Laboration 4

Continuous Integration

Med Continuous Integration menas att kod från sido-grenar i repot merge'as in i main-grenen med (oftast) ganska korta intervaller. Upplägget kan skilja sig mellan team beroende på vald branch-strategi och annat. Ju längre arbete som utförs i en dev-gren desto större blir risken att main-grenen har ändras så pass mycket sen man grenade av att merge-konflikter uppstår. Dessa går ofta relativt enkelt att lösa, men ibland händer det att rätt komplicerade konflikter dyker upp (om t.ex. flera feature-team har ändrat mycket på samma ställe) vilket då leder till att man kan behöva sammankalla alla inblandade för att gemensamt lösa konflikten.

Genom att istället försöka göra mindre merge'ningar ofta minskar man risken för detta. Ett vanligt arbetsflöde är att utvecklaren grenar av koden, gör sina ändringar och verifierar att alla enhetstester går igenom lokalt *innan* man skapar en PR mot main-grenen. Vanligtvis har man sedan ett antal tester/checkar som går mot PR'en, och krav på att dessa ska gå igenom för att kunna göra en merge. Efter detta granskas PR'en för att sedan merge'as in till main-grenen, där ytterligare tester körs.

Ibland händer det att vissa features måste merge'as in fastän de inte är helt färdiga 😲, och de inaktiveras då mha. feature-toggling innan koden rullas ut i produktion.

Continuous Delivery

Continuous Delivery är att automatisera flödet från källkod till release-bar produkt där varje körning av ett flöde ska vara en förutsägbar och kontrollerad process. Detta brukar innebära att merge'ning till main-grenen triggar att koden byggs och paketeras för att sedan deployas till t.ex. en staging/QA-miljö för vidare testning.
Efter att releasen har blivit godkänd/validerad och alla release-grindar har passerats krävs att en människa manuellt godkänner släppet till produktion.

Det viktigt att varje utvecklare är medveten om att allt som comittas/merge'as till main-grenen potentiellt kan släppas ut till produktion nästan direkt.
För mitt team är detta en kvalitetshöjande insikt. Ingen vill vara den som "sköt sönder prod" och därmed lägger man automatiskt mer tid på testning och validering innan man merge'ar in sin PR.

Continuous Deployment

Continuous Deployment är Continuous Delivery minus den mänskliga inblandningen vid släpp till produktion.
Utvecklarna och de automatiserade testerna ges så mycket förtroende att ett mänskligt godkännande inte krävs.
Man kan säga att Continuous Deployment är den naturliga fortsättningen av ett lyckat Continuous Delivery-arbete.

En tanke jag ofta hör om automatiserad deployment är att *"ingen är bättre på att hitta buggar än våra kunder/användare"* och att det är bättre att dessa buggar upptäcks kort efter släpp då de oftast är lättare att hitta.

Inom vissa områden finns strikta regelverk (militär, medicin, telekom etc.) som förhindrar användningen av CD.

Agila metoder

Agila metoder är iterativa och flexibla med fokus på samarbete och snabba leveranser. Istället för att följa en linjär plan från början till slut arbetar man styrt av feedback och anpassning under hela projektet.

Arbetet delas upp i iterationer där varje iteration levererar en fungerande del av produkten.
Man strävar också hela tiden efter att förbättra både produkten men även processerna kring arbetet.

Scrum

Scrum är en metodik där man jobbar i sprintar (iterationer) om 1-4 veckor och team-storleken ska i idealfallet vara 3-9 personer.
Under varje sprint planerar, utvecklar, testar och levererar teamet en fungerande del av produkten. Teamet är självorganiserande och ska helst bestå av alla de olika kompetenser som krävs för att nå sprint-målet.

I teamet finns två tydligt definierade roller:

- Produktägaren ansvarar för att prioritera och representera kundens intressen.
- Scrum-mastern organiserar sprintens alla ceremonier (planering, daily, review och retro) och har även till uppgift att hjälpa teamet genom att undanröja ev. hinder.

Kanban

I Kanban finns inte samma sprint-tänk som i Scrum. Istället visualiseras arbetet som ska utföras som kort på en Kanban-tavla uppdelad i sektionerna "Att göra", "Pågående" och "Klart".
Teamet sätt också upp begränsningar för hur många kort som kan vara pågående samtidigt, vilket undviker överbelastning och sätter fokus på att slutföra pågående arbeten.
Även Kanban strävar efter kontinuerlig förbättring genom att mäta t.ex. genomströmning och tidsåtgång för kort. Med denna data kan teamet hitta flaskhalsar men också förbättringar i arbetsflödet.

SAFe

SAFe är ett ramverk för att skala upp agila metoder för att hantera stora och komplexa projekt. Precis som i alla agila metoder betonar även här vikten av kontinuerlig förbättring.
SAFe är uppbyggt kring en hierarkisk struktur med tre nivåer:

- Portföljnivån - fokus på strategisk planering och prioritering.
- Programnivån - planering och koordinering.
- Teamnivån - självorganiserande utveckling.

DevOps

DevOps kombinerar utveckling (Dev) och drift (Ops) med målet att skapa strömlinjeformade processer för kontinuerlig leverans (CI/CD) och drift.
En av grunderna i DevOps är just automatiseringen av bygg-, test- och distribution- och övervakningsflöden. Det täta samarbetet mellan utvecklarna och driftsteamet leder till korta feedback-loopar, där buggar och förbättringar snabbt återkopplas.

QA (Quality Assurance)

Kvalitetssäkring (QA) är en process för att säkerställa att en produkt uppfyller de fastställda kraven, och eventuella andra standarder. Syftet är att upptäcka och åtgärda eventuella buggar eller fel i en produkt innan den släpps ut till kund.
Inom QA ingår att definiera kvalitetskrav, skapa testplaner och testfall, genomföra tester. Även prestanda- och säkerhetstester brukar vara QA's ansvar.
Genom utvärdering av resultaten kan kvaliteten kontinuerligt förbättras.

Exempel på testverktyg + CI-verktyg som kan användas för att automatisera

Här kan jag ha missförstått. Du kanske vill ha mer beskrivande exempel än bara verktygsexempel?

Enhetstester

Testverktyg: JUnit och NUnit.
CI-verktyg: TeamCity, Jenkins och Azure Pipelines.

Integrationstester (API)

Testverktyg: Selenium (UI) och Postman/Newman (API).
CI-verktyg: TeamCity, Jenkins och Azure Pipelines.

Systemtester

Testverktyg: Cucumber och Ranorex.
CI-verktyg: TeamCity, Jenkins och Azure Pipelines.

Acceptanstester

Testverktyg: Appium och Selenium.
CI-verktyg: TeamCity, Jenkins och Azure Pipelines.

Ge exempel på ytterligare två typer av mjukvarutester och hur de kan genomföras (manuellt eller automatiserat)

Lasttestning

Lasttestning används för att hitta buggar och begränsningar/flaskhalsar i ett system eller en applikation.
Exempelvis kan man simulera ett hundratal användare som samtidigt loggar in på en sajt och sedan navigerar till en sida där flera databasanrop görs. Genom att mäta den latency'n som kan uppstå kan man bedöma om t.ex. databasen behövs skalas upp eller om andra åtgärder krävs.
Ett liknande testfall är lasttestning av mikrotjänster, där man "bombar" tjänsten med anrop för att säkerställa att den inte går ner.
Lasttestningen är i sin natur lämpad för automatisering då den ofta simulerar ett mycket stort antal användare.

Monkey testing

Monkey-testning är en typ av stresstestning där en stor mängd pseudo-slumpvisa händelser skickas till en applikation i snabb följd för att verifiera att den inte kraschar. Eftersom händelserna är pseudo-slumpvisa går det i efterhand att återskapa sekvensen som ledde fram till en krasch.
Testerna delas oftast in i två kategorier:

- Smarta tester har kännedom om applikationen som testas och baserar sitt beteende på tidigare skickade händelser. Syftet med testerna är vanligtvis att på något sätt knäcka applikation.
- Dumma tester har ingen kännedom och skickar händelserna slumpvis.
  Oftast hittas betydligt färre buggar än de smarta testerna, men de som hittas är istället otroligt svåra att komma på testfall för.
  Monkey-testning är en typ av testning som måste automatiseras pga. av den stora, och snabba, mängden av händelser som skickas.

Beskriv hur CI/CD kan användas tillsammans med olika branscher i ett repository för att skapa ett arbetsflöde för att utveckla, testa och publicera kod enligt en agil metod. Använd gärna illustration.

Här beskriver jag vår tolkning av trunk-baserad utveckling, och den kanske skiljer sig den "officiella" versionen, men den funkar för oss.

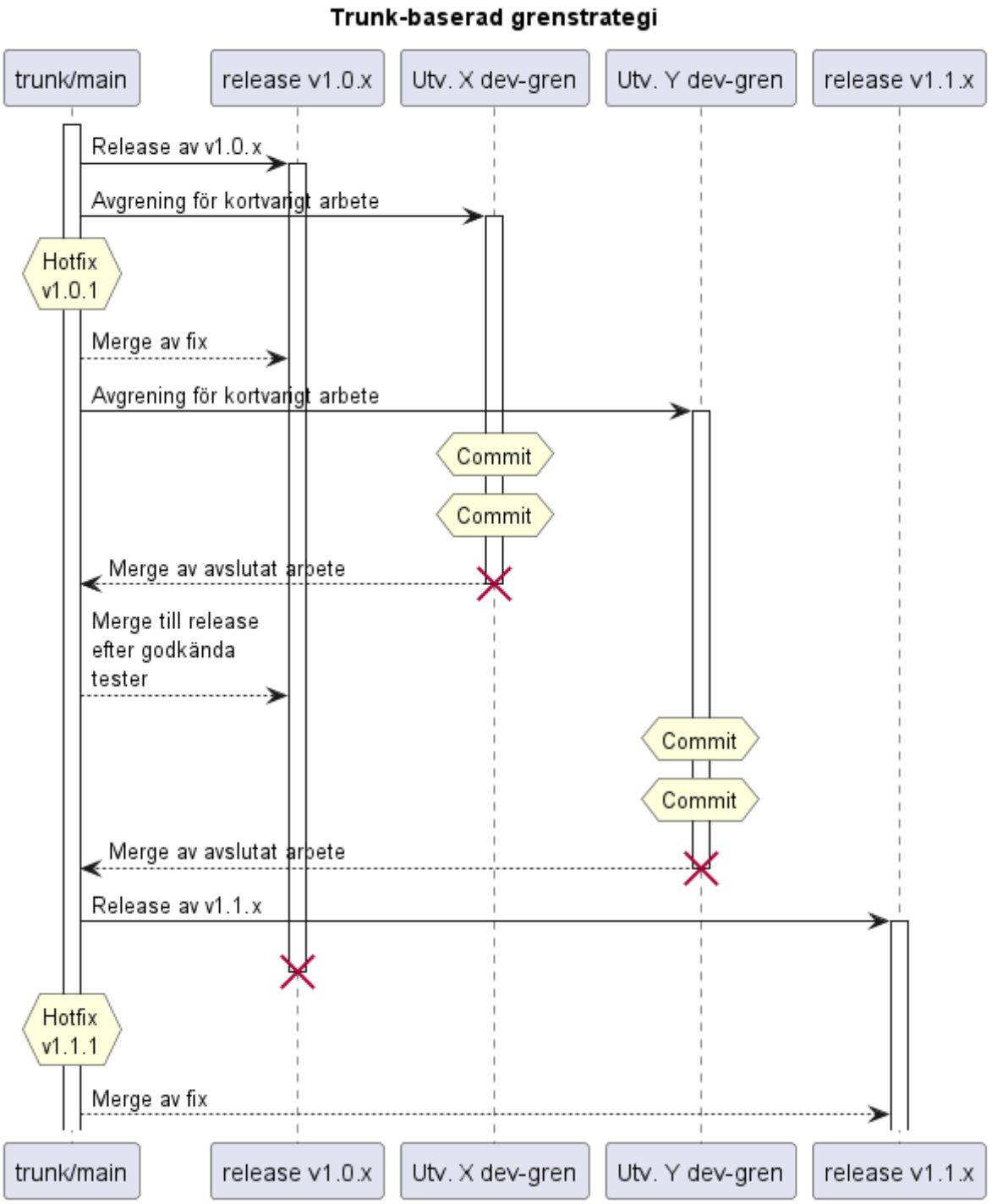
För att nyttja CI/CD fullt ut bör man använda en trunk-baserad grenstrategi, och inte en GitFlow-variant med långlivade dev- och feature-grenar.
Koden i trunken (main) går att deploya när som, men det är vanligt att man ändå använder sig av release-grenar för att hålla lite koll på vad som ligger ute. Det underlättar även (tycker vi) när man ska deploya ny funktionalitet till test-/QA-miljöer.

Vid utveckling av ny funktionalitet grenar utvecklaren av från trunken och skapar en personlig dev-gren. Ett antal commit'ar görs och utvecklaren ansvarar även för att berörda enhetstester fortfarande går igenom. Vid avslutat arbete görs en PR tillbaka mot trunken, och den merge'as efter att ha blivit godkänd. Testarna ska nu ha skrivit klart sina tester för funktionaliteten, och som en del av PR-godkännandet körs de nya och befintliga automatiserade tester.

Efter godkännande kan funktionaliteten befordras/taggas (promoted) till releasekandidat. Vid denna taggning startar ett automatiserat produktionsbygge som deployar till en QA-miljö om allt går bra. Tillgängligheten av den nya funktionaliteten styrs av feature-flaggor så det skulle gå att skicka ut i produktion direkt om funktionaliteten stängs av (togglas).

Fler tester, både automatiserade och manuella, görs mot QA-miljön och om alla tester går igenom befordras/taggas bygget som stabilt (stable) och deployas automatiskt ut i produktion.

Rinse and repeat.


Trunk-baserad grenstrategi
trunk/main
release v1.0.x
Utv. X dev-gren
Utv. Y dev-gren
release v1.1.x
Release av v1.0.x
Avgrening för kortvarigt arbete
Hotfix
v1.0.1
Merge av fix
Avgrening för kortvarigt arbete
Commit
Commit
Merge av avslutat arbete
Merge till release
efter godkända
tester
Commit
Commit
Merge av avslutat arbete
Release av v1.1.x
Hotfix
v1.1.1
Merge av fix
trunk/main
release v1.0.x
Utv. X dev-gren
Utv. Y dev-gren
release v1.1.x