Sensorveiledning Software Engineering H2022

Symbol	***Betegnelse***	***Generell, ikke fagspesifikk beskrivelse av vurderingskriterier***
A	Fremragende	Fremragende prestasjon som klart utmerker seg. Kandidaten viser svært god vurderingsevne og stor grad av selvstendighet.
B	Meget god	Meget god prestasjon. Kandidaten viser meget god vurderingsevne og selvstendighet.
C	God	Jevnt god prestasjon som er tilfredsstillende på de fleste områder. Kandidaten viser god vurderingsevne og selvstendighet på de viktigste områdene.
D	Nokså god	En akseptabel prestasjon med noen vesentlige mangler. Kandidaten viser en viss grad av vurderingsevne og selvstendighet.
E	Tilstrekkelig	Prestasjonen tilfredsstiller minimumskravene, men heller ikke mer. Kandidaten viser liten vurderingsevne og selvstendighet.
F	Ikke bestått	Prestasjon som ikke tilfredsstiller de faglige minimumskravene. Kandidaten viser både manglende vurderingsevne og selvstendighet.

Det vil i mange av oppgavene være mulig å begrunne andre løsninger enn de foreslåtte, da det bare er hovedtrekkene som beskrives for hver oppgave.

Emnene er såpass omfattende at det ikke vil være forventet at kandidaten går i dyptgående detaljer om hvert av punktene i hvert av svarene, men at kandidaten viser detaljert kunnskap der det er nødvendig for å forklare forskjeller mellom ulike metoder og konsepter.

Sensuren skal vurdere i hvilken grad kandidaten *i helhet* (og ikke utelukkende som en summering av “poeng” for hver oppgave, men omfanget angitt kan brukes som et utgangspunkt for vekting) har kunnskap om emnene til å vite når og hvordan de bør benyttes og hvordan de bygger opp under trygg og stabil programvareutvikling.

Oppgave 1

1. Hovedmålet er at kandidaten skal vise hvordan bruken av applikasjonen henger sammen med virkeligheten. Oppgaveteksten tilsier at scenarioet som beskrives først minst bør ha med opplisting av arrangement, filtrering på sted, valg av antall billetter, bestilling, betaling og levering av billetter. Deretter bør det beskrives hvordan inngangen på konsertstedet håndteres, og ev. om det gjøres noen oppfølging etter at konserten er over.

   I del 2 skal kandidaten hente ut de relevantene kravene til applikasjonen ut fra sin beskrivelse. Kravene skal være beskrevet med en tydelig og konsis tekst. Hvilke krav som er forventet her avhenger av hvordan scenarioet til kandidaten er utformet, men det er forventet at kravene dekker typiske brukssituasjoner for applikasjonen.

2. Ikke-funksjonelle krav beskriver egenskaper ved systemet, krav som gjerne styrer hvordan andre krav blir utviklet. Funksjonelle krav beskriver faktiske funksjoner i systemet; de tingene som gjerne må kodes (og funksjonelle krav er som oftest knyttet til en brukergruppe).

3. Ikke-funksjonelle krav må beskrives på en slik måte at de er testbare.

Oppgave 2

1. Containerization handler om hvordan vi bundler applikasjonen, miljøet og avhengighetene dens til en pakke - en container. Dette er selve applikasjonen og det den trenger for å kjøre (bibliotek, ev. interpreter for programmeringsspråket, innebygd webserver, etc.)

   Orchestration handler om hvordan vi distribuerer og styrer containerene våre, og om hvordan vi kan automatisere strukturen av "datasenteret" vårt - Infrastructure as a Service. Orchestration koordinerer containere og ressursene, mens containerne er selve applikasjonene vi ønsker å kjøre.

2. Versjonskontroll gjør at vi kan spore endringer i oppsettet og miljøet rundt både applikasjonen (endringer i containeren), og endringer i infrastrukturen den kjører på og hvordan den distribueres og gjøres tilgjengelig (orchestration). Dette fungerer både som dokumentasjon, om informasjon om hvilke endringer som har blitt gjort når, av hvem og hvorfor - og gjør at vi kan rulle tilbake til tidligere versjoner ved behov.

Oppgave 3

1. Semantisk versjonering går ut på at vi er enige om en konvensjon angående for hva de ulike delene av et versjonnummer betyr. Versjonsnummeret er hovedsaklig beskrevet på formen <major>.<minor>.<patch>. <patch> er kun bakoverkompatible bugfikser, <minor> kan introdusere nye features så lenge de nye egenskapene er bakoverkompatible, mens endringer i <major> kan introdusere endringer som ikke er bakhoverkompatible (unntak er når <major> er 0, da er det antatt at endringer som ikke er bakoverkompatible kan introduseres når som helst - men dette skjer som oftest ikke i praksis).

   Dette kan gi oss mer stabil programvare fordi vi kan ha en formening om hvorvidt endringene i bibliotek vi avhenger av vil brekke vår egen programvare (som testene våre bør avsløre uansett, men det gir oss en forventning om hva som skal skje).

2. Målet med en låsefil er at nøyaktig de samme avhengighetene som ble installert første gang blir installert senere, og de lar oss foreta oppgradering av avhengighetene som en eksplisitt handling - og ikke noe som skjer uten at vi er klar over det eller har kontroll på det.

   Signaturen i låsefilen sørger for at ikke avhengigheten som ble lastet ned første gang kan bli byttet ut på serveren vi laster ned avhengigheten fra (men fortsatt ha samme versjonsnummer). Den garanterer at det som blir lastet ned senere er nøyaktig det samme som det ble lastet ned første gang.

Oppgave 4

Branching handler om at vi kan splitte utviklingsløpet (versjoneringa) ut i forskjellige grener (branch) som kan utvikles uavhengig av hverandre. Endringer som gjøres i en gren påvirker ikke andre grener, med mindre utviklerne gjør et bevisst valg om å hente inn endringer eller slå sammen grenen med andre grener (merging).

Merging handler om at en gren slås sammen ved at en gren flettes (merges) inn i en annen gren. Dette gjør det mulig å drive utviklingen av forskjellige nye features uavhengig av andre som jobber med andre ting i samme programvareprosjekt.

Branches danner også grunnlaget for at vi kan gjennomføre pull requests, og at vi har muligheten til å gjøre code review etc. før merging blir gjennomført.

Oppgave 5

Det skal ikke legges vekt på syntaksfeil eller trivielle småfeil. Pseudokode er greit.

Det forventes at hvert krav og de to underpunktene har egne tester som bekrefter at oppførselen er korrekt. For full uttelling bør det også ha blitt skrevet tester som viser at de negative responsene i metodebeskrivelsene fungerer som de skal (slik som at kandidaten tester hvorvidt det å legge til samme bok to ganger gir `false` ved andre forsøk).

Pass på at testene har fornuftige navn som forteller hva som testes, og at disse beskrivelsene henger sammen med kravene eller metodebeskrivelsene.

Et eksempel på hvordan testene kan se ut i Python:

Vi antar at bokdatabasen er tom foran hver test, f.eks. ved at det er satt opp via en fixture eller i en setup-metode. Kandidaten står fritt til å gjøre samme antagelse.

```
def test_add_book():
     assert add_book("a_book_id")


def test_book_should_not_be_added_if_book_already_exists():
     assert add_book("a_book_id")
     assert not add_book("a_book_id")


def test_lender_can_lend_a_book():
     assert add_book("a_book_id")
     assert lend_book("a_book_id")


def test_lender_can_not_lend_a_book_already_lent_out():
     assert add_book("a_book_id")
     assert lend_book("a_book_id")
     assert not lend_book("a_book_id")


def test_lender_can_not_lend_a_book_that_doesnt_exist():
     assert not lend_book("a_book_id")


def test_added_book_is_shown_as_in_library():
     assert add_book("a_book_id")
     assert has_book("a_book_id")


def test_not_added_book_is_not_shown_as_in_library():
     assert not has_book("a_book_id")
```

```
def test_lender_can_return_book():
     assert add_book("a_book_id")
     assert lend_book("a_book_id")
     assert return_book("a_book_id")


def test_lender_can_not_return_a_book_already_returned():
     assert add_book("a_book_id")
     assert lend_book("a_book_id")
     assert return_book("a_book_id")
     assert not return_book("a_book_id")


def test_lender_can_not_return_a_book_never_lent():
     assert add_book("a_book_id")
     assert not return_book("a_book_id")


def test_lender_can_not_return_a_book_that_doesnt_exist():
     assert not return_book("a_book_id")
```