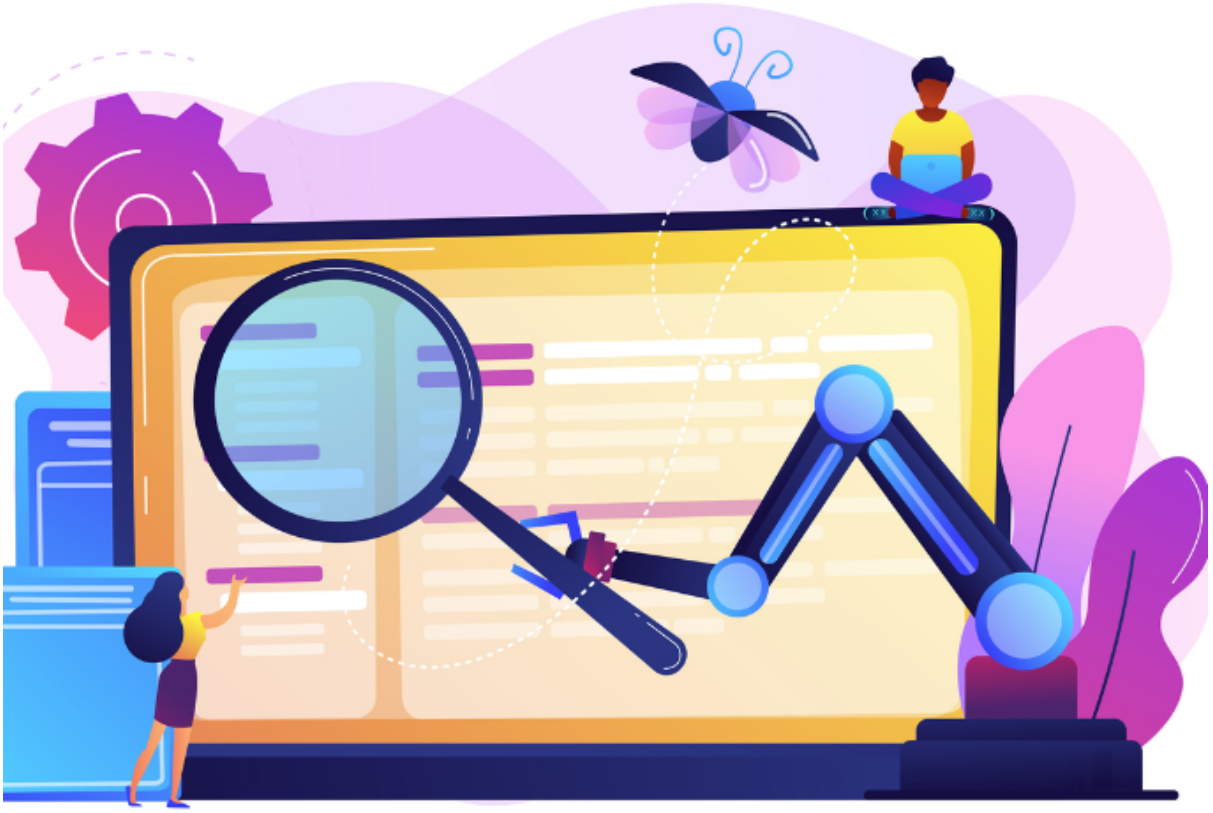
Lab 3: Automatiserad testning

Kursnamn: Kvalitetssäkring i systemutveckling

Namn: Casey Oyugi



När jag själv skulle installera automation servern Jenkins och sedan konfigurera miljön på min dator, slog det mig att sätta upp och underhålla en sådan miljö kan vara en stor utmaning. Där satt jag och följde en video som dessutom hanterade installationen av en äldre version och varav jag dessutom saknade någon nödvändig kompetens och kunskap om automatisering-testverktyg och metoder. Med det sagt är arbete med test och kvalitetssäkring svårt där det krävs både utbildning och erfarenhet för att bli en bra testare. Det krävs dessutom gott samarbete med de övriga delarna av organisationen och konsekvent uppföljning som kan vara en tidskrävande process (Gustavsson & Görling, 2019). Det tar även tid och resurser att ta fram automatiska testfall där man bör se över ifall de faktiskt lönar sig eller är relevanta.

Mc Connel (2004) skriver om olika tekniker i syfte att förbättra mjukvarukvalitet, där han bland annat nämner test strategier. Han menar att utförandet av testning kan ge en detaljerad bedömning av produktens pålitlighet där det är ytterst viktigt att förstå att en del av kvalitetssäkring är att utveckla en teststrategi som samarbetar med produktkraven, arkitekturen och designen. Utvecklare i många projekt förlitar sig på testning som den primära metoden både för bedömning av kvalitet och förbättring av kvalitet. Det här har starka band till Mc Connels koncept av den interna kvaliteten som bland annat är relaterad till arkitekturen och hur lätt det är att underhålla och vidareutveckla produkten. Därför bör man även tänka på komplexiteten i testmiljön varav en komplex testmiljö kan vara ett tecken på bristande intern kvalitet, eftersom det kan tyda på en otydlig eller komplicerad arkitektur. Oavsett omfång för en testplan och dess strategier behöver den svara på frågor som, hur ska systemet testas? När ska det testas? Vem ska testa systemet och vilka användargrupper behöver vara delaktiga i testandet? (Mc Connel, 2004).

När man utvecklar en teststrategi bör det dessutom bygga sig på en utvecklingsprocess som Test Driven Development (TDD). TDD är en agil utvecklingsmetodik där tester skrivs innan koden utvecklas till skillnad från traditionell testning som utförs efter att koden har skrivits (BrowserStack, 2023). Fördelarna med TDD är att den främjar till att skriva en ren och strukturerad kod från början eftersom utvecklarna måste skapa tester för att testa sina funktioner. Detta leder till att det blir lättare att se varför koden misslyckades och kräver dessutom mindre felsökning. Dessutom kan detta leda till minskat teknisk skuld och färre buggar i koden (Dahlin, 2023)**.** TDD passar dessutom bra med Continuous Integration (CI) processer, eftersom de automatiserade testerna som skapas kan köras regelbundet, vilket hjälper till att upptäcka problem tidigt (Dahlin, 2023).

Framtagandet av en testplan gör att man kan få en tydlig plan för när och hur mycket resurser som behövs för testverksamheten. Att ha testverksamhet inplanerad precis som utveckling är ett bra sätt för att se till att den verkligen blir genomförd. En annan fördel med att diskutera testarbetet tidigt är att det då finns möjlighet att utforma programvaran på ett sådant sätt att den är lättare att testa som som medför till mindre utmaningar när testmiljön behöver underhållas (Gustavsson & Görling, 2019).

Automatiserings servrar som Jenkins bygger sig på öppen källkod och om man ska använda sig av öppen källkod är det viktigt att känna till att alla inte får ta och använda programkoden hur man vill. Källkoden som skapats skyddas nämligen av upphovsrättslagstiftningen på samma sätt som till exempel musik. Programvara som kallas öppen källkod förhåller sig ofta under någon form av licens som reglerar användningen som till exempel organisationen Open Source Initiative (OSI). Om man inte känner till dessa licenser kan flera problem uppstå (Gustavsson & Görling, 2019)

Inför installation, konfiguration och driftsättning av en miljö för automatattest kan det bli en utmaning att välja rätt verktyg som passar ett specifikt projekt och utvecklarteamets behov. Det kan väl så vara att olika projekt kräver olika verktyg och utvecklare i projekten har olika erfarenheter och kunskaper om automatiserade tester. För att skapa effektiva tester och en samarbetsvillig miljö krävs det att hela teamet, om inte organisationen, har en förståelse för systemets arkitektur, krav och funktioner. Man bör skapa en testplan i syfte att säkerställa att man har tillräckligt med resurser för att utföra automatiserade tester som kan vara en utmaning. Under installation, konfiguration och driftsättning kommer det säkerligen bli en stor svårighet med konfigurationen och integrationen av verktygen där testmiljöerna kan ha en stor teknisk komplexitet. Efter all installation, konfiguration och driftsättning kommer det bli utmaningar med att underhålla de automatiserade testerna då de kräver kontinuerligt underhåll, som inkluderar uppdateringar av testskript, ändringar i testdata och buggar. Här är det även viktigt att ha ett bra underlag för samarbetsvilja inom teamet som kommer ge positiva konsekvenser som att snabbt kunna hantera fel i programvaran eller integrera nya funktioner på ett effektivt sätt.

Att kvalitetssäkra införandet av en miljö för automattester i ett team med flera systemutvecklare är en viktig process för att säkerställa att automatiserade tester fungerar effektivt och ger pålitliga resultat. Därför krävs många viktiga steg och åtgärder för att uppnå detta. Först behöver man en tydlig strategi och planering. I denna planering fördelar man roller och ansvar och fördjupar sig i frågor som vad målet är med testningen eller vilka områden och funktioner i systemet som ska täckas av de automatiserade testerna. Roller och dess ansvar kan exempelvis vara testledare som planerar och leder testarbetet bland utvecklarna eller teststrateger som ansvarar för testmetodik och processerna (Dahlin, 2023). I planeringen är det dessutom bra att skapa en detaljerad plan som bör innehålla en lista på beskrivningar och testfall att genomföra. Dessa kan vidare grupperas i övergripande sviter av tester för att skapa en bättre överblick och sedan skapa en prioritetsordning och hur resultatet ska noteras. Framtagandet av en testplan gör att man kan få en tydlig plan för när och hur mycket resurser som behövs för testverksamheten (Gustavsson & Görling, 2019).

I ett utvecklingsteam är det även viktigt att säkerställa att teammedlemmarna har den rätta kompetensen och kunskapen om automatiserade tester och de verktyg som används i miljön. Därför kan det vara bra att inkludera utbildning och träning för att försäkra att alla är väl insatta i de mest effektiva och bästa teknikerna som krävs för att skapa och underhålla de automatiserade testerna. Med det sagt är det viktigt med kontinuerlig kompetensutveckling. Teamet består troligast av både seniora och juniora utvecklare med olika arbetslivserfarenheter. Detta bör utnyttjas då man kan främja en kultur med kunskapsdelning inom teamet där man kan dela med sig av sina erfarenheter och kunskaper om automatiserade tester (Gustavsson & Görling, 2019).

Det kan även vara bra att etablera standarder och riktlinjer för automatiserade tester. Detta kan bekräftas med en checklista som kan inkludera att namn-och kodstandard följs, att det finns felhantering där det krävs, katalogstrukturer, att en kod har blivit granskad av en kollega och att dokumentation finns. Tydlig och välstrukturerad dokumentation är avgörande och särskilt användbart när andra utvecklare arbetar med samma projekt. Testfallen fungerar som levande dokumentation för koden och beskriver som sagt hur det ska användas (Dahlin, 2023).

Strategier i ett team för versionshantering är viktigt då man bör se till att det versionshanteringssystem som valts ut används på ett genomtänkt sätt. Det är bra att i ett utvecklarteam bestämma sig för en gemensam samt tydlig praxis när testskript och testdata ska sparas, hur man ska arbeta med förgreningar, buggar etc (Gustavsson & Görling, 2019).

Som jag nämnt lite kort tidigare i rapporten är det viktigt att försäkra sig om att det lönar sig att införa en miljö av automattester innan det är bestämt. För att kunna utvärdera en miljö för automattest från början till slut bör man noggrant se över, analysera och utvärdera de affärsbehov, krav och mål som ska uppfyllas. Redan i planeringsfasen där uppdraget specificeras och beslutas om hur projektet ska genomföras bör varje krav för projektet utsättas för granskning. Det kan inkludera att identifiera vilka områden av systemet som ska täckas och definiera tydliga mål för prestanda och pålitlighet i testerna. Detta bekräftar Alhassan, Alzahrani och AbdulAziz (2017) där de understryker att testaktiviteter måste planeras och hanteras på rätt sätt redan från början av en mjukvaruutveckling.

När planeringsfasen har passerat bör noggrann övervakning och kontroll utföras i syfte att säkerställa att testerna utförs korrekt och täcker de krav som lagts. Detta kan bekräfta att testmiljön är stabil och att integrationen med CI fungerar som förväntat (Dahlin, 2023). Efter implementeringen är det viktigt att kontinuerligt övervaka prestandan och effektiviteten av de automatiserade testerna, som kräver mycket tid, arbete, bra dokumentation och samarbete inom teamet(Gustavsson & Görling, 2019). Kontinuerlig feedback och samarbetsvilja inom teamet är mycket värdefullt då man samlar in åsikter från olika utvecklare, testare eller andra intressenter om användbarheten och effektiviteten i testmiljön.

Sammanfattningsvis så kräver en helhetsutvärdering, från början till slut, av automatiserad testmiljö en noggrann, strategisk och metodisk ansats genom hela processen. Det är en ständig strävan för att säkerställa att testmiljön möter affärsbehoven och ger positiva resultat för utvecklingsteamet och organisationen som helhet.

Innan jag påbörjade den här uppgiften trodde jag att införandet och underhållet av en automatiserad testmiljö var en enkel process. Min uppfattning var att automatiserade tester alltid var det självklara valet inom testning då det alltid skulle medföra omedelbara och fullständiga pålitliga resultat. Med andra ord att det skulle vara den mest effektiva och okomplicerade vägen för hantering av testning i ett programvaruprojekt. Fast desto mer jag lärde mig och undersökte ämnet, desto tydligare blev det att verkligheten är mycket mer komplex än så. Det är inte bara att skapa några testfall och köra dem, utan det kräver en djup förståelse av systemet, kontinuerlig övervakning och anpassning till ändrade krav och förutsättningar. Man måste noggrant utvärdera och överväga när och hur de ska användas samt veta om det är mer värt än manuella testningar. Det är dessutom ett bredare koncept som inte är begränsat till testverksamhet, utan inkluderar alla aktiviteter som görs för att minimera risken att ett system brister. Enligt Gustavsson och Görling (2019) är det viktigaste med testverksamheten att man försöker angripa den med en strukturerad ansats som inkluderar en utarbetad strategi för hur testandet ska gå till. Det har även lärt mig att införandet av en automatiserad testmiljö är en pågående process som involverar flera steg och aspekter.

Avslutningsvis tar detta tillbaka mig till begreppet Total Quality Management (TQM) vars huvudsakliga fokus är att fokusera på kontinuerlig förbättring av förmågan att leverera högkvalitativa produkter och tjänster till kunder. Det föreslår att varje förbättring som görs inom företaget, såsom en tydlig strategi för testplanen, bra samarbete inom ett team, definition av mål eller krav eller en noggrann metodisk ansats genom hela processen, kommer att bidra till att förbättra organisationens totala kvalitet och kvaliteten på den slutliga produkten (Alhassan, Alzahrani & AbdulAziz, 2017).

REFERENSLISTA

Alhassan, A., Alzahrani, W., & AbdulAziz, A. (2017). Total Quality Management for Software Development. *Total Quality Management*, 158(5).

Download Programvarukvalitet - externa och interna kvalitetsegenskaper Mc Connell, S. (2004). *Code Complete*, Microsoft Press.

Gustavsson, T., & Görling, S. (2019). *Att arbeta med systemutveckling*.

BrowserStack (2023-06-14). *What is Test Driven Development (TDD)?* Tillgänlig: https://www.browserstack.com/guide/what-is-test-driven-development#:~:text=TDD%20Vs.-,Traditional%20Testing,after%20the%20code%20is%20written. [Datum för åtkomst: 2023-10-05]

Dahlin, M. (2023). *Kodskuld och test, TDD, JUnit* [PowerPoint-presentation] Studium. https://uppsala.instructure.com/courses/80109/files/5507791?wrap=1

Dahlin, M. (2023). *Software Quality Assurance* [PowerPoint-presentation] Studium. https://uppsala.instructure.com/courses/80109/files/5499406?wrap=1

Dahlin, M. (2023). *Source Control & Continuous Integration* [PowerPoint-presentation] Studium. https://uppsala.instructure.com/courses/80109/files/5574847?wrap=1