Continuous Integration(CI):

CI är en metod för att integrera alla kodändringar i huvudgrenen i ett delat källkod-arkiv, automatiskt testa varje ändring när man lägger in eller sammanfogar dem och automatiskt starta en byggning. Med kontinuerlig integration kan fel och säkerhetsproblem identifieras och åtgärdas lättare och mycket tidigare i utvecklingsprocessen.
Genom att slå samman ändringar ofta och utlösa automatiska test- och valideringsprocesser kan man minimera risken för kod konflikter, även när flera utvecklare arbetar med samma program. Detta minskar väntetiden och korrigering av buggar eller säkerhetsproblem kan ske nästan omedelbart .

Continuous Delivery(CD):

CD är en metod för mjukvaruutveckling som fungerar tillsammans med CI för att automatisera processen för tillhandahållande av infrastruktur och lansering av applikationer.
När koden har testats och byggts som en del av CI-processen tar CD över under de sista stegen för att se till att den är paketerad med allt den behöver för att kunna distribueras till vilken miljö som helst när som helst. CD kan täcka allt från tillhandahållande av infrastrukturen till distribution av applikationen till test- eller produktionsmiljön.
Med CD byggs programvaran så att den kan distribueras till produktion när som helst. Sedan kan du utlösa drift sättningarna manuellt eller gå över till kontinuerlig driftsättning, där drift sättningarna också automatiseras.

Continuous Deployment(CD):

CD gör det möjligt för organisationer att distribuera sina applikationer automatiskt, vilket eliminerar behovet av mänsklig inblandning. Med kontinuerlig distribution fastställer DevOps-teamen kriterierna för kodreleaser i förväg och när dessa kriterier är uppfyllda och validerade distribueras koden i produktionsmiljön. Detta gör det möjligt för organisationer att snabbare få nya funktioner i händerna på användarna.
Man kan göra CI utan kontinuerlig leverans eller distribution, men man kan inte göra kontinuerlig integration utan att redan ha CI på plats. Detta beror på att det skulle vara extremt svårt att kunna distribuera till produktion när som helst om man inte tillämpar CI-grunderna, som att integrera kod till en delad repo, automatisera testning och byggning och göra allt i små omgångar dagligen.

Agile Metoder:

Från den gamla vattenfallsmetoden ville föregångarna till den agila metoden slippa kodprodukter av dålig kvalitet och minska kostnaderna för detta. Vattenfallsmetoden gick ut på att utvecklarna skulle utveckla produkten utan att någon testning skulle göras under utvecklingsprocessen, och i slutet när produkten är redo att gå ut till användarna skulle den testas, vilket ledde till olika typer av resursslöseri som kunde ha minskats eller undvikits om den hade testats i ett tidigt skede av utvecklingen. År 2001 träffades de 17 medlemmarna, som är grundarna av den agila metoden, på Snowbird resort under helgen. Efter stora meningsskiljaktigheter och fler förhandlingar kom de fram till de tolv agila manifestationerna som gäller än i dag och många fler dagar framöver. Agile är en iterativ metod för projektledning och programvaruutveckling som hjälper team att leverera värde till sina kunder snabbare och med mindre huvudvärk. I stället för att satsa allt på en "big bang"-lansering levererar ett agilt team arbete i små, men konsumerbara, steg. Krav, planer och resultat utvärderas kontinuerligt så att teamet har en naturlig mekanism för att reagera snabbt på förändringar.

SCRUM:

Scrum är en delmängd av den agila metoden och är det mest använda ramverket i dagens värld. Scrum-teamet består av Scrum Master, produktägare och utvecklare. Scrum-teamet ansvarar för all produktrelaterad verksamhet. Teamet arbetar i sprintar som är högst en månad långa och uppnår målet för sprinten inom den givna tiden. Utvecklarna bestämmer hur mycket arbete som kan utföras under en viss sprint och produktägaren ordnar produktbackloggen så att det viktigaste står högst upp på listan för utvecklarna för att bestämma sprint-målet. Scrum master är till nytta för Scrum-teamet och organisationen. Scrums tre pelare är öppenhet, inspektion och anpassning. Händelserna i en sprint är Sprint Planning, Daily Scrum, Sprint Review och Sprint Retrospective. Varje händelse är ett formellt tillfälle att inspektera och anpassa Scrum-artefakter, som är Product Backlog, Sprint Backlog och Increment. Scrum-teamet använder tavlor som hjälper dem att hålla arbetets framskridande öppet och tillgängligt för hela scrum-teamet så att de kan anpassa sig i enlighet med detta. Product Backlog är den enda källan till det arbete som scrumteamet utför. Scrum Master underlättar för teammedlemmarna att sköta sig själva och fungera över teamgränserna. De tar också bort eventuella hinder för scrumteamet som hindrar scrumteamet från att fungera effektivt.

Kanban:

Kanban är ett populärt ramverk som används för att genomföra agil och DevOps-programvaruutveckling. Det kräver realtidskommunikation av kapacitet

och full insyn i arbetet. Arbetsmoment representeras visuellt på en kanban-tavla, vilket gör att teammedlemmarna kan se läget för varje arbetsmoment när som helst, precis som i Scrum. Kanban-metodiken för arbete har mer än 50 år på nacken. I slutet av 1940-talet började Toyota optimera sina tekniska processer utifrån samma modell som stormarknader använde för att fylla på sina hyllor. Toyota-arbetarna kommunicerade kapacitetsnivåer i realtid på fabriksgolvet (och till leverantörerna), arbetarna skickade ett kort, eller "kanban", mellan lagen.Även om signaltekniker för denna process har utvecklats sedan 1940-talet, är samma "just in time"-tillverkningsprocess (eller JIT) fortfarande kärnan i den. Agila mjukvaruutvecklingsteam kan idag utnyttja samma JIT-principer genom att anpassa mängden pågående arbete till teamets kapacitet. Detta ger teamen mer flexibla planeringsalternativ, snabbare resultat, tydligare fokus och öppenhet under hela utvecklingscykeln.Kanban-metodiken bygger på fullständig öppenhet i arbetet och kommunikation i realtid om kapaciteten. Därför bör kanban-tavlan ses som den enda källan till sanning för teamets arbete.

SaFe:

Scaled Agile Framework (SAFe) är en uppsättning organisations- och arbetsflödesmönster för att genomföra agila metoder på företagsnivå. Ramverket är ett kunskapsunderlag som innehåller strukturerad vägledning om roller och ansvar, hur arbetet ska planeras och ledas och vilka värderingar som ska upprätthållas. SAFe främjar anpassning, samarbete och leverans mellan ett stort antal agila team. Det bildades kring tre primära kunskapsområden: agil programvaruutveckling, lean produktutveckling och systemtänkande. När företag växer i storlek erbjuder SAFe en strukturerad metod för att skala upp agila team. Det finns fyra konfigurationer i SAFe för att tillgodose olika skalnivåer: Essential SAFe, Large Solution SAFe, Portfolio SAFe och Full SAFe. SAFe’s kärnvärden beskriver den kultur som ledarskapet måste främja och hur människor ska bete sig inom den kulturen för att effektivt kunna använda ramverket. I SAFe-ramverket får smidighet aldrig ske på bekostnad av kvalitet. det finns fem viktiga dimensioner av inbyggd kvalitet: flöde, arkitektur- och designkvalitet, kodkvalitet, systemkvalitet och kvalitet vid lansering.

DevOps:

DevOps är en uppsättning metoder, verktyg och en kulturell filosofi som automatiserar och integrerar processerna mellan programvaruutveckling och IT-team. Den betonar teamets egenmakt, kommunikation och samarbete mellan olika team och automatisering av tekniken. DevOps-rörelsen började omkring 2007 när mjukvaruutvecklings- och IT-verksamheterna uttryckte oro över den traditionella mjukvaruutvecklingsmodellen, där utvecklare som skrev kod arbetade åtskilda från verksamheten som distribuerade och stödde koden. Termen DevOps, som är en kombination av orden utveckling och drift, speglar processen att integrera dessa discipliner i en enda kontinuerlig process.Ett DevOps-team består av utvecklare och IT-drift som samarbetar under hela produktlivscykeln för att öka hastigheten och kvaliteten på programvaru installationen. I en DevOps-modell är utvecklings- och driftsteam inte längre "siloade". Ibland slås dessa två team samman till ett enda team där ingenjörerna arbetar under hela applikationens livscykel - från utveckling och test till driftsättning och drift - och har en rad tvärvetenskapliga färdigheter. devOps-team använder verktyg för att automatisera och påskynda processer, vilket bidrar till att öka tillförlitligheten. En DevOps-verktygskedja hjälper teamen att ta itu med viktiga DevOps-grundprinciper, inklusive kontinuerlig integration, kontinuerlig leverans, automatisering och samarbete. DevOps värderingar tillämpas ibland på andra team än utvecklingsteam.

QA:

Med Agile utvecklas lösningarna med ett inkluderande tillvägagångssätt och med tvärfunktionella team insatser. Nya organisationer med föränderliga behov inser fördelarna med Agile för snabbare applikationsutvecklings cykler och snabbare omställning i krävande situationer. Kortare och snabbare utvecklingscykler ifrågasätts dock i allmänhet när det gäller kvalitet, och det är där kvalitetssäkring (QA) kommer in i bilden. Även om kvalitet ger validering får den inte försämra programvaruutveckling processen och dess takt. Icke desto mindre blir kvalitetssäkring en integrerad del av ekvationen, framför allt för att ge validering och säkerställa stabilitet för applikationen. Det hjälper till att bygga en applikation som är robust, tillförlitlig och tillgänglig även under oförutsedda omständigheter. Testning kan därför inte ske i faser, utan måste planeras som en process. QA’s roll i agila lösningar kan omfatta både testning och utveckling. Tanken är att utvecklare och testare aktivt måste samarbeta för att leverera koden och slutföra projektet enligt kundens uppdrag. QA hjälper till att proaktivt ta itu med problem

och potentiella fel i en applikation under utvecklingscyklerna. Det kan också hjälpa till att åtgärda problem med funktionalitet, prestanda eller säkerhet. Detta kommer inte bara att säkerställa stabiliteten i applikationen utan också minska testarbetet när applikationen väl hamnar i användarnas händer. QA-teamet skulle vara bättre rustat för att ge snabb feedback genom ett effektivt samarbete med utvecklings folket.

Tools:

Enhetstester	Integrationstester (API)	Systemtester	Acceptanstester
JUnit, NUnit, TestNG ,Mockito, PHPUnit	Postman, Apigee, JMeter, Assertible, REST-assured	Katalon Platform, Selenium,Appium, Watir, Eggplant	Selenium, Testim, Katalon, Squish

WhiteBoxTesting(Manual Testing) :

White box Testing, som också kallas testning av glaslådor, strukturell testning, testning av tydliga lådor, testning av öppna lådor och testning av genomskinliga lådor. Den testar en programvaras interna kodning och infrastruktur och fokuserar på att kontrollera fördefinierade indata mot förväntade och önskade resultat. Den är baserad på en applikations inre arbete och kretsar kring testning av den interna strukturen. Vid denna typ av testning krävs programmerings färdigheter för att utforma testfall. Det primära målet med white box-testning är att fokusera på flödet av in- och utdata genom programvaran och att stärka programvarans säkerhet. Termen "white box" används på grund av systemets interna perspektiv. Namnet clear box eller white box eller transparent box betecknar förmågan att se genom programvarans yttre skal in i dess inre. Utvecklarna gör tester i vit låda. I detta test testar utvecklaren varje rad i programmets kod. Utvecklarna utför White-box-testningen och skickar sedan programmet eller mjukvaran till testteamet. White box-testningen innehåller olika tester, t.ex: Testning av banor, testning av slingor, testning av villkor, testning

baserad på minnesperspektivet; Testning av programmets prestanda. JUnit är ett exempel på white box-testning.

Smoke Test:

Ett Smoke Test är ett funktionellt test som avgör om en byggd version är stabil eller inte. Det verifierar funktionen hos viktiga funktioner för att se till att programmet kan klara ytterligare tester. Ett Smoke test utförs i början av livscykeln för programvaruutveckling (SDLC). Detta test bör alltid göras med varje nybyggd version eller utgåva som integreras med befintlig programvara. Smoke Testing utförs alltså före alla detaljerade regressions- eller funktionstester. Genom att utföra röktester tidigt i SDLC kan utvecklare eller QA-testare snabbt verifiera kvaliteten på byggnaden och utförandet av uppgifter. Builds med eventuella fel skickas snabbt tillbaka till utvecklingen innan tid slösas bort med mer uttömmande tester.Förbered dig för testning: Se till att du ställer in den atmosfär som du föredrar för röktestet. Detta innebär att du förbereder alla filer, servrar och licenser som du kan behöva för testet. Skapa kopior av dina filer och bygg också, så att du har säkerhetskopior om något skulle hända.
Samla in alla nödvändiga filer: Hämta alla bygg- eller kodfiler som du behöver för testet.
Skriv testskript: Använd ett enda skript för att köra testerna. Se dessutom till att ditt skript är skrivet så att det skapar och sparar en rapport efter varje test. På så sätt kan eventuella byggfel rapporteras korrekt och noggrant till utvecklarna.
Rengör data: Se till att testet körs i en ren miljö. Ta bort alla ovidkommande filer som kan påverka röktestet. Detta inkluderar även att stoppa servern och tömma databastabeller.
Selenium och PhantomJS är två av de bästa och mest populära verktygen för automatiserade Smoke test.

CI/CD tillsammans:

CI/CD är en bästa praxis för devops och agil utveckling. CI/CD omfattar en kultur, verksamhetsprinciper och en uppsättning rutiner som används av programutvecklingsgrupper för att leverera kodändringar oftare och på ett tillförlitligt sätt. Genom att automatisera integration och leverans kan CI/CD låta programvaruutvecklingsteam fokusera på att uppfylla verksamhetens krav samtidigt som man säkerställer kodkvalitet och programvarusäkerhet.

CI/CD-verktyg hjälper till att lagra de miljöspecifika parametrar som måste paketeras med varje leverans. CI/CD-automatiseringen gör sedan alla nödvändiga tjänsteanrop till webbservrar, databaser och andra tjänster som behöver startas om. Den kan också utföra andra förfaranden efter distributionen. Eftersom målet är att leverera kvalitetskod och -applikationer kräver CI/CD också kontinuerlig testning. Vid kontinuerlig testning utförs en uppsättning automatiserade regressions-, prestanda- och andra tester i CI/CD-pipelinen. Ett moget devops-team med en robust CI/CD-pipeline kan också genomföra kontinuerlig distribution, där applikations ändringar körs genom CI/CD-pipeline och övergående builds distribueras direkt till produktionsmiljön.
Grupper som implementerar kontinuerlig integration börjar ofta med konfigurationen av versionskontrollen och definitioner av praxis. Även om kontroll av kod görs ofta, utvecklar agila team funktioner och korrigeringar inom kortare och längre tidsramar. Utvecklingsteam som praktiserar kontinuerlig integration använder olika tekniker för att kontrollera vilka funktioner och vilken kod som är redo för produktion. Många team använder feature flags, en konfigurationsmekanism för att aktivera eller inaktivera funktioner och kod vid körning.I en automatiserad byggprocess paketeras all programvara, databas och andra komponenter tillsammans.Kontinuerlig integration paketerar inte bara all programvara och alla databaskomponenter, utan automatiseringen utför också enhetstester och andra typer av tester. Testning ger viktig återkoppling till utvecklarna om att deras kodändringar inte förstörde något. Med de flesta CI/CD-verktyg kan utvecklarna starta byggnationer på begäran, som utlöses av kodkommiteringar i versionskontrollförteckningen, eller enligt ett definierat schema.
Kontinuerlig leverans är automatiseringen som driver applikationer till en eller flera leveransmiljöer. Utvecklingsteam har vanligtvis flera miljöer för att lägga upp ändringar i applikationer för testning och granskning.En typisk pipeline för kontinuerlig leverans har bygg-, test- och distributionssteg.
När utvecklingsteamet har valt ett CI/CD-verktyg måste det se till att alla miljövariabler konfigureras utanför applikationen. CI/CD-verktyg gör det möjligt för utvecklingsteamen att ställa in dessa variabler, maskera variabler som lösenord och kontonycklar och konfigurera dem vid tidpunkten för distributionen för målmiljön.

Verktyg för kontinuerlig leverans tillhandahåller också instrumentpanels- och rapporteringsfunktioner, som förbättras när utvecklingsteam implementerar

observerbara CI/CD-pipelines. Utvecklarna får en varning om en byggning eller leverans misslyckas. Instrumentpanelen och rapporteringsfunktionerna integreras med versionshantering och agila verktyg för att hjälpa utvecklarna att avgöra vilka kodändringar och användarhistorier som byggdes.