UPPSALA UNIVERSITET CAMPUS GOTLAND

Institutionen för informatik och media

Utbildningsprogram: Kandidatprogram i Systemvetenskap (inriktning Programvaruteknik)

Kursansvarig lärare: Maritha Dahlin

Datum: 21 oktober 2022

Kvalitetssäkring, 7,5 hp

Sophia Quirk, Jennifer Wåhlin, Daniel Jari och Carl-Johan Johansson

1 Inledning

Kvalitetssäkring handlar om att säkerställa att kvaliteten i systemutveckling och rutiner är tillräckligt hög. Exakt vad "tillräckligt" innebär är emellertid svårt att säga på förhand eftersom det skiljer sig från system till system, och även från betraktare till betraktare. Att kvalitetssäkring är viktigt för att uppnå en god standard inom utveckling är emellertid odiskutabelt. Gustavsson & Görling (2019, s. 13) skriver exempelvis att: "Anledningen till att många IT-satsningar i dag misslyckas är inte att det saknas teoretiska modeller och ramverk för hur vi borde agera. Anledningen är snarare att vi ofta har problem med att tillämpa dem på verkligheten med all den komplexitet som omger oss."

Av intresse för den här rapporten är därför hur mycket av forskningen som tillämpas i verkligheten, och om några av de rutiner som arbetats fram på arbetsplatser för att höja standarden på slutprodukten är understödda i forskningen eller ej.

Kvalitet är ett brett begrepp som kan delas in i mindre delar, eller kvalitetsegenskaper. Idén är att förbättringen av en kvalitetsegenskap också höjer kvaliteten på slutresultatet (Gustavsson & Görling, 2019; Alhassan, et al, 2017). Kvalitetssäkring är ett område som har utvecklats och omdefinierats genom införande av ny teknik. Det är inte länge sedan versionshanteringsprogram som exempelvis Git blev en standard inom IT-industrin. Det är ännu färre år sedan Continuous Integration (CI-linjer) med automattester av kod började användas i stor omfattning. Även systemutvecklingsmetoder som Scrum, Kanban, osv används i dag med avsikten att garantera en högre kvalitet i produkten.

Rapporten fokuserar på kvalitetssäkringsprocesser inom Svenska Spel som är ett statligt ägt spelföretag inom den reglerade spelmarknaden i Sverige. Svenska Spel är indelat i tre affärsområden: 1) Spel & Casino, som erbjuder spel på sport, nätcasinon, poker och dylikt; 2) Tur, som behandlar nummerspel och lotter; 3) Casino Cosmopol & Vegas, som ansvarar för landbaserade kasinon och värdeautomater.

Det är kvalitetssäkringen för digitala spel som är av intresse för rapporten, och för dessa ansvarar i huvudsak affärsområdena Spel & Casino och Tur.

1.1 Syfte

Rapporten är menad att ge en bättre förståelse för hur företag arbetar med kvalitetssäkring i praktiken. Metoder och standarder som följs på företaget kopplas till kurslitteraturen för att skapa jämförelser mellan kursens teoretiska delar och företagets praktiska rutiner. Rapporten är en kvalitativ studie där två respondenter har intervjuats om de kvalitetssäkringsmetoder som används i respektive arbetsteam.

1.2 Metod

Vi har genomfört en intervju med två utvecklare på Svenska spel. Nedan följer en kort presentation av respondenterna:

Respondent 1: Lukas Björkman har arbetat som backend-utvecklare på företaget i två år. Han jobbar med programmeringsspråket Python i ett team som heter Application Platform.

Respondent 2: Linus Löfgren har arbetat som frontend-utvecklare på företaget i tre år. Han arbetar i programmeringsspråken JavaScript och C# i ett team som heter kundsidor och snabbspel.

Intervjun genomfördes via Microsoft teams vilket möjliggjorde att respondenterna kunde ställa upp under arbetstid. Intervjun pågick i en timme och samtalet spelades in med hjälp av en diktafonapp. Inspelningen transkriberades (se bilaga 2) och bearbetades om till en intelligent ordagrann transkription (Semantix, 2022).

Inför intervjun utformade vi en arbetsmodell och formulerade frågor som resulterade i en intervjuguide (se bilaga 1). Vi valde att avgränsa frågorna till tre kategorier: kvalitetssäkring, arbetsprocess och tester. Vi ställde även ett par inledande frågor som berörde respondenternas yrkesroller. Eftersom vi inte visste hur väl respondenterna känner till modeller och begrepp som berör kvalitet försökte vi att anpassa frågorna till en utvecklare, som i sin yrkesroll främst har ett praktiskt förhållningssätt till kvalitetssäkring. Intervjuguiden skickades till respondenterna innan intervjun med en förhoppning om att det skulle förhöja kvaliteten och relevansen av svaren.

I enlighet med Vetenskapsrådets forskningsetiska principer (Vetenskapsrådet, 2016) har vi informerat respondenterna om deras roll i uppgiften och hur informationen kommer att användas för att uppfylla informationskravet. I början av intervjun informerades respondenterna om att deras medverkan är frivillig och att de har rätt att avbryta den när de vill. De lämnade sitt samtycke till att medverka och uppfyllde därmed samtyckeskravet. Nyttjandekravet och konfidalitetskravet har inte varit aktuella inom ramarna för arbetet eftersom uppgifterna inte kommer att användas i andra syften än just denna rapport och respondenterna inte lämnat information av särskilt känslig art som motiverar någon form av konfidalitet. Båda har godkänt att deras namn används i texten.

Vi har använt deduktiva och induktiva analysmetoder för att relatera respondenternas svar till de kvalitetsbegrepp som definieras i forskningslitteraturen (McConnell, 2004; Alhassan, et al, 2017; Delone & McLean, 2003; m. fl). För en komplett förteckning över litteratur som använts för att definiera kvalitetssäkring och kvalitetsegenskaper, se referenslistan.
Texten tar upp begrepp och termer som ISSM (Information System Success Model), TQM (Total Quality Management), Scrum och DevOps (Developer Operations). Dessa är etablerade begrepp kopplade till kvalitetssäkring och systemutvecklingsmetoder men åtföljs för det mesta av litteraturreferenser som förtydligar sammanhanget.

2 Resultat

Nedan sammanfattas det som framkom ur olika delar av intervjun under tre rubriker som representerar olika frågor ur arbetsmodellen.

2.1 Kvalitetssäkring

Svenska Spel har olika dokument med riktlinjer som rör kvalitetssäkring. Under intervjun framkommer det att dokumenten inte nödvändigtvis används. De anställda är medvetna om att dokumenten finns men i olika team följs dokumenten olika noggrant. Respondent 2 är väl insatt i vilka typer av riktlinjer som finns men menade att de inte följdes till punkt och prickar i hans team. Respondent 1 är inte bekant med riktlinjerna utöver att han är medveten om att de existerar. Han menar också att eftersom hans team främst består av tekniker så efterföljs inte systemteorin lika noggrant som i andra team (se bilaga 2, fråga 4).

Respondenterna uttrycker att de önskar förbättring vad gäller inblick i riktlinjer för kvalitetssäkring med förhoppning om bättre samsyn och kommunikation kring den kod som skrivs (se bilaga 2, fråga 6).

Båda respondenter är överens angående hur temporär eller långsiktig en lösning bör vara. De menar att det inte finns några kortsiktiga lösningar eftersom chefer inom företaget inte vill ägna resurser åt att förbättra redan fungerande kod. Respondent 2 ger ett exempel då du kommenterat din egna kod med “‘Fix this’ – Två år senare och ingen som har brytt sig” (se bilaga 2, fråga 5).

2.2 Arbetsprocess

Både Respondent 1 och Respondent 2 anger att respektive team arbetar mer eller mindre enligt Scrum. De har sprintar som är olika långa beroende på vilken typ av projekt de arbetar med. Just nu arbetar de med 2- respektive 3-veckorssprintar. Varje morgon har de så kallade daily-standups; korta morgonmöten på stående fot där alla får berätta hur de ligger till med specifika arbetsuppgifter och vad de har för olika hinder. I varje team arbetar olika arkitekter som ansvarar för kod på olika nivåer i systemet. Det finns även arkitekter som arbetar mer heltäckande över flera team. Innan en utvecklare laddar upp sin kod visas och diskuteras koden med rätt mjukvaruarkitekt under en så kallad code-review (se bilaga 2, fråga 7).

Flera olika DevOps-relaterade verktyg används för CI (Continuous Integration). Jenkins är ett av dessa verktyg som bland annat används för körning av automattester. Respondent 1 berättar att Svenska Spel till stor del har ett hemmabyggt CI/CD-flöde (Continuous Integration/Continuous Delivery) men att de just nu ser över möjligheten att byta ut dessa delar mot de vältestade tjänster som finns på marknaden.

Respondent 2 berättar att utöver de Jenkins-loggar han ser över “då Trinidad inte vill bygga” (se bilaga 2, fråga 8a) så behöver de inte arbeta aktivt med Jenkins. Fungerar allt är Jenkins endast något som sker automatiskt i bakgrunden efter att man pushat via Git. Respondent 1 tillägger att det är smidigt att hela företaget arbetar i samma flöde via konfigurationssystemet för Kubernetes (se bilaga 2, fråga 8a).

De nämner både för- och nackdelar med att arbeta i ett automatiserat flöde. Då utvecklaren inte testar koden manuellt kan det vara svårt att förstå vad som händer, och Respondent 1 menar då att det blir lite för icke-transparent. En stor fördel med det automatiserade flödet är att de tester som laddats upp med tidigare kod då också körs, vilket gör att problem oftast upptäcks innan de hinner ställa till problem (se bilaga 2, fråga 8b).

Respondent 1 berättar att de på svenska spel arbetar i moduler och genom att göra det så bidrar man till att det blir enklare att byta till att arbeta i andra miljöer och även byta programmeringsspråk om det skulle behövas. Respondent 2 instämmer och anser att arbeta med moduler också leder till att man enklare kan återanvända kod (se bilaga 2, fråga 9).

2.3 Tester

Utöver autotester uppger respondent 2 att det finns anställda på företaget som specifikt ägnar sig åt att göra GUI-tester. Dessa frontend-testare är inne på sajten och testar att inloggningarna fungerar som de ska och att knapparna triggar rätt funktioner. Utöver autotester och GUI-tester utförs även enhetstester. Respondent 2 uppger att Svenska Spel har som mål att täcka upp 80% av systemen med enhetstester men att det varierar i hur mycket tid hans team ägnar åt att skapa och utföra dessa. Han poängterar att enhetstester i vissa fall kräver mer än vad det ger. Det är nödvändigt att jämföra nyttan med kostnaden då enhetstester kräver både tid och resurser. Respondent 2 säger att det vore nästan omöjligt att arbeta efter ett sådant mål och påpekar att koden kan vara återanvänd eller datamodeller, i båda fallen har koden redan genomgått flera tester (se bilaga 2, fråga 10).

De har många testmiljöer som täcker olika områden. Vissa är säkert till och med lite överflödiga uppger Respondent 2. Testmiljöerna har olika egenskaper och funktioner, de liknar mer eller mindre produktionsmiljön. Respondent 2 säger att de har en “vilda västern”-miljö som är en säker miljö där man ändå kan börja testa delar av koden. De har även en testmiljö som ska vara så publik som möjligt där allt ska vara inställt som det är i produktion.

På frågan om det finns några nackdelar med de tester som utförs idag svarar Respondent 1 att det är svårt att utföra automattester på den typen av spel som Application Platform utvecklar, och att Continuous Integration därför ofta kommer in för sent i utvecklingsprocessen (se bilaga 2, fråga 13). Det är svårt och tar lång tid att hitta på fejkdata som ska skickas in för att få ut fejkdata, och det finns funktioner som inte kan testas eftersom de är beroende av externa system. Respondenten säger att exempelvis JUnit-tester inte är värdefulla i det här avseendet

och att det är bättre att ha användartester som man kör när alla delar av programmet är på plats.

Respondent 2 säger att man på frontend-sidan av utvecklingen delar kodbas med väldigt många utvecklare, och att upp till 50 personer dagligen är inne och petar på samma kod. Små problem kan därför stoppa produktionsflödet under långa stunder och man lägger mycket tid på att åtgärda dem. Respondent 2 säger också att kodbasen är så stor att det ibland tar 30 minuter efter att man pushat sin kod till Continuous Integration-linjen innan man får tillbaka ett negativt testresultat, och då tvingas man avbryta vad man gör för tillfället och återuppta vad man gjorde tidigare. Respondent 2 påpekar också att automatiska GUI-tester ofta går sönder utan att det är något faktiskt fel på koden. Utvecklaren tvingas att skriva en så kallad merge request många gånger innan den till sist går genom utan att något förändrats.

På frågan om hur mycket utvecklingstid som ägnas åt testning (se bilaga 2, fråga 14) svarar Respondent 1 att man testar mycket, men att mycket av testningen på backend-sidan handlar om ”slänga skit på väggen och se vad som fastnar”. Det är därför svårt att tidsuppskatta hur lång tid testerna tar, men att man ägnar mycket tid åt detta. Det handlar dock om manuell testning, och sedan får specifika testare gå in och se till att koden fungerar med allting annat i ett makroperspektiv.

Respondent 2 säger att de testar mycket, men att det är lätt att missa saker och att det är därför företaget har specifika testare som utför så kallade regressionstester, som utförs om och om igen för att se till att tidigare funktionalitet bibehålls. Man testar det man gör för tillfället “så gott man kan” medan det skapas. Respondent 2 säger att han ägnar väldigt lite tid åt stresstester och GUI-tester och uppger att det är någonting man eventuellt skulle behöva ägna mer tid åt. Uppskattningsvis ägnas mindre än en procent av utvecklingstiden åt de här typerna av tester.

På frågan om vad de skulle vilja jobba mer med för tester framöver (se bilaga 2, fråga 15) säger Respondent 1 att han är sugen på fler metamiljötester, där man snurrar upp en separat utvecklingsmiljö med egna oberoende på en avgränsad server. Det gör det lättare att testa var man befinner sig i utvecklingen just nu utan att riskera att ha sönder någonting i koden för något annat arbetsteam. Isolerade testmiljöer innebär också att man kan göra mer aggressiva tester som försöker ha sönder systemet för att kontrollera dess robusthet.

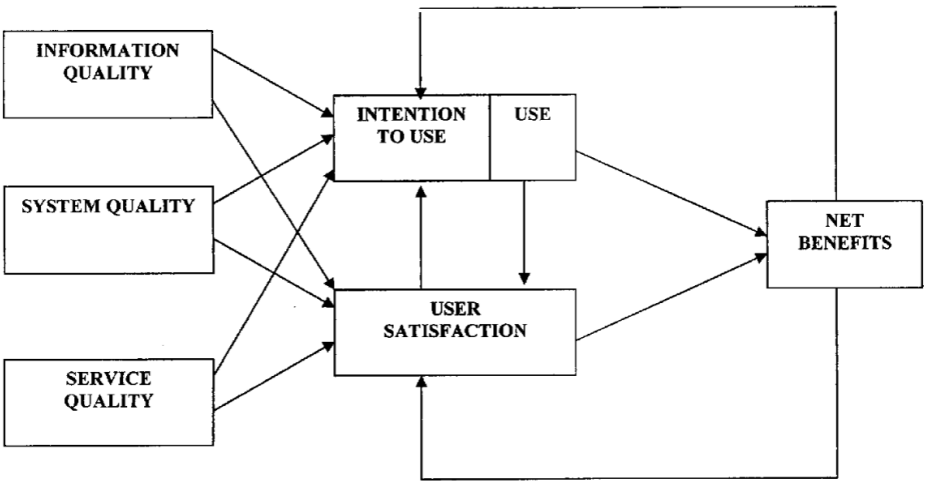
Respondent 2 uppger att de i frontend-utvecklingen i framtiden behöver jobba mer med så kallade device-tester, där man testar på specifika plattformar som exempelvis en iPad.

3 Diskussion/Reflektioner

Genom intervjun med utvecklarna uppstod flera intressanta diskussioner som kan knytas till olika koncept och modeller inom både kvalitetssäkring och systemutvecklingsmetoder. Resultatdelen av rapporten ger en ofiltrerad översikt av både frontend- (se svar från Respondent 1) och backend-utveckling (se svar från Respondent 2) i mindre team på Svenska Spel.

Kvalitetssäkring är ett paraplybegrepp som innefattar många olika modeller och metoder. Det kan vara svårt att avgöra vad kvalitetssäkring innebär eftersom det finns många dimensioner, det vill säga att finns många olika faktorer som bidrar till att en produkt eller tjänst uppnår en högre kvalitet. Kvalitetssäkring kan till exempel innebära dokumentation av resultaten man kommit fram till, eller att införa automatiska test som försäkrar att programkod fungerar som den ska och följer en enhetlig standard. Svenska spel använder ‘sig av kvalitetssäkring på olika nivåer, och beroende på vad man jobbar på och hur man jobbar använder man sig av olika kvalitetssäkringsmetoder enligt respondenterna.

ISS-modellen som arbetats fram av DeLone & McLean (2003) beskriver förhållandena mellan sex kvalitetsdimensioner som anses vara kritiska för framgångsrika informationssystem: Informationskvalitet, Systemkvalitet, Servicekvalitet, Systemanvändande/Användaravsikt, Användarbelåtenhet, och så kallade Net benefits som påverkas av både användandet och användarupplevelsen av systemet. Av dessa sex dimensioner var frågorna som utgör grunden för rapporten avgränsade till att behandla de tre första eftersom båda respondenterna som intervjuades arbetar med utvecklingssidan av verksamheten. Det finns dock en användaraspekt av kvalitetssäkring som är minst lika viktig som utvecklarsidan. Ett rimligt antagande är att ett företag av Svenska Spels storlek utför regelbunden uppföljning med sina kunder för att se hur användarna uppfattar systemet i användning.



Figur 1: *Updated D&M IS Success Model*

Systemkvaliteten är den dimension som är enklast att koppla till utvecklingssidan eftersom den innefattar många av de kvalitetsegenskaper som är ett resultat av kvaliteten på programkoden. Båda respondenterna tar upp exempel på externa egenskaper som robusthet, användbarhet och effektivitet. Även inre kvalitetsegenskaper som testbarhet och återanvändbarhet (se bilaga 2, fråga 12-14) diskuteras.

Företaget utför regressionstester (se bilaga 2, fråga 13) eftersom systemen måste fungera rätt även när de är hårt belastade, vilket styrker systemets servicekvalitet. Svenska Spel har ett stort kundfokus och systemen måste leverera för att inte riskera att förlora användarnas förtroende. Respondent 2 uttrycker en önskan om att utföra device-tester i framtiden (se bilaga 2, fråga 14). Detta skulle höja användarupplevelsen för kunden vilket går i linje med företagets kundfokus. Att utforma krav efter kundens behov som systemet ska leverera bidrar till en högre informationskvalitet.

Båda teamen arbetar med Scrum, ett ramverk som består av ett antal principer, processer och rutiner som med fördel kan anpassas inom systemutveckling för att säkerställa att projektet upprätthåller en hög kvalitet (Gustavsson & Görling, 2019). Det som lyfts i intervjun som en del av företagets rutiner är sprintar (två eller tre veckor långa) och *daily-standups*. Huruvida de tillämpar andra processer och principer, däribland *product backlog* och *sprint planning*, är okänt utifrån empirin. Båda respondenterna uppger att det varit en positiv upplevelse att arbeta utifrån Scrum.

För att kvalitetssäkra tester behövs en strukturerad och utarbetad strategi för hur utförandet ska gå till, vilket innebär att testverksamheten bör få ta plats redan i planeringsstadiet (Gustavsson & Görling, 2019). Testerna finns till för att säkerställa att kraven på systemet uppfylls. Därför behöver kraven vara detaljerade och inte innehålla några motsägelser som kan leda till olika tolkningar för olika parter (Gustavsson & Görling, 2019). Genom att formulera tydliga krav skapas rätt förutsättningar för att utforma tester, vilket i sin tur sparar både tid och resurser. Kostnaden för att korrigera fel ökar ju senare de upptäcks. Därför är det nödvändigt att kontinuerligt utvärdera och förbättra testprocedurerna så att det inte blir en belastning för företaget.

Respondenterna uppger att det finns anställda testare på företaget som har hand om den huvudsakliga delen av auto- och GUI-testerna, men att utvecklarna själva har ansvar för att formulera och utföra enhetstester. Det framkommer inte i intervjun hur kommunikationen mellan utvecklare och testare sker men både Respondent 1 och Respondent 2 uppger att det finns fel som regressionstesterna fångar upp. De tar även upp att teamen lägger mindre tid på testning än vad Svenska Spel har som mål, ett rimligt antagande är därför att det skulle vara möjligt att upptäcka fel i ett tidigare stadie (genom enhetstester och code review) om teamen skulle lägga mer tid på att planera och organisera sin testverksamhet.

Total Quality Management (TQM) är ett teoretiskt ramverk för företag som är “aimed at satisfying all the customer requirements, needs and expectations using a continuous

improvement approach" (Alhassan, et al, 2017, s. 2). Författarna menar att ordet total syftar till alla aktiviteter inom företaget:

> *"It suggests that any improvement that is made in the business, be it a better design of a component or a better process of a system, will help to improve the 'total quality of the organization and the quality of the final product."*
> (Alhassan, et al, 2017, s. 1)

Analysen av Svenska Spel är begränsad i omfång till utvecklarsidan, och därför finns det många andra aspekter av företaget som också verkar för kvalitetssäkring utöver de vi har försökt identifiera. Svaren från båda respondenterna pekar emellertid på att flera av TQM-principerna så som de definieras av Wang, et al (1995) enligt Alhassan, et al (2017) kan kopplas till aktiviteterna på utvecklingssidan.

Respondent 1 och 2 uppger exempelvis på flera ställen i den insamlade empirin att de arbetar både med CI-linjer (Continuous Integration) och individuella tester i sina arbetsrutiner. Detta knyter an till TQM-principen om att mäta ("Measurement") utvecklingen för att kunna få en idé om kvaliteten på systemet just nu.

Båda uppger också att de arbetar med iterativa processer i form av Scrum som bygger på TQM-konceptet om en "continuous improvement process" (Wang, et al, 1995), dvs en pågående förbättringsprocess där produkten aldrig anses vara helt färdig och ständigt byggs på, förändras och utvecklas. Utvecklingen av digitala spel inom Svenska Spel måste på liknande vis ständigt underhållas och uppdateras för att uppfylla krav på säkerhet och kompatibilitet med mjukvaruuppdateringar i både operativsystem och webbläsare. Mycket av arbetet är drivet av kundfokus ("Customer focus") för att utveckla användarupplevelsen och anpassa den till nya plattformar och format som dyker upp på marknaden. För bara drygt tolv år sedan fanns exempelvis inte smarta telefoner. Så kallade device-tester där man testar mot många plattformar är även ett område som Respondent 2 påstår har förbättringspotential (se bilaga 2, fråga 14).

3.1 Induktiv och deduktiv analys

Med deduktion menas att man utgår från en eller flera premisser och sedan härleder slutsatser från dem. Med induktion menar vi analyser som i stället utgår från utfall, eller empiriska observationer; slutsatsen kommer alltså först och sedan generaliseras premissen fram i efterhand (Johansson, 2020).

En induktiv analys av Respondent 2:s svar på frågan om det finns några nackdelar med testen som man utför i dag är exempelvis att GUI-testerna ofta går sönder trots att det inte är något fel på dem, och att det därför inte är förtjänstfullt att utföra så många GUI-tester (se bilaga 2, fråga 13).

Samtidigt svarar dock Respondent 2 på frågan om hur mycket utvecklingstid som ägnas åt testning (se bilaga 2, fråga 14) att han vill ägna mer tid åt GUI-tester och att dessa bara utgör mindre än en procent av all utvecklingstid. En deduktiv analys skulle därför kunna härleda att GUI-testerna är viktiga, men att själva testmetoderna behöver förbättras för att inte gå sönder hela tiden.

Detta ger även upphov till en ny fråga: är det alla GUI-tester som går sönder eller bara vissa specifika GUI-tester? Man kan också fråga sig om GUI-tester utgör en så liten del av effektiv utvecklingstid eftersom utvecklarna räknar med att de kommer gå sönder och därför drar sig för att köra dem, trots att de egentligen är viktiga. Ward Cunningham myntade enligt Gustavsson & Görling (2019, s. 21) begreppet teknisk skuld som syftar till "dålig eller kortsiktig utveckling som framöver kommer att innebära en belastning" och därför inte är eftersträvansvärd. Man kan här reflektera över att den tekniska skulden inte bara sker i form av dålig programkod utan även i form av ineffektiva testmetoder.

Vi ställde även frågan: "Har ni fått vägledning eller informerats om några generella riktlinjer för kvalitetssäkring i ert arbete?" till respondenterna. Vi upplever i efterhand att frågan är för bred för att svara på under den korta tiden vi har med dem. Detta medför att slutsatser som baseras på svaren inte innefattar vilka riktlinjer som finns i företaget. Svaret innefattar inte heller tydlig information om hur företaget arbetar med att informera de anställda kring dessa. Det som går att uttyda är att de tillhandahåller någon form av information och dokument rörande riktlinjer som nyanställda, och att det varierar hur väl de granskar och följer dessa.

Om vi jämför respondenternas svar på frågan ser vi att Respondent 2 är betydligt mer insatt än vad Respondent 1 är, i vilken typ av dokumentation som finns i företaget. I början av intervjun framkommer det att Respondent 1 arbetar som ensam utvecklare i sitt team medan Respondent 2 arbetar tillsammans med flera andra utvecklare i sitt team. Vi kan göra antagandet att utvecklare som arbetar i team med fler utvecklare oftare nås av information om riktlinjer och är mer benägna att av sätta sig in i de dessa. Den deduktiva slutsatsen blir då att det finns en koppling mellan hur insatt en utvecklare är och antalet utvecklare i teamet.

Det framkommer i de personliga introduktionsfrågorna (se bilaga 2) att Respondent 1 är självlärd, till skillnad från Respondent 2 som har genomgått en utbildning i programvaruutveckling. Även här kan vi göra ett antagande, att en självlärd utvecklare tolkar och värderar riktlinjer rörande kvalitetssäkring annorlunda än en utbildad utvecklare. Den deduktiva slutsatsen blir då att det finns en koppling mellan utvecklarens utbildning och hur utvecklaren tar till sig och värderar olika riktlinjer från företaget.

4 Rekommendationer & Slutsatser

Det är lätt att som artikelförfattare övertolka sin egen roll i rapporten när det kommer till att forma rekommendationer och dra slutsatser utifrån diskussionen som förts kring empirin. Denna empiri är resultatet av en timmes intervju och därför begränsad i både omfång och djup. Det vore även naivt att tro att ett företag av Svenska Spels storlek inte redan har ägnat mycket tid åt att utreda och utvärdera sina arbets- och testrutiner. Vi har därför utgått från respondenternas egna tankar om områden som har förbättringspotential i våra rekommendationer. Slutsatserna är separata från rekommendationerna och bör också tolkas på så vis.

4.1 Slutsatser

Det finns en allmänt känd diskrepans mellan befintlig forskning och dess faktiska tillämpning i praktiken ute på arbetsplatser, av förståeliga skäl: alla utvecklare är inte akademiker, och det står vanligtvis inte i deras jobbeskrivning att de ska hålla sig uppdaterade på befintlig forskning. I diskussionen framgår det att svaren går att relatera till aspekter av informationskvalitet, systemkvalitet och servicekvalitet inom ISS-modellen.

En slutsats som dras utifrån svaren på frågan om respondenterna "är medvetna om generella riktlinjer för kvalitetssäkring" (se bilaga 2, fråga 4) är att det finns en koppling mellan hur pass insatta de är i ämnet och antalet utvecklare i deras arbetsteam.

En annan slutsats utifrån svaren på samma fråga är att det även finns en koppling mellan utvecklarens utbildning och hur utvecklaren värderar och granskar krav och riktlinjer för kvalitetssäkring.

Respondenterna nämner inte TQM som begrepp specifikt, men flera av deras arbetsrutiner och strukturer matchar modellens principer och aktiviteter. Det framgår att kundfokus ("customer focus"), Scrum ("continuous improvement process") och mäta och testa ("Measurement") är centrala delar inom båda teamens arbetsmodeller.

4.2 Rekommendationer

En rekommendation som vi format utifrån en deduktiv analys av empirin är att man bör se över hur GUI-testerna är utformade och om de går att förbättra för att göra dem mindre benägna att gå sönder.

Ytterligare en rekommendation är att göra en uppföljning på Respondent 2:s resonemang (se bilaga 2, fråga 14) om att det skulle behövas fler device-tester i framtiden för alternativa plattformar såsom surfplattor. En utvärdering kan behöva göras för att se om den extra testkostnaden detta skulle medföra är motiverad.

Respondenterna nämnde var att de önskade att företaget skulle bli bättre på att faktiskt sätta sig in i, och följa, de riktlinjer för kvalitetssäkring redan som finns (se bilaga 2, fråga 6). Ännu en sak som nämndes var att de ville öka samsynen på den kod som skrivs genom förbättrad kommunikation. Detta skulle till exempel kunna skapa bättre flexibilitet (McConnell, 2004, s. 501) men också öka kvaliteten på slutprodukten.

Referenslista

Alhassan, A., Alzahrani, W. & AbdulAziz, A. (2017). *Total Quality Management for Software Development*. Tillgänglig via Uppsala Universitet: https://uppsala.instructure.com/courses/65158/files/3440411/download?download_frd=1 [2022-10-21]

Delone, W. & McLean, E. (2003). *The DeLone and McLean Model of Information Systems Success: A Ten-Year Update*, Journal of Management Information Systems. Tillgänglig via Uppsala Universitet: https://uppsala.instructure.com/courses/65158/files/3689735?wrap=1 [2022-10-21]

Gustavsson, T. & Görling, S. (2019). *Att arbeta med systemutveckling*, Studentlitteratur AB Lund, Upplaga 1:2.

Johansson, A. (2020). *Vad är deduktiv forskning?* Cafealar.se. Tillgänglig: https://www.cafealar.se/vad-ar-deduktiv-forskning/ [2022-10-21]

McConnell, S. (2004). *Code Complete*, Microsoft Press, Andra upplagan. Tillgänglig: https://uppsala.instructure.com/courses/65158/pages/vecka-36-informationskvalitet?module_item_id=524339 [2022-10-21]

Semantix (2022). *Transkription - transkribering?* Semantix.se. Tillgänglig: https://www.semantix.com/se/transkription-transkribering [2022-10-21]

Vetenskapsrådet (2016). *Forskningsetiska principer*. Tillgänglig via Uppsala Universitet: https://uppsala.instructure.com/courses/44110/pages/system-och-kunskapsutveckling-oversikt-och-litteratur?module_item_id=262465 [2022-10-21]

Wang, R. Y., Storey, V. C., & Firth, C. P. (1995). *A framework for analysis of data quality research*. IEEE transactions on knowledge and data engineering, 7(4), s. 623-640.

Bilagor

Bilaga 1 - Arbetsmodell

Nedan syns de intervjufrågor vi arbetat fram under QiP 1.

Introduktionsfrågor

#	Fråga	Källa	Motivering
1.	Vilken typ av team arbetar du i?		Personlig introduktionsfråga
2.	Vad är din roll i teamet?		Personlig introduktionsfråga
3.	Hur länge har du arbetat på företaget?		Personlig introduktionsfråga

Kvalitetssäkring

#	Fråga	Källa	Motivering
4.	Har ni fått vägledning eller informerats om några generella riktlinjer för kvalitetssäkring i ert arbete?	McConnell (2004)	Det finns många olika systemegenskaper (externa och interna) som vi ofta tar upp i kursen.
5.	Hur drar man gräns för hur temporär eller långsiktig en lösning behöver vara med tanke på hållbarhet och robusthet?	McConnell (2004)	Hållbarhet och robusthet är två viktiga egenskaper inom systemkvalitet.
6.	Vad ser du/ni för brister och förbättringspotential i kvalitetsarbetet?		Värdefullt att veta hur det långsiktiga tänket ser ut.

Arbetsprocess

#	Fråga	Källa	Motivering
7.	Hur ser arbetsprocessen ut? Arbetar ni med någon särskild agil systemutvecklingsmetod som exempelvis Scrum eller liknande? Har ni regelbundna möten för att stämma av utvecklingen, etc?		Relevant att veta vilken typ av utvecklingsmetod som används.
8a.	Arbetar ditt team med verktyg för DevOps såsom CI/CD, exempelvis i Jenkins eller Microsoft Azure?	Tomas Gustavsson och Stefan Görling, Att Arbeta Med Systemutveckling (2017)	Eftersom vi arbetat med detta i kursen
8b.	Följdfråga: Erbjuder Jenkins värdefulla funktioner i verksamheten?		Uppföljningsfråga på föregående
9.	Är det möjligt att flytta produkten ni arbetar med till andra miljöer och programmeringsspråk?	McConnell (2004)	Intern kvalitetsegenskap: Portabilitet, återanvändbarhet, hållbarhet

Tester

#	Fråga	Källa	Motivering
10.	Kan du/ni ge något konkret exempel på ett par tester som ni utför på koden?	McConnell (2004)	Intern kvalitetsegenskap: testbarhet
11.	Vad finns det för fördelar med de tester ni gör?		Uppföljningsfråga på föregående
12.	Finns det några nackdelar med testerna?		Uppföljningsfråga på föregående
13.	Hur mycket tid ägnar ni åt testning under utvecklingen?		Intressant att veta vilken avvägning som görs mellan att spendera tid på tester och att spendera tid på utveckling i allmänhet
14.	Hur tror ni att arbetet med tester kommer att utvecklas i framtiden? Exempelvis, vilka		Värdefullt att veta hur det långsiktiga tänket ser ut.

	typer av tester kommer ni arbeta mer/mindre med?		

Bilaga 2 - Transkription

Fråga 1

1.	Vilken typ av team arbetar du i?		Personlig introduktionsfråga

Lukas: Mitt team heter Application Platform, och det är väl främst ett driftteam där vi underhåller plattformar, där teamet främst består av tekniker och inte utvecklare.

Linus: Mitt team heter kundsidor och snabbspel, vi håller först och främst på med frontendutveckling. Några snabbspelsprodukter är bingo, poker och nätcasino, och så håller vi på med kundsidor. Det kan till exempel vara inloggningssida det kan vara gränssättning. Vi håller även på med pauser, där man känner att “nu tycker jag att jag spelar lite för mycket jag kan inte riktigt kontrollera det här” så kan man pausa lite på olika sätt. Och sen så håller vi också lite på med de här produkterna så finns det lite backend moduler i C# och (JS)node.

Fråga 2

2.	Vad är din roll i teamet?		Personlig introduktionsfråga

Lukas: Jo men jag är då ensam utvecklare i teamet . Jag har varit tekniker så till en viss procent hjälper jag fortfarande med våra teknikerdelar, driftbehov i Kubernetes. Sen jobbar jag främst med att bara utveckla Moduler, eller flöden, för att se grejer för liksom själva miljön, och det gör jag ensam.

Sophia: Hur stort är teamet?

Lukas: Vi är, jag tror att vi är 8 pers, och som jobbar och så eller så i princip.

Linus: Mitt team? Eh, vi är 7 utvecklare för tillfället. 3 st produkterna / 4 vi har en inlånad nu som håller på med kontodelar.

Jennifer: Vilken roll har du i din grupp?

Linus: Jag jobbar som, eh, jag är systemutvecklare, så jag gör både lite frontend och backend. Jag kanske inte måste gå in på detaljer, jag gör helt enkelt allting som har med utveckling att göra.

Fråga 3

3.	hur länge har ni jobbat på Svenska spel?		Personlig introduktionsfråga

Lukas: I två år.

Linus: Tre, ganska precis tror jag, det är nog där.

Carl: Har ni jobbat någon annanstans innan Svenska Spel, typ med liknande saker?

Lukas: Jag har arbetat med IT, framför allt IT-support, så ganska annorlunda från nuvarande IT.

Linus: Jag kommer tidigare från Skatteverket, där jobbade jag i två år. Innan dess var jag lärarassistent 4 månader innan jag fick jobba på Skatteverket.

Fråga 4

4.	Har ni fått vägledning eller informerats om några generella riktlinjer för kvalitetssäkring i ert arbete?	McConnell (2004)	Det finns många olika systemegenskaper (externa och interna) som vi ofta tar upp i kursen.

Lukas: Det finns ju, det finns ju nedskrivet någonstans. Det är väl bokfört på min nivå men, för vårt team, eller vår grupp så har vi ett eget system och det är väldigt unika för Svenska spel, så det är ju ingen annan som kan det på det sättet och därför sätter vi själva upp kraven. Kodmässigt så är det globala grejer för det finns ett sånt här dokument för mina saker. Det är lite mer vilda västern hos oss.

Linus: Ja men det finns ju jättemycket egentligen, som man bör följa liksom. Huruvida det har följts är ju en helt annan fråga. Det är nog väldigt olika från team till team. Lite konkreta saker som vi har är att vi har utvecklingsprocessdokumentet. Ett levande dokument där man kan skriva upp: “nu finns det en ny standard där” men när vi får en nyanställd ska de ha gått igenom här dokumenten och ta del av det i alla fall.

Och sen så har vi ju också vår webbserver för hemsidan, kan man väl säga, och den har ju en egen SDK som också har jättemycket att en matnyttig info. Typ: “Hur bygger man de här interna konceptet som vi kanske kallar komponent eller widget eller något sånt,” “Vad bör man följa när man bygger en sån”, “Vem behöver man prata med?”, “Vem äger det?” Sådana saker.

Och sen ere på kodnivå så finns det ju till exempel de här linterprogrammen, som är typ en mjukvara som sitter i texteditorn. Den berättar t,ex “nu har du inte indenterat koden korrekt”. Om du till exempel skriver long int istället för int så försöker den säga åt dig att det inte är enligt standard för företaget typ. Utöver det så har vi SonarQube, vet inte om ni har stött på det, men det är ett sånt här kodanalysprogram, på sourcen kan man säga, som säger: “Här kan det finnas potentiella säkerhetsbrister för ni använder inte SSL i det här fallet”

Och så har vi Code-abuser-teamet där man på Svenska Spel inte kan skicka ut kod utan att någon annan utvecklare tittar på den.

Fråga 4a

4a.	När reagerar systemet på att kod måste analyseras?		

Lukas: I Github-flow blir det att man jobbar i en “future branch”, och varje gång man commitar så triggas ett bygge trots att det inte är till “master-branchen”. När ett bygge triggas så sker autotesten samtidigt och har man någon linter så sker det också där.

Linus: Det finns också en så kallad realtidslinter som tittar på vad du gör just nu, och den kan också stoppa bygget ifall den ser att du inte följer kodstandard. Linter är något som man måste installera i sitt IDE eller texteditor, och har folk det inte installerat på ett korrekt sätt så kan man tro att allting gått rätt ända till tills man märker att den inte var det vid första bygget, 30 minuter senare, och då har man slösat jättemycket tid. Vi har ett kommando som kontrollerar att tester och Linter gått igenom innan man behöver ha pushat upp till ett Repo.

Fråga 5

5.	Hur drar man gräns för hur temporär eller långsiktig en lösning behöver vara med tanke på hållbarhet och robusthet?	McConnell (2004)	Hållbarhet och robusthet är två viktiga egenskaper inom systemkvalitet.

Lukas: Jag tror inte på att man kan göra generella lösningar eftersom allting tillfälligt blir permanent. Det är verkligheten. Jag tror inte det finns tillfälliga lösningar för allting som fungerar, fungerar. Kod är så, skriver du kod som fungerar så är funktionellt kod. Kodmässigt så tror jag inte att det finns någon gräns det finns bara ett ja eller nej. Fungerar det och alla tester går igenom så fungerar det. Annars är det inte okej och man ska nog vara beredd på att koden, även om den är ful, är permanent. Det som kan göra att koden blir tillfällig är att de har stor arbetsbelastning på systemet och då måste man gå igenom den och göra den funktionell vid ett senare tillfälle.

- Sen kanske vi ska nämna det här med teknikerdelarna. Vi har typ en serverpark där vi behöver flytta runt maskiner, och det där är helt en tillfällig lösning. Men där är det inte riktigt det här… för ,mycket av problemet med det här, varför “tillfälligt blir permanent”, är ju oftast att man har en ganska stor arbetsbelastning. Att man har x antal system, att man har tid att göra tillfälliga saker. Man gör någonting sen så går man vidare till nästa grej.

Jobbar man mer som systemläran, mer frenetiskt, som det inte blir i teknikerarbetet, så har man ju samma sak så då är det mer att man har liksom det här, gigantiska bubblan i stället för massa småbubblor. Där tycker jag att det går bättre att göra tillfälliga lösningar, för där har man mer så här: Okej, vi kanske har en brist på resurser så att då kanske vi kan, men den här

går rätt bra, så vi kan sno några servrar från det och flytta in här så vi kan få lite mer liksom hjälpkraft och sånt.

Jag tror mer, det är möjligtvis om du har som ett större system där man är mer medveten om vad allt är, där kan det nog gå bättre att göra mer tillfälliga saker, annars: Allt är permanent.

Linus: Jag tror jag instämmer ganska precist i det där faktiskt. Det är, det händer att man tror att man gör saker typ tillfälligt men det är alltid så att det blir permanent. Jag vet inte det någonting i det här att man liksom, det är ju ändå någon som betalar; affären betalar ju för att vi ska göra någon viss utveckling, och det är inte så ofta som det dyker upp en *JIRA* som säger: "Ta hand om era dåliga lösningar ni gjorde för två år sen!" ifall inte någon utvecklare verkligen tar tag i det där. Så det är inte realistiskt att man någonsin får tid att göra det. Så lösningen man gör för tillfället, den kommer ju vara i princip permanent. Det har man exempel på överallt i koden med kommentarer och allt var det nu kan vara. Typ "Fix this" – Två år senare och ingen som har brytt sig.

Fråga 6

6.	Vad ser du/ni för brister och förbättringspotential i kvalitetsarbetet?		Värdefullt att veta hur det långsiktiga tänket ser ut.

Linus: Jag tror att för mig känns det också som att det är liksom att få typ teamet på sen med nivå. Jag tycker det känns som att det ibland finns väldigt olika nivåer av engagemang eller hur mycket man bryr sig. Vissa kanske tycker det är jättekul med de här lite mjukare frågorna, typ som vi sitter och pratar om nu, och vissa avskyr allt sånt och vill bara skriva kod hela tiden och det går ju inte. Det blir, eller har jag upplevt att man ser olika på *code reviews* ibland, och det blir en ganska… det är ju inte alls att man följer den här utvecklingsprocessen som man ska göra varenda gång, för den är så otroligt detaljerad och det är så mycket som man egentligen borde göra så att man skulle aldrig ha tid att skriva någon kod om man följde den till punkt och pricka. Och där kanske det finns en förbättringspotential till processen, att man kanske kan ha en stor process, men att man kanske har lite guldkorn som man kan ta ut ur den som faktiskt är mer applicerbara i vardagen.

Men sen, så åter till de här *code reviewsen* i teamet, så - jag tror att, eller, det känns som att det är så svårt att få alla på samma nivå, och vissa uppgifter kan också vara typ att någon har gjort en lösning som man kanske inte håller med om, eller som man inte tycker är speciellt bra i grunden, och sajten har tappat tre dagar i utvecklingen. Är det då rimligt att bara hacka ner på hela den lösningen? Den funkar ju liksom. Ska jag komma och säga att jag tycker inte det här var bra? Det är en jättesvår avvägning. Jag tror man måste på något sätt synka i sitt team. Har man varit med ett par år så kanske man har samsyn på såna här saker, men sånt där kan man vara jätte-, jättesvårt kan jag tycka.

Lukas: Ja, ja, jag har lite underförstått att det är väldigt viktigt med… eller i och för sig, i mitt fall, i de miljöerna jag existerar och skriver kod i, är det ganska klart, för vi har inte dåligt

med testning eller *autotestning* och sådär, eller det är mycket av en del testning och om man vill testa någonting så får man det manuellt testat. Men det är också ganska bristande, ramverksmässigt då, att det finns inte något just nu större som är utvecklat till, att, liksom, enkelt bygga tester eller autotesta. Det är ju ett ganska stort behov att testa. Tester är ju tråkigt att göra. Det är såna saker, att skriva tester. Det är ju väldigt viktigt att lägga tid och energi i ett tidigt skede för att ramverket du bygger och ska jobba i får ett enkelt sätt att utforma testerna. Det är ju något jag själv tittar på från och till, och försöker hjälpa till att implementera, där är det enkelt att göra. Om det tar tid och det blir jobbigt att göra - få till ett sånt [test] automatiskt! Så det behöver man inte lägga tid på att testa flera typer av lösningarna. Man gör samma sak med testgrejer också, att man slipper lägga den tiden på att testa massa eller på att bygga kropptester för att man tycker att det här borde lösas av sig självt. Sånt är viktigt att få till automatiserat, så att man man själv slipper lägga tid på det. Det är inte kul.

Fråga 7

7.	Hur ser arbetsprocessen ut? Arbetar ni med någon särskild agil systemutvecklingsmetod som exempelvis Scrum eller liknande? Har ni regelbundna möten för att stämma av utvecklingen, etc?		Relevant att veta vilken typ av utvecklingsmetod som används.

Linus: För att svarar på det första: Vi arbetar enligt *Scrum*, och för oss är det treveckorssprintar där vi brukar sätta upp veckovis mål eftersom vi jobbar med lite större projekt. Så att beställaren kan säga att vi är, *”Demo bar progress”.* En gång i veckan brukar vi försöka stämma av. Det är ett ganska schysst sätt att jobba. Det är första gången jag testar att jobba så, men det har varit riktigt schysst faktiskt. Vi har *daily standups* också där vi brukar ta varvet runt, vi säger lite vad man har för hinder och såna saker och övrig information. Vi har ju *lösningsarkitekter* och *software architects* också som är arkitekter i olika nivåer som brukar kunna dela lite info på de här mötena.

- Det här med *code reviwes*, att det riskerar bli för mycket kanske - för jag tycker inte heller att man ska, liksom, när koden funkar, så är inte jag någon att säga: “Det här var för dålig lösning” eller sådär. Men vi har ju den här *software-arkitekten*. Han är den som vi ska gå till för att stämma av sådana här saker och han är den som har sista ordet i vad som är okej och inte och sådär. Men jag vet inte, det funkar lite si och så med det här. Det är ju en övermäktig uppgift egentligen för en person att ha koll på växande utvecklare som gör lite vad dom vill.

Lukas: Yes, men det är väl *Scrum* i botten i alla fall, två veckors sprintar ungefär. Sen är det ju också, om vi går tillbaka, eftersom majoriteten av teamet är tekniker, vi kör tvåveckorssprintar. Vi kallar det projektplanering eftersom det i teknikervärlden går ganska långsamt. Det är ju det som är lite, då handlar det om att man felsöker eller bygger upp nya byggblock. Det går jättelångsamt, men vi håller ändå tvåveckorssprintar enligt *Scrum*-metoden.

- Det är mest jag som jobbar med kod, och då får upp ett flow, det är ju som sagt hur man ska jobba i *Git* och *code reviews*. Sen just mitt fall så är det så att vi är inte jättemycket folk som håller på att jobba med det här systemet, och vi som jobbar med det är lite utspridda över diverse team. Och där blir det också så att man bara skickar runt frågan om *code reviews* och sånt. Vi har en så kallad domänarkitekt då med, ja, men den form av *lead*-arkitekt eller vad man ska kalla det. En extra *arkitektig* arkitekt. Så man jobbar mycket så, sen kan man kanske skicka till dom som jobbar med själva vanliga kvalitet-teamet som finns. Så det blir ju lite på så vis med det här med att höra av sig.

Fråga 8a

8a.	Arbetar ditt team med verktyg för DevOps såsom CI/CD, exempelvis i Jenkins eller Microsoft Azure?	Tomas Gustavsson och Stefan Görling, Att Arbeta Med Systemutveckling (2017)	Eftersom vi arbetat med detta i kursen

Lukas: Ja, men okej! Jag kan väl börja. Det är ju inte någonting som vi utvecklar, men på Svenska Spel överlag är vi ju… Vi är ju väldigt roliga så, vi har ju kört främst hemmabyggt *ED*-flöde, där vi har Jenkins i botten och sen har vi en *eventkö*.

Och sen har vi ett stort *event*-byggt mikroserviceramverk, eller system med ramverk, som skriver pipeline. Det är egentligen bara hemmabyggda moduler, som system då, som utgör hela liksom Kuba-flödet [Kubernetes] och byggflödet. Det enda som är från gatan just nu är typ Jenkins, hela vägen ner till applikationen som installerar allting. Det är ett använt verktyg som kan kombineras. Men de bakas liksom in i de här applikationerna.

Just nu sitter vi och kollar lite på att byta ut det här mot saker, mer från gatan. Där är det liksom mer vanliga CI/CD -verktyg. Men vi kör ju på *on thread*, alltså saker i våra egna datahallar. Så då är det ju mer saker som går att köra i våra egna kluster, så då var det till exempel.. justerbara servrar som faktiskt är på CD eller +CD.

Jennifer: Vill du tillägga något?

Linus: Eh, ja… Jag har ingenting smart att säga om det här. Jag är ganska skonad från att vara inne i hela DevOps-flödet. Jag kollar bara lite Jenkins-loggar ibland när Trinidad inte vill bygga och, jag vet i alla fall att förr i tiden så releasade vi lite av våra backend-moduler genom att starta igång en Jenkins-pipeline. Annars så gör jag nästan inte någonting med Jenkins.

Lukas: Men egentligen så är det samma flöde för alla. Det är det som är enkelt, ett generellt flöde som hela företaget använder. Det som är jättesmidigt är att hela företaget nu körs på Kubernetes och alla kan skapa egna helm charts, etc.

Sophia: Så alla använder Jenkins men alla behöver inte tänka på att man använder det för det sker automatiskt?

Lukas: Exakt, så vad som händer är att vi kör Git bash som Git-system, så alla projekt, alla repos, skickas iväg som listade event så fort man gör en ändring, och då förstår den att: “Ah, men nu har någon gjort en ny commit här då gör vi ett nytt bygge automatiskt”, och sen drar Jenkins igång och och börjar bygga koden på just den nya commiten. Så på så sätt så är det väldigt väldigt mycket magi, som vi kan kalla det. Om du inte har jobbat med det [Jenkins] så kan man gömma sig ifrån det rätt bra om man vill.

Fråga 8b

8b.	Följdfråga: Erbjuder Jenkins värdefulla funktioner i verksamheten?		Uppföljningsfråga på föregående

Lukas: Ja, jo, men exakt. Poängen är ju att man inte behöver göra någon manuell handpåläggning mer än nödvändigt och en del av allt det här är ett flöde som håller koll på vilken commit eller version av koden det är. Ett problem är att det är lite för icke-transparent idag, det är lite svårt att veta vad som händer.

Fråga 9

9.	Är det möjligt att flytta produkten ni arbetar med till andra miljöer och programmeringsspråk?	McConnell (2004)	Intern kvalitetsegenskap: Portabilitet, återanvändbarhet, hållbarhet

Lukas: Alltså, andra miljöer ska inte vara något problem. Skulle man vilja så skulle det nog inte vara något jätteproblem, det är väl ramverket som finns i en annan språkkostym, men jag tror inte att det är några konstigheter att byta språk om man skulle vilja.

Linus: Mm, jag tror jag instämmer. Frågan är varför man skulle vilja byta programmeringsspråk kanske. Men då handlar det om ett annat läge.

Sophia: Ja det är väl kanske om företaget gör någon stor förändring i så fall och det kanske är då man får anledning till att titta på den där gamla koden man har skrivit med alla “hitta alla felen”.

Jennifer: Den temporära koden.

Linus: Rätt mycket skulle säkert gå att återanvända eller i alla fall koncepten bakom, men det skulle ju ändå bli nästan som att skriva en ny modul från början tänker jag men i ett annat programmeringsspråk.

Carl-Johan: Vilka språk jobbar ni i förresten?

Lukas: Jag skriver i Python.

Linus: Jag skriver JavaScript och C#.

Fråga 10

10.	Kan du/ni ge något konkret exempel på ett par tester som ni utför på koden?	McConnell (2004)	Intern kvalitetsegenskap: testbarhet

Lukas: Ja senast så… Jag har ett projekt som jag jobbar med just nu där jag har byggt en enkel Python-testaktivitet, som jag kör manuellt, men det är egentligen bara att den anropar modulerna i rätt ordning och kollar att det blir rätt. Jag har nog inga jättespännande saker just nu, Linus ni har ju väldigt mycket vettigt där. Jag jobbar bara med API:er faktiskt och skickar in en förfrågan, “Blev det bra?” “Jajämen” , det är ungefär det jag kan göra.

Linus: Ja men vi gör ju rätt mycket tester skulle jag ändå säga men sen har vi ju också folk som är anställda för att testa våra grejer. Både front-end testare som är inne på sajten och kollar att det fortfarande går att logga in och den här knappen ser fortfarande ser bra ut och den här menyn ser ut som den ska och allt sånt där. I kodbasen har vi enhetstester och auto-tester, eller GUI-tester. Enhetstesterna är väl lite si och så med hur mycket folk som brukar skriva det.

Egentligen så har vi sagt att vi ska ha ungefär 80 % coverage men det är också lite drömsiffror som man typ aldrig kommer upp i och det är inte alltid så användbart heller. Om man då sätter det kravet, som vissa tycker att man ska ha eller i bästa av världar skulle man ha, så blir det också tyvärr nästan omöjligt arbete med. Vissa saker kanske bara är copy-paste eller datamodeller som inte behöver några tester. Men vi har ju backend-modulerna, de exponerar ju API:en, där har vi back-end testare som skriver auto-tester så kommer det till oss per automatik. De här testerna svarar fortfarande med samma fel, det är de här parametrarna som är fel och den kan också svara med rätt svar.

Vi har alla de här enhetstesterna, vi har de här GUI-testerna som är de som körs när man till exempel releaser trinidad, den här webbservern, då snurrar det upp en headless Chrome som det heter, en lättviktig variant av chrome. Jag tror inte att man ser någonting i den utan den kör koden i chrome-motorn och så är det som att alla saker finns på plats som att det vore en webbplats genom till exempel element-id i HTML eller klasser på element i HTML och sådana saker. Om jag trycker på den här knappen så borde den här knappen vara synlig och

aktiv. Och sådant körs ju hela tiden. Så vi har både folk som manuellt sitter och klickar runt och sen så har vi också ganska mycket automatiska konfigurationer.

Fråga 11

11.	Vad finns det för fördelar med de tester ni gör?		Uppföljningsfråga på föregående

Linus: I front-end så finns det otroligt mycket fördelar tycker jag. Vi har ju också en rad olika testmiljöer som täcker olika saker, vissa är säkert lite överflödiga också. Vi har en pre-produktionsmiljö som ska vara mest publik. Testarna kollar alltid där innan saker skickas ut i prod som är sista steget innan vi går live. Innan dess har man en, jag vet inte om det alltid kallas *staging* men det kan man väl säga, som i casinot så brukar vi mest utveckla där på grund av leverantörgrejer. Testmiljöer som de kan tillhandahålla och så där. Det mest rätta sättet att göra det på är att man utvecklar i en miljö ännu tidigare än dess, så att man har en ganska “vilda västern”-miljö som är en lite mer säker miljö där man ändå kan börja testa saker, det bör funka i den här miljön. Sen har man en ännu mer pre-prod-miljö som är så publik som möjligt. Här ska allt vara inställt som det är i produktion. Sen kommer det ut i prod. På front-end är det väldigt väldigt viktigt och väldigt ofta som vi säger att den här saken funkar inte riktigt nu, den här menyn kommer inte upp nu när man scrollar så det finns super super mycket fördelar med testerna, alla testerna känns jätteviktiga. Enhetstester känns kanske inte alltid känns super super nödvändiga men det är de säkert ändå, de ligger där i bakgrunden och säkerställer att den här funktionen ändå gör typ vad det är tänkt att den ska göra.

Sophia: Vill du tillägga något Lukas?

Lukas: Nej, men det är som sagt, speciellt med enhetstester, särskilt fördelen där man faktiskt går in och gör om koden. Då är det ju väldigt trevligt att veta liksom att testerna är på en lagom hög nivå i koden, att gör du något längre ner – att framför allt veta att någon har sett till att det här fungerar. Så om jag laddar upp nytt innehåll då är det väldigt skönt om den meddelar: ’Oj, nu gick något sönder’. Så där är det väldigt skönt med *refactoring* och sånt där.Men, sen är det också mycket det här att kunna göra, i det fallet man gör nyutveckling, alltså innan jag skrivit programmet, att jag testar det. Mäta liksom att nu är appen klar. Då vet jag att här ska jag skicka in den här datan, och sen ska jag få ut den här datan. Då sätter jag upp de testerna och sen börjar jag skriva programmet.

Fråga 12

12.	Finns det några nackdelar med testerna?		Uppföljningsfråga på föregående

Linus: Ska du börja Lukas?

Lukas: Jag har inga problem direkt, men ett problem som mitt team som jag jobbar med har är att Continuous Integration kommer in för sent. Den typen av spel jag jobbar med är ganska

svårtestade, det är svårt att automatiskt testa såna spel eftersom du kan inte bara snurra upp applikationen. Det blir väldigt mycket jobb att skicka in all låtsasdata den behöver. På så sätt kan det vara att man behöver lägga väldigt mycket tid på att få ut all relevant data för att kunna mocka och skicka in sin fejkdata och få ut fejkdata. På så sätt vet jag att det finns vissa grejer som är helt hopplösa att fixa eftersom de är så beroende av externa system. Och så kan det vara ganska slitsamt att få upp vissa tester. Men där kanske man också får kolla på, där behöver man kanske inte ha, ehm, JUnit-tester och sådana saker, utan där är det kanske bättre att ha tester som man kör när allt är uppe.

Linus: För oss frontendutvecklare så har vi Trinidad… den här webbservern där man delar kodbas med så otroligt många. Där är det flera hundra utvecklare som varit inne i den koden. Dagligen är det säkert femtio pers som är inne och rör på den här koden. Så här är det väldigt viktigt att linters funkar, och att kodstandarden följs. Sen funkar det lite si och så (i verkligheten). Det innebär också att man vill ”deploya” samma sorts kodbas i slutet. Och hade jag exempelvis gjort någonting som tar sönder något som Lukas har byggt… nu jobbar vi inte ens i samma team, men jag kanske hade gjort något som tar sönder Lukas kod. Då är det ingen som kan gå runt med hela vår produktion innan det här är åtgärdat. Nån måste helt enkelt fixa det här. Och då är det stopp i det här flödet som ändå ska kunna gå hela tiden att ”releasa”. Och där har vi något som varit problematiskt. Ibland är det större problem, ibland är det mindre problem. Men det är viktigt att alltid kontrollera att man inte haft sönder testen, för då är det ingen som releasar någonting och då är inte folk glada.

- Vad finns det mer… Det börjar gå ganska långsamt då och då, det är ju en väldigt stor kodbas, och ibland kanske man inte får reda på att testerna inte funkar förrän trettio minuter efter man pushat sin kod. Och då har man väntat på det här i trettio minuter, och då får man återuppta var man var, och man ba ”Jaha nu får jag fixa det här”. Och rätt så ofta går dessutom de här GUI-testerna sönder utan att det är något fel på dem. Och då får man sitta och skriva ett test hundra gånger i en sån här merge request. Till sist går den genom utan att koden ändrade sig. Och då har man spenderat en halv dag på det här. Det är definitivt en stor nackdel.

Fråga 13

13.	Hur mycket tid ägnar ni åt testning under utvecklingen?		Intressant att veta vilken avvägning som görs mellan att spendera tid på tester och att spendera tid på utveckling i allmänhet

Lukas: … Nej, men det gör man väl. Testar man inte så vet man inte hur det går. Med jobb där man bygger saker brukar det vara mycket av den här ”slänga skit på väggen och se vad som fastnar”-metoden. Man tror att det gick att köra men det gick åt helvete, okej, då testar vi igen. Det är svårt att säga någon slags tid, men testar man inte så vet man ju inte var felen finns eller ens om det fungerar. Det är klart att man lägger en rimlig tid i apputvecklingen på

att se att det blir rätt. Men då är det lite mer att man manuellt testar. Där lägger vi ganska mycket tid, på saker man kanske inte kan autotesta, utan det är mer… fungerar det? Nej. Ok.

Linus: Jag tror också att det handlar om hur man definierar att testa. Det är klart att man testar det man gör, men kanske inte alla scenarion som man kan tänka på kanske, det är därför man har de här regressionstesterna som de [personerna] som är avsedda för att testa gör. Men samtidigt, jag skriver inte koden och utan att köra den en enda gång sen skickar den vidare till test, så funkar det ju inte. Man testar ju allt man gör medan det skapas så gott man kan. Och sen får [testarna] kolla så att det funkar med allt annat i ett makro[perspektiv], liksom.

- Och sen är det det här med att skriva in stress- och GUI-tester och sånt, det är det otroligt sällan som jag gör. Det borde man nog göra oftare. Men där lägger jag väldigt lite tid. Mindre än en procent av all utvecklingstid i alla fall.

Fråga 14

14.	Hur tror ni att arbetet med tester kommer att utvecklas i framtiden? Exempelvis, vilka typer av tester kommer ni arbeta mer/mindre med?		Värdefullt att veta hur det långsiktiga tänket ser ut.

Lukas: Jag blir lite sugen på mer meta-miljötester (?) alltså att man dynamiskt snurrar upp en helt separat miljö med eget beroende. På Svenska Spel har vi ett gäng permanenta… testmiljöer kan man kalla det. Ett antal uppsättningar som gör likadant. Man snurrar upp den med de beroenden man tror att man har. Behöver man en databas för produktionen så snurrar man upp en tillfällig databas. Man snurrar upp allting man tror man behöver och sen skickar man in någon form av testsystem i det. Då blir det ganska enkelt att testa var man är just nu i en isolerad miljö, utan att man förstör för någon annan på något annat ställe. Och då kan man även köra lite aggressivare tester, typ ”Vad händer om jag spränger databasen?” Typ Chaos Engineering där man försöker knäcka systemet för att se hur robust det är.

Linus: Vi måste jobba lite mer i framtiden på devicetester. Det är ett ganska stort problem för frontend. Att kunna testa på ipad bland annat och andra plattformar som vi inte kodar direkt på.