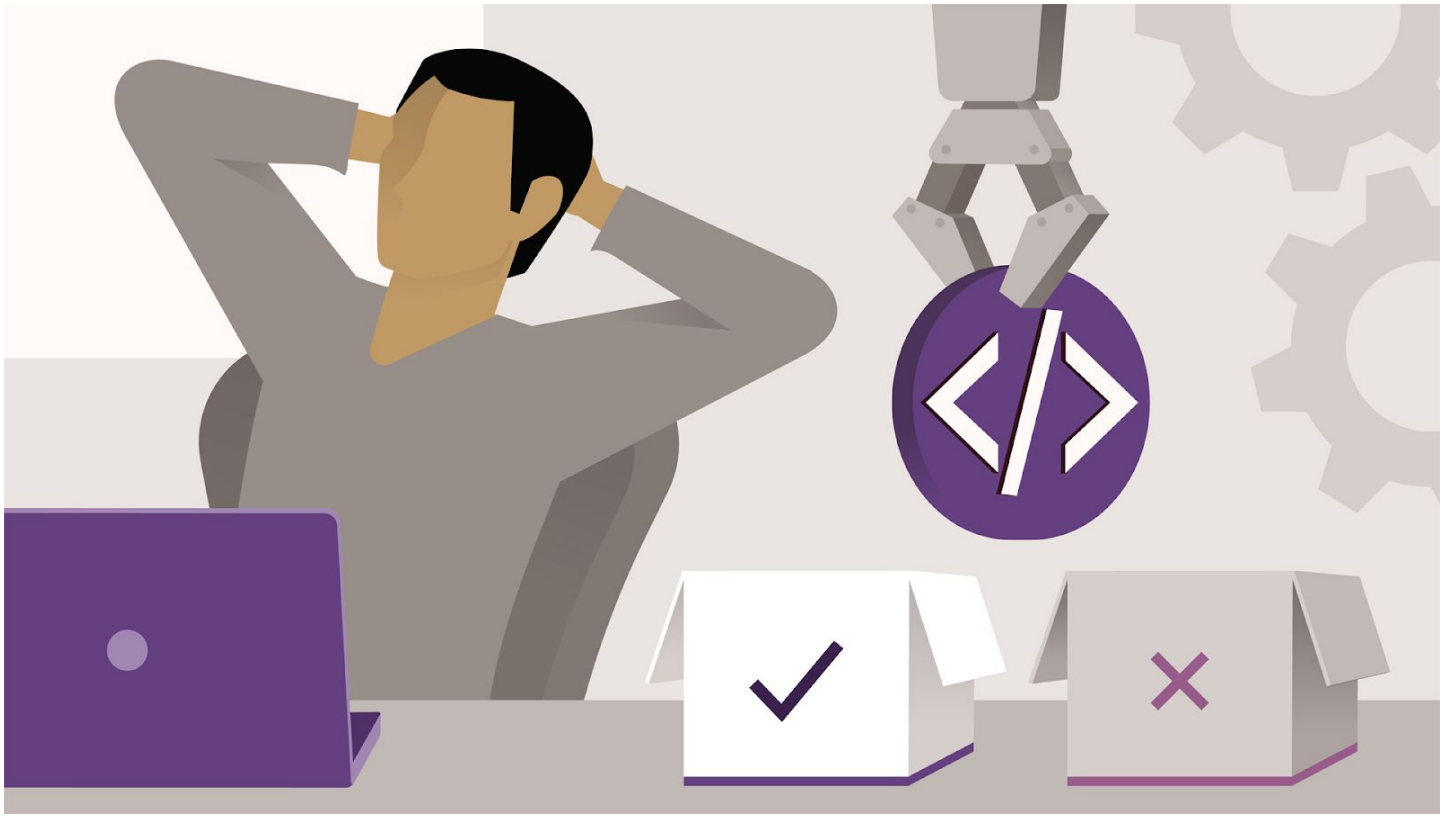
Newton

Testautomationsprocesser

[2020-05-28]



Examensarbetet
Testare 1
vårterminen 2020

Författare:

Rizgar Teimouri
Vidar Zingmark
Kevin Zhang

Newton™

Sammanfattning

Vi har undersökt på hur man applicerar testautomatiseringsprocesser i ett projekt på företag. Testautomatisering är ett ämne som på senare tid har blivit väldigt hett i arbetsmarknaden. Detta har lett till att arbetsuppgifter inom test har möjligheten att effektiviseras markant jämfört med tidigare perioder.

Test säkerställer att produkten som utvecklas håller den önskade kvalitet som kunden förväntar sig.
Automatiserad testmiljö är många företag intresserad att investera i men att kunskapen och hur man ska gå tillväga för att implementera det och bibehålla testmiljön effektivt är den svåra delen som uppstår för många företag.
En av anledningarna till att vi valde just detta ämne att djupdyka i är att det är extremt relevant för vår kommande yrkesroll och vi alla är på ett eller annat sätt intresserade av att jobba med testautomation.

Med denna rapport och vår undersökning ska vi genom litteraturstudie och föreläsningar komma fram till hur man uppnår en effektiv automationsprocess och vad en effektiv automationsprocess innebär inom vårt ämne. Vi har avgränsat arbetet till detta specifika område, vi tar upp manuella tester som ett argument för hur dessa två kan komplementera varandra för en effektiv testautomationsprocess.

Newton™

Innehållsförteckning

Newton™

Begreppsförklaring

Nedan förklarar vi termer som används i rapporten. Förklaringen finns för att underlätta för läsaren.

Annuitet
Inbetalningsöverskott per år

ROI (Return On Investment)
Avkastning på investering. Används för att beräkna ifall en investering är värd i längden.

GDPR (General Data Protection Regulation)
Dataskyddslagen för personliga uppgifter

Enhetstest
Tester som utförs på kodnivå

Integrationstester/API tester
Tester som utförs på systemnivå där olika system kommunicerar med varandra för att hitta eventuella fel/buggar.

Miljö/miljöer
Miljö där tester utförs

DevOps (Dev - development, Ops - Operations)
En metodik för snabba cyklar

Outsourcing
Man låter annat företag sköta en eller flera processer åt en för att man ska kunna fokusera på sin kärnverksamhet.

Testfall
Begrepp inom programvarutestning där det beskriver en händelse och hur produkten ska bete sig vid givna parametrar. Det används för att säkerställa att produkten fungerar som det är tänkt genom att man inför steg som ska reproduceras och resultatet ska stämma överens med det förväntade resultatet.

Problemformulering

Introduktion

Automationsprocessen är en process som på senare tid spridit sig över de flesta arbetsmarknader och har effektiviserat alla möjliga arbetsuppgifter kring olika yrkesroller. Automationsprocesser består av datorer som oftast gör retroaktiva uppgifter, men kan även genomföra fullständiga flöden. I dagens läge finns det köpsystem som är automatiserade, fordon med automatiserade styrsystem, men även automatiserade tillverkningar av dessa maskiner. Man kan nästan säga att allt är på väg till ett automatiserat samhälle. Där arbetsgivare vill uppnå automation för att uppnå kostnadseffektivitet (Inkomst/utkomst) eller (Prestation / lön).

När det ständigt lanseras nya uppdateringar blir systemen allt mer komplexa. Det behövs mer verifieringar att systemen och processerna fungerar som förväntat, annars kan det förekomma oväntade dyra kostnader. Därför behöver man testare som genomför tester för att säkerställa att systemen fungerar som de ska. Dock kan det ta upp emot veckor eller månader beroende på vilka omständigheter testerna har, för att kunna genomföra tester av olika slag. Därför bör man värdesätta att implementera automatiska flöden.

Inom detta projekt har vi valt att djupdyka in i just automationsprocessen inom testning, då det är relevant för vår yrkesroll och är ett område många företag har fått upp ögonen för.

Frågeställningar

I projektets gång kommer vi att besvara och diskutera dessa frågeställningar:

- Vad kan testautomatiseras för att det ska bli en så effektiv automationsprocess som möjligt?
- Vad innebär en effektiv automationsprocess inom respektive område för när man testar...?
- När är det mer effektivt att bibehålla manuella tester?

Avgränsningar

Eftersom området är väldigt stort och kan expanderas lätt, vill vi avgränsa projektet till en viss grad. I projektet kommer vi att beskriva översiktligt om testautomationsprocesser för att skapa en uppfattning för läsaren så den kan ska förstå helheten med projektet. Vi kommer dessutom bidra med information kring subjektet för att styrka textens förståelse men behålla fokusen på frågeställningarna.

Denna rapport kommer endast gå in på testautomation. Mer specifikt går vi in på hur man uppnår en effektiv automatiseringsprocess samt hur manuella tester kan bidra till en effektiv automationsprocess.
Vi kommer även gå in på vad en effektiv automationsprocess innebär.
Vi kommer inte gå in på andra områden inom test eftersom det inte kommer bidra till våra frågeställningar och det som vi undersöker för denna rapport.

Examensarbetets Mål

Syftet med detta examensarbete är att beskriva hur testautomationsprocesser kan appliceras i ett projekt på företag. Dessutom skapas en uppfattning om vad som kan automatiseras och vad som bör behållas manuellt. Målet med projektet är att ge läsaren en förståelse för de frågeställningar som har ställts upp och vad de slutsatser och analyser som vi har kommit fram till medger.

Nulägesbeskrivning

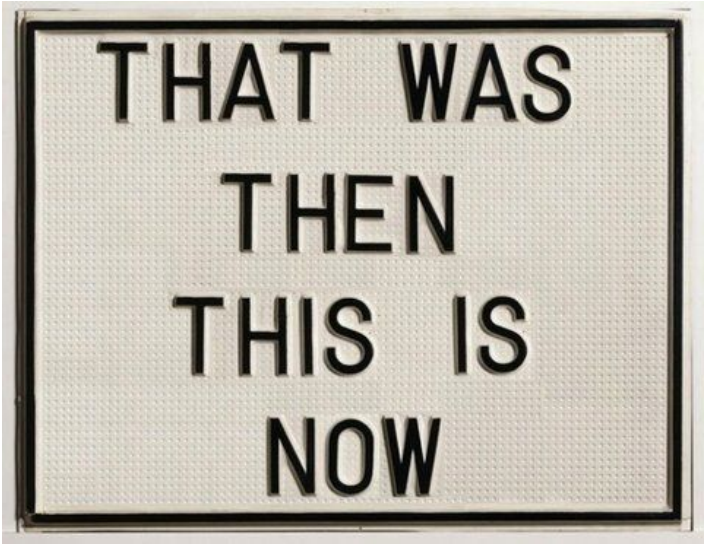
Företag har sedan flera decennier tillbaka försökt att minska sina utgifter markant, genom att använda sig av olika typer av metoder som maskiner och redskap. I nuläget har vi kommit till ett läge där företag vill implementera “automatiska flöden”, som resulterar till att allt sker per automatik. Ju mer som automatiseras desto viktigare blir det att våra IT-system fungerar som de ska, då IT-systemen utvecklas och uppdateras mer frekvent än förut. Systemen blir även allt mer komplexa vilket resulterar till att testprocesser tar allt längre tid att genomföra.

I nuläget jobbar många företag fortfarande i den gamla standarden genom att testa IT-system manuellt, som drar en mycket större kostnad än om man skulle automatisera testningen.

Det finns många fördelar, men även en del nackdelar med att implementera automatiska testflöden som bör diskuteras. Om vad som överväger vad och huruvida det är värt att implementeras eller inte.

Några nackdelar med att implementera automatiska processer är att det lätt kan gå över projektets budget. Det blir också allt svårare att hitta kompetenserna till arbetet ju mer komplext systemet är. Dessutom kan det även bli dyrt att underhålla beroende på hur bra planeringen till projektet genomförs och så kan det ta tid att implementeras.

Fördelar med att implementera automatiska processer är att det sparar tid åt testarna, utvecklarna får snabbare respons, mindre anställda behövs då färre mänskliga behov finns och testerna kan köras när som helst på dygnet. Generellt sätt sparar företag både tid och pengar i längden, vilket gör det väldigt attraktivt för många företag att implementera dessa processer.



Problemet i nuläget:

Problemet i nuläget som sagt är att företag fortfarande arbetar i de gamla standarderna och har ibland inte hört talas om “investering i testning”, då det är utvecklare som testar majoriteten av tiden. Det man vill genomföra med testning är att validera att ett system fungerar som det ska, och verifiera att utvecklaren har utvecklat korrekt samt att kunderna fått det dem vill ha.

Vissa chefer uttalar sig “Varför ska vi spendera mer pengar på testning, när utvecklarna redan testar sin kod?”. Det här är en vanlig situation i en testares vardag och kan besvaras med “Det är lättare att korrekturläsa en annans text än ens egen”. Detta appliceras även på kod och kodens funktionalitet. Även projektledare inom applikationsutveckling upplever att arbeta i team är alltid bättre än att arbeta ensam, genom parprogrammering eller andra eventuella par uppgifter.[1]

Just nu levereras systemen kontinuerligt och det blir allt svårare att hänga med i utvecklingen med endast manuella tester, eftersom behovet att utföra tester i samma utsträckning är mycket högre nu än vad det har varit tidigare. Allt fler företag outsourcar sin kod/produktion för att kunna spara in pengar och därmed lättare kunna fokusera på det dom är bra på. Dock följer det med nackdelar som: “Om de inte håller kvaliteten uppe kan det ställa till med problem gentemot slutkund.”[2]

Det här gäller även kommunikationen mellan företag som har outsourcat då det behövs valideringspunkter att systemet är korrekt enligt avtalen. Dessa valideringspunkter är tester, och utförs av testarna på respektive bolag. Om en fördröjning skulle uppstå skapas en kedjereaktion[3] som kan leda till projektförseningar, eller ännu värre missnöjda slutkunder. Valideringspunkter kan även fördelas i olika nivåer från hela projekt till en liten funktionalitet i ett system.

Oftast slutar det med att testarna inte har tiden till att testa allt, och måste fokusera på det viktiga där tid är viktigare än kvalité. Det här kan vara ett problem då utvecklingsteam har höga krav på kvalité och därmed kanske inte uppnår de standarder som är satta.

[1] Oförberedd Intervju med Projektledare
[2] Redaktionen i Affärsutveckling 2009 paragraf nackdelar med outsourcing https://www.foretagande.se/outsourcing-aven-for-sma-foretag
[3] Begrepp https://sv.wikipedia.org/wiki/Kedjereaktion

Konsekvenser och möjligheter som vi upplever i nuläget:

Eftersom företag i nuläget lider av tidsbrist för att uppnå den kvalité som efterfrågas så gäller det att arbeta mer metodiskt, d.v.s. “work smarter, not harder”. Människan har alltid varit duktig på att förbättra tekniker för att gynna tiden och prestationen till arbetet. Till exempel har vi inte bara en gaffel, utan även en kniv och en sked. Där alla föremålen har en betydelse, men vi använder fortfarande våra “biologiska redskap” händerna till att äta saker.

Samma sak gäller för automatiska tester, vi använder de redskap vi behöver när de är gynnsamma. Men med alla förändringar som kommer, så följer det med för- och nackdelar. Så vilka för- och nackdelar ser man utifrån de konsekvenser företag upplever?

Nackdelarna med att implementera automatiska tester är att:

- Det kostar pengar då kompetens kan vara limiterat, dyrt att underhålla beroende på projektändringar eller komplexitet.
- Att implementera automatiska tester kan ta tid och därmed behöver man utvärdera vilka tester som bör automatiseras.
- Målet med automatiska tester är en vinning i framtiden, om något är väldigt aktuellt kan det dra fokuset från nuvarande releaser och därmed inte leverera lika bra om tiden blev tillbringad på att testa.

En annan nackdel är att en del företag tror att man kan automatisera alla sina tester och att de inte behöver några manuella tester alls. Det stämmer inte eftersom automatiserade tester ska kompletteras med manuella tester och inte ersätta manuella tester helt. Om man har både manuella och automatiska tester kommer det att resultera i att testprocessen kommer fungera bättre eftersom båda dessa testmetoder kompletterar varandra.

Dessa nackdelar är något som tas in i beräkning innan man bestämmer sig för att automatisera testprocesser. Det kostar mycket pengar att implementera automatiska tester och det drar resurser för att underhålla automatiska tester. Alla automatiska tester ska inte automatiseras eftersom ibland har inte en dator samma kapacitet som en människa att se förändringar/fel, vilket innebär att vissa tester är mer lämpliga att utföra manuellt. Men man ska absolut ha som åtanke att implementera automatiska tester då det blir mer produktivt när man väl finner lösningen till det. Dock ska man även fundera över ifall det är värt det och ifall man verkligen sparar mycket på det.

Fördelar med att implementera automatisera tester är att:

- I längden sparar man arbetskraft då testarna kan fokusera på att producera mer tester eller förbättra/underhålla existerande tester.
- Alla i teamet får en snabbare feedback, t ex till utvecklare, kravanalytiker eller andra intressenter. Detta leder till mer kontinuerliga flöden.
- Mindre mänskliga fel, mindre differenser då en dator alltid utgår från samma punkt.

En annan fördel med automatiserade tester är att de kan bli exekverade när som helst på dygnet, kan pågå längre än vad en arbetsdag på 8 timmar och kostar nästan inpå ingenting att genomföra. Man ska ta in som åtanke att inte ersätta alla manuella tester med automatiska tester då det kräver kompetens att skapa dessa testprocesser och underhålla, där tiden och pengar är stora faktorer.

Innan man implementerar testautomatisering ska man tänka på hur det kommer att löna sig långsiktigt i tid och pengar. ROI (Return Of Investment) är ett begrepp som används för att beräkna lönsamheten. ROI beräknas genom att man dividerar vinsten från en investering med kostnaden för investeringen (annuiteten / grundinvesteringen).

En annan faktor att tänka på är att maskiner inte har människans beslutstagande, det innebär att en människa kan tänka utanför ramarna. En maskin kan inte utforska ett system som en duktig testare skulle kunna göra i nuläget. Exempelvis kan inte en maskin reflektera över visuella aspekter på ett system, hur det ser ut som exempel. Man måste också reflektera över vilka tester som man ska automatisera samt vad man kommer tjäna på i längden.

Idag finns det även många olika enheter som ett system måste fungera på och i sin tur ställer höga krav på testning av ett system. Det innebär att man måste testa mer frekvent och på många olika enheter. Därför vill många företag automatisera många av deras tester. I överlag får man bättre kvalitet och man sparar tid och pengar i längden.
"Testautomatisering blir snarare ett viktigt stöd och komplement till den manuella testningen och ytterligare ett verktyg i vår testlåda i vår jakt på bättre kvalitet."[4]

[4] https://www.frontit.se/inspiration-kunskap/artiklar/testautomatisering-det-bra-det-daliga-och-den-stora-fragan/

Metodbeskrivning

Hur kommer vi ta in information?

Metoderna till att samla information angående ovanstående punkter kommer att genomföras genom internet och från föreläsningar inom och utanför skolan. Sedan finns det även teorier och metodologier kring olika typer av testning som vi kommer att dra upp. De har en påverkan/utveckling inom ämnesområdet och kan skapa intressanta tankebanor och diskussioner.
Informationen tas in från besök av föreläsningar och information från internet.

Litteraturstudie

Denna rapport består av och samlar information från litteraturstudier. Litteraturen består av information om automatiserade tester, hur man effektiviserar automatiska testprocesser och vilka verktyg/metoder som är lämpliga för en effektiv automatiseringsprocess.

Föreläsningar

Information som vi har tagit del av är i form av seminarier där vi har fått viktig information från erfarna testare som arbetar inom testautomation. Dessa föreläsningar går igenom hur man kan implementera automatiska tester, vad man bör tänka på och vilka typer av automatiska tester som ska göras. Den här typen av information är bra för att koppla med nuvarande arbetsprocesser och hur företag arbetar.

Föreläsningen hölls av en konsult från företaget ADDQ och föreläsningen pågick i en timme. Denna föreläsning gav ett stort underlag för vårt arbete och vi kunde utföra en fördjupad undersökning gällande automatiska testprocesser.

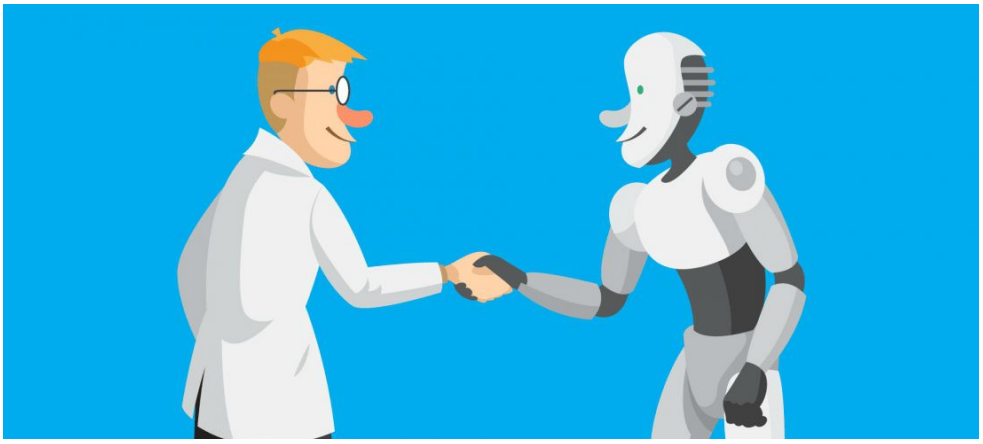
Resultatredovisning

Föreläsningar

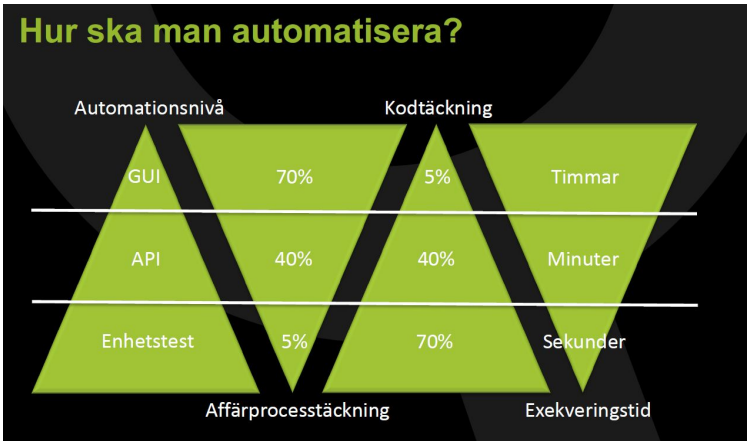
Föreläsning på ADDQ

En av föreläsningarna som vi har tagit information ifrån är besöket på ADDQ i Stockholm. Föreläsningens agenda var “Så kommer du igång med testautomatisering” och presenterades av en testautomatiseringskonsult. Det som presenterades på ADDQ var en introduktion till testautomatisering, hur man kommer igång med automatiska processer och rekommendationer för en effektiv testautomatiseringsprocess. Föreläsningen var informativ och gav oss en hel del information inom testautomationsprocesser och mycket runt omkring subjektet.

En viktig punkt som togs upp på föreläsningen var balansen mellan GUI (Graphical User Interface) tester, API (Application Programming Interface) tester och enhetstester (Programmeringskod). Där benämndes det att prioriteringen av de olika nivåerna som ska testas är oberoende av varandra. Varje nivå har sina prioriteringar och för att fylla en bra grund ska de tyngst påverkande testerna prioriteras först.
En annan punkt som togs upp på föreläsningen var fördelningen mellan manuella och automatiska tester. Där jämförde man om det var mer effektivt att implementera automatiska tester jämfört med manuella. Jämförelsen hade olika faktorer som påvisade när man borde/inte borde försöka sig på att automatisera ett område. En av föreläsarens teser var att “alla tester ska inte automatiseras, då automatiska tester komplementerar manuella tester “.[5]



[5] https://www.addq.se/inspiration-kunskap/testautomatisera-strategiskt-sa-ska-du-tanka



På presentationen visades en bild på hur man ska tänka när man prioriterar vilka tester som ska automatiseras. Syftet med bilden är att man ska ge publiken en uppfattning på hur man ska gå tillväga för att uppnå en effektiv testautomatiseringsprocess.

Bilden ovanför representerar en bild från föreläsningen.

I sektionen “automationsnivå” ser man tre typer av tester; enhetstest, API-test och GUI-test’

- GUI-nivån är det som användaren kan se d.v.s. användargränssnittet på ett system eller en webbsida.
- API-nivån är det som användaren inte kan se. API består främst av kommunikationen mellan program.
- Enhetstest-nivån är den “lägsta” nivån där koden till programmet skrivs och testas.

I den andra sektionen “affärsprocesstäckning” handlar det om hur stor del av arbetet som slutkunden värdesätter. T.ex. om ett fel skulle uppstå i GUI-testerna skulle det vara mer kritiskt då kunden har en närmare koppling till GUI:et än funktionerna i sig.

I den tredje sektionen “kodtäckning” har vi de olika nivåerna och deras påverkan på kodtäckningen. Kodtäckningen är antalet kodrader exekverade dividerat med antalet kodrader producerade i ett projekt. “Genom enhetstester kommer vi med liten ansträngning kunna exekvera cirka 70 procent av alla kodrader, 40 procent via API-tester och 5 procent via GUI-tester.”

Den fjärde och sista sektionen visar på hur lång tid det tar att testa respektive område. Enhetstester tar bara några sekunder att utföra eftersom man tester koden och dess funktioner. API-testerna som är ett steg upp i pyramiden kan ta några minuter att utföra eftersom man testar kommunikationen mellan system och dess funktioner. GUI-testerna är nivån som tar längst tid, då man testar användargränssnittet på ett system. Dem här typerna av tester kan ta flera timmar att utföra.

“Genom att arbeta enligt ett strategiskt tankesätt med en bra fördelning mellan de olika nivåerna, kommer det bli enklare för dig att lyckas med automatisering. Ingen nivå är i sig

den bästa, eller den som är mest “lagom”. Det är kombinationen av alla pyramider - automationsnivå, affärsprocesstäckning, kodtäckning och exekveringstid.”[6]

Projektstyrning

DevOps

DevOps (Dev - development, Ops - Operations) är en metodik inom agil utveckling för att det finns stora behov av snabba cykler i arbetsmarknaden. DevOps handlar om samarbetet mellan utveckling och drift. Där för man samman de som utvecklar och de som ansvarar för infrastrukturen. Man bygger, testar och lanserar ofta, snabbt och tillförlitligt.[7] Tanken är att man förenar människor, processer och teknik för att ge ett kontinuerligt värde till kunderna.

DevOps för teamen kan göra så man får bättre samordning och samarbete för att producera bättre och pålitligare produkter. Teamen kan lättare leverera efter kundernas behov, öka tillförsikten i applikationerna, samt skapa och uppnå snabbare affärsmål. Fördelarna med DevOps resulterar i att teamen blir högpresterande och utifrån metoderna skapas bättre produkter.

De här förbättrade funktionerna för samarbete och produktivitet är också essentiella för att uppnå följande affärsmål:

- Kortare tid till marknaden
- Anpassning efter marknad och konkurrens
- Bibehåller systemets stabilitet och tillförlitlighet
- Förbättrar snittiden för återställning

DevOps har en applivscykel bestående av planerings-, utvecklings-, leverans- och körningsfas. Varje fas är beroende av varandra och faserna är inte rollspecifika.

Plan

I planeringsfasen skapar teamen en uppfattning om funktioner och egenskaper hos applikationer och system de skapar. Här spåras hela förloppet - från uppgifter för enskilda produkter till uppgifter som rör flera produkter.

Utveckling

I utvecklingsfasen kodar, testar, granskar samt integration utförs av teammedlemmar. Man distribuerar koden till olika miljöer. I utvecklingsfasen vill man skapa innovationer snabbt utan att offra kvalitet och stabilitet samt hålla upp produktiviteten. För att uppfylla dessa mål använder man högproduktiva verktyg, man automatiserar exempelvis repetitiva manuella steg och itererar i små steg genom automatiserad testning och kontinuerlig integrering.

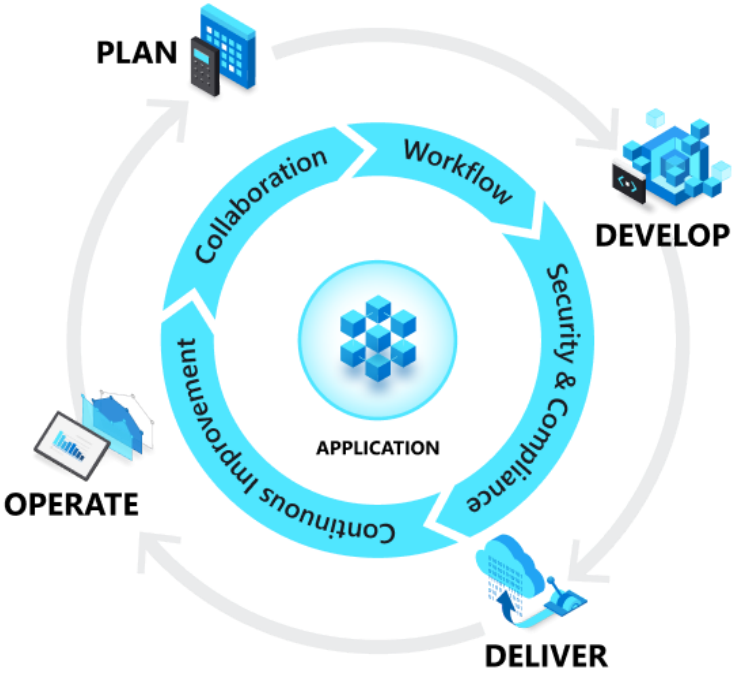
[6] https://www.addq.se/inspiration-kunskap/testautomatisera-strategiskt-sa-ska-du-tanka
[7] Frontit. DevOps, CI, CD – vad betyder det egentligen? https://www.frontit.se/inspiration-kunskap/artiklar/devops-ci-cd-vad-betyder-det-egentligen/ (Hämtad 2020-02-04)

Leverans

I leveransfasen distribuerar man applikationer till produktionsmiljöer på en så konsekvent och tillförlitligt sätt som möjligt. Här distribuerar och konfigurerar man grundinfrastrukturen som utgör miljöerna. Teamen definierar en versionshanteringsprocess och flyttar applikationer mellan olika steg tills de blir tillgängliga för kunder. Det är med fördel att automatisera dessa processer för att de blir skalbara, upprepningsbara och kontrollerade. Detta säkerställer också att produkter levereras med lätthet och trygghet.

Körning

I körningsfasen underhåller, övervakar och felsöker man applikationer i produktionsmiljö. Teamen arbetar för att säkerställa systemtillförlitlighet och siktar på att systemen är tillgängliga hela tiden. Här vill man identifiera problem innan de påverkar kunderna och åtgärda problemen snabbt när de inträffar.[8]



[8] Microsoft. Vad är DevOps? https://azure.microsoft.com/sv-se/overview/what-is-devops/ (Hämtad 2020-02-04)

CI/CD Continuous Integration, Continuous Delivery & Deployment

Utöver att etablera DevOps-kulturen implementerar man vissa metoder i applivscykeln. En metod som man brukar använda sig av är CI/CD, vilket är en metodologi för att effektivisera flödet mellan kund och utvecklingsteam. Den här typen av metod används idag inom många organisationer och kommer säkerligen vara aktuell inom de kommande årtiondena.

Så, vad är CI/CD? CI som står för continuous integration och CD för continuous delivery samt deployment är ett koncept som används för att skapa så kallade Continuous deployment (CD).[9] Continuous deployment är ett tillvägagångssätt mot programutveckling där funktionaliteter är frekvent levererade genom automatiserade leveranser.[10]

CI/CD är hela processen och i en liten del av processen ingår det så kallade microservices som är en programutvecklingsteknik där testning är ett delmoment.[11] I CI/CD brukar man säga att processen är “pipelines” som på svenska motsvarar “rörledning”. Om man stoppar dessa processer genom att göra saker manuellt säger det sig själv. D.v.s. hela processen blir inte kontinuerlig och CI/CD står för “continuously”, som förstör betydelsen med hela metodologin.

CI är en metod där ett team av utvecklare integrerar sin kod tidigt och ofta till “code repository” som motsvarar “kodförvar” på svenska. Varje integration verifieras av ett automatiserat bygge, som också exekverar tester för att snabbt, tidigt och automatiskt identifiera eventuella integrationsfel. Detta implementerar man för att reducera risken för “Integration Hell” som syftar på att varje teammedlem integrerar sin egna kod i slutet av en deadline. Integration Hell leder i slutändan till att det tar mycket längre tid att integrera all kod med varandra för att man behöver fixa buggar.[12]
En av de bästa fördelarna med CI är att det sparar tid under utvecklingscykeln genom att identifiera konflikter tidigt. Dessutom leder det till bättre teamkoordination vilket resulterar till bättre kvalitet och snabbare “releaser” på mjukvarusystemen än om man skulle använda traditionella programvaruutvecklingsmetodiker.

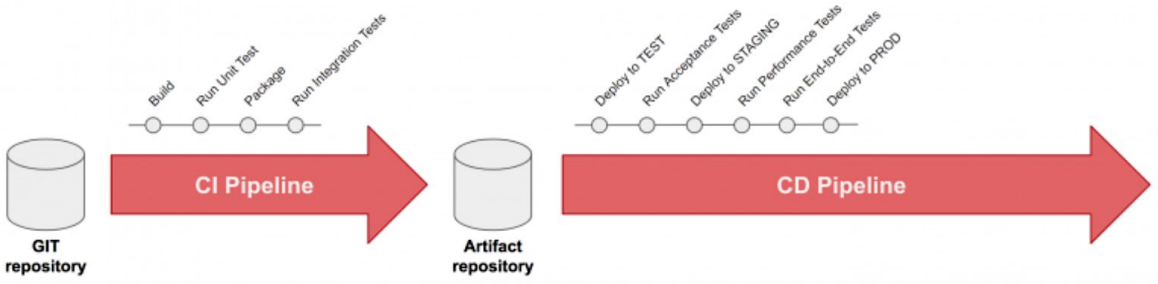
CD är efterföljaren till CI där man bygger continuous delivery-metoder ovanpå CI-metoder för att distribuera koden till slutanvändarna. Efter att man automatiserat sina tester i CI, automatiserar man releaseprocessen och man kan distribuera sina system när som helst genom ett knapptryck.
Genom att utnyttja fördelarna med continuous delivery distribuerar man sina system så ofta som möjligt i små batcher så att man kan felsöka ifall det uppstår buggar i systemen.

[9] https://en.wikipedia.org/wiki/CI/CD
[10] https://en.wikipedia.org/wiki/Continuous_deployment
[11] https://en.wikipedia.org/wiki/Microservices
[12] https://www.solutionsiq.com/agile-glossary/integration-hell/

Continuous deployment går ännu djupare än continuous delivery. Här distribueras alla ändringar man gör i “prodkutionspipelines” automatiskt till kunderna. Det är bara ändringar som stoppats av testerna som inte går till produktion.
Continuous deployment tar bort trycket från teamet för att det inte finns någon releasedag. Utvecklarna kan fokusera på att utveckla deras produkter och dom ser sitt arbete gå i produktion minuter efter att dom blev klara.

Sammanfattat är CI både en del av continuous delivery och continuous deployment. Continuous deployment är samma som continuous delivery förutom att i continuous deployment distribuerar man sina byggen automatiskt.

En övergriplig bild på hur CI/CD fungerar:



Vad är så bra med det?
Fördelar med CI:

- Automatiska tester exekverar regressionstester så fort något nytt läggs till i systemet och fångar upp eventuella buggar vilket leder till att man kan åtgärda det snabbt och man blir mer säker på att systemet blir så buggfritt som möjligt när den sätts i produktion.
- Det är lätt att bygga releasen då all integrationsproblem blivit lösta tidigt i projekten
- Utvecklare blir varnade så fort de förstör bygget och kan fokusera på att åtgärda det innan de går över till nästa uppgift.
- Testkostnader blir mycket lägre då det bara tar några sekunder för CI-servern att köra testerna.
- QA-teamet spenderar mindre tid på testning och kan fokusera på förbättringar på systemet.

Fördelar med continuous delivery:

- Komplexiteten med att distribuera systemen försvinner helt. Teamet behöver inte spendera flera dagar på att driftsätta systemen och kan fokusera på att bygga ett så bra system som möjligt.

- Man kan driftsätta kontinuerligt, därav namnet, vilket leder till att man får feedback på systemen oftare av kunder.

Fördelar med continuous deployment:

- Man behöver inte pausa utvecklingen mitt i en release. Distribuering sker automatiskt när man gör en ändring.
- Eftersom man distribuerar små ändringar åt gången blir det enklare att åtgärda eventuella buggar som uppstår. Dessutom blir det mindre riskfyllt.
- Systemkvaliteten ökar varje dag istället för varje månad eller varje kvarter. Detta leder till att kunderna ser förbättringarna som görs hela tiden.[13]

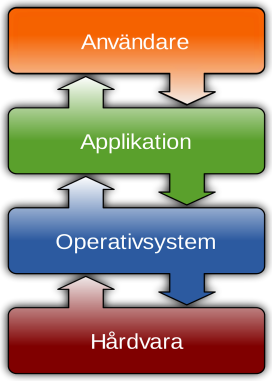
13

https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Datalager

IT-världen är enormt stor där system är byggda på olika sätt, olika programmeringsspråk, gemensamma databaser, icke-gemensamma databaser etc. Det finns miljontals olika skillnader på system som även härstammar på samma problem. Dock har vi alla en gemensam grund och det är IT-arkitekturen.

IT-arkitekturen är ett sätt att organisera resurserna och komponenterna i ett datorsystem.[14] Det beskriver de olika komponenterna mellan datorer där komponenter delas in i mjukvarumoduler. Mjukvarumoduler kan även beskrivas som en "organiserad samling av maskininstruktioner" som har en avsedd uppgift på ett datorsystem.[15]



Bilden till vänster beskriver de olika skikten som man kan kategorisera dem olika komponenterna i där programvara skapas med hjälp av programmeringsspråk.

Dem flesta utvecklingsprocesser genomförs i applikationslagret, men det finns även företag som arbetar med operativsystemlagret, men dessa är ytterligt få jämfört med dem som jobbar i applikationslagret.

Även om IT- arkitekturen är organiserad blir det fortfarande en vag bild på vad respektive område representerar. Därför fördelar man olika arbetsuppgifter till två olika kategorier som är frontend som står för "framdel" och backend "bakdel". Dessa är generella beskrivningar då arbetsprocesser oftast utvecklas efter dessa beskrivningar då det är enklare att kategorisera. När man som testare arbetar inom automatiserade processer kan man göra tester både mot framdelen och bakdelen.

Frontend är den del av systemet som användaren kan se, t.ex. användargränssnitt och backend är allt som användaren inte kan se som t.ex. en databas eller kod.

Bilden till vänster är en visuell beskrivning på de två områdena.

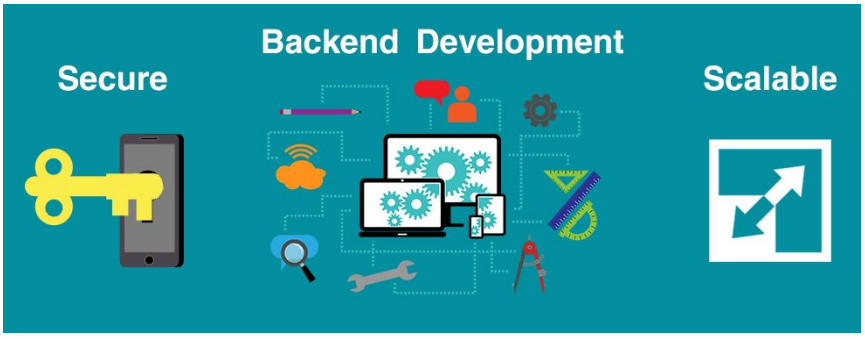
[14] https://sv.wikipedia.org/wiki/IT-arkitektur
[15] https://sv.wikipedia.org/wiki/Programvara

Frontend

Frontend associeras ofta med webbaserade mjukvarumoduler som är baserade på **HTML** (Hyper Text Markup Language), **CSS** (Cascading Style Sheet) och **JavaScript**.[16] Frontend associeras ofta med vad användaren ser och gör men inte själva funktionernas djup. Dessa saker kan t.ex. vara texter, bilder, länkar, formulär eller knappar etc.[17]

Backend

Backend associeras ofta som serverside eller databashantering som baserar sig på SQL (Structured Query Language), Python, C#, Ruby och PHP (Personal Home Page Tool).[18] Backend associeras ofta med “det programmet gör” eller “det användaren inte ser”. Dessa saker kan t.ex. vara databasinformationtaganden, kalkyleringar eller basfunktionaliteter.



[16] https://sv.wikipedia.org/wiki/Front-end_och_back-end
[17] https://www.weahead.se/vad-ar-front-end
[18] https://www.avense.se/artiklar/skillnaden-mellan-front-och-back-end/

Tre typer av automatiska tester:

Det finns många olika sätt på hur automatiska tester genomförs i en testautomationsprocess men de vanligaste brukar oftast vara:

- **Enhetstester**
- **Integrationstester/API tester**
- **Gränssnittstester**

Enhetstester:

"Testerna skrivs i samma språk som programmeringen görs i och syftar till att kontrollera att specifika delar av koden (metoder eller komponenter) beter sig på ett förväntat sätt och körs varje gång ett bygge görs. På så vis säkerställs att kodändringarna inte har påverkat funktionaliteten i någon annan del av kodbasen. Det är ofta i detta sammanhang man pratar om code coverage, d.v.s. hur mycket av koden som har testats igenom."
Enhetstester är tester på programmeringsnivå vilket innebär att man testar metoder och komponenter i programmeringskoden.
Enhetstester utförs tidigt i utvecklingen och det är oftast programmerare som skriver dessa tester själva men i vissa fall skrivs dessa tester av en testare.

Integrationstester:

Integrationstester testar att alla enheter i en applikation kan kommunicera med varandra. Integrationstester är bra för regressionstestning och kan byggas för att hitta buggar så tidigt som möjligt. En förutsättning för att kunna testa integrationen är att man belyser testbarheten i applikationen.[19] Man testar funktioner, prestanda, säkerhet eller att få status på hur systemet fungerar. Dessa tester utförs under hela systemets utveckling.

Gränssnittstester:

"Till sist har vi nivån som ligger närmast det som vi normalt kallar testning. På den här nivån körs testerna mot det slutliga användargränssnittet och tanken är att det är en riktig användares beteende som ska simuleras."
Med dessa tester kollar man att alla funktioner fungerar som de är tänkta och det ska finnas minimalt med buggar.
Gränssnittstester utförs i slutet av en systemutveckling eftersom man testar användargränssnittet av ett system.

[19] https://blogg.addq.se/inspiration-kunskap/vad-ar-enhetstest-och-integrationstest

Analys och Slutsatser

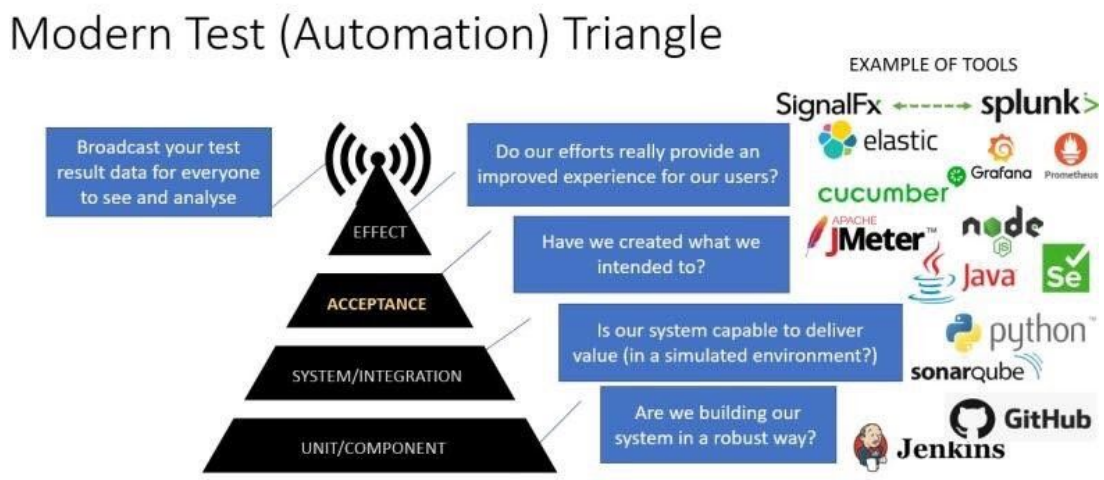
Vad för resultatdel har vi? Vad kan vi dra för slutsatser mot frågeställningarna?

Efter de resultat vi har fått från föreläsningar och litteraturstudier har vi fått en fördjupad förståelse för testautomatiseringsprocceser och kan därmed dra slutsatser/analyser om det är värt att implementera automatiska tester eller inte. Vi går även in på hur vi ska uppnå en så effektiv testautomatiseringsprocess som möjligt.

När man kollar på om det är värt att implementera automatiska tester finns det många aspekter som går in på hur man går tillväga för att uppnå en effektiv testautomatiseringsprocess. Processerna kan appliceras på de olika lagren frontend, API och backend. På frontendlagret utförs t.ex. användargränssnittstester och på backend utförs enhetstester, men även där de olika lagren skiljer sig ifrån varandra som innehåller API tester. De tre lagren täcker majoriteten av applikationsutvecklingen och blir därmed en väsentlig del att undersöka.

Balansen mellan de olika lagren är olika prioriterade beroende på företag. Alla företag har olika prioriteringar men alla lagren ska ha en viss balans för att uppnå en effektiv testautomatiseringsprocess. Balansen gäller även mellan manuella och automatiska tester då undersökningen från ADDQ visade på när man ska/inte ska ersätta manuella tester med automatiska. Det här är för att det kan bli kontraproduktivt, man förlorar både resurser och tiden det tar att implementera. Som i problemformuleringen är manuella tester ett komplement till de automatiska tester som även påvisades under föreläsningen.

Bilden nedan är en representation av en automatisk testprocess, där olika testverktyg mellan olika skikt kan användas.



[User interface testing & kanske en lösning till det]

GUI-testning (användargränssnitt/frontend) är en del av ett större system som kan automatiseras.
Under ADDQ föreläsningen togs det upp att användargränssnittstester ska utföras automatiskt i så hög grad som möjligt. Idag har konsumenter många olika enheter och dessa enheter har också olika operativsystem med olika versioner på dessa operativsystem.
Detta leder till att man måste utföra GUI-testning på många enheter.
Men det som påpekades också var att detta inte betyder att alla tester ska automatiseras, utan man ska prioritera vilka tester som företaget vinner mest på att automatiseras.
Företaget vill inte förlora tid och resurser på automatiska tester som inte är nödvändiga och det kan leda till en ineffektiv automationsprocess.

Tanken med automatiserade testprocesser är att kunna leverera resultat mer kontinuerligt, det ska vara en effektiv testprocess och många företag vill ha resultat snabbt. Det som också togs upp på föreläsningen var att man måste ha i åtanke att inte automatisera alla manuella tester. Det här var en viktig punkt som togs upp på föreläsningen.
Vi analyserade detta djupt under vår undersökning för att komma fram till resultat om hur man bibehåller en effektiv automatisk användargränssnittstestning.
Under vår undersökning kom vi fram till att det främsta som man bör tänka på är att alla GUI-tester i en testmiljö inte ska automatiseras.
Detta var också en viktig punkt som togs upp under ADDQs föreläsningen.
Ett exempel är om en GUI-test går ut på att klicka på några knappar i ett system behövs inte sådana tester automatiseras eftersom de kan utföras manuellt. Detta resulterar i att det blir både billigare och tar mindre tid. På detta vis kan man effektivisera testerna i en testmiljö.

Användargränssnittstestning innebär att man testar ett systems funktioner, flödet på systemet och allt som användaren kan se.
Automatiserade testprocesser för ett användargränssnitt innebär att datorn kommer följa de kommandon som en testare eller utvecklare har programmerat. Datorn kommer att utföra dessa kommandon exakt likadant varje gång de körs, datorn kan också utföra testerna mycket snabbare än vad en människa kan göra. Alla i teamet kan exekvera koden och se resultaten, vilket blir mycket svårare när man utför tester manuellt.
Detta leder till att man kan köra samma tester flera gånger om dagen, man kan även köra automatiska tester efter att man har lämnat kontoret för dagen. Repetitiva arbetsuppgifter behövs inte utföras av en testare utan testaren kan starta upp de automatiska testerna och låta datorn utföra det. Detta leder i sin tur till att testaren kan lägga mer tid på andra arbetsuppgifter som t.ex. UX samtidigt som dessa tester körs av datorn. Man får eventuellt bättre slutprodukt av automatiserade tester.
I ADDQ föreläsningen togs det också upp hur man uppnår en effektiv automatiserad process för GUI-tester. När man ska automatisera GUI-tester måste man först prioritera vilka tester som bör automatiseras och vilka som inte ska automatiseras. Man måste också utvärdera vilka automationsverktyg som man ska använda eftersom en del av dessa kostar pengar, vissa stora summor. Det är därför väldigt viktigt att man bara använder de verktyg som gör så att teamet uppnår så nära full täckning som möjligt.

Teamet räknar ur ROI (Return on investment)
En annan viktig punkt att tänka på när det gäller prioritering av automatiserade tester är att det är dyrt att skapa och underhålla automatiserade tester. Därför tycker vi att man ska prioritera vilka tester som bör automatisera och vilka tester som bör utföras manuellt utan att företaget förlorar mycket tid och pengar. För att prioritera vilka tester som bör automatiseras diskuterar testarna i teamet med varandra och kommer fram till en slutsats om automatiseringsprocessen. Detta är ett bra sätt att få fram konsensus inom organisationen tycker vi.
Ett exempel på detta är om man har ett testscenario där testaren ska avgöra om en webbsida/system är estetiskt tilltalande. Exemplet på testscenariot ska inte automatiseras eftersom testscenariot kräver mänsklig observation vilket inte kan automatiseras.
Ett annat exempel är ett testscenario i en webbsida där man ska lägga ett objekt i varukorgen och sen slutföra köpet. Detta testscenario kan bli väldigt repetitivt och tar längre tid att utföra. Detta testscenario bör man överväga att automatisera för att man i längden kommer att spara både tid och pengar. Testaren behöver inte heller utföra väldigt repetitiva testscenarion utan kan fokusera på andra arbetsuppgifter.

Automatiserade testprocesser är det rätta vägen att gå inom testning tycker vi efter djupare undersökning på frågan.
För att få en så effektiv testautomatiseringsprocess som möjligt är det en del faktorer som spelar in i hur effektiv processen blir. Effektiva metoder för testautomatisering är att man använder CI/CD pipeline metoden för att kunna leverera och testa produkten kontinuerligt.

Idag är system som utvecklas mycket mer komplexa och det utvecklas i en snabbare takt än vad det gjorde förr i tiden.

Därför är det viktig att testa ett system kontinuerligt, testaren får sitt testresultat kontinuerligt och ger feedback till utvecklarna med jämna mellanrum. För att uppnå detta måste man implementera automatiska tester i en testmiljö samt automatisera testprocessen. Men det betyder inte att alla tester ska automatiseras, en testmiljö bör bestå av både manuella och automatiska tester. På så sätt får vi det bästa av två världar. Ett företags tester blir effektivare, testaren får mer arbete gjort och testaren får göra mer varierande arbete.

[4 djupa exempel på när man ska/inte ska införa automatiska tester]

- Brist i resurser, kostar att implementera:

För att implementera automatiska testprocesser krävs det resurser från företaget, kunskaper inom automatiska testprocesser och tid. Nu ska vi ge er ett djupt exempel på brist i resurser vilket kan leda till att ett företag inte kan automatisera sina testprocesser.
“40% of a test automation person time is spent on authoring tests.”[20]
Automatisering av ett företags testprocesser är dyrt, personalen som ska skapa dessa tester kommer kosta pengar, det tar tid att skapa automatiska tester vilket resulterar i att det kommer kosta mycket i längden. Underhålla automatiska tester är dyrt, i en agil värld ändras system konstant och nya funktioner implementeras vilket leder till att de automatiska tester som har implementerats måste underhållas. Idag finns det också många olika enheter (Mobiler, datorer) och versioner på dessa enheter vilket också leder till att automatiska tester måste kunna testa flera enheter än vad det behövdes förr.
“In the agile world, UI changes are frequent, This implies that changing the UI requires changes in the UI-tests. 30% of a test automation person time is spent on maintaining tests” [21]

Ett företag som har brist på resurser bör inte implementera automatiska tester om de inte har gjort en redovisning på vad kostnaden kommer vara. Företaget måste också räkna ut effektiviteten av att automatisera testprocessen. När detta har gjorts kan man ta ett beslut om man ska automatisera testprocessen.
Det krävs expertis inom testautomation, de rätta verktygen, och planering för att lyckas med implementation av automatiska testprocesser. Verktyget som ett företag använder kostar också pengar, antingen genom att betala för tjänsten (verktyget) eller att man betalar för underhåll av tjänsten (verktyget). Med underhåll av tjänsten menar vi att man betalar ett företag som underhåller verktyget så att det har den senaste versionen. Fel som uppstår med verktyget fixas och att det företag man betalar har ansvar om eventuella fel uppstår med verktyget.
CI/CD pipeline kostar mycket att implementera i ett utvecklingsteam. Det är inte optimalt att implementera detta om ett företag har brist på resurser även om det kommer med en del fördelar.
Ett företag med låga resurser kan förlora pengar på att automatisera testprocessen och då är det rätta valet att fortsätta utföra sina tester manuellt, då det blir billigare för företaget.

[20] https://www.testim.io/blog/test-automation-costs/
[21] https://www.testim.io/blog/test-automation-costs/

- Komplexa miljöer, tar tid, och kanske mer tid än själva utvecklingen

Komplexa miljöer kan bestå av flera hundra system som är integrerade med varandra och kommunicerar med flera olika enheter. Implementation av automatiska tester kan ta mycket tid, vilket eventuellt resulterar i att det blir stora kostnader.
Komplexa miljöer kan bestå av en massa system som är integrerade med varandra och jobbar tillsammans, detta innebär att strukturen på ett system kan bli väldigt komplext.
Om en sådan situation uppstår att man har komplexa miljöer att jobba i. Då får utvecklingsteamet utvärdera om man ska utföra så många manuella tester som man hinner, det är fortfarande möjligt att uppnå en hög kvalité med systemet. Men enhetstester är väldigt viktigt att utföra och om systemet är komplext blir det ännu viktigare med enhetstester. Även om utvecklarna får lite mer jobb tycker vi att enhetstester är ett måste. För att om man vill göra ändringar i koden eller något fel uppstår blir processen att göra dessa ändringar mycket enklare och det blir lättare att spåra felet. Det blir även billigare om man ändrar fel tidigt i utvecklingen, speciellt om man har en eller flera komplexa system. Enhetstester är det billigaste testautomationsprocessen att utföra, det tar minst tid att utveckla och det täcker största delen av koden.
Enhetstester är ett måste men utöver det måste man se över vilka resurser man har och hur mycket tid testarna kan lägga på att automatisera sina tester.

- Planerade utveckling, GDPR med att skapa testdata, eller andra funktioner som måste ändras

I och med att GDPR trädde i kraft i maj 2018 får man exempelvis inte spara testdata om personer utan deras tillåtelse, man får bara spara nödvändig information om personen som verkligen behövs och man får inte spara det i längre än så länge det behövs.
Vad GDPR innebär är att man inte får använda riktiga personuppgifter som testdata när man testar ett system. Syftet med GDPR är att skydda personuppgifter inom Europa, det innebär att företag och organisationer ska absolut inte använda personuppgifter oavsett omständighet. Exempel på detta är om ett företag vill använda våra personuppgifter i gruppen för att testa deras system. Företaget måste få godkännande från oss innan de kan använda våra personuppgifter. Annars kommer de bli bötfällda eller bli tvingad att betala böter för personen.
“Det kan kosta upp till fyra procent av omsättningen eller 20 miljoner euro i böter, det som är högst.”
Syftet med GDPR är att skydda personuppgifter från organisationer som behandlar dessa uppgifter. “I Sverige har vi tidigare haft en förenkling av personuppgiftslagen. Har man behandlat personuppgifter i löpande text och enklare listor har den hanteringen kunnat göras

med stöd i missbruksregeln. Det har gjort det möjligt att hantera personuppgifter så länge det inte är kränkande för någon. Den regeln försvinner helt och hållet i och med den nya lagen."[22]

Därför måste man skapa testdata för att kunna testa sitt system. Detta är väldigt viktigt om systemet skulle vara en bokningswebbsida eller liknande.

Allt detta innebär mer utvecklingsarbete, det ställs en massa nya krav på hur data ska hanteras och helt plötsligt måste man utveckla datan för att uppfylla GDPRs regler.

"Skriva koden helt enkelt, bygga om formulär, skriva nya sql-satser, samt tänka ut vilken information som skall sparas och på vilket sätt."[23]

All data som användes innan denna lag infördes måste analyseras och utvärderas, eftersom det är en riskfaktor. All data som inte uppfyller GDPRs krav måste bytas ut mot ny data som uppfyller dessa krav. Detta påverkar både tidsplanen och budgeten för projektet, det kommer både kosta mycket pengar och ta tid att implementera.

I de flesta fallen används testdata för att utföra automatiska tester. Dessa testdata som man använder måste uppdateras så att de följer GDPRs lagar. Detta kommer bli dyrt för utvecklingen eftersom man måste först skapa nya testdata. Sedan måste man uppdatera sina automatiska tester och implementera nya testdata till de.

När man inför automatiska tester i en testprocess måste dessa tester användas av fixerat data så att de följer GDPR lagen. Annars blir det väldigt kostsamt för företaget när det gäller böter och att göra ändringar på testerna.

- Automatiserade tester på enhetsnivå

Automatiserade tester på enhetsnivå är ett måste. Enhetstester utförs oftast av utvecklaren men i vissa fall gör testaren det. Enhetstester fokuserar på klasser och metoder i koden som man ska testa, det säkerställer att koden uppför sig som förväntat.

Enhetstester går ut på att testa koden man har skrivit själv och verifiera funktionaliteten.

Det finns många fördelar med att utföra enhetstester och det absolut största fördelen med detta är att man kan hitta defekter tidigt i utvecklingen. Detta innebär att det blir billigare att åtgärda problemen tidigt i utvecklingen än vad det hade kostat om man hade hittat defekten senare i utvecklingen.

Enhetstester som utförs av utvecklaren ses av några som en nackdel för att utvecklaren får göra en extra uppgift. Men det är en fördel eftersom vi hittar defekter tidigare i utvecklingen som kan rättas till och ändringar i koden blir lättare att utföra med väl utvecklade enhetstester. Ju tidigare man åtgärdar en defekt desto billigare blir det. Vi ser väldigt få nackdelar med automatiserade enhetstester och många fördelar som vi har tagit upp i texten.

Enhetstester är en viktig del för att uppnå en effektiv automatiseringsprocess. Det är ett av dom viktigaste delen i en testprocess eftersom man hittar defekter tidigt i utvecklingen. Det leder till att rättning på defekter som man hittar blir billiga och ändringar som leder till fel blir inte lika dyrt att åtgärda. Allt detta leder till att testprocessen blir effektivare.

[22] https://cio.idg.se/2.1782/1.674864/gdpr
[23] https://cio.idg.se/2.1782/1.674864/gdpr/sida/2/del-2-har-ar-7-gdpr-konsekvenser-for-utvecklare

Rekommendationer

Vår undersökning gällande detta ämne leder till att företag bör applicera automatiska tester i sin testmiljö. För att implementera automatiska tester behöver testaren goda kunskaper inom ämnet och implementationen ska ske gradvis. Vår rekommendation är om ett företag ska automatisera sin testmiljö ska företaget först välja ut några ur sitt utvecklingsteam med rätt kompetens. De ska redovisa hur man ska gå tillväga för att automatisera testmiljön, vilka eventuella konsekvenser det kan uppstå, fördelar som kommer med en automatiserad testmiljö och eventuella nackdelar med implementering av en automatisk testmiljö.
Det finns tre nivåer av automatiska tester; Enhetstester, API-tester och GUI-tester. När ett företag ska automatisera dessa tre nivåer av tester bör man redovisa och prioritera vilka testfall som ska automatiseras. Detta är för att man ska ha rätt balans mellan automatiska och manuella tester.

För att uppnå en effektiv automatiserad testmiljö rekommenderar vi baserade på våra undersökningar att man ska implementera automatiska tester i en testmiljö. Dock bör man redovisa huruvida det blir lönsamt i längden eller inte och att balansen mellan manuella och automatiska tester är viktiga för en lyckad automatisk testmiljö. Att bibehålla manuella tester utöver de automatiska testerna är viktig för en effektiv automatiserad testmiljö. Vår undersökning visade att inte alla tester ska automatiseras, vilket resulterar i att manuella tester också är viktiga att bibehålla.
Den stora faktorn för en effektiv automatiseringsprocess är att rätt kompetens och kunskaper finns inom organisationen.

Newton

Källförteckning

https://www.frontit.se/inspiration-kunskap/artiklar/testautomatisering-det-bra-det-daliga-och-den-stora-fragan/

https://smartbear.com/solutions/api-testing/

https://smartbear.com/learn/automated-testing/what-is-automated-testing/

https://smartbear.com/learn/automated-testing/best-practices-for-automation/

https://www.guru99.com/gui-testing.html

https://smartbear.com/resources/ebooks/6-ways-to-measure-the-roi-of-automated-testing/

https://www.testim.io/blog/test-automation-costs/

https://lemontree.se/lemontree-speed-quality/lemontree-event-kommande/gdpr/

https://codefresh.io/continuous-integration/continuous-integration-delivery-pipeline-important/

https://cio.idg.se/2.1782/1.674864/gdpr/sida/2/del-2-har-ar-7-gdpr-konsekvenser-for-utvecklare

https://cio.idg.se/2.1782/1.674864/gdpr

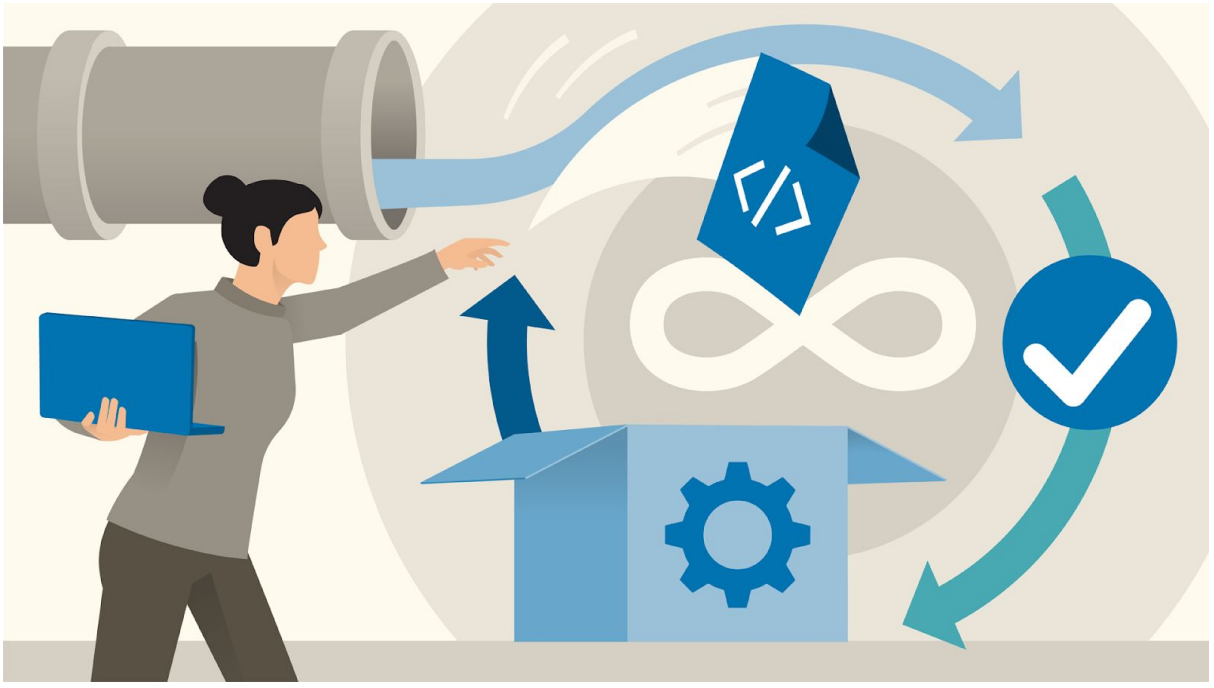
https://blogg.addq.se/inspiration-kunskap/testautomatisera-strategiskt-sa-ska-du-tanka

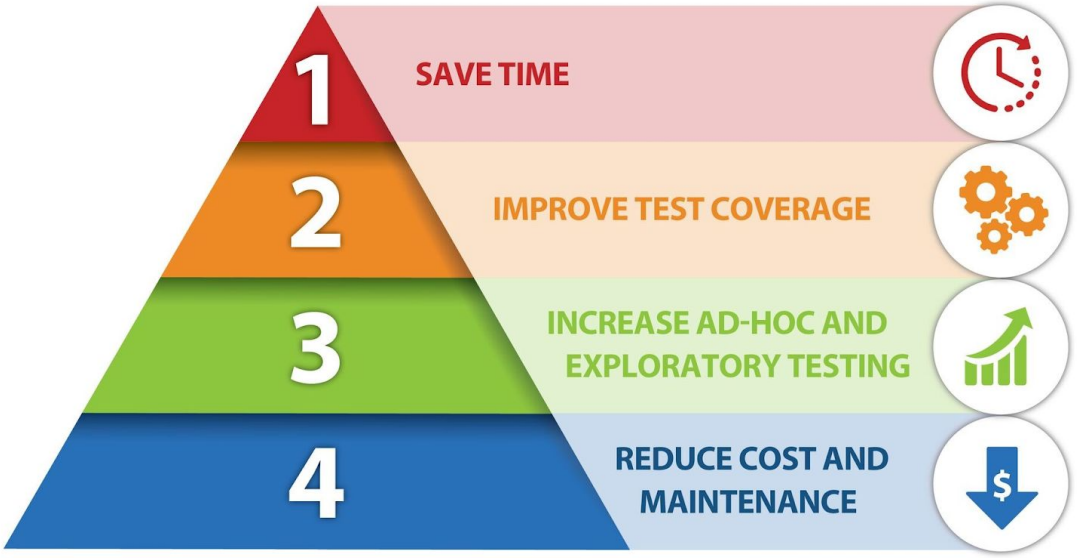
https://www.addq.se/inspiration-kunskap/vad-ska-du-inte-automatisera?utm_campaign=Testautomatisering&utm_content=125181094&utm_medium=social&utm_source=linkedin&hss_channel=lcp-87387

https://www.ques10.com/p/2315/strength-and-weakness-of-manual-and-automated-te-1/

https://www.atlassian.com/continuous-delivery/principles

Bilagor




AUTOMATED SOFTWARE TESTING
1
SAVE TIME
2
IMPROVE TEST COVERAGE
3
INCREASE AD-HOC AND EXPLORATORY TESTING
4
REDUCE COST AND MAINTENANCE