(Løsningsforslag) Ukesoppgaver uke 14: DevOps and Code Management

Oppgavene er hentet fra Kap 10 i Sommervilles bok Engineering Software Products som ligger tilgjengelig fra kurssiden ved innlogging.

- 10.1. Explain why adopting DevOps provides a basis for more efficient and effective software deployment and operation.

DevOps gir hurtigere og bedre utgivelser fordi – når alle er med på alt – reduserer det tiden man bruker på å kommunisere, man kan sette alle til å fikse bugs, alle i teamet har like stort eierskap og forståelse for hva som er lurt og ikke, og små endringer i hver utgivelse gjør fatale feil mindre sannsynlig. Automatiseringen av det meste gjør også at man kan jobbe effektivt med de viktigste arbeidsoppgavene.

- 10.2. Briefly explain why it is essential to use a code management system when several developers are involved in creating a software system. What are the benefits of using a code management system if only a single developer is involved?

Det er essensielt å bruke et kodehåndteringssystem når man er flere utviklere for å kunne kontrollere koden som styrer programvaren. Med et håndteringssystem kan man jobbe parallelt, unngå å skrive over viktig kode, se endringshistorikk, legge til funksjonalitet uten å endre master-koden, håndtere ulike versjoner av koden, dobbeltsjekke og teste før man legger til endringer i masterkoden og håndtere lagring.

Fordelene med å bruke et kodehåndteringssystem når man jobber alene er de samme. Det er lurt å kunne se tidligere versjoner av sin egen kode og man kan da lettere backtracke, rette opp i og unngå feil.

- 10.4. What does creating a new branch in a code management system mean? What problems can arise when more than one developer is working on the same code and they try to merge their changes with the project master branch?

Når man lager en ny branch i kodehåndteringssystemet Git lager man en ny versjon av masterfilen/filen man brancher ut av. Dette gjør det mulig å endre på funksjonene i masterfilen uten å ødelegge/endre det som ligger i der.

Når man er fornøyd og har testet funksjonaliteten man har lagt til i branchen forsøker man gjerne å merge den inn i masterfilen. Dersom masterfilen da er annerledes enn den versjonen man lagde branchen sin av, skaper dette problem. Dette kan skje dersom en utvikler har merged sin branch inn i masterfilen i mellomtiden. Problemet oppstår fordi koden i din branch, som skulle vært lik med masterfilen, ikke er det lenger, og det kan oppstå konflikter som skaper nye bugs og feil. Du og den andre utvikleren må da gå gjennom koden manuelt og velge ut hva som skal være i masterfilen og hva som skal slettes.

Evt også denne:

- 10.9. What are the differences between continuous integration, continuous delivery, and continuous deployment?

Continuous integration handler om kodehåndtering - at man automatisk integrerer ny, ferdig testet kode i programmet etterhvert som utviklere gjør seg ferdig med sine deler av koden. Det kan være fra ny funksjonalitet til bare en endring av variabler. Prinsippet er at koden skal være så oppdatert som mulig, godt integrert med alt man har av kode, hele tiden.

Continuous delivery handler om å sørge for at programmet alltid er klart for å slippes. Dette gjøres gjennom kontinuerlig testing, både av selve programmet, men og av funksjoner og kodesnutter som legges til. En masterbranch skal alltid være ferdig testet, klar for levering.

Continuous deployment handler om å alltid kunne - i prinsippet - rulle ut programmet til brukerne. Man sørger for dette ved å kjøre automatiske tester av programvaren på maskinene og plattformene den er ment å fungere på.

Alle disse DevOps metodene har rot i automatisering. Fordi for å sørge for kontinuitet bør man automatisere aktivitetene. Forskjellen ligger både i hvilke automatiseringverktøy man kan bruke for å oppnå kontinuitet, og hvilke steg i utvikllingsprosessen de er spiller størst rolle. Continuous integration kan man sørge for med kodehåndteringsverktøy som Git, og er noe utviklerne må jobbe for å opprettholde (de må committe og pushe koden). Continuous delivery kan man sørge for med testeverktøy som SquareQube og Jenkins og som må få tilgang til master branchen i Git. Mens Continuous deployment må ha hardware/miljø å teste programvaren i - og har et høyere abstraksjonsnivå enn de to andre metodene.

En tidligere eksamensoppgave som er relevant er:

- Forklar hvorfor versjonshåndtering er viktig.

Versjonshåndtering er viktig fordi vi ofte gjør feil når vi koder. Versjonshåndtering gjør det mulig å se tilbake på hva som er blitt lagt til fra koden fungerte til den ikke fungerer lenger. På den måten kan vi lettere kontrollere og fikse feil og bugs.