, vars tester exekveras kontinuerligt i utvecklingsprocessen såväl som i leveranspipelinen under leveransprocessen. I en DevOps-organisation präglas mjukvarutestning i stor utsträckning av automation, däremot finns det fortfarande behov för mänsklig intuition och manuell kvalitetssäkring i form av explorativ testning och kontinuerlig referentgranskning av kod. Infrastruktur behandlas som kod och automatiseras med hjälp av konfigurationshanteringsverktyg och kodscript. Detta, i samband med den tvärdisciplinära organisationsstruktur som återfinns DevOps-organisationer, medför att uppsättning och konfiguration av infrastruktur kan hanteras och kvalitetssäkras av teamen själva. Ifall problem med koden inträffar i drift kan teamen snabbt hitta orsaken eftersom att de vet vilka förändringar som nyligen driftsatts.

Diskussioner kring design- och releasebeslut understödjs om möjligt alltid av oomtvistlig data från de återkopplingsslingor som är ett resultat av att DevOps innefattar kontinuerlig övervakning och kartläggning av realtidsprestanda och användarbeteenden. De tvärdisciplinära teamen äger kollektivt dess design- och releasebeslut och teammedlemmarna har lika stort inflytande då alla besitter unika perspektiv och domänkunskap. För att fatta releasebeslut utifrån data rullas nya funktioner ut gradvis och A/B-testas inom utvalda användargrupper. Övervakning och återkoppling av användarnas beteenden visar funktionens genomslagskraft och ligger till grund för beslut kring huruvida funktionen ska släppas till samtliga användare eller reverseras. Med hjälp av mörklagd produktlansering släpps sedan funktionen osynligt för användarna och synliggörs därefter gradvis i syfte att minimera sprängradien för potentiella prestanda- och driftsproblem.

Studien har påvisat att organisationer som tillämpar DevOps bedriver kvalitetsassistans snarare än kvalitetssäkring. När kvalitetssäkring övergår till att vara en kollektiv insats ställs det krav på att alla som är involverade i produktutvecklingen kontinuerligt arbetar med, samt tar ansvar för, kvalitet. QA-funktionen understödjer och coachar diskussioner, beslut och aktiviteter relaterade till kvalitet och testning, därmed assisteras kvalitet proaktivt snarare än säkras reaktivt. Studien har vidare påvisat att denna typ av proaktiv kvalitetassistans till stor del understödjs av testautomation, samt data som återkopplas från kontinuerlig övervakning av användarbeteenden och realtidprestanda.

7.1 Vidare forskning

Vi har med denna studie undersökt hur kvalitetssäkringsaktiviteter bedrivs i organisationer som tillämpar DevOps. Vi har i detta sammanhang valt att avgränsa bort hur slutproduktens kvalitet påverkas av dessa kvalitetssäkringsaktiviteter, något som vi anser vara en naturlig förlängning av studien. Det skifte mot kvalitetsassistans istället för kvalitetssäkring som studiens resultat utvisar utgör nya perspektiv att bedriva forskning utifrån inom ämnesområden som mjukvarukvalitetssäkring, mjukvarukvalitet och testning.

Bilagor

B1 Intervjuguide - Svenska

Bakgrund

- Vilken är din arbetsroll?
- Hur länge har du arbetat i den rollen?
- Har du någon annan bakgrund inom företaget?
- Vad har du för bakgrund innan du började på företaget?
- Vilka är dina främsta ansvarsområden?
- Vilka är dina typiska dagliga aktiviteter?

Organisationsstruktur och kultur

- Kan du kortfattat beskriva företagets organisationsstruktur för produktutvecklingen?
- Vilken avdelning och/eller vilket team ingår du i?
- Vilka andra roller ingår i ditt team/avdelning?
- Vilka roller skulle du säga är involverade i arbetet med kvalitetssäkring av produkten?
- Vilka roller skulle du säga är involverade i arbetet med kvalitetssäkring av processen?
- Hur involverade i arbetet med kvalitetssäkring är de personer i teamet/organisationen som inte är formellt inriktade på QA - t.ex. utvecklare och systemadministratörer?
    - På vilket sätt samarbetar ni?
        - Verktyg
        - Arbetssätt
        - Kommunikation
- Finns det några definierade målsättningar och/eller incitament för kvalitetsförbättringar?
    - Individnivå
    - Teamnivå
    - Organisationsnivå
- Har företaget någon typ av belöningsanordning (både monetär och icke-monetär)?
    - Vilka omfattas av detta?
    - Vilka faktorer baseras den på?
- Upplever du att företaget uppmuntrar till experimenterande och innovation?
- Hur drar ni lärdom av och delar kunskap kring framgångar och eventuella misslyckanden?
- Hur upplever du QAs har inflytande över beslut gällande mjukvaruproduktens design och arkitektur?
- Hur upplever du QAs har inflytande över beslut gällande huruvida en funktion är färdig för release eller inte?

Kontinuitet och automation

- Kan du kortfattat beskriva hur företaget arbetar med kvalitetssäkring av nya funktioner?
- Vilka typer av tester kör ni, samt när och hur frekvent körs dessa?
    - Vem eller vilka ansvarar för utformningen av de olika testerna?
    - Vem eller vilka ansvarar för exekveringen av de olika testerna?
- Hur stor andel av testerna är automatiserade?
    - Vilka är inte automatiserade?
- Vilken eller vilka metoder och tillämpningar för mjukvaruutveckling använder företaget?
- Finns det några specificerade kvalitetskrav som måste tas hänsyn till vid utveckling och kvalitetssäkring av produkten?
- Sker det någon referentgranskning av koden innan den checkas in?
- Använder ni er av konceptet infrastruktur som kod (Infrastructure as Code) för att automatisera konfigurationen av infrastruktur?
- Använder ni er av konceptet beteendedriven systemadministration (Behaviour Driven Operations), dvs. beteendebeskrivningar för hur infrastruktur ska konfigureras, liknande BDD för källkod?

Mätning, övervakning och återkoppling

- Hur arbetar företaget med mätning och optimering av själva utvecklings- och leveransprocessens kvalitet?
- Kan du kortfattat beskriva hur företaget arbetar med mätning och återkoppling av produktens kvalitet och prestanda?
    - Vilken typ av data samlas in och analyseras gällande produktens kvalitet och prestanda?
    - Vem eller vilka ansvarar för och/eller utför detta arbete?
    - I vilket ändamål och på vilket sätt utnyttjas denna typ av data?
- Kan du kortfattat beskriva hur företaget arbetar med mätning och återkoppling av användarupplevelser?
    - Vilken typ av data samlas in och analyseras gällande användarbeteende?
    - Vem eller vilka ansvarar för och/eller utför detta arbete?
    - I vilket ändamål och på vilket sätt utnyttjas denna typ av data?
- Kan du kortfattat berätta hur ni resonerar kring och går till väga med defektprioritering?
    - Hur bedömer ni huruvida en defekt bör anses som kritisk eller inte?
- Använder ni er av mörklagd produktlansering, det vill säga produktionsdriftsatta men för användaren dolda funktioner?
- Hur säkerställer ni att nya funktioner som släpps mottas av användare med det genomslag ni önskar (exempelvis gradvis utrullning och A/B-testning)?

B2 Intervjuguide - Engelska

Background

- What is your job role?
- How long have you had that role?
- Do you have any other background within the company?
- What is your other background prior to the company?
- What are your main responsibilities within the company?
- What are your typical day to day activities?

Organizational Structure and Culture

- Can you briefly describe the company's organizational structure for product development?
- Which department and/or which team are you part of?
- What other roles are part of your team/department?
- What roles would you say are involved in the quality assurance of the product?
- What roles would you say are involved in the quality assurance of the process?
- How involved in quality assurance are the team roles that are not formally dedicated towards quality assurance – e.g. development and operations?
    - In what way do you collaborate?
        - Tools
        - Ways of working
        - Communication
- Are there any specifically defined goals and/or incentives for quality improvement on:
    - Individual level
    - Team level
    - Organizational level
- Does the company have some kind of a reward system (monetary and/or non-monetary)?
    - Which roles are included in that?
    - What aspects are the reward based upon?
- Would you say that the company encourages experimentation and innovation?
- How do you learn from, and share knowledge about, success stories and failures?
- How do you experience QA's influence on decisions regarding software design and architecture?
- How do you experience QA's influence on decisions regarding whether or not a feature is ready for release or not?

Continuity and Automation

- Can you briefly describe how the company works with quality assurance of new feature?
- What types of tests do you run, when and how often are they run?
    - Who or whom are responsible for creating those tests?
    - Who or whom are responsible for executing those tests?
- At large, how many of these tests are automated?
    - Which are not automated?
- Which method(s) and practices for software development does the company use?
- Are there any specified quality requirements that need to be considered in development and quality assurance of the product?
- Is there any peer review of the code before it's committed?
- Do you use the concept of Infrastructure as Code for automating the configuration of infrastructure?
- Do you use the concept of Behaviour Driven Operations, i.e. writing behavioural descriptions for infrastructure configuration, similar to BDD for source code?

Measurement, Monitoring and Feedback

- How does the company work with measurement and optimization of the quality in the development and delivery process?
- Can you briefly describe how the company works with measurement and feedback on the product quality and performance?
    - What kind of data is gathered and analyzed regarding product quality and performance?
    - Who or whom are responsible for and/or executes this?
    - To what purpose and in what way is this data used?
- Can you briefly describe how the company works with measurement and feedback on user experience and behaviour?
    - What kind of data is gathered and analyzed regarding user behaviour?
    - Who or whom are responsible for and/or executes this?
    - To what purpose and in what way is this data used?
- Can you briefly describe how you reason around and carry out defect prioritization?
    - How do you assess whether a defect should be considered as critical or not?
- Do you use dark launching, i.e. features are deployed in production but hidden from users?
- How do you assure that newly released features are embraced by the users with the kind of impact/uptake that you wish?

B3 Transkriberingsprotokoll - P1

Företag: Spotify
Intervjuperson: P1
Arbetsroll: Team Lead Quality Assistance
Tid och plats: Torsdag 21 april 2016, kl 13:00-14:00, Google Hangouts

(S) - Samarbetspräglad organisationsstruktur och kultur = Grön
(K) - Kontinuitet och automation = Blå
(M) - Mätning, övervakning och återkoppling = Röd

Rad	*Frågor och svar*	*Kategori*
1	**Vilken roll har du och hur länge har du arbetat med den rollen?**	
2	Min roll idag är som så kallad Chapter Lead på Spotify. Ett Chapter är ett litet team som jag har linjeansvar för, d.v.s. personalansvarsrollen, och vi samlar våra Chapters runt olika discipliner som i vårt fall då är QA. Andra exempel på Chapters är Web Chapter, Package Chapter, Mobile Chapter och så vidare. Vi har valt att organisera oss på det sättet. Så det är min roll. Min roll är *inte* att vara testledare eller liknande. Utan det är just personalansvarsbiten som personlig utveckling, lönesamtal och allt sånt där liksom. Samt att då vara en delaktig QA-resurs i ett utvecklingsteam också, i den mån jag hinner och kan.	
3	**Har du alltid jobbat med den rollen på Spotify?**	
4	Jag började på Spotify för fem år sen och då var jag en testare i teamet som då var ett team med tio testare ungefär. Sen dess har vi vuxit ungefär tio gånger så många gånger på Spotify, och vi har väl ungefär tio till QA-team idag. Sen fyra år tillbaka är Lead för ett av de teamen. Så jag jobbade först ett år som testare och sen fyra år i den rollen jag har idag.	
5	**Vad har du för annan bakgrund innan Spotify?**	
6	Jag började jobba inom IT-världen fem år innan det ungefär. Jag har jobbat som testare under dessa år, både på produktbolag och på konsultbolag. Huvudsak för mig tidigare har varit mer frontend-testning eller slutanvändarnära testning så att säga. Mer åt att sitta och klicka runt i webbsidor och program från ett slutanvändarperspektiv. Vilket är lite intressant med tanke på att det jag jobbar med idag är Payments-bitarna på Spotify, så det är mer backend-tungt. Det innebär mer att kolla så att webbsidan är uppe och snurrar och att våra system interagerar bra med betalningslösningar och alla såna där saker. Så det är lite skillnad mot vad jag själv jobbade med när jag var testare.	
7	**Så man kan säga att du är mer av en personalchef över testare och QA-personal.**	
8	Precis. Vi jobbar ju som en matrisorganisation på så sätt, med Chapters och Squads som vi kallar det. Chapters är då QA, mobil, webb eller liknande, och Squads är utvecklingsteamen. Tillsammans bildar dessa en Tribe, och en Tribe består av ungefär 50 personer. Det finns Tribes som har 20 personer och det finns Tribes som har 100, men ungefär 50 är det normala. Då är jag personalansvarig för alla som vi kallar QA här, men alla dem sitter i var sin egen Squad, och det är där de gör 80-90 % av det dagliga arbetet, som då är QA inom ett utvecklingsteam.	S
9	**Så det finns alltså folk som har dedikerade QA-roller.**	
10	Precis, Vi har dedikerade QA-roller. Sen har vi definierat vad en QA är enligt vår definition. Exempelvis, vi använder inte begreppet Quality Assurance, vi kallar det Quality Assistance. För att just ordet “assurance” är lite knepigt. Är quality assurance ens någonting man ska försöka göra? Man kan ju inte “assure quality” - det är en holistisk sak som måste ägas av alla i teamet. Däremot har vi sett ett värde i att ha folk som fokuserar på kvalitet, på samma sätt som att man har Agile Coaches och Scrum Masters och liknande som fokuserar på processer. En Agile Coach äger liksom inte att teamet levererar, men däremot är det de som fokuserar mest på det och har intresse och kunskap om det. Därför har vi QA på det här sättet. Men det är definitivt en intern diskussion som pågår. Det finns vissa Tribes som inte har QA alls, och det finns dom som har mer av vad vi skulle kunna se som traditionella testare. Vi ligger nog lite mitt emellan i vårt team.	S

11	**Säger du på sätt och vis att kvalitetssäkringen genomförs på olika sätt beroende på vilken Tribe eller Squad man sitter i?**	
12	Det finns en generell sak att säga om Spotify: The Squad is responsible for the quality that they produce. Jag försöker hamra in det när jag får chans. Det är lite av ett kärnbudskap som innebär att ni som utvecklingsteam bär ansvaret för er kvalitet, sen hur ni löser det är väldigt mycket upp till er. Det finns ett värde i det. På samma sätt som att ett utvecklingsteam är ansvarigt för sin egen process, men att det kan ha en Agile Coach som hjälper till med att coacha processen. På samma sätt tror jag det finns nytta i att ha någon som hjälper till med kvalitetstänk, för det är lite att tänka på. Men det är fortfarande teamet i helhet som äger det. Det är inte heller bara någonting som påtrycks ovanifrån, utan det är också en chans för teamet att tänka på såna här saker. Det är upp till er att komma på hur ni ska göra det här. Det är inte något test-team någon annanstans som korrekturläser era saker och sen får ni tillbaka resultatet efter en månad. Ni äger det här, ni behöver komma fram till vad som funkar för er, vad som är intressant i er kontext, vad kvalitet är för er, och vad för kvalitet som är intressant för er som utvecklingsteam. Det är sådana saker som en QA-person kan hjälpa med att coacha. Sen i praktiken tar ju QA mycket av det jobb som någon testare annars kunde ha gjort någon annanstans: automatisering av tester, sitta och köra manuella tester, hålla i regressionstestning inför en release och så vidare. Men det är inte det enda QA gör. Det gäller att ha bang for the buck, alltså att anpassa oss till vad varje team behöver mest. Det kan handla om att få det här teamet att försöka förstå vilka stakeholders dom ens har, varför teamet finns och varför de leverar de saker dem gör. Det kanske är det som bidrar till kvalitet, då är det det QA fokuserar på. Eller så kan det handla om att sätta upp en automatiserad regressionstest-stream varje gång vi gör en release, då fokuserar man på det. Så QA är ganska brett.	S
13	**Vilka roller är det som typiskt ingår i en Squad?**	
14	I en Squad så har man en produktägare. Sen har man någon sorts Agile Coach, i vissa fall kanske någon jobbar heltid med det, eller så delar man Agile Coach med fyra andra team eller liknande. Vår approach är att man har en QA i varje Squad, men just nu har vi inte det, vi behöver anställa ikapp. Sen ingår även den uppsättning av utvecklare som behövs för att lösa problemet. Det finns alltid en diskussion kring hur specialiserade utvecklare som behövs. Det vill säga, behöver teamet IOS-utvecklare, Android-utvecklare, backend-utvecklare eller utvecklare som är fokuserade på data? Vi har ju några miljoner användare och massa transaktioner varje månad så det blir mycket data. Då kan det behövas analytiker och liknande. Men det beror helt på vad Squaden har att lösa. I de Squads som jobbar med klienterna håller utvecklarna på med mer frontend-nära saker. Däremot kan det finns andra Squads som bara bygger de tunnlar som vi skickar användardata genom, till exempelvis Finance så att de kan analysera vilka vi skickar pengar till och vilka vi får pengar av. Då är det ju ren backend-saker. Men grundkärnan i varje Squad kan man säga är en uppsättning utvecklare av olika smak beroende på vad Squaden gör, en produktägare, någon sorts Agile Coach, och en QA. Det är vår grundförutsättning. Designers är också roller som ingår i vissa Squads.	S
15	**Det verkar ju vara ganska mycket kollektivt ansvar för teamet som helhet. Finns det någon typ av belöning eller incitament för att ett team ska prestera, får teamet någon typ av recognition så att säga?**	
16	Inte strikt uttalat så att det handlar om monetära belöningar eller liknande. Det är mer klapp på axeln och att vissa team blir inkallade till ett möte och får beröm när de har gjort något bra. Detta handlar ganska mycket om företagsvärderingar och kultur, och det finns både starka för- och nackdelar med att ha teambaserade incitament. Det är ju väldigt svårt med incitamentsstyrning av kvalitet i och med att det bygger på att man mäter. Det gör att folk kanske tänker: “Vi vill få belöning, vilket är lättast - att lösa buggarna eller att aldrig rapportera dom?”. Det gör att folk drar sig för att rapportera dom, så min starka avsikt är att försöka undvika detta. Det ska inte vara belöningar och bonusar som är målet. Det ska istället vara så att: “Tänk så många coola grejer vi gör och så många användare som kommer ha nytta av dem”.	S
17	**Hur upplever du att teamen vågar testa nya saker och experimentera utan att känna att de är rädda för att misslyckas? Hur tacklar ni det?**	
18	Det är en intressant fråga. Vi gör enkäter internt på bolaget för att försöka mäta hur folk upplever att det är att jobba här. Det varierar lite från år till år. Vi frågar: “Do you dare to fail?”. Det är fortfarande en majoritet som känner att det inte är något problem att misslyckas på jobbet. Men fler och fler känner nog lite mer osäkerhet kring det, och det grundar sig nog i hur stort bolaget börjar bli. Ju mindre bolaget är desto bättre känner man varandra. Om någon gör någonting man inte riktigt förstår anledningen till, så kanske man ändå tycker att det är en schysst kille och han vill nog inget illvilligt eller dumt med det. Det ger en känsla av att “ingen kommer se ner på mig om jag misslyckas, de förstår det”. Ju större företaget blir desto svårare blir det att behålla en sån transparens. Det är någonting vi kontinuerligt jobbar med kulturmässigt. Vi kommer misslyckas, och det är okej att misslyckas, så länge man lär sig någonting av det. Faktum är att i jämförelse med Google och Amazon så är vi ju	S

	småttingar, och om vi ska kunna vara snabbare än Google då måste vi kunna chansa mer. Sen handlar det mycket om: "Vad är ett misslyckande och inte?". Att lansera en feature som kanske inte får det uptake som vi tror att det ska få är inte ett misslyckande. Det är en bra sak, då vet vi ju att folk gillar inte den här grejen. Vi provar alltid saker. När vi tittar på och ändrar utvecklingsprocesser så försöker vi undvika att planera för mycket. Det är mer såhär att vi byter i en månad och testar, sen analyserar vi det och ser vad som hände. Det kan aldrig bli ett misslyckande, däremot kanske vi lärde oss att det här ska vi inte göra igen. Det handlar om att bygga den typen av kultur och det är knepigt, men viktigt.	
19	**Hur gör ni för att dra lärdom kring det här och dela denna kunskapen?**	
20	Att kunna analysera något bygger på att man har data som man kan analysera. Sen i slutänden blir det alltid någon slags magkänsla som spelar roll utöver den analyserade datan. Att samla in data är emellertid ohyggligt viktigt. Det är någonting vi blir mer och mer strikta med. Om Daniel och Martin, våra grundare, frågar vad vi gör och varför vi gör det så undrar de: "Bygger ni det här på magkänsla eller har ni data på det?". När det gäller slutanvändarprodukten handlar det väldigt mycket om att analysera mönster kring hur folk använder den. Att logga allt. Då kan man se: "Om vi har den här presentationen så tar det folk 13 sekunder att bestämma sig för att köpa, om vi däremot har den här andra så tar det 6 sekunder". På de 13 sekunderna hinner folk hamna i telefonsamtal, tröttna eller ångra sig. Att analysera sådana saker är viktigt, för då kan man prova saker och sedan faktiskt kolla på resultatet. Sen är det inte lika enkelt att applicera detta på våra egna processer. Hur vet till exempel våra utecklingsteam att de blir mer effektiva med Kanban istället för Scrum? Det är svårt att mäta. Att vi får flera användare är en liten del av det som visar att vi gör något bra, det gäller dock att ständigt hitta nya saker att mäta. I slutet av dagen är det vi mäter kanske någon sorts fuzzy sak, men vi måste gå efter någonting. Det gäller att kontinuerligt tänka: "Hur kan vi veta om vi faktiskt lyckas eller inte med en förändring?".	S, M
21	**Ni mäter alltså slutanvändarnas upplevelse, och om den är bra är det ett resultat av att ni har en välfungerande utvecklingsprocess. Finns det någon annan dedikerad typ av mätning, exempelvis ledtider för själva utvecklingsprocessen?**	
22	Om man ser till övergripande saker som vi mäter på bolaget är det hur många användare vi har, hur mycket tid de spenderar i tjänsten, hur många av våra betalande användare vi förlorar varje månad, sådana saker. På en lägre nivå som i ett enskilt utvecklingsteam bland alla 100 team på Spotify är det omöjligt att veta vilken påverkan just det enskilda teamet har. Jag jobbar just nu mycket med kundsupport-sidan, där går det inte att mäta vår just vår inverkan på hur många slutanvändare tjänsten har. Då kan det istället handla om att mäta ledtider. Utvecklingsteam som jobbar med att effektivisera utvecklingen av klientbyggen och liknande tittar mer på siffror kring hur fort det går att introducera en ny feature eller hur fort det går att fixa en bugg. Där mäter man hur lång tid det tar från att ha börjat skriva kod tills det att man faktiskt ser resultatet av koden i feedbackloopen. Det går att exempelvis att mäta hur länge man behöver vänta på kompilering och hur lång tid det tar för byggmiljön att bygga. Hur får vi till exempel tidsåtgången för den automatiska testsviten, som alltid körs vid varje incheckning, att vara så kort som möjligt? Det är någonting som utvecklingsteamen mäter.	M
23	**Hur upplever du att QAs inflytande över designbeslut är, har ni mycket input där?**	
24	Vi har Design väldigt nära, men vi samarbetar inte så bra som vi borde. Det är någonting vi funderar på ofta. Det skulle vara bättre om vi kunde ha mer input till dem, och vice versa. [---]. Ett typexempel är att vi ofta får specifika designskisser från Design på hur något ska se ut, så att vi ska ha något konkret att testa mot. Att däremot komma med input till Design och säga: "Det här kommer inte att funka tillsammans med resten av klienten", eller "Vi vet att våra slutanvändare blir förvirrade av det här" hade vi kunnat bli bättre på. Då är kanske vi på QA de personerna som borde hjälpa dem att se det. Så överlag skulle jag säga att vi har bra möjligheter att ha inflytande över designbeslut, och vi sitter väldigt nära Design, men jag tror de facto inte att vi utnyttjar det som vi skulle kunna göra. Design och QA är roller som bör sitta bredvid varandra och prata mycket.	S
25	**Tänker du på UX-design nu, eller mer på design som i kodstruktur och arkitektur?**	
26	Ah, okej, du menar så. De som kallas Design på Spotify håller på med UX-design, det var dem jag tänkte på. Men du har helt rätt att det är en begränsande sak att enbart kalla det design.	
27	**Nejdå, inte alls, jag försökte bara förstå vad du syftade till.**	
28	Jaja, absolut. När det gäller systemdesign och liknande är det någonting vi är bättre på att vara närvarande vid. Vi sitter bredvid utvecklarna och en del av vår uppgift är ju att tänka på att just testabilitet och liknande. Det är definitivt något vi har inflytande över. Så fort det finns ett state (tillstånd) som kan förändras vill jag på något sätt kunna överrida det så att jag kan testa det härifrån. Extremfallet är att det får inte vara så att jag måste åka till Tyskland och köpa ett telefonabonnemang för att kunna testa hur en feature fungerar på mobiler i Tyskland. Jag måste på något sätt kunna sitta på kontoret och simulera de här sakerna, och det handlar om att bygga in testbarhet och det är typiskt sådana de-	S

	signbeslut vi påverkar. [---] vi måste tänka på sådant. Det fungerar just eftersom att vi sitter så pass nära utvecklarna, och också att vi har en ganska decentraliserad systemdesignprocess hos oss - det är inte någon Chief Architect som sitter någonstans långt bort och bestämmer, utan här har vi koll och det är i våra team som sådana här saker bestäms.	
29	**På tal om inflytande, hur är QAs inflytandet över releasebeslut? Till exempel bedömning huruvida någonting är klart för release, eller om det är för mycket defekter, eller om ni kan släppa ändå?**	
30	Det är en intressant fråga, för den är lika mycket teknisk som den handlar om psykologi och socialt samarbete. En sak man vill undvika som QA är att vara gatekeeper eller polis på så sätt att alla måste komma till mig och fråga om det är okej att släppa eller inte. För då hamnar man in en svår situation. QA kan inte ta de besluten själva, dels för att vi inte har all input som behövs från business-sidan. Så det är produktägaren som får ta releasebeslutet, det är ändå produktägaren som vet mest och bäst om det. Det är definitivt så att QA ska försöka ha stor impact och stort inflytande över releasebeslut, men QA ska inte vara de som bestämmer saker. Här gäller det även att utbyta sina insikter **[---]**. Att ta sådana beslut är liksom, vissa älskar det och tycker att det är skitkul, andra avskyr det. Det är så att hela teamet gemensamt äger kvaliteten och äger således gemensamt beslutet kring att släppa eller inte. Om någon från QA kommer in väldigt strikt och säger ja eller nej, så lär sig antagligen inte resten av teamet att ta det ansvaret. Jag tycker att QA har ett visst ansvar i att lära teamet att inse saker. Om jag som QA tror att det är en dum idé att släppa nu, men teamet vill släppa ändå, och Spotify sedan går ner i tre dagar… Så kan jag fortfarande inte låta dem göra det, för min egen skull. I det här fallet överdriver jag lite. Men lite utbildningsmässigt kan man göra så ibland för att flina lite och säga: “Ja, titta vad som hände. Jag yrkade i hela teamet att vi borde göra såhär och såhär - nu gjorde vi inte det och här är konsekvensen, nu analyserar vi vad som hände så att vi kan lära oss av detta för framtiden”. Det finns ingen strikt process som säger att QA måste ge klartecken till en release, utan det är Squaden som gör det. Däremot inser ju folk generellt sett att QA ibland är bra att ha i ryggen.	S
31	**Okej, jag förstår. Jag kan tänka mig att ni baserar ganska många av releasebesluten på någon slags datadriven impact-analys, att ni bedömer vilken påverkan det kommer ha och hur kritisk en defekt faktiskt är för slutanvändarna?**	
32	Om man tar sådana saker som kända buggar. Där finns ofta en större möjlighet att mäta hur många av användarna som kommer påverkas av det, hur stor impact det har. Är det så att de inte kan starta klienten så är det katastrofalt för varje individ, är det däremot så att några pixlar förskjuts så att det ser fel ut är det en mindre impact. Man måste bedöma hur stort problem det är för den övergripande kvaliteten, det är inte svart och vitt. Visst, det är fortfarande dåligt - det ser inte bra ut om det är fult, men det är inget stort problem [---]. Så i de fallen är det oftast lättare att göra en analys och bedöma hur stor impact en känd bugg har. Det som är knepigast att analysera är när vi har en ny feature som vi vill släppa, som vi tror är en bra idé och att folk kommer använda. Då måste vi bedöma hur stor den är, hur färdig den måste vara innan den går ut i jämförelse med andra features, samt vad systemrisken är att vi bryter någonting annat så att det inte fungerar längre. [---]. Det är svårt att säga, men i de fall där det går att ta beslut datadrivet så gör vi det, då vet vi ju om det är ett problem eller inte. Så stora som vi är idag, så är det ju så att om vi har 10 000 kunder som lider av ett visst problem, så är det ju dumt. Men jämfört med att vi har knappt 100 miljoner användare i månaden så kanske det finns bättre saker för oss att ägna oss åt. Men man får heller inte köra över de där 10 000 användarna heller, man ska ju vara schysst mot dem för de har ju betalat pengar. Då kanske de inte heller kommer rekommendera tjänsten. Man får göra en värdering där, men ju mer data vi har desto enklare blir ju den värderingen. Då kan man se vad som är rätt sak att göra här.	M
33	**Nu har vi pratat lite om kultur, beslut och inflytande, organisering och struktur, och så vidare. Vi tänkte gå in lite närmare på testning. Kan du börja med att kortfattat beskriva hur ni arbetar med testning av nya funktioner eller features?**	
34	Som sagt, det är inte QA som äger kvaliteten på det sättet. Men likväl så finns det ju oftare en ganska naturlig uppdelning av när, var och på vilket sätt testerna körs. Enhetstester körs ofta för att se om koden är okej i sig, men man kan egentligen aldrig bevisa att businesslogiken fungerar som den ska med enhetstester. Enhetstester är ju väldigt lämpligt av praktiska och tekniska skälet att utvecklaren skriver såna själv i samband med att de jobbar med TDD, känner ni till det?	K
35	**Jo, absolut.**	
36	Så det är ju en typiskt sak som är lätt, att skriva sina egna tester och sen låta koden passera testerna tills den faktiskt uppfyller dess syfte. Enhetstesterna skrivs ju förhoppningsvis i samband med koden, så de är ju incheckade **[---]**. Så enhetstester är sådant som ligger på utvecklarna, vi förväntar oss att de ska göra det. Vi på QA kan ju komma in och coacha utvecklarna och säga att det är bra att göra enhetstester, även om utvecklaren tycker det är jobbigt och inte ser värdet av det just då. **[---]**. Enhetstester är ett fantastiskt bra sätt att dokumentera vad koden gör och varför den gör det, så det har	S, K

	ju ett värde ur en dokumentationssynpunkt. Där kan vi på QA komma in och coacha. Tittar man exempelvis på helhetsöversikt och systemtester så är det kanske något vi på QA jobbar mer med. Vi håller mer på med exploratory-based testning, som exempelvis kan vara: "Okej, jag som användare försöker gå igenom ett flöde här, sen så ändrar jag mig mitt i, vad händer då?" **[---]**. Sen kan vi göra slumpartade automatiserade tester som kommer täcka liknande saker, men det kommer inte att byta ut en mänsklig hjärna som kan sitta och fundera på: "Vänta nu här, det här var knasigt, det här är jag intresserad av att titta på mer". Det finns en mogul i USA som heter James Bach som pratar om Sapiens Testing, alltså sapiens som i homo sapiens, tänkande testning. Det är vad man skulle kunna kalla det. **[---]**. För att komma tillbaka till frågan, det här är olika beroende på vad teamet utvecklar. Har teamet mycket backend-system som vi har, så kanske man inte har någon frontend att testa i. Då har vi mockup-klienter som vi använder för att försöka testa så att vi har ett bra API. Man behöver ju inte testa med det riktiga systemet som egentligen konsumerar APIt. Då kan man skicka in vilken knasig data som helst och se hur vårt system reagerar på det. Så det är ju fortfarande Sapiens Testing, det är en mänsklig hjärna som funderar på vad som kommer hända, men det kräver ju att man bygger ett verktyg för att göra det här. Sen krävs såklart testautomatisering vid incheckningar och liknande, som ser till att passera alla enhetstester, vissa integrationstester, vissa deploytester för att se exempelvis hur det går att deploya mot flera miljöer samtidigt **[---]**.	
37	**Okej. Då är jag med. De här explorativa testerna du pratade om, skrivs de ner eller definieras de på något sätt, eller inte alls?**	
38	Hos oss idag så är det ganska lite att de skrivs ner, men vi använder oss av något, som i alla fall tidigare kallats "[---]", där det skrivs ner tips och tricks helt enkelt som vi delar mellan oss QA. Vi jobbar både med enskilda saker med våra egna utvecklingsteam men ändå också i stort sett med samma system. Det är exempelvis viktigt att köpa Spotify eller säga upp Spotify på liknande sätt. De systemen är liknande, så om någon sitter i ett team och har byggt ett verktyg som gör att vi kan [---] 40 användare samtidigt från en kalender, ja men bra, sprid då den kunskapen! Någon annan kan behöva ungefär samma. Men vi skriver inte ner, vi kör till exempel inte strikt sessionsbaserad testning. På tidigare ställen innan jag började jobba på Spotify så jobbade vi på lite andra sätt, då bestämde vi till exempel att i 45 minuter ska du sitta och testa det här och ha klart för dig att du skriver ner det du testat och sedan diskutera med en kollega bredvid. Jag tror det här delvis kommer från att vi sitter som ensamma QA i våra team, speciellt om du sen hittar något nyttigt att dela med dig av. I utsträckning så är ju explorativ testning naturligt att ha i då det finns några kunskapsansvariga.	S, K
39	**Okej, men om man tittar på andra typer av tester som enhetstestning, integrationstestning och prestandatester som kan automatiseras, skrivs de ner eller utformas mer?**	
40	I stor utsträckning har utvecklarna varit väldigt bra på att tänka på att göra det här själva, de har inte haft så mycket hjälp av QA tidigare. Mycket av det här är som sagt gjort mycket av utvecklarna själva, däremot huruvida eller när de körs eller liknande; är det till exempel något som vi kontinuerligt kör varje deploy? Är det någonting som man bara får för sig när det verkar gå segt? Ja, men varför då? Varför hänger databasen alltid ihop sig? Vi kanske borde versionstesta för att se var vi är? Det ligger för ad-hoc här och det gör att QA ska visa värdet av kvalitet. Tänk om vi inser det här när vi står där en dag och inser att databasen är problematisk för nu försöker 100 000 kunder signa upp, det är inte så bra. Det handlar mer om att utbilda värdet, så låt oss nu göra det här kontinuerligt och på ett sätt som är enkelt att göra. Men om det är en QA eller en utvecklare som gör det diffar lite mellan utvecklingsteam och utvecklingsteam. Men det är viktigt att göra.	S, K
41	**Okej jag förstår. Generellt sett, hur stor andel av testerna överlag är automatiserade och inte? Eller rättare sagt, vilka är?**	
42	En svår fråga, för vad definierar ett test eller inte? Om en utbildad duktig QA sitter och testar någonting i två timmar så kan du kalla det för ett test eller för tusen test. Man kanske kan prata om hur mycket "effort" som ligger i det. Det är en annan fördel med automattester, om du gör rätt så är effort ett (1); du sätter dig ner och skriver det här en gång och sen förhoppningsvis behöver du aldrig röra det igen. Det kommer köras på varje commit, och det är kanske något vi ska automatisera. Jag kan nog inte ge ett bra svar på hur stor proportion, men jag kan säga att jag önskade att vi hade mer automattester då jag tror det kan finnas ganska mycket "låghängande frukt" som vi skulle vilja kunna se mer av. Du kan använda tester för att dels under utvecklingens gång se om det vi bygger kommer fungera i live-miljön. Ifall du kör livetester kontinuerligt kan du se om vi fortfarande är uppe och snurrar, eller om den senaste commiten du skickade in slog ut all sign up i Australien. Det var kanske för att vi uppdaterade en config-fil som vi inte trodde skulle påverka det området. Då kan man ha automattester som monitorerar saker så att vi får tidigare feedback på det, istället för att customer support ska börja ringa och säga att det är problem i Australien, och om turnaround-tiden ligger på sex timmar så skulle vi istället kunna se direkt med ett automatiskt test att man inte kan signa upp i Australien. Så vad är test på så sätt? Vad är syftet med testet? Är det för att testa feature-utveckling eller för	K, M

	att testa hur vi faktiskt mår i verkligheten?	
43	**Okej. Du nämnde TDD innan, är det en standard eller tillämpas några andra metoder i företaget som stort?**	
44	Företaget som stort är svårt att svara på, vi är för stora, men jag vet att det finns olika förespråkningar för TDD, medan vissa utvecklingsteam kanske kör ganska strikt, men jag kan inte nämna något specifikt. Vi har inga hos oss som gör det, vilket jag tycker är lite synd. En stor friktion mot TDD är att det är svårt att börja jobba med. Det är ganska långt ifrån vad man gör när man börjar programmera själv, då slänger man mest ihop något. Det är ett annat mind set och det är jobbigt att ta sig över den tröskeln. Även att hitta tiden till det är svårt, man kan tycka att man ska bygga en feature istället för att lägga tid på processen. Det finns också BDD, behaviour-driven development, som man försöker lösa liknande problem med; mycket för mig med TDD och BDD handlar om att man tänker till mer före. Det finns två stora nyttigheter som jag kan se med det. Det ena är att du tvingas tänka till över vad det här egentligen ska göra; väldefinierade tester som vi kan ha bra diskussioner om och vad vi egentligen är ute efter samtidigt som man kan tänka: “Vad händer om?”, och: “Vad ska vi göra då?”. Detta springer du annars på först i efterhand när du deployar och då inser problemet. Den andra stora fördelen är att du lämnar efter dig tester som du kan lita på fungerar. Har du skrivit bra TDD så i hög grad så kan man säga att även om du refaktorerar om koden, men alla tester sitter uppe i miljön, så gör alltså den här koden vad den egentligen är ute efter, men uppenbarligen på ett annat sätt då.	K
45	**Okej, jag är med. Men utöver de här testmetoderna ni använder er av, utför ni någon typ av referentgranskning av koden mellan varandra? Någon typ av parprogrammering eller liknande för att försäkra er om att koden faktiskt håller den kvalitet ni önskar?**	
46	All kod som checkas in genomgår en code review av någon annan, ibland fler personer. Vi började med att all kod skulle strikt genomgå en review av någon annan, så i början var det ganska mycket formatering och sådana saker, vilket inte alls egentligen var det vi var ute efter. Vi drev en bra diskussion när vi ville införa mer automatiska checkar som kollar att du skrivit koden enligt den bakgrund som du skulle skriva. Det är för att göra den läslig för någon annan, men vi gick över mer till att diskutera designprinciper och se om något kommer fungera eller om något har glömts bort. De här kodgenomgångarna görs av vem som helst i teamet, så det är inte något QA nödvändigtvis måste göra, men många vill gärna be QA göra det och QA gör det gärna, men det är inte en bottleneck som säger att det måste gå via dem. Det skulle leda till att det blev en roll folk skulle förlita sig för mycket på, typ: “Jaja, men det där kommer QA ta hand om”, men det skulle också ta för mycket tid. Men att koden ändå ska genomgå code review är annars en ganska strikt regel.	K
47	**Ett sätt att “få det på köpet” är ju parprogrammering. Är det någon som använder det, eller inte alls?**	
48	Jag skulle önska att fler utförde parprogrammering. Jag vet att när vi partestar saker så är det väldigt givande. Däremot kan parprogrammering för mig inte ersätta en code review. Däremot löser det liknande problem. Det är en risk att tänka att de gör samma sak. Du blir ju hemmablind när du sitter och tänker på ett program i 45 minuter, och kommer det då in någon annan som inte fastnat i din tankebana och tittar på det kan de se var man tänkt fel. Som par kommer man tänka i samma bana. Men ja, det är bra för att dela kunskap. Chansen för att din kamrat bredvid faktiskt kan fixa en bug när du är hemma sjuk är betydligt mycket större om den sett koden innan. Det blir lättare att skriva kommentarer i koden, då du själv kan tycka att det är uppenbart vad din kod betyder. Har du någon annan som tittar på det samtidigt skulle den kunna säga att: “Det här skulle inte jag förstå”, och skriver då in en kommentar där. Det händer att vi utför parprogrammering, ofta kanske på initiativ bara att: “Nu ska vi köra parprogrammering!”, och så gör vi det i en vecka. Det handlar ju också om att ha förtroende att skriva med andra; “Don’t fear to fail”. Det är okej att misslyckas. Det kan även ibland också kännas som ett missförstånd att slå flest tangenter i timmen, det är väldigt sällan det är begränsningen för kompetensen att skapa bra saker. Vill ni veta exakt vilken kod ni själva skrev skulle ni kunna ta fem minuter var och komma fram till vad ni ska skriva. Parprogrammering är också är bra till att korta ner tiden till att två ska tro på den skrivna koden.	S, K
49	**Okej, men till era code reviews, används några checklistor som ni utgår ifrån, till exempel en Definition of Done? Används någon definierad kodstandard? Skriver ni icke-funktionella krav eller utgår ni från mer generella kvalitetskrav som man ska ha i åtanke?**	
50	Definition of Done används mer på en story-nivå, vilket är något som tar tre-fyra personer några dagar att bygga. Storyn ska lösa ett problem, så när vet vi att problemet är löst? Det innebär att koden är deployad, att vi har kollat med customer support att vi inte fått massor av klagomål, att vi har dokumenterat det vi borde och sådana saker; Definition of Done är på den nivån. Men i och för sig när jag tänker efter, under code reviews eller liknande kan man kolla på koden och se om det finns några tester på den. I vissa fall kanske en mer formell process: “Vad är rimligt? Vilka tester förväntas läggas in här?”. Så det hade kunnat vara en Definition of Done, att om det inte finns några tester så ska de läggas in.	K

51	**Är det upp till granskaren att komma på det eller finns det nedskrivet?**	
52	Jag känner inte till att det finns nedskrivna guidelines. I vissa fall skulle det säkert hjälpa med en checklista.	
53	**Okej jag förstår. Om vi snabbt går över till nästa punkter, koncept så som "Infrastructure as Code" och "Behaviour-Driven Operations". Du nämnde ju BDD inom utveckling, men använder ni liknande koncept på Ops-sidan?**	
54	Jag känner inte igen något av begreppen faktiskt. Det jag kan nämna kort om Ops är att då jag började för fem år sedan hade vi en operations-avdelning. Då var vi kanske sammanlagt 60-70 personer, medan vi nu är 700. Förr fanns det då 5-10 personer som jobbade med operations som såg till att servrarna snurrade och uppmärksammade och försökte lösa fel, men det är en genomgående trend att, likt DevOps, försöka flytta ut det här arbetet närmre utvecklarna. Jag har varit med om samma resa tidigare, då det största skälet är att det både är friktions- och kostnadsmässigt dumt att ha operations för sig. De kan se att någonting är fel, men inte veta vad. Om du inte själv drabbas av ett problem när din kod inte gör vad den borde göra så bryr du dig mindre. Det är inte du som väcks mitt i natten när servern är nere, men det är det man nu löser med DevOps. Det är inte en separat Ops-avdelning som försöker lösa problemet, det är lättare då vi i teamet vet vad som deployades och vilka förändringar vi gjorde då vi är närmre problemet.	S, K
55	**Vi tänkte gå in lite snabbt på det vi kallar för mätning, övervakning och feedback. Det handlar ju mycket om insamling av data vilket du tidigare nämnt, så vi kan försöka koppla tillbaka till hur ni samlar in användardata och varför ni gör det. Berätta lite kort om hur ni går tillväga.**	
56	Det finns olika sorters data. En sorts data kan vara click through-rate på vår sida, en annan hur länge användarna är inne och en tredje vart de kommer ifrån. Det svåra är att hitta vad man vill ha i den. Sen har du slutanvändardata, det är ju användartester då man sätter sig ner och kollar vilka som försöker använda vår sida och då kan se vilka som använder den. Det är inga stora datamängder men fortfarande information. Sen kan vi också prata ledtider. Då försöker vi samla vår utvecklingstid vilket kan vara så enkelt som ett streck på en white board för varje dag vi arbetar på en story, så kan vi se att vi ligger till exempel på 7,2 dagar i snitt på en story. Det kan ge indikationer på om det går långsamt och vad vi då kan göra för att förhindra det. Vi monitorerar även att våra servrar är uppe och snurrar, det sitter ganska många stora skärmar på kontroret som visar statistik om till exempel CPU- och databasbelastning i form av spikes och så vidare. [---] Sen får vi inte hantera kontokort, själva siffrorna i sig, så det är specialiserade bolag som gör det åt oss, "payment providers" som det kallas. De hjälper oss dock genom att skicka mail och alerts om till exempel antal köp av Spotify går ner då de historiskt sett vet vad som är rimligt en tisdag klockan två. Då kan vi ta reda på om det grundar sig i något vi känner till, eller om det är ett nytt problem. Vi får hjälp av många tjänster på det sättet, av underleverantörer.	M
57	**Vem är det som ansvarar för att hamstra den här datan och göra något med den sen?**	
58	Det beror lite på. Ansvaret ligger hos den som är intresserad, det kan vara till exempel ett team som är fokuserad på retention; till exempel hur många procent av de som har en månadsprenumeration som fortsätter varje månad. Om vi är mest intresserade av det är det vårat ansvar att få tag i den datan. Rent tekniskt hur det går till så har vi en ganska stor del av [---] department som kallas IO som just jobbar med intern tooling av monitorering och liknande, så alla utvecklingsteam inte själva måste bygga från scratch. "Vi vill ha den här sortens data samlad på den här sortens databas, samt kunna presentera på den här sortens dashboard", då finns det mer eller mindre förberett helt enkelt. Men vill vi som är intresserade av vissa saker måste vi se till att logga rätt data.	M
59	**Då används datan sen till att ta beslut gällande förändringar, eller leda till nya stories eller feature requests?**	
60	Absolut. Det handlar lika mycket om att döda features som man inte ska hålla vid liv för att inte slösa pengar. Men också om man upptäcker till exempel en stor betalkanal i ett land så kan vi inse att vi borde jobba mer på den. Jag gissar att ni har använt Spotify själva? Då kanske ni känner till veckans tips, Discover Weekly. Det var först en skämtfunktion som sen lanserades till alla anställda vilket vi tyckte om, och senare släpptes till resten av världen. Det var en sak som vi lärde oss av att samla in data, då det blev mycket större än vi trodde. Våra back-end system orkade inte riktigt med det här, att lansera en ny playlist med 30 nya låtar till 100 miljoner användare är ett ganska stort jobb. Då fick teamet direkt inse att om vi ska lansera det här till alla användare i världen så måste vi rikta om våra back-end system. Det var ju inte ett misslyckande utan om folk är beroende av det här måste vi se till att det fungerar, och då kan vi med hjälp av data se hur många användare som lyssnar på den här playlisten varje måndag; "Jaha okej, det var sjukt många, intressant!". Det hade varit svårt att uppmärksamma annars, men nu kan vi se det samma dag.	M
61	**Det här handlar ju om experimenterande funktioner, men säg att ni har en funktion som ni faktiskt vill släppa. Hur arbetar ni då för att se till att genomslagskraften eller intresset blir så**	

	stort som ni hade önskat?	
62	En sak vi mäter noga är retention, activation eller liknande. Till exempel om någon signar upp sig, hur stor chans är det att den använder Spotify tre dagar senare? Varje gång vi introducerar en ny feature utgår vi mycket från om den är komplex eller svårförstådd, om den är svår att använda leder till att färre faktiskt stannar kvar. Vi monitorerar alla sådana features just utifrån vilken grad av retention vi har. Då är det oftast till 1% av användarna först och mäter på det sättet. Dels kanske vi får klagomål på att det kraschar eller att något inte fungerar. Vad händer med till exempel activation på de här procenten? Är det så att folk plötsligt har gett upp efter tre dagar? I så fall var det för komplext och då måste vi rita om. Förhoppningsvis har vi fått data utifrån användartester från tio personer som har gjort intervjuer innan, men det är först när man provar det som man vet. Det är något vi använder oss väldigt mycket av, just features som är lanserade till 1% av användarna för att kunna monitorera hur de beter sig och sen ta beslut utefter det. Då kan man sedan gå ut till 100% av användarna för då vet vi, vetenskapligt visat, att det fungerar.	M
63	**Ni rullar alltså ut gradvis till olika grupper av användare, men använder ni samma metodik för att testa och jämföra olika typer av features?**	
64	Vi A/B-testar väldigt mycket. Jag vet inte hur många hundratals A/B-tester vi jobbar med samtidigt. Jag gissar på att om ni båda har uppdaterat Spotify-klienten så kan vi hitta något som inte är likadant i dem. Vi jobbar mycket med det, just för att få det här nära slutkunden, det är först då man verkligen vet. Detta gör att man inte stannar kvar i sin känsla för det här är ju det faktiska testet.	M
65	**Vi pratade förut om hur man bedömer hur kritisk en bug är inför release. Vi tänkte även koppla detta till en fråga om mörklagd produktlansering, även kallat feature toggles. Hur arbetar ni med det?**	
66	Ett A/B-test är ju strikt att jämföra population A med population B och hur de beter sig. Sen kan du bestämma dig för att du vill lansera en viss tjänst, men vi kanske inte vågar rulla ut den till alla användare än, just för risken att det kan leda till för många kraschar, och då kan vi använda oss av feature toggles. Egentligen utför det samma funktion som ett A/B-test, men det handlar mest om syftet bakom det. Du kanske har bestämt dig för att lansera en funktion, men släpper till få kunder sakta men säkert. Om den inte funkar så kanske du råkar krascha alla iPhones, men då kraschar du bara några få istället. Så det är också något vi jobbar mycket med. Vi började nyligen använda Google Cloud-plattformen, och det är något vi fortfarande labbar sakta men säkert med just nu, att rulla ut till fler och fler användare till Google Cloud, men det är inget som slutanvändaren nånsin ser. Musiken strömmas från A eller B, och det är ett ganska stort steg för vi vet att vår back-end rullar bra så vi vill inte lämna den för tidigt, men med tiden vill vi gå över helt till Google Cloud. Vi började med att rulla över 1% av användarna, och även alla anställda. Man kollar först om det funkar och sen kan man dra över användare. Det är ett fantastiskt sätt att kunna göra det på, istället för att tvingas gå över 100% eller inte.	M

B4 Transkriberingsprotokoll - P2

Företag: Spotify
Intervjuperson: P2
Arbetsroll: Quality Assistance Engineer
Tid och plats: Fredag 22 april 2016, kl 13:00-14:00, Google Hangouts

(S) - Samarbetspräglad organisationsstruktur och kultur = Grön
(K) - Kontinuitet och automation = Blå
(M) - Mätning, övervakning och återkoppling = Röd

Rad	*Frågor och svar*	*Kategori*
1	**Vad är din arbetsroll och titel? Vad jobbar du med?**	
2	Vi jobbar ju väldigt lite med titlar på Spotify. Behöver man en titel så pratar man med sin anställande chef om det vid det tillfället, till exempel om man ska på en konferens eller liknande där det kan vara nödvändigt för att kommunicera utåt. Jag är ingenjör på Spotify, eller så väljer jag att se det i alla fall. Ingenjör med inriktning på kvalitet.	
3	**Hur länge har du jobbat med just den inriktningen på Spotify?**	
4	I tre år.	
5	**Du jobbar alltså med kvalitet. Kan du förklara några typiska aktiviteter eller ansvarsområden?**	
6	Jag ser det som att vi försöker sträva efter ett pragmatiskt och agilt sätt att jobba på, men jag tycker inte om ordet agilt för det har fyllts med så mycket dåliga saker under de åren som det har varit ett buzzword. Vi försöker vara en lättrörlig och pragmatisk tech-organisation. Vi är organiserade i squads, och när man försöker arbeta med kvalitet på det sättet som jag försöker arbeta på så behöver man vara generalist, och man behöver kunna fylla väldigt många olika roller, men även ha en infallsvinkel som hela tiden ställer frågor om “fit for use”. När jag pratar om kvalitet, och framför allt när det gäller mjukvara och intern kvalitet, så brukar jag använda uttrycket “fit for use” istället för kvalitet. Kvalitet är svårt att fylla med innebörd, då det ofta blir en väldigt subjektiv tolkning, och så är det svårt att ha det när vi pratar om mjukvara och om att leverera värde. Vi måste sätta oss ner tillsammans och ta allt från business-organisationen till ingenjörerna och verkligen titta på vad det är vi försöker lösa just nu, och vad som står på önskelistan. Tickar vi alla de boxarna och skeppar någonting, även om vi har utestående krav eller om vi har saker som vi vet är trasiga så kan vi fortfarande leverera med kvalitet. Kvalitet eller “fit for use” är någonting som vi som ingenjörsorganisation tillsammans bygger upp. Det är enda sättet vi kan skapa oss något som är kvantifierbart.	S
7	**Fungerar det som någon typ av checklista inför en release, i likhet med en Definition of Done?**	
8	Det hade vi kunnat jobba med om vi jobbade med release-cyklar, men det gör inte vi. Vi jobbar med rullande releaser, så när vi har någonting där vi kan fastställa om det är “fit for use” eller om det levererar något värde, då kan vi släppa det i produktion. Det är något vi måste hålla löpande hela tiden. Jag jobbar med finansiella system, och det är ett väldigt speciellt sätt att jobba med en kodbas mot ett sådant system.	K
9	**Det är en intressant koppling till mätvärden för kvalitet, men innan vi går in på det, kan du bara beskriva kort hur ditt team ser ut?**	
10	På ett övergripande plan jobbar jag med initiala betalningar, allt i från gränssnittet där du genomför din betalning till integration till en payment gateway.	
11	**Du nämner att du är en ingenjör med inriktning på kvalitet, men skulle du säga att de andra i ditt team då inte delar något ansvar för kvalitet, eller att alla har något typ av kollektivt kvalitetsägarskap?**	
12	Jo men absolut, det är helt omöjligt för en person eller till och med ett utvecklingsteam att äga kvalitet. Det beror också på vad man menar med att äga kvalitet, men i och med att teamet auto-	S

	nomt inte kan sätta kraven för kvalitet eller “fit for use” så kan man helt enkelt inte äga kvalitet. Det behöver expanderas, det är helt omöjligt att en individ kan äga kvalitet.	
13	**Men har ni dedikerade QA-roller?**	
14	Vi kallar det inte Quality Assurance, utan Quality Assistance. Det kan man tycka vad man vill om, men den rollen som är dedikerad har andra funktioner än vad en klassisk Quality Assurance-roll har.	S
15	**Kan du ge exempel på några typer av aktiviteter eller ansvar som du tänker på i förhållande till detta?**	
16	Då vi är organiserade på ett sådant sätt att vi äger en kodbas och system, och därmed implementationen, i teamet så kommer de tekniska kraven för någon som håller på med Quality Assistance vara större. De kommer behöva gräva mer själva i koden och förstå vad som behöver utforskas, och även implementera verktyg för att kunna hjälpa ditt team att vara kapabla till att utföra testarbete.	S
17	**Så varje team är ansvariga för den kvalitet som de själva producerar?**	
18	Teamet är ansvarigt för att det värdet som är tänkt att leverera blir levererat. Saker måste vara “fit for use” och de måste vara uppe och snurra.	S
19	**Så kvalitet är en team effort?**	
20	Ja, absolut. Men kvalitet är mer än en team effort, för utan någon som sätter riktningen för affärsinitiativet kommer du hamna i en del konstiga diskussioner efter leverans om du inte inte har dragit den delen av organisationen väldigt nära och se vilket problem vi faktiskt försöker lösa. Kan vi implementera en lösning på det problemet som levererar det värdet på det tilltänkta sättet?	S
21	**När det kommer till designen av kodbasen, alltså arkitekturen, skulle du säga att du som med inriktning på kvalitet har större ansvar än resten av ditt team? Vem eller vilka har inflytande när det kommer till dessa designbeslut?**	
22	Det handlar om att individer i ett team kan använda sig av olika verktyg eller roller. För att sätta upp ett framgångsrikt team så ska man anställa ingenjörer som är duktiga generalister men som också har en spetskompetens. Du har ansvar som individ att ta beslut som påverkar arkitekturen, framför allt när du jobbar i en agil struktur. Annars kommer du bli överkörd och inte kunna sätta upp några spärrar. Om allting måste routas genom en person för att den ska signa av så kommer vi aldrig komma någonstans. Det kommer aldrig fungera om du vill röra dig snabbt.	S
23	**Om vi pratar om inflytande, och inte så mycket ansvar, skulle du då påstå att du som har en kvalitetsinriktning kan ifrågasätta vissa designbeslut utifrån ett kvalitetsperspektiv?**	
24	Absolut. Jag förväntar mig att jag är mer vokal och försöker tränga mig in i sådana diskussioner, men jag har exakt samma förväntningar på alla andra i mitt team. Sitter man och jobbar tillsammans i en grupp om åtta personer så kommer du överhöra väldigt mycket och delta aktivt i informella diskussioner, vilket är otroligt värdefullt. Men jag skulle inte säga att det är något som faller mer på mig i mitt dagliga arbete än på någon annan. Däremot kanske jag mer aktivt söker upp de diskussionerna; till exempel code reviews och arkitekturfrågor. Inte för att jag tror att jag kommer överglänsa någon och ge bäst input, utan för att jag vill suga åt mig av de här diskussionerna. De ger mig bättre möjlighet till att utföra experiment på mjukvara.	S
25	**När du nämner experiment, hur stor möjlighet skulle du säga att ni har till att testa nya saker? Finns det några begränsningar eller möjligheter?**	
26	Vi har extremt goda testmöjligheter. Men precis som alla andra som håller på med betalningar så springer vi in i tredjepartsintegrationer. Där har vi en gateway, en payment service provider, som slussar oss vidare till ett hundratal olika betalmetoder ute i världen. Det blir lite stökigt då det är så många och de håller väldigt olika kvalitet. Däremot allting som ligger under vårat tak har vi väldigt bra miljöer för. Skulle man känna att man saknar någonting så hoppas jag på att vi anställer duktiga ingenjörer som ser problemet och tar tag i det.	S
27	**Hur innovativ tillåts man vara?**	
28	Jag skulle vilja säga att ansvaret ligger väldigt mycket hos individen. Det är ingen som kommer driva dig, även om vi har avsatt tid under vissa perioder där du har möjlighet att experimentera. Alla tar till vara på den tiden på olika sätt, vissa väljer att bara jobba på som vanligt under de här perioderna.	S
29	**När man då testar nya saker, finns det begränsningar i att våga testa nya saker och samtidigt misslyckas? Finns det en sådan mentalitet?**	
30	Den största delen av vårat dagliga arbete handlar om just detta, att göra bort sig. Det som står	S

	högst upp på vår önskelista är att tiden från en idé går från tanke till produktion ska vara så kort som möjligt. Det kommer ju också med ett pris, för ibland går ju då grejer sönder. Men det finns en otroligt stor förståelse från medarbetare om att det är så det fungerar om man vill driva mjukvaruutveckling på det sättet som vi gör.	
31	**Kan du beskriva kortfattat hur ni jobbar med testning?**	
32	Vi jobbar väldigt mycket som coachande individer. Testningen som vi bedriver skiljer sig från om vi testar i grupp för att till exempel sprida kunskap om nya features och förändringar i kodbasen eller om vi testar själva. Det är väldigt stort fokus på explorativ testning när vi arbetar i grupp. De sakerna som vi inte kan automatisera får åka ut så får vi använda data istället. Vi har en data- och statistikdel vilken vi får snabb feedback ifrån, om vi till exempel skickat ut något som verkligen sänker våra KPI:er. Allting från automatisering till sessioner till samtal innan vi börjar bygga saker; driva samtal om vilka problem vi har och se om vi kan kommunicera ut mer tid om att göra kvalitetsförbättringar i den här koden. Som sagt, explorativ testning och mycket automatiserad testning där data kan plockas ut direkt och börja lägga ut den för att se om vi har problem med olika saker.	S, K, M
33	**Vilka typer av tester försöker ni automatisera?**	
34	Allting ifrån enhetstester, integrationstester och end-to-end-tester så långt vi kan driva det. End-to-end-testerna tar oftast stopp om vi vill kunna köra dem kontinuerligt med tredjepartsintegrationer. I övrigt försöker vi hålla en triangelformation på våra tester, så flest enhetstester, sen integrationstester och sen ett tiotal smoke-tester.	K
35	**Vilka är det som bygger dessa tester?**	
36	Det är de som utvecklar funktionen. Det är också en del av code reviewen som blir öppen för hela teamet. I en sådan code review faller det sig naturligt att individer med olika specialiseringar tittar på olika saker, beroende på omfattning.	S, K
37	**Vad använder ni er av för olika utvecklingsmetoder?**	
38	Jag anser utvecklingsmetoder vara ganska mycket ritual och ger ganska lite värde. Jag tror att om man har åtta passionerade ingenjörer tillsammans i en grupp så kommer de sitta med varandra, de kommer sitta isär, de kommer kommunicera och så vidare. Ju längre man låter samma grupp sitta tillsammans, desto mer av dessa synergieffekter kommer man få. Det finns inget uttalat om specifika utvecklingsmetoder, utan vi är väldigt pragmatiska. Är det så att man kör fast så är det bara att prata med varandra, det sker ganska organiskt.	S, K
39	**Har ni några typer av formella kodgenomgångar eller referentgranskningar av kod?**	
40	Absolut, vi har ju code reviews som måste godkännas av två individer från teamet, annars mergear vi inte den koden.	K
41	**Utgår dessa kodgenomgångar från någon lista med punkter man ska kolla efter?**	
42	Alla de sakerna som kan vara nedskrivna automatiseras. Struktur av kod och statisk checkning kan du ju automatisera. Sätter du på mänskliga ögon handlar det om intuition och domänkunskap som du vill åt, annars slösar du tid om det handlar om att leta stavfel eller syntaxfel. Om du automatiserar de här bitarna kan jag då stänga av den delen och då fokusera på det som är mer intiutivt.	K
43	**Om ni bygger en feature som har starkt fokus på ett särskild kvalitetskrav, till exempel säkerhet, hur kommuniceras det kravet till teamet?**	
44	Så som vi jobbar nu faller nog det ansvaret på mig, att vokalisera den biten till teamet. Beroende på hur stor omfattning det rör sig om kan det bli tickets eller inte. Säkerhet är ett väldigt specifikt område, där har vi ett säkerhetsteam som vi tar in i vissa reviews om vi känner att vi behöver några extra ögon. Antingen kommer de och sitter med oss, eller så gör de bara en review.	S, M
45	**Hur fungerar uppsättningen av infrastruktur?**	
46	Vi har en configuration manager som kallas för Puppet, vilket är en uppsättning av Ruby-klasser som går att ärva från och sedan skapar en definition för hur infrastrukturen ser ut.	K
47	**Beskrivs de på ett så att säga tekniskt språk, eller finns det domänspecifika eller talspråksnedskrivna sådana beskrivningar också?**	
48	Nej. Det är ju uppkommenterat, men de här definitionerna är ganska lättlasta skulle jag säga. Jag tycker att Ruby är bra på det sättet. Det som är svårt att hålla kolla på är arven och var saker och ting kommer ifrån. Men det är samma sak som i Chef eller andra liknande konfigureringsramverk.	K
49	**Kvalitetssäkring och testning hör ju ihop ganska mycket med att samla in data, så vi går över lite till det området. Kan du förklara hur ni samlar in data och vad ni använder den till?**	

50	I vårt fall jobbar vi med någonting som är ganska tacksamt på det sättet. Vi sköter om integrationer ut mot olika betalningsmetoder, vi bedriver helt enkelt en checkout-verksamhet. Kunden går in, väljer en produkt och betalar. På det sättet har vi en ganska tydlig tunnel dit men sen förgrenar sig den tunneln ut i flera smågångar, beroende på vilket land du är i och vilken betalmetod du är intresserad av att betala med. Vill du betala med någonting som är online eller offline, vill du betala med ett kreditkort där vi kan göra auktoriseringen direkt, eller vill du till exempel få en QR-kod som du går ner med till din lokala 7Eleven-butik, scannar och sen betalar med cash. Det finns en hel uppsjö av det här. Skulle vi då gå igenom de här sakerna manuellt och sitta och forska i skulle det ta väldigt lång tid. Vi har dashboards där vi jobbar med realtidsdata. Vi jobbar med loggar och parsear våra loggar med Stash och Kibana. Varje loggrad blir en JSON-struktur, vilket gör att vi kan använda Kibana för att bygga dashboards. Då kan vi slänga upp alla våra betalmetoder och sedan titta på loggarna till exempel över sju dagar. Droppar vi på ett visst datum kan vi gå tillbaka och titta på vad vi deployade där, alltså vad var det för någonting som gick ut där som kan ha påverkat droppet. Det här kan vi använda som en charter för vår explorativa testning.	K, M
51	**Då kan jag tänka mig att ni använder och drar ganska mycket slutsatser från sådan typ av data när ni börjar bygga på nya features.**	
52	Ja, det gör vi. Sen så testar vi ju mycket i ett A/B-format. Samtidigt som vi gör annat - de här sakerna är inte exklusiva.	M
53	**Vi pratade en del med Olof igår om det ni kallar Feature Toggles, det vill säga mörklagd produktlansering. När bestämmer ni hur ni ska gå till väga med sådant - vilka faktorer avgör om ni vill släppa någonting men gömma det, så att säga?**	
54	Risk, skulle jag säga. Vad riskerar vi att påverka och i vilket skede vi är. Vi sitter ju ihop med tre andra team till exempel. Då måste man tänka på vad de har i pipelinen, vad som kommer gå ut och vilken tågordning vi måste ha på sakerna som går ut. Är det okej att våra saker går ut samtidigt men att vi dark launchar våra grejer? Är det okej att vi ligger på fem procent och testar eller stör det de grejer ni försöker göra? Sådana saker.	M
55	**Säg att ni har kända defekter i någonting och så funderar ni på huruvida ni ska ge ut det eller inte. Vad har du som QA-roll för inflytande över den diskussionen och det beslutet?**	
56	Väldigt, väldigt mycket. Där får man luta sig tillbaka på det som vi åtog oss att göra när vi i början pratade om fit for use. När vi gör en bedömning: "Vad är det minsta vi måste uppfylla för att vi ska leverera så mycket av det här totala värdet som vi vill åt?". Det är det jag försöker luta mig så mycket emot som möjligt, att alltid titta på det värde vi levererar för organisationen som helhet. Om vi lanserar det här och låter det lägga ut ett känt fel, men att det låser upp så att vi kan fortsätta på något annat, så är det viktigare. Vi vill kontinuerligt kunna fortsätta leverera värde.	S, M
57	**Skulle du säga att du utgår mycket från data då? Historiskt sett kanske du har sett att man borde kunna släppa eftersom att den här typen av defekter ligger långt ner på prioriteringslistan?**	
58	Kvalitet måste vi dela in i mindre fack, för kvalitet omfattar egentligen allting. Även för den minsta lilla av features måste vi ändå dela dess kvalitet i fack. På ena strecket har vi sådana saker som konsumenten kommer bedöma oss på. Vad bryr sig den personen om? På det andra strecket har vi intern kvalitet som är längst åt andra hållet. Beroende sen på vad det är så kan det finnas andra fack i mitten, i det här stora kvalitetsbegreppet. Det är olika saker. För någon som ska sitta och testa det här åtta timmar om dagen så finns det många aspekter, och det är så många olika roller som dömer kvalitet. Där måste man hitta någon form av balans.	M
59	**Du nämnde A/B-testning tidigare. Ni rullar alltså ut features gradvis till en liten mängd användare till att börja med. Hur går det till när ni bedömer och väljer huruvida ni ska välja A eller B, om ni ska strunta i båda, eller köra fullt ut?**	
60	Vi definierar någon form av prestandametrik. För oss på Payments är det ganska enkelt i och med att både A och B ser ganska likadana ut. Flest köp vinner. Det är ju ett väldigt tacksamt KPI. Skulle vi sedan vilja så kan vi bryta ner det här på ålder och andra aspekter också. Om vi vill göra någonting där vi exempelvis vill rikta oss till ungdomar så måste vi ju titta på åldersaspekten också. Vi gör egentligen bara dashboards och rullar ut funktionerna och sedan stämmer vi av genomslagseffekten mot datakällan tillsammans med den slutliga rapportering som vi får från våra Payment Service Providers. På det sättet kan vi se vilket av alternativen som ger bäst konvertering. Sen kan det finnas andra analyser som vi måste göra. En grej kanske går jättebra, men den kanske kannibaliserar på någonting annat som är ännu bättre. Så kan det ju vara också. Det är lite olika beroende på vad det är för typ av feature.	M
61	**Jag förstår. Nu har vi pratat lite om mätning och resultatet av produkten i sig. Hur jobbar ni med mätning och optimering av själva processen?**	

62	Det är lite olika. Vi har ju Agile Coaches på Spotify som driver det här arbetet. Sen så är det väldigt mycket upp till Squaden, sammansättningen av individual contributors. Det enda riktigt återkommande som vi har i mitt team är retrospektiva möten och planering därefter. Där har vi diskussioner kring vad som gick bra, vad som gick dåligt och vad vi kan göra bättre. Sen har vi återkommande uppföljning och action points därefter.	M, S
63	**Har ni några återkommande mätvärden eller enheter som man kan mäta konkret?**	
64	Nej, bara diskussion. Alla har en ganska god känsla för hur mycket vi borde leverera under en iteration. Min känsla är i alla fall att de sakerna som måste komma upp till ytan för att vi ska kunna leverera bättre faktiskt kommer upp till ytan naturligt, utan att behöva mätvärden och estimeringar.	M
65	**Det måste vara svårt att hålla på med konkret mätning av en sådan mjuk sak som just en laginsats.**	
66	Ja, precis. Det kommer se olika ut i alla olika team. Det räcker med att du byter ut en teammedlem och stoppar in en annan, så kan dynamiken se helt annorlunda ut. Där får man hitta verktyg som fungerar för sitt team, att vara pragmatisk. Vad tror vi kommer hjälpa? Finns det någonting vi kan prova? Kan vi utvärde att ha estimat, till exempel, eller var det skulle kunna vara.	S, M
67	**Om ni då gör väldigt bra ifrån er under en iteration, belönas ni på något sätt? Kanske inte just monetärt, men överlag. Hur fungerar det?**	
68	Vi belönar oss själva (skratt). Vi har diverse aktiviteter som vi planerar in - vi har ganska stor frihet att styra vår egen belöning och när vi ska fira att vi har gjort någonting. Om vi vill gå ut och käka med teamet och dricka några bira så har vi ganska stor möjlighet att styra det själva.	S
69	**Nu har vi gått igenom alla förskrivna frågor. Har du någonting du känner att vi inte har frågat om? Någonting centralt kring kvalitet kanske, som du känner är värt att nämna?**	
70	Vi har kommit in på de sakerna lite grann. Någonting jag skulle ta med mig, om jag vore er är att ställa frågor kring hur man ser på intern och yttre kvalitet - hur man bestämmer kvalitetskrav runt användarperspektiv, kontra internt i organisationen. Jag skulle fråga ifall man upplever att kodkvalitet, hastighet och defekter hänger ihop. Det är sådana saker som jag försöker jobba mycket med och sprida kunskap kring. Kvalitet är mycket mer nyanserat än binärt, alltså: “Är det fel eller inte?”. Det finns andra vertikaler som intern och extern kvalitet och så vidare.	
71	**Intressant. Hur ser du på de här sakerna?**	
72	För det första är det väldigt viktigt att definiera både extern och intern kvalitet. Intern kvalitet handlar dels om testbarhet: “Går det här att testa och hur enkelt är det att testa?”. Dels handlar det om hur enkelt det de är att komma igång och göra förändringar om vi skulle hitta ett problem. Det är också ett kvalitetsmätvärde. Intern kvalitet är ofta kopplad till hastighet: “Hur snabbt kan vi, från att vi hittar någonting som vi vill förändra, leverera förändringen till någon som kan bedöma den med externa kvalitetsglasögon?”. Hela den kedjan hänger ihop. Det är någonting som är väldigt viktigt för mig. Det är mitt lilla mantra, att man måste tänka på alla de här sakerna. Jag har varit med i många projekt där man har negligerat intern kvalitet till förmån för extern kvalitet, utan att egentligen ställa sig frågan vad tradeoffen är. Att alla animationer ska bete sig på ett korrekt sätt och att gränssnittet ska vara komplext, men ändå enkelt. Att de sakerna har varit så viktiga att man har lagt nästan all sin tid där, och inte tittat på den interna kvaliteten. Efter ett tag går det så otroligt långsamt att leverera mjukvara för att man har byggt ett kråkslott bakom gränssnittet och komplicerat alldeles för mycket. I de fall där man har gått ut till kunden och frågat så har de väl sagt: “Ja, men det här är väl ett okej interface”. De kommer aldrig att vara mer exalterade än så. Det händer när man inte har återkopplat med kunden kring vad den faktiskt bryr sig om sig. De vill bara ha en stor röd knapp som det står “do it” på och vill hellre att det ska gå snabbt - att det man frågar efter ska levereras fort och att det ska vara bra säkerhet. Den interna kvaliteten och den yttre kvaliteten kan vara mer ihopkopplade än vad man tror.	M
73	**Hur skulle du säga ni går tillväga för att göra det här, alltså att leverera snabbt men ändå hålla en god kvalitet?**	
74	Alla väsentliga funktioner som arbetar kring det vi ska leverera måste acceptera att det kommer ta tid i anspråk. Beställare, produktägare, businessägare, eller vem det nu är, måste spendera tid tillsammans med ingenjörerna och QA-funktionen. Alla måste göra väldigt mycket saker tillsammans och förstå att det här kommer ta tid i anspråk. Det är först när man har fått samman alla som man kan börja ha produktiva diskussioner som kan hjälpa oss framåt. Det är då vi kan lyfta sådana här aspekter som var vi sätter vi vår interna kvalitetsribba, och var vi sätter kvalitetsribban ut mot kunderna. Det är viktigt att begära ut information från användarna, alltså få feedback. Ibland kan vi bygga något i liten skala först för att korta ner feedbackloopen. Men att initialt hålla alla organisatoriska funktioner ansvariga för att man medverkar. Om man tittar på konsultföretag så är det vanligt att man tar ett uppdrag, får någonting kastat över staketet som ingenjör, sitter och bygger, och så	S, M

	blir det pannkaka för att man inte fick veta hur man navigerar i kravställningen.	
75	**Där måste ju samarbete vara A och O mellan de olika parterna och rollerna.**	
76	Och en grundläggande förståelse för alla som ska vara med i det här. Då måste vi förstå alla andra. Man behöver inte ha en djupgående förståelse, men en grundläggande förståelse för vad alla funktioner gör. Då tror jag att samarbetet och förståelsen blir så mycket bättre när man kommunicerar med varandra. Det här är kanske det svåraste med mitt jobb idag, att få ihop det här. Att jobba för att detta ska ske organiskt.	S
77	**Det måste vara en balansgång mellan intern kvalitet och yttre, man vill ju undvika teknisk skuld så mycket det går men samtidigt få ut saker snabbt.**	
78	När det gäller teknisk skuld beror det helt på vad tradeoffen är. Är det så att, ja men vi skjuter ut det här imorgon, då kommer vi kunna skriva på [---]. Jag vet ju att våra businessmänniskor är jätteduktiga på det de gör, de är inte där för att förstöra mitt liv. Om de säger till mig: "Vi kan krita på det här kontraktet med företag X och de har committat att lägga 300 miljoner i marknadsföringsbudget för Spotify och deras produkt", då är ju den där fula lilla kodsnutten eller konstiga if-satsen inte så viktig. Då kanske vi får acceptera att det är så, men vi kanske får tracka den på ett bra sätt. Det måste hela tiden vara en pragmatisk inställning. För det finns inga idealscenarion när man håller på med de här sakerna. Man måste hela tiden försöka hålla en hög ribba, men man måste även vara villig att sänka den vid vissa tillfällen. Men då är det jätteviktigt att man sätter ett slutdatum på när vi höjer ribban igen. Om vi merge'ar det här, ska det vara adresserat om två veckor. Det här är sådant man får komma överens om. Businessen måste acceptera: "Vi skjuter ut det här nu och då får vi det här värdet, men då behöver vi lägga en vecka efter att det här har gått ut åt att förstå hur vi ska göra det här på rätt sätt istället". Det är ett givande och tagande och det är en viktig diskussion, det är viktigt att få fram en förståelse i. Det handlar om hur mycket man jobbar med kravställare, då inte funktionen kravställare, utan den personen som ställer krav eller som spaltar upp en lista av möjligheter som teamet sen konsumerar. Hur mycket man jobbar däremellan med förståelse utav vad kvalitet är för varje item på den här listan. Det är jätteolika, men varje specifik story och varje specifikt item har speciella kvalitetsaspekter som vi kan ta hänsyn till och ha en diskussion kring. Det räcker inte med att tänka på samarbetet, man måste utöva det också när vi gör implementationer.	K, S

B5 Transkriberingsprotokoll - P3

Företag: Klarna
Intervjuperson: P3
Arbetsroll: Team Lead System Test
Tid och plats: Måndag 25 april 2016, kl 09:00-10:00, Skype

(S) - Samarbetspräglad organisationsstruktur och kultur = Grön
(K) - Kontinuitet och automation = Blå
(M) - Mätning, övervakning och återkoppling = Röd

Rad	*Text*	*Kategori*
1	**Vad jobbar du med, vad har du för arbetsroll?**	
2	Jag är Team Lead för ett team som kallas för System Test. Vår uppgift är att genomföra tester på systemnivå som täcker alla olika produkter som klarna bygger. Vi samarbetar med andra teams för att åstadkomma överlappande test coverage, både på komponent- och subsystemnivå, såväl som på systemnivå. Det är det som jag och mitt team gör på Klarna.	S, K
3	**Hur länge har du jobbat på Klarna, och har du haft samma roll hela tiden?**	
4	Jag har varit på Klarna i lite över fem år. Jag har haft ett flertal olika jobb på Klarna. Jag började som Merchant Support 2nd Line, vilket är någon slags butikssupport. Jag jobbade mest mot Tyskland, från Schweiz, då jag pratar tyska också. Det handlade om att hjälpa butiker som har problem med sina integrationer. Efter ett tag blev jag tillfrågad om jag skulle vara intresserad av att testa. Jag började då en kort tid som manuell testare, som jag sedan [---] med automatisering, det var för browser-tester i en av våra produkter. Sedan blev jag någon slags Lead för vår nya plattform som vi lanserade i UK, det handlade om mycket koordination och att bygga ett end-to-end-framework av tester. Det var mycket koordination och rollen var lite blandning av Delivery Lead och QA. För lite mer än ett år sedan fick jag förfrågan om jag skulle vara intresserad av att leda ett eget team inom Engineering och bygga upp det. Det var samtidigt som en ganska stor omorganisation, vi har ingen dedikerad QA-avdelning längre sedan dess. Innan dess var vi ungefär 15 personer som jobbade med QA. Nu har vi istället bara Software Engineers in Test. De flesta flyttade in i team inom respektive domän som de jobbade med tidigare, medans jag och några andra bildade ett team som började jobba med testautomatisering på systemnivå.	S
5	**Okej, jag förstår. Kan du beskriva kortfattat hur organisationsstrukturen ser ut för produktutveckling och test?**	
6	Vi har olika domäner. Det finns flera teams inom varje domän, och varje domän ska ha domänkunskap och vara specialiserade inom ett visst område. En domän kan vara betalningar, själva slutkundsupplevelsen vid köptillfället, kommunikation och service mot slutkunden, och såna saker. Det brukar vara mellan två till tre, teams inom varje domän, ibland flera. Det finns en Engineering Manager som leder varje domän. Sen har vi Team Leads som både har management-ansvar, men ändå ska vara hands-on och teknisk, som leder de respektive teamen. Teamen är Scrum-size, mellan fyra och åtta personer. Så fort de bli större brukar de splittras upp. Lite utanför har vi arkitekter som ska stödja de respektive domänerna. Alla de rapporterar till en VP, sedan högre upp har vi en CIO. Alltså, CIO -> VP -> EM -> TL.	S
7	**Vilka roller ingår typiskt i ett team?**	
8	Det beror mycket på. Vi har inte så att alla teams jobbar med till exempel Scrum, det är alltid någon form av Agile Methodology men det betyder inte att alla har en Scrum Master. Man kan ha det, men alla har inte det. Sen är det så att alla produktorienterade team har en Product Manager. Sen på domännnivå har de i sin tur en Group Product Manager. De kommer från produktorganisationen, vilken inte är samma som Engineering, men de jobbar såklart väldigt nära tillsammans. Det finns alltså en PM vanligtvis, sen en TL per team, sen finns det utvecklare. De är så att säga “bara utvecklare”, det låter lite dumt men, de flesta utvecklare är cross-functional, vilket innebär att varje team har end-to-end ansvar. Det betyder att teamet ansvarigt för att bygga sin pipeline, testa, utveckla, deploya, och även supporta - de har alltså on-call support också. Det beror emellertid lite på vilken service de bygger, vissa services behöver inte ha någon on-call. Detta betyder alltså att ett team är	S, M

	ansvarigt för hela livscykeln av en produkt, från början till slut. Det är liksom inte så att nån annan får testa åt dem. Det här betyder dock inte att det inte finns något samarbete mellan teamen. Det finns exempelvis team utanför produktutveckling som kallas för ESub och Nobel, som ska möjliggöra bättre deployment och så. De jobbar mycket med de verktyg som finns för att enable'a teamen. De kan hjälpa till om ett team exempelvis behöver ett federated Continuous Integration Jenkins-verktyg, eller använda sig av Job DSL'er, alltså en Job Builder för att bygga automatiserade Jenkins-jobb. De hjälper till med att kunna testa det och hjälpa teamet att deploya. Sen så finns det såklart också team som kan säga: "Vi behöver ha bättre monitorering". Då bygger det supporterande teamet en lösning, som sedan alla teams som använder liknande tech-stack kommer använda sig av, för att inte behöva återuppfinna hjulet varje gång.	
9	**Okej. De teamen som jobbar lite mer med verktygen, vad kallas de?**	
10	Det finns lite olika sådana team, vårt team som heter System Test är ju också ett slags supporterande team. System Test and Performance kallas domänen jag jobbar vi, och där ingår också ett Performance-team. Vi båda ligger under produktutvecklingsavdelningen, trots att vi på något sätt bara utvecklar verktyg som möjliggör att vi kan testa saker på systemnivå. Sen har vi Engineering Support, och de kommer från avdelningen IT Ops. Där finns folk som jobbar med nätverk och infrastruktur och alla sådana saker. Trots att det egentligen är olika avdelningar är det ganska tajt samarbete när det behövs.	S
11	**Det är en intressant struktur, som samtidigt kan vara lite svår att greppa.**	
12	Det enklaste sättet att se på det är någon form av matrisorganisation. Vi har Birth Teams vertikalt, sen har vi sådana teams som supportar dessa och går genom alla Birth Teams horisontellt. Sen hur mycket de påverkar... I slutändan är det ju teamet som får bestämma: "Vill vi använda det här eller inte?". Vilket har både för- och nackdelar. Vi får folk att vara mer engagerade i att påverka och utveckla sina egna verktyg, men det betyder också att [---] så att ett team inte nödvändigtvis använder samma verktyg som de andra, vilket kan göra det mer komplicerat.	S
13	**Skulle du säga att varje team är ganska empowered på det sättet?**	
14	Definitivt. Teams anses vara autonomous. De kan fatta sina egna beslut men de får också stå sina egna kast. Det finns ändå krav mot team, de måste ändå leverera bra produkter som är vältestade och så vidare, så det är inte bara att göra vad man vill.	S
15	**Vart kommer de kraven ifrån inom organisationen?**	
16	Det finns lite olika. När det kommer till produktkrav, alltså hur en produkt ska bete sig, så har Klarna som företag en product roadmap som beskriver de initiativ vi jobbar mot. Vi försöker hålla antalet pågående initiativ väldigt lågt så man kan fokusera på en grej i taget. Varje initiativ är nedbrutet i olika mål, och de målen definieras mest av Product Managers. Sen dras det in utvecklare för att bestämma detaljer kring målen, vad som är rimligt och hur vi borde göra det på bästa sätt. Så det är mycket samarbete mellan arkitekter, Product Managers, Group Product Managers, Team Leads och utvecklare. Sen är det ju inte så att vi kan sätta ett mål och att det aldrig kommer ändra sig, och att vi kan releasa ett halvår senare. Det är en väldigt moving target. Det rör sig kontinuerligt, och man måste vara flexibel och anpassa hur man jobbar därefter.	S, M
17	**Har ni några belöningssystem för teamen, om det går bra eller dåligt så att säga?**	
18	Inte på ett sätt som man kan direkt se så. Man har som team så kallade OQR's, det vill säga Objective Queue Results. Det är ett sätt att mäta om ett team lyckades eller inte. Sen har man personliga mål som man sätter, de påverkar sedan belöningen.	S, M
19	**De här personliga målen, innefattar de något specifikt område, som exempelvis att man ska främja produktkvalitet till en viss nivå? Hur ser målen ut?**	
20	De målen är väldigt personliga. Det beror på hur personen jobbar. Så de är gjorda från en Team Lead tillsammans med respektive utvecklare, och kan variera. Det handlar mycket om hur kan vi utveckla alla som jobbar här och samtidigt ändå på något sätt att företaget kan profitera av det. Kvalitet, att skriva... Det är svårt att kvantifiera att till exempel skriva mer tester. Det hjälper inte så mycket. Men att man säger till exempel: "X person har ansvar för att halvera tiden för alla tester utan att man tappar coverage". Det är ganska tangible. Att man säger att man vill få ner testsvitens tidsåtgång från 15 minuter till 7 minuter. Det kan vara ett sådant exempel. Men det kan också vara att en person säger att man vill bli bättre på att kommunicera. Då kanske vi kan få den personen att tala på en konferens eller liknande. Så det beror på vad personen vill också.	S, M
21	**Då har vi pratat lite om struktur och så. Ni verkar ha ganska dynamiska roller, och kollektivt ansvar i teamen. Om vi tittar lite mer inom teamen, och ansvarsområden där. Om vi ser till de som fokuserar på kvalitetssäkring och testning, hur mycket inflytande har de över beslut när det kommer till arkitektur och design av mjukvara?**	

22	Det är lite svårt att svara på, för ofta har vi inte ofta en dedikerad roll som bara är ansvarig för kvalitet. Det finns vissa team som har dedikerade testare eller Software Engineers in Test, och de försöker ju bygga produkten på ett sätt som gör att man kan hooka in och göra white box-testning på ett mycket enklare sätt. Samtidigt ligger också ansvaret hos respektive utvecklare att det är så. Mycket fokus ligger på att testa och testa rätt, men vad rätt är skiljer sig ganska mycket från tidigare erfarenheter eller produkten i sig. Man får inte glömma att, inom Klarna där vi är 30-40 olika teams just nu, jobbar alla med lite olika produkter. Vissa arbetar bara med stateless microservices, och att testa det är en helt annan värld än [---] testing platform som har states, vilket är mycket mer komplicerat och kanske har en del som är inköp som byggs runt omkring allting. Vad som anses vara testning och minimum coverage skiljer sig enormt.	S
23	**Det blir ju så när alla roller i teamet har kvalitet och kvalitetssäkring organiskt i sig.**	
24	Testing har ju lite att göra med vad man försöker mitigera. Man vill ju kunna säkerställa att funktionalitet fungerar - att slutkunden eller en butik kan göra en viss sak. Så vad är det vi definitivt måste kunna veta varje gång vi releasar, när det kommer till regressionstester? I slutändan är det ett priority call från produktägaren och Team Lead när man säger: “Vi har idag inget test coverage för detta, jag tycker vi borde ha det”. Sen får man kolla på hur stora riskerna är att fel kommer hända, att se hur många kunder vi har som beter sig på ett visst sätt, till exempel hur många som använder Internet Explorer 6. Man försöker alltid gå tillbaka till data för att kolla varför man gör någonting. Att försöka testa allt och alla scenarion är orimligt.	S, M
25	**Vad är det för typ av data ni har tillgång till för att göra dessa beslut?**	
26	Vi försöker spara hur kunder och butiker beter sig, alla dessa anrop kan vi göra statistik av. Vilka butiker använder sig av vilken sorts anrop och under vilka tidspunkter? Det gör att man kan se trender.	S, M
27	**Testning och data går ju ihop ganska mycket. Hur arbetar ni med testning? Finns det någon bestämd metod, som till exempel TDD, eller kör alla team olika?**	
28	De flesta team kör olika. Vissa teams kör till exempel alltid någon typ av Mob Programming, så att det alltid är flera som kodar tillsammans. Vissa teams pairar alltid, medan vissa team aldrig pairar men är mer rigorösa när det kommer till code review och sådana saker. All kod är i alla fall versionskontrollerad, och det finns en pipeline för releaser, men hur det exakt fungerar beror lite på. Eftersom att just vi gör systemtester brukar vi ofta starta med ett Birth Team-scenario som beskriver ett high level-krav. Vi utgår från det och [---], sen pratar vi om vad en slutkund, butik, eller vem det nu är som är stakeholder, förväntar sig. Vi går igenom detta scenario och försöker fånga de flesta sakerna som kan inträffa och hur rimligt det är att det händer inom ett scenario. Utan det för den delen leder till någon dum dependency, att en grej failar för att man har pluggat in för många tester. Man vill ju ändå alltid hooka in sig på lite olika steg. Vi brukar utgå från ett Birth Team-scenario, så börjar vi skriva testerna efter det. Vi har byggt ett, eller rättare sagt extendat ett existerande, ramverk för att möjliggöra det för oss. Vi har kontroll exakt över vilken service som beter sig hur och i vilket läge.	K, M
29	**Ni mappar alltså ut vilka scenarion som är användaren sannolikt kommer hamna i, eller?**	
30	Ja, exakt. När ett nytt initiativ startas möts vi med Group Product Manager och pratar om exakt vilka förväntningar som finns på hur produkten ska fungera. Sen när vi har bättre på koll på det, brukar vi komma med förslag på hur vi borde testa det på bästa sätt. Sen implementerar vi det. När vi har implementerat det går vi med vårt scenario till respektive team som påverkas av detta scenario och frågar hur deras coverage ser ut, så att vi kan säkerställa att de har tänkt på alla corner cases och sådana saker, medans vi försöker hålla oss på en högre nivå. Sen kan det vara så att ett team säger: “Vi skulle gärna vilja gå lite djupare med det här”, och då hjälper vi dem med det.	S
31	**Jag förstår. Alla de här testerna du pratar om är automatiserade med andra ord.**	
32	Ja, nästan alla tester är automatiserade.	K
33	**Gör ni någon typ av manuell eller explorativ testning?**	
34	Ja, vi har lite olika saker. Vissa av våra Software Engineers in Test jobbar med de olika [---]. De har olika sessions innan release. Ett team skapar en release branch, sen testar dem då. Ett annat alternativ är att vi brukar ha möten innan release av större features, där vi brukar bjuda in stakeholders och team som har varit med och utvecklat - egentligen är det öppet för alla som är intresserade. Det vi brukar göra då är att vi först går igenom hur en “happy path” ser ut. Sen tar vi oss tid och pratar exempelvis om huruvida vi ska fokusera närmare på ett visst område och göra explorativ testning där. Vissa saker är lite enklare att göra när det gäller explorativ testning än andra.	K, S
35	**Så det är inför release, det du pratar om nu?**	
36	Ja. När vi på System Test är involverade är det oftast minst två eller tre andra teams involverade. När	

	det är större än bara på en integrationsnivå.	
37	**Då förmodar jag att det är en del beslut som måste tas. I ett fall där det finns kända defekter, vad har Engineers in Test för inflytande på releasebeslutet?**	
38	Det vi brukar göra är rätt simpelt. Baserat på den information vi har brukar vi göra någon slags riskanalys - hur sannolikt det är att problem uppstår och hur stort av ett issue det är. Det brukar inte vara jätteformellt hela tiden, men i slutänden blir det ändå så att vi kan ge en rekommendation: "Go eller no go", för respektive release. Om vi säger no go brukar det vara baserat på denna data som vi har. Så det blir inte bara: "Jag tror, kanske". Det handlar om att försöka göra informed decisions eller enligt best guess. Det som är viktigt är att vara jättetransparent mot Product Managers. Ibland kan det vara så att vi rekommenderar att en release blockas men att den releasas ändå - vilket ibland skiter sig. Det kan hända. Men ibland händer ingenting och buggen fixas en vecka eller en dag senare. Ibland kan det också vara så att man väljer att blocka releasen och att den istället flyttas fram. Det beror mycket på situationen. Ju mer argument man har, eller rättare sagt ju mer man kan backa upp dem med data, desto enklare är det att ta sådana beslut. Det är inte så att testarna anses vara "the bad guys" som gör sönder saker, utan det är mer så: "Jaha, se där. Kul, hur hittade du det?". Det blir mer en dialog än att folk är otrevliga mot varandra, för det är inte det man är ute efter. Det är ett väldigt outdated sätt att se det som utvecklare vs. testare. Vi jobbar mycket med att försöka få teams att förstå exakt vad vi gör, hur vi testar och varför. Att också få dem att kolla på vår kod och prata ut om vad de tycker är bra och dåligt, sedan anpassa det. Vi har också en rapport som vi skickar ut varannan dag med en översikt av issues som vi hittar. Vårt testningsramverk är byggt på ett sätt som gör att vi rapporterar varje steg det gör inom ett system, hur lång tid det tog, om det var pass eller fail, samt lite extra information. Det gör det väldigt enkelt att aggregera och säga: "För två månader sedan hade team X alltid två sekunder för en viss integration, nu tar det tio sekunder". Eller: "Detta team har alltid klockat nio sekunder, har de mer issues nu än vad de vanligtvis har?".	S, M
39	**Ni verkar ha en delande kultur där man är öppen med vad man gör.**	
40	Jag tycker inte alltid att det har varit så, eller ja, jag tycker inte vi har varit så bra på det tidigare. Vi har spenderat mycket tid på att visa och inse hur man upplever kvalitet i förhållande till testning eller verifikation av funktionalitet. Kvalitet är ju egentligen hur slutkunden upplever helheten, medan verifiering handlar mer om saker fungerar som de borde. Jag blandar de begreppen lite, jag ber om ursäkt för det. Känslan av kvalitet är ju svår att automatisera och testa på så sätt. Vi har insett att det är enklare om vi från början säger hur vi ska testa något och vad vår output ska vara, sen hur slutkunden upplever det… De säger om det är bra eller dåligt, om de tycker att det fungerar eller inte. I slutändan så bryr sig slutkunden inte så mycket om enskilda services. Om en butik eller en intern agent som ska sitta och klicka runt, så bryr de sig inte om en viss service fungerar, utan de bryr sig bara om helheten. Så att bara kunna säga det att Klarna fungerar, ja eller nej, brukar vara rätt värdefullt. Sen kan man ju ha ett flow där man kan peka att vissa services fungerar bättre än andra, det gör det också lättare att pinpoint'a vad som har gått fel.	S, K
41	**På tal om kultur, skulle du säga att ni tillåts experimentera eller vara innovativa?**	
42	Det tycker jag. Det beror väldigt mycket på vilket team man jobbar i också. Vissa teams har dedikerade dagar där de har möjlighet att bygga ett proof of concept för förbättringar av vissa saker. Det finns hackathons också som kan vara företagsrelaterade, eller samarbete med något externt företag. Så länge man har en bra anledning att säga: "Jag vill testa det här nya frameworket, jag har hört att det är jättebra och jag tror det kommer förbättra någonting med vår produkt". Sen behöver man kunna sälja det mot ett produktbygge, mot en ny feature. På samma vis som med technical debt - att man vill jobba med det istället för en ny feature, så handlar det om hur mycket man vill investera framåt. Jag tycker det är svårt att hitta en bra balans där. Ibland jobbar man bara med features och bygger technical debt och känner att man aldrig får tid att bli av med det, men ibland känner man att man kan fokusera på det. Man har möjlighet, men det kan ibland vara svårt.	S
43	**Om man testar en massa nya grejer och sedan misslyckas, vilket såklart händer, hur hanteras det av företaget?**	
44	Såklart är det okej att misslyckas, men det är viktigt att man inte gör samma fel två gånger. Det handlar om att lära sig från det man gör fel och sedan förbättra det. I slutändan är det inte en person som är ansvarig för ett fel, utan det är ett team som har byggt en grej som inte har upptäckts i code reviewen, under testning av till exempel unit testning, integrationstestning eller performance testning, och som ändå smugit sig in live. Det är ganska mycket som måste gå fel, vilket det ibland gör. Då får man kolla vad som har hänt och varför det inte har upptäckts. Vi får göra en root cause analysis och säkerställa att man har ett regressionstest på plats som testar det bättre nästa gång. Även kolla om något test har givit ett false positive eller liknande. Inom mitt team har vi en gång varannan vecka en timmes clean up sessions, där vi börjar på toppen av våra tester och går igenom alla långsamt för att kolla vad de faktiskt gör. Vi kollar så det inte dupliceras några tester och om de är skrivna på bästa sätt, annars tar vi bor dem. Det är en ganska viktig ritual för oss att verkligen säkerställa att vi bara	S, M

	testar det som är relevant. Om vi tar bort någonting brukar vi även meddela något team så att de inte har några förväntningar.	
45	**Det låter som en formell genomgång av testerna. Du nämnde förut att ni hade kodgenomgångar, men hur mycket av detta utförs formellt?**	
46	Vi har som sagt code reviews, och då behöver minst två personer godkänna koden som inte har varit involverade i att skriva den. Det varierar ganska mycket, det finns vissa teams som har en review, andra teams kan ha en person som är ansvarig för att klona ner hela koden och köra och testa den. Men ja det finns formella genomgångar på plats.	K
47	**Du nämnde att vissa parprogrammerar, och då kanske man inte behöver lika extensiv code review?**	
48	Jag vet inte. Jag är inte helt såld på att man inte behöver göra en code review. Det är väldigt enkelt att man får en person att gå med på någonting om man är jätteentusiastisk, och då kanske båda snöar sig in helt åt ett håll. På det sättet är code reviewen väldigt bra, för då är det en person som aldrig har sett vad problemet är och hur det uppläses. Om den personen då ser: "Okej men varför gör ni såhär?", så hjälper code reviewen ganska mycket. I fall någonting går sönder om en vecka som två andra har rört så kan man gå tillbaka och kolla vad de har gjort och vad de har ändrat. Vi jobbar med så många olika områden samtidigt och då är det ibland att hålla koll på alla saker, och då tycker jag code reviews hjälper att hålla oss uppdaterade utan att behöva göra det i realtid.	K
49	**Utgår code reviewsen från någon slags checklista, eller är det bara från huvudet?**	
50	Det finns ingen typ av formell lista. De flesta team har någon typ av linter som kollar hur koden är skriven och vilken kvalitet den håller. De flesta teams har det så att när en pull request skickas så triggas ett Jenkins-jobb som kör ett test, som sen rapporterar tillbaka till review-verktyget och säger till exempel att alla tester har fungerat eller vilka fel som uppstod. Det handlar mycket om att förstå och kunna läsa syntax.	K
51	**Du nämnde Jenkins i samband med pipelinen, men har ni några andra verktyg för uppsättning av infrastruktur? Till exempel Puppet.**	
52	Vi brukar använda Chef. Eller det finns några team som fortfarande gör det. En hel del team använder sig av Ansible. Vi har också team som använder sig av Docker Compose. Det varierar lite. Det är många team som har börjat flytta och använda Docker mer och mer. Vi har också en hybrid cloud som vi använder oss av.	K
53	**Det är det vi menar med "Infrastructure as Code".**	
54	Jag tycker det är en väldigt viktig del inom testning också, till exempel hur man ger feedback så snabbt som möjligt. Det är bara möjligt om man bygger en infrastruktur som möjliggör det. Vi har nyligen flyttat från Jenkins som exekverar tester och skickar alla tasks med Docker Container till Mesos för att kunna göra det. Det fungerar väldigt smidigt. Mycket av vårat jobb är inte enbart testning, men även testningsverktyg och allting som är runt omkring. Som jag nämnde tidigare är rapportering jätteviktigt, och att vi visar det på ett trevligt sätt på till exempel snygga dashboards.	S, K, M
55	**På tal om infrastruktur, finns det någon BDD-motsvarighet, där man skriver beteendebeskrivningar i domänspråk? Eller är det syntax?**	
56	Vi har haft en intern [---] runt Ansible som tog hand om hela moln-API:et, där man kunde skriva: "Skapa den här maskinen eller provisionera den här", men det är ändå baserat på Ansible-syntax, och jag tycker personligen att det är väldigt smidigt. Man har ett provisioneringsscript där man beskriver vilken maskin som ska göra vad eller i vilken ordning. Jag tycker det löser sig ganska trevligt. I Chef är det ju Ruby, så det är inte så jättesvårt. Det finns ju för- och nackdelar med domänspråk som i Cucumber. Man kan ju abstrahera bort saker på grund av att man skriver en label som kanske inte exakt matchar vad man egentligen gör. Det kan vara bra att återanvända vissa steg, men samtidigt vet man aldrig exakt om det verkligen är det man vill göra. Jag gillar BDD, men ibland inte.	K
57	**Vi nämnde det snabbt förut, men kan du beskriva hur ni går till väga när vi ska släppa något nytt? Hur kan ni samla in data för att avgöra genomslagskraften?**	
58	Oftast är vi med från början. Det är ett grundkrav. Vi kan inte komma in senare och säga att vi vill testa något nytt från ingenstans. Det är inte okej helt enkelt. Vi brukar bli involverade tidigt, vilket också gör att vi kan få en bra uppfattning om hur testdatan ska se ut. Vi brukar göra våra tester samtidigt som respektive team ska utföra sina tester. I fall då några är redo för sin del, men inte de andra tre teamen, så kan vi i de fallen skicka våra anrop till en server bara för att se om allting fungerar, utan att behöva deploya till staging eller liknande.	M
59	**Du nämnde tidigare också att ni kollar mycket på hur användarna beter sig. Om vi pratar om prestanda och den typen av kvalitet, är det något ni tittar på?**	

60	Vi har ett dedikerat team för just performance testing, och de jobbar väldigt nära ihop med oss som gör functional testing. Så vi har functional och non-functional. Sen har vi ett team som är ansvarigt för security operations. Performance-teamet återanvänder sig av vårat framework och kör funktionella tester fast med stor belastning. De behöver inte bry sig så mycket om hur kunden beter sig, men bara vad som händer om kunden gör något några tusen gånger. De jobbar även båda med baseline performance tester in i respektive team pipelines, som även har dedikerade performance-miljöer som de kan köra våra tester mot.	S, K
61	**Så ni använder er också av den typen av data när ni designar nya saker?**	
62	Kanske inte så mycket när man designar. Vid design försöker vi få en uppfattning om vad som är kraven och förväntningarna på kapacitet hos en service. Till exempel om det är en fil som kommer in varje dag, eller om några miljoner kunder försöker göra ändringar. Det beror mycket på om det är en exponerad realtidskomponent som måste svara omgående, eller om det är något som över huvud taget är synligt för kunden. Jag skulle säga att performance testing som är inbyggt från början är väldigt svårt. Testning brukar vara mycket enklare.	S
63	**Vi pratade tidigare om hur ni resonerar kring prioriteringar av defekter inför en release, men händer det att ni släpper någonting, men samtidigt döljer det för slutanvändarna?**	
64	Ibland har vi feature flags, det finns absolut. Vi har A/B-testning som körs. Sen har vi dedikerade testgrupper, men även det skulle kunna kallas för A/B-testning, då man kanske gör en beta-lansering av en ny produkt som bara en dedikerad kund kommer att få se.	M
65	**I vilket syfte gör ni dessa små utrullningar?**	
66	Det handlar ju om att se hur kunden reagerar på det och hur det påverkar conversion. Det finns jättemånga olika saker. Det kan till exempel vara för att se skillnader mellan hur sannolikt det är att kunden klickar på en knapp eller en annan. Det kan vara små saker, och det kan vara stora nya features. Det kan även vara så att man vill släppa en feature trots att den inte är helt färdig i backenden.	M
67	**Hur gör ni då i så fall?**	
68	Teams brukar själva få ta hand om det. För testning behöver vi bara ha ett sätt att trigga respektive feature flag. Återigen, det handlar mycket om hur vi kommunicerar mellan teams och veta exakt vad vi ska testa. Ska vi testa alla möjliga features eller ska vi bara testa vissa? Det beror väldigt mycket på.	
69	**Okej jag förstår. För att kort återkoppla till mätning då, hur gör ni för att mäta och optimera själva utvecklingsprocessen?**	
70	Det handlar mycket om att samla information om till exempel vilka team som har de flesta felen eller om det är mismatch mellan förväntningar. Det kanske handlar om att teamet inte har tillräckligt med tid för att skriva bra tester. Eller har någonting gått helt fel? Som ett resultat för förbättring brukar det oftast handla om samarbete och gå igenom respektive produkt och tester. Kanske inte på enhetstestnivå, men komponenttester och sådana saker. Man kan också kolla på ett teams pipeline. Om ett team tar väldigt lång tid på sig och har många manuella steg brukar fler fel uppstå. Hur kan vi då hjälpa teamet att bli av med detta? Det är sådana saker vi kollar på. Det är ganska många faktorer. En root cause analysis kan visa att något team inte fick tillräckligt med information, eller att de inte hade tid för planering och hade en stor backlog.	M

B6 Transkriberingsprotokoll - P4

Företag: Klarna
Intervjuperson: P4
Arbetsroll: Software Engineer in Test
Tid och plats: Måndag 25 april 2016, kl 16:00-17:00, Skype

(S) - Samarbetspräglad organisationsstruktur och kultur = Grön
(K) - Kontinuitet och automation = Blå
(M) - Mätning, övervakning och återkoppling = Röd

Rad	*Text*	*Kategori*
1	**Can you tell us what your job role is?**	
2	I'm part of the System Test team, which was created maybe one year ago. There was a shift in the organization. Before that there was a centralized QA-area that provided resources to different teams; testers basically. So the QA was already in the teams basically, because every person worked on different projects so in my opinion there was no [---] of a team. In order to have the testers be an actual part of the product development teams, the QA-testers became part of the teams that they were working for. Of course product development teams are responsible for their own services and take care of their services all the way from development to live operations, and if there is an issue they have to be on call and respond to that. They are responsible for the testing, basically everything. Still we were in need of a centralized, well not centralized, but an end-to-end system testing team. There are things that connect services with other services and some end-to-end requirements that can't be the teams' responsibility, because it goes beyond that. So basically my team's responsibility is to provide testing end-to-end.	S
3	**Do you sit in your team?**	
4	Yes I sit with my team mates.	
5	**In the other product teams, there are testers sitting in those teams?**	
6	Exactly. But I'm in charge of testing only those services that specifically connect everything. That basically means we play the role of users, we test everything end-to-end, the end users will be purchase consumers, or a bank, so we play the role of people. We have to have everything automated.	
7	**How long have you had your current role at Klarna?**	
8	About a year.	
9	**Have you worked at Klarna before that?**	
10	Yes, I've been at Klarna for two and a half years. I have a background in security testing, penetration testing basically so from time to time I also do penetration testing, but that's my own activity and not a team activity.	
11	**Would you say that other engineers in the product teams are responsible for, or do quality work?**	
12	Exactly. Every team is responsible for the services they build. They are responsible end-to-end basically. We support teams with extra confidence when they deploy. They have to test their own service end-to-end, they have to run some tests like unit test and all the levels of testing. But we have to see that the service works in the whole system.	S, K
13	**So everyone is responsible for the quality itself?**	
14	To test every single combination of data is up to them [---]. Let me know if I should go more into details.	K
15	**I think that is a good overview. We spoke to Daniel this morning and he explained your structure as kind of a matrix organization with Birth-teams and supporting teams.**	
16	Yes, we are making that shift. We are a support team basically, we need to work with a lot of teams all the time. We spend a lot of time just walking to different areas to talk to people because we have	S

	this end-to-end understanding of the system. We need to try to understand how all the services are connected.	
17	**So are you kind of a provider of tools or communication between teams?**	
18	We try to shift to that as well. At the moment we have our own infrastructure for testing, so we run the tests continuously all the time and report results about the service tests to dashboards that we are doing in different environments. At the moment, teams can use our information or data to see how their services are doing. We have been working with all the teams to set up their testing frameworks, but we're also trying to provide tools, structures and methodologies so that they can do better tests.	S, K, M
19	**Maybe we should move on to more in depth about testing and how it works. Do you in your team have any special method of doing your tests or does it vary from time to time?**	
20	We all come from different background so we all have our own ideas, so we like to discuss a lot. I have a strong interest in automation, but we all agree to try different things. [---] We interact with all the services, and sometimes we breach the services just to verify what happens. When you do end-to-end, it's like a black box. You have some input, you have some output, and if the output is not what you expected, it's hard to pinpoint where the failure is because there could be four or five different services. We split our testing in what we call steps, and every step is encapsulated in this framework so that every single step is measurable. You can tag and put labels on them, so even though the test passes or fails, you can report that during a step when talking to a specific service, it failed. This information can be used to pinpoint the cause. There is also information about the response time for instance, so every action is measured independently.	K, M
21	**So what kind of tests are these? Are they unit tests or integration tests?**	
22	No, I'm talking about automated system level testing. We don't do any unit testing at all in my team. Well, we do unit testing for our tools, but not for Klarna services. We unit test our libraries and frameworks but not the Klarna services. That's up to the teams. We have the whole system, like a version copy, and we integrate it through web sites, APIs, files etc.	K
23	**But that's mostly automated?**	
24	Yeah, almost everything is automated.	K
25	**Do you do any manual testing?**	
26	Not much, but whenever there is a new feature that involves many teams to be coordinated, like a feature that spans over many services, we organize [---]. We set up some tools and gather people from different teams and spend a few hours doing exploratory testing. We try whenever possible just to go through teams' tests as well and to see if new tests are to be added or fixed. But it's up to the teams to do the unit level testing at least.	K
27	**Do any teams follow Test-Driven Development or Behaviour-Driven Development?**	
28	They are free to choose their own methodologies. Many of them do TDD, pair programming and mob programming, but I can't speak for every single team. It's the same for us. We have our own tools, but for instance I do TDD when I get in [---], but it depends on the time or the complexity of what you're building.	K
29	**I presume that you have code reviews?**	
30	Yes. We are a development team basically, since most of our effort is coding. As I told you we have a lot of tools and frameworks. Whenever there is a change and we have a pull request, someone is supposed to review it. We cannot merge it to the master branch until it's been reviewed by X number of people. When it's merged to the master branch there is a pipeline that will deploy it to unit testing, and if it works it will go to the next stage. As a development team we have [---] for our tests, so we test our tests basically. Sometimes our end-to-end test is part of another team's pipeline, so they run all their levels of testing and the last one is probably our test. That means we don't want to have flaky tests, because that will stop them from deploying to production, so we need to be very careful and test our tests. Make sure they don't have false positives. We are a very technical team, like a development team, we develop tests and tools.	K, S
31	**Do you have a predefined checklist or any Definition of Done before you actually commit something? Or do you do the code reviews from the top of your heads?**	
32	There is no documentation or Definition of Done, that is more for infrastructure. The code will not be merged unless it's reviewed, or if the builds in Jenkins have been unsuccessful, so you are forced to review your code. When it comes to a more complex feature that requires a different level of review you have a person assigned that doesn't just review the code, but also run the code and execute it to review it in a deeper way.	K, M

33	**Talking about infrastructure, could you go into detail on how you configure it? Do you use any sort of configuration tools like Puppet or Chef?**	
34	We have libraries that you use, for example Nexus, and then Jenkins to run the unit tests. The libraries are in Ruby, so it's a gem. You can do a change, update the version, pull a request, and if the pull gets approved, it is merged. Then there is a Jenkins job that will get the library and run the unit tests. If it passes, it will publish the new version to Nexus, which is a repository of libraries. That's the case for libraries. Then we also have some services which are used to help in testing basically. For instance there is a service that keeps track of e-mails that we are expecting from tests, because [---] e-mail may take 24 hours to arrive, so we don't want to have a test that lasts 24 hours. So we run the test [---] to create a purchase and that will create an e-mail. So this is the service, that within 24 hours you should receive an e-mail with the subject "blabla". So there is a service to test errands like these offline. So whenever we update a new version of a service we run the unit test, and if it passes, we build this library and push it to Nexus, and then we need to deploy it. We use Ansible at the moment, and we also have a Klarna cloud which is a cloud hybrid, where we [---] and deploy the new service and check that it's fine.	K
35	**So how do you configure the deployment itself? Are you aware of the concept "Infrastructure as Code"?**	
36	Yes. So all the scripts in Ansible and Jenkins are code, we use Jenkins Job Builder for that. So whenever there is a change in the build itself in Jenkins or in the Ansible scripts, there is also a request so that people need to review it. So the infrastructure is code for us a well.	K
37	**You know BDD for developing, do you have anything similar for the infrastructure provisioning, like writing behavioural descriptions?**	
38	No, in our case it's Jamel files. We are moving forward to Job DSL but it's basically the same. We don't have an urge to do that. We use Docker a lot for services, so if we need to mock to Amazon instead of Klarna cloud it's easy to do the switch. It's easier to run tests locally on your machine if you need to, you just run the Docker image and you have all the dependencies on there.	K
39	**We want to talk a bit about data and quality assurance based on feedback. Can you describe how you work with measurement and feedback on quality and performance?**	
40	Sure. I think there is a cost of running a test. When you write a test you need the time from a person to write the code, and also maintenance, so there is a cost of time and money that you could spend somewhere else. So you need to take the most you can from that test. It's valuable information. The way we take a lot of value from the tests is to register them. Every test we run, we store the results in Elasticsearch at the moment. If you don't store results you basically you just run a bunch of tests that will pass or fail, but it doesn't give you much information. If it fails it might be a bug in the test or the system, some environmental issue. If you want to take a lot of [---]. The execution doesn't tell you much, but if you have ten thousand executions of a test and it always passes, but then it fails, it will give you more than… I mean, not more but it's different than a pass or fail every now and then. So history gives you a lot of information if your test is flaky or not. It gives you a good idea if the failure is caused by an actual bug or an issue in the test. So every single action we do, every single step, every single service we interact with is reported to Elasticsearch, so we index everything. Then it's easy to see the trends or patterns in graphs. For instance we have e-mail reports every 48 hours, then we do some queries in the data race, where we try to accumulate different kinds of failures in order to pinpoint the team that is responsible for that. We create an email with sample data so that they can debug the issue. That's done automatically using history.	M
41	**Do you use data about users and user scenarios?**	
42	We generate synthetic data for our tests. For our tests we use random data, so it's not the same every time. We'll have a range and pick a random number in that range, and then generate synthetic data for different countries so if a test happens against a German checkout, it will generate German names and addresses. Every order or whatever we create will look different. So pros and cons? The con being that you may run that same test twice and get different results because you chose the wrong ranges for your data, so that it went through a different path in the system. The benefit being that, if you run the same tests with the same data all the time you might lose specific paths for combinations of data, but with random data we spot a lot of bugs that only happens because of the random data. So in order to repeat the same test twice we just log all the data we use so we can repeat it manually. We also log the ID of the random number generator so we can use the same ID to generate the same data again. Usually it works for us to use random data.	
43	**Speaking of bugs and known defects, how do you prioritize them? How do you decide if they're critical, or if they can be released even though there are known defects?**	
44	So we are not gatekeepers. Everything is a puzzle of releasing to production or not. We try to make	S

	all the teams run our tests and empower them as much as we can, to have them make the decision to release or not. As we do end-to-end testing, let's say that we have a [---] with A, B and C. The team that developed C want to go to production, they run a unit test and they pass, they run integration test and they pass, then run our system end-to-end test and they fail because of A or B. The test will fail, but should they deploy to production? We try to provide them with the information. "The test failed because of A and B, it's up to you to release or not". We basically try to provide them with information only, not to be gatekeepers. As I told you, we have a lot of knowledge about the whole system, so we basically know if something is high priority or not. So we might have a suggestion to release or not, we give our input basically. It's not our call.	
45	**So you have a lot of influence on the decision, but you don't make the decision?**	
46	Exactly.	S
47	**So how do you decide if it's critical or not? Do you use any user case scenarios mappings?**	
48	So you have the business critical scenarios. But I don't think we have any methodology or checklist, we just know that it's critical because it affects the most important service. The idea is that we don't catch any bugs, it should be caught by the different teams on different test levels so we try to push testing to the teams as much as we can. We should not test for them. So whenever there is a test that fails because of a bug and it has been deployed into staging or production we enforce them to implement a test so it doesn't happen. It shouldn't be on us. We support extra confidence.	S
49	**So if you have a bug but the business side really wants to deploy into production, do you ever release bugged features but hide them with feature toggles?**	
50	Usually when there is something important going on there is a "go or no go"-meeting that both the Leads and different members of the teams involved in the feature attend. If there is a bug included in production that is discovered, we could fix it right away, but it depends on the bug. A newly deployed feature is disabled by default, so A/B-testing basically, so it can be enabled in certain [---] to verify that it's working. But it depends on the team, each team owns their service and are responsible of how they deploy features or not. It differs.	S, M
51	**What kind of information can you get from deploying disabled features?**	
52	Well basically they are disabled for a specific user so that we can see if it works, or test it in staging, or do a real purchase for specific users only. It's not available for the public, but the code is deployed to production. There is a lot of A/B-testing on the front end, on websites, so a feature is displayed for a certain number of people, and for other it is not. So we gather metrics to see how it works.	M
53	**So if you deploy two features, A and B, and then feature B is the most successful, do you deploy it to everyone or do you do it gradually?**	
54	That's up to product managers and the teams to decide.	M
55	**We've talked about measuring quality and how to discover bugs and functional testing, but how do you measure your development process to make sure it's effective?**	
56	As we work with a lot of teams and as we are a support area, we cannot plan ahead too much. We could try to have a backlog but there are changes all the time as we talk to different teams all the time. Their deliveries may get delayed, so that's why we don't use Scrum for instance. We use something more similar to Kanban. A sprint of two weeks would be too long for us, because it would change. We try to measure against KPI's and goals, as a support team for other services. We also try to measure how many tests fail because of issues in our tests or libraries versus issues in the system, and having those metrics so we can fulfill our goal of reducing the amount of failing tests caused by us, by 50 %. But we're working on finding other ways of gathering metrics and using that. But again, our goals change all the time, so it's hard to keep track of.	M
57	**You have personal goals. Are those unique for every person, or do they focus on any specific quality aspects?**	
58	So the personal goals are aligned with the team goals, so we may share some of the goals. There are also personal goals that are used for self development, like learning more about this or that technology that could be important for conferences or trainings. But the goals are shared over X services or a specific service so there might be a joint effort. And we also have team goals that are aligned with the company goals.	S
59	**So what happens if you reach the goal, or if you don't?**	
60	We try to make the goals ambitious, but basically we have to make them reachable. I think that nothing bad happens if you don't, but you have to look into why you didn't reach it. Maybe because you have something else that is more important that came up, so we try to have many goals not just one, and we try to update them. We have one on one meetings every week with our Team Lead Daniel	S

	and go through the goals and adjust them. If I didn't reach my goal of, I don't know, support for X features, it could be because something has changed, or maybe because we didn't perform. In that case we need to take action and see what's going on.	
61	**How are you rewarded when you do something great?**	
62	Personally I don't look for rewards in that case, I just do my work and don't need my Team Lead to tell me every day that I'm doing great. I just want to know if I'm doing something wrong. Usually juniors may need more positive feedback, so I think it's up to every single person individually. I guess the way of rewarding is giving new challenges and responsibilities.	S
63	**On the other hand, if something goes really bad and you basically fail, would you say that there's an open culture at Klarna that supports failing?**	
64	In my experience Klarna does support failing. We sometimes break some stuff to make something fail, or try to come up with new ways of testing or new tools. We spend time on researching and trying new things. If something goes wrong we don't need to blame someone, we just need to see what went wrong so it doesn't happen again. I've never seen anyone point fingers because someone screwed up. It's the team's responsibility.	S
65	**So would you say that innovation and experimentation is supported?**	
66	Yes.	
67	**So all of our topics are very broad so to speak, but you get the picture we're looking for it's all these tool chains and feedback loops that are supported by the culture. Is there any topic that you think we missed out on?**	
68	Let me think. We spoke about the technologies that every team is free to choose, but we have to set some limits. We use Java, Ruby or Redlang and try to keep it among those three. We share the same languages more or less, so it's easy to shift teams and move around. But then again, if your team wants to use Go or whatever language they are free to try. The same with infrastructure and those methodologies, it's up to the teams to decide. New things have to be approved, but other than that there is a lot of freedom.	S
69	**The teams must be very empowered in that sense, that they're free to choose how to work as long as they deliver the expected quality.**	
70	Exactly.	
71	**It must be nice to have so much freedom and such an open way of working.**	
72	Yes, but the consequences of that is also that a framework for a certain technology cannot be used for other technologies, so maybe you need two versions of it. If you look at someone else's code you want to know the language it's written in, so it might take more time to understand it. Maybe you want to use someone else's pipeline, then you might have to do some rework. But other than that I think there are a lot of benefits in choosing technologies.	S

Referenser

Adams, B. & McIntosh, S. (2016). Modern Release Engineering in a Nutshell: Why Researchers Should Care, *Future of Software Engineering (invited paper), Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, ss. To appear

Armenise, V. (2015). Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery, *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, ss.24–27

Bang, S. K., Chung, S., Choh, Y. & Dupuis, M. (2013). A Grounded Theory Analysis of Modern Web Applications: Knowledge, Skills, and Abilities for DevOps, i *RIIT 2013 - Proceedings of the 2nd Annual Conference on Research in Information Technology*, 2013, ss.61–62

Beck, K., Beedle, M., Bennekum, A. Van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001). Manifesto for Agile Software Development, *The Agile Alliance*, Tillgänglig Online: http://agilemanifesto.org/ [Hämtad 2016-05-02]

Bhasin, S. (2012). Quality Assurance in Agile: A Study towards Achieving Excellence, i *Proceedings - Agile India 2012, AgileIndia 2012*, ss.64–67

Cao, L., Mohan, K., Xu, P. & Ramesh, B. (2009). A Framework for Adapting Agile Development Methodologies, *European Journal of Information Systems*, vol. 18, no. 4, ss.332–343

Cho, J., Huff, R. A. & Olsen, D. (2011). Management Guidelines for Scrum Agile Software Development Process, *Issues in Information Systems*, vol. 12, no. 1, ss.213–223

Claps, G. G., Berntsson Svensson, R. & Aurum, A. (2015). On the Journey to Continuous Deployment: Technical and Social Challenges along the Way, i *Information and Software Technology*, Vol. 57, ss.21–31

Collins, E. F. & de Lucena, V. F. (2012). Software Test Automation Practices in Agile Development Environment: An Industry Experience Report, *2012 7th International Workshop on Automation of Software Test (AST)*, ss.57–63

DeGrandis, D. (2011). DevOps: So You Say You Want a Revolution?, *Cutter IT Journal*, vol. 24, no. 8, ss.34–39

Diabol (2016). Diabol hjälper Klarna utveckla en ny plattform och att bli experter på Continuous Delivery, [blogg], Tillgänglig Online: http://blog.diabol.se/?p=894 [Hämtad 2016-04-28]

Di Bella, E., Fronza, I., Phaphoom, N., Sillitti, A., Succi, G. & Vlasenko, J. (2013). Pair Programming and Software Defects - A Large, Industrial Case Study, *IEEE Transactions on Software Engineering*, vol. 39, no. 7, ss.930–953

Edelen, C. (2014). Balancing Act: Software Quality Vs. Time To Market, *WallStreet & Technology*, 24e mars, Tillgänglig Online: http://www.wallstreetandtech.com/risk-management/balancing-act-software-quality-vs-time-to-market/d/d-id/1268821 [Hämtad 2016-04-09]

Erich, F., Amrit, C. & Daneva, M. (2014). Cooperation between Information System Development and Operations: A Literature Review, i *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*, ss.1–1

Feitelson, D. G., Frachtenberg, E. & Beck, K. L. (2013). Development and Deployment at Facebook, *IEEE Internet Computing*, vol. 17, no. 4, ss.8–17

Fitzgerald, B. & Stol, K.-J. (2015). Continuous Software Engineering: A Roadmap and Agenda, *Journal of Systems and Software*

Gohil, K., Alapati, N. & Joglekar, S. (2011). Towards Behavior Driven Operations (BDOps), i *Proceedings of the 3rd International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2011)*, ss.262–4

Hedström, J. (2014). DevOps lyfter Spotify, *TechWorld IDG*, 18e november, Tillgänglig Online: http://techworld.idg.se/2.2524/1.594442/devops-lyfter-spotify [Hämtad 2016-04-19]

Huizinga, D. & Kolawa, A. (2007). Automated Defect Prevention: Best Practices in Software Management, Hoboken, NJ: Wiley-IEEE Computer Society Press

Humble, J. & Farley, D. (2011). Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Upper Saddle River, NJ: Addison-Wesley

Huo, M., Verner, J., Zhu, L. & Babar, M. a. (2004). Software Quality and Agile Methods, *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, vol. 1, ss.520–525

Hüttermann, M. (2012). DevOps for Developers, Dordrecht: Springer

IEEE. (2014). IEEE Std 730-2014 - IEEE Standard for Software Quality Assurance Processes (Revision of IEEE Std 730-2002), *IEEE Std 730-2014 (Revision of IEEE Std 730-2002)*

Jacobsen, D. (2002). Vad, hur och varför? - Om metodval i företagsekonomi och andra samhällsvetenskapliga ämnen, Lund: Studentlitteratur

Kaleshovska, N., Josimovski, S., Pulevska-Ivanovska, L. & Janevski, Z. (2015). The Contribution of Scrum in Managing Successful Software Development Projects, *Economic Development*, vol. 17, no. 2, ss.175–194

Khalane, T. & Tanner, M. (2013). Software Quality Assurance in Scrum: The Need for Concrete Guidance on SQA Strategies in Meeting User Expectations, i *IEEE International Conference on Adaptive Science and Technology, ICAST*, 2013

Klarna (2016). Open Positions, Tillgänglig Online: https://www.klarna.com/careers/open-positions [Hämtad 2016-04-28]

Liu, Y., Li, C., Liu, W. (2014). Integrated Solution for Timely Delivery of Customer Change Requests: A Case Study of Using DevOps Approach, *International Journal of U-& EService, Science & Technology 7*, ss.41–50

Lwakatare, L. E., Kuvaja, P. & Oivo, M. (2015). Dimensions of DevOps, i *Lecture Notes in Business Information Processing*, vol. 212, 2015, ss.212–217

Nabulsi, M. & Hierons, R. (2015). A Test Framework for Communications-Critical Large-Scale Systems. *IEEE Software*, vol. 32, no. 3, ss.86-93

Ozer, M. & Vogel, D. (2015). Contextualized Relationship Between Knowledge Sharing and Performance in Software Development, *Journal of Management Information Systems*, vol. 32, no. 2, ss.134–161

Roche, J. (2013). Adopting DevOps Practices in Quality Assurance, *Communications of the ACM*, vol. 56, no. 11, ss.38–43

Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J. M. & Oivo, M. (2016). Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study, *Journal of Systems and Software*

Schaefer, A., Reichenbach, M. & Fey, D. (2013). Continuous Integration and Automation for DevOps, i *Lecture Notes in Electrical Engineering*, vol. 170 LNEE, ss.345–358

Scharff, C. (2011). Guiding Global Software Development Projects Using Scrum and Agile with Quality Assurance, *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, ss.274–283

Smeds, J., Nybom, K. & Porres, I. (2015). DevOps: A Definition and Perceived Adoption Impediments, i *Lecture Notes in Business Information Processing*, vol. 212, 2015, ss.166–177

Solís, C. & Wang, X. (2011). A Study of the Characteristics of Behaviour Driven Development, i *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011*, ss.383–387

Spinellis, D. (2012). Don't Install Software by Hand, *IEEE Software*, vol. 29, no. 4, ss.86–87

Spotify (2014). Spotify Engineering Culture (Part 1), [video], Tillgänglig Online: https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/ [Hämtad 2016-04-15]

Spotify (2014). Spotify Engineering Culture (Part 2), [video], Tillgänglig Online: https://labs.spotify.com/2014/09/20/spotify-engineering-culture-part-2/ [Hämtad 2016-04-15]

Sumrell, M. (2007). From Waterfall to Agile - How Does a QA Team Transition? i *Proceedings - AGILE 2007*, 2007, ss.291–294

Virmani, M. (2015). Understanding DevOps & Bridging the Gap from Continuous Integration to Continuous Delivery, *Fifth International Conference on Innovative Computing Technology (INTECH)*, Galcia, Spanien, ss.78-82, 160

Waller, J., Ehmke, N. C. & Hasselbring, W. (2015). Including Performance Benchmarks into Continuous Integration to Enable DevOps, *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 2, ss.1–4

Walls, M. (2013). Building a DevOps Culture, Sebastopol, California: O'Reilly Media

Wettinger, J., Breitenbücher, U. & Leymann, F. (2014). DevOpSlang - Bridging the Gap between Development and Operations, i *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8745 LNCS, ss.108–122

Wiegers, K. & Beatty, J. (2013). Software Requirements, 3.uppl., Redmond: Microsoft Press