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 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 - Prosjektform

a)
Eksempler
Planbasert

- Rigid prosess
- Distinkte faser i prosjektet
- Programvare implementeres etter en overordnet, oppsatt plan
    - Krav og spesifikasjon er satt før kode skrives
- Rammebetingelsene settes tidlig

Iterativ

- Løsere prosess
- Enklere endring av krav og implementasjon
- Tilbakemelding fra kunde oftere
- Levere tidlig verdi til kunde
- Testing fortløpende i prosessen

b)
Eksempler
Planbasert - typisk integrasjonsoppgavene bakover i systemet

- Delene av systemet som krever sertifisering mot bankens betalingssentral, kortleverandører og tilsvarende
- Delene av systemet som er underlagt lover med gitte krav som må følges
    - Lovverket
    - Finanstilsynet
- Moduler med stor økonomisk konsekvens, slik som låneutbetaling
- Rapportering til staten (årsoppgaver, etc.)

Iterativ og inkrementell

- Hovedfunksjonaliteten for brukeren i nettbanken
    - Låne/sparekalkulatorer
    - Prosessene med kjøp av fond etc. fram til kjøpet prosesseres

- Lånesøknader for brukere og kundebehandlere
- Kundestøttefunksjonalitet
- Kontoopprettelse, etc.

Det er under forelesningene lagt spesiell vekt på at utviklingsløp kan følge forskjellige modeller, både innenfor samme prosjekt og innenfor samme modul - f.eks. med planbasert utvikling i de tidligere fasene, og mer iterativ og inkrementell utvikling i senere faser.

c)
Planbasert utvikling benyttes mye i store og "trege" industrier med enten lang produksjonstid eller høyt krav til trygghet og avsjekk av rutiner, der kravene ikke endrer seg vesentlig over tid.

Eksempler:
- Embeddet programvare / større maskinvareprosjekter med lang produksjonstid og liten mulighet for å endre oppførsel etter produksjon
- Kritiske systemer med fare for liv og helse
- Flyindustrien
- Militæret
- Kjernekraftverk
- Koordinasjon mellom helt separate grupper med utviklere, eller der maskinvare og programvare utvikles av separate grupper

Oppgave 2 - Krav

a)
Kandidaten bør beskrive at funksjonelle krav handler om funksjonaliteten i systemet mens ikke-funksjonelle krav beskriver "alt rundt". Ikke-funksjonelle krav bør kunne kvantifiseres / være målbare.

b)
Kandidaten står fritt til å finne opp egne krav som er relevante for scenariet.

Mulige funksjonelle krav fra scenarioet:	*Mulige ikke-funksjonelle krav:*
- Se saldo - Ta i mot overføring til konto - Betale regning - Overføre penger mellom kontoer - Se utestående balanse på kredittkort - Kjøpe/selge fond - Kontakte kundeservice - Søke om lån - Se innvilget lån	- Kjøre på <programmeringsspråk> og ev. Versjon - Kjøre på plattform/operativsystem xyz - Følge aktuell lovgivning og godkjenninger (overordnet) - Ikke skape mer enn en henvendelse pr. bruker pr. år til kundeservice - Ha x% oppetid - Kunne oppgraderes uten nedetid / med maks xx minutters nedetid

- Se nedbetalingsplan for lån - Se priser - Kommunikasjon fra banken - Årsoppgave	- Kjøre på alle nettlesere med minst x% markedsandel - Kjøre på serverer med 8GB minne - Gi kontooversikt i løpet av ett sekund - Ingen sidevisninger skal ta mer enn fem sekunder

c)
Ved planbasert utvikling er kravene definert som en stor jobb tidlig i prosessen, mens under Agile utvikling registreres og endres kravene fortløpende etter hvert som de oppdages - f.eks. i starten av en sprint.

Oppgave 3 - Estimering

a)
Metoder som er forelest

- Sammenlign med en enkelt faktorer som representerer prosjektstørrelse fra tidligere ("Count, Compute, Judge" og "Estimate by proxy")
- Grovestimering og klassifisering ("T-Shirt sizing")
- Story / Effort points - planning poker
- Dekomposisjon og rekomposisjon (mange små vs et stort estimat)
    - Dele opp krav i mindre biter
- Bygg en prototype
- Algoritmisk estimering

b)
Det viktigste for å få bedre nøyaktighet er å måle tid brukt og justere fremtidige estimat ut fra kjente avvik. Andre muligheter, men som ikke handler om tid:

- Flere personer estimerer samme oppgave
- Bruke flere estimeringsteknikker på samme oppgave

Oppgave 4 - Testing

a)
Kandidaten bør nevne de forskjellige nivåene og innholdet i testene (testing av avhengigheter) og rollen med akseptansetesting foran endelig overlevering til kunde.

Eksemplene bør være noe som er isolert og enkelt for enhetstestene og overordnet funksjonalitet for integrasjons- og akseptansetestene.

b)
Testdrevet utvikling i hovedsak at testen skrives før kode skrives eller endres (spesielt for legacy-systemer). Koden skrives deretter for at testene skal passere. Testene beskriver kravene til funksjonen i (tilnærmet) helhet. Red - Green - Refactor som teknikk innenfor TDD er også lagt vekt på.

c)
Test doubles som fakes, stubs og mocks hjelper oss med å skrive tester der koden har avhengigheter, gjerne de som har sideeffekter eller er eksterne.

Disse teknikkene gir oss metoder å teste denne oppførselen, uten at vi avhenger av, eller må bry oss om den eksterne avhengigheten.

Oppgave 5 - Containerization

Her bør kandidaten beskrive at omgivelsene og avhengighetene til applikasjonen blir stabile og at vi får en “enhet” (container) som ser identisk ut uavhengig av om det er lokal utvikling, testing eller produksjon.

Fordeler kan være:

- Isolasjon
    - Applikasjoner vet ikke om hverandre
    - Avhengighetene er utelukkende begrenset til containeren selv
- Stabilitet
    - Oppgraderinger av en container påvirker ikke en annen
    - Containeren kan restartes i sin initielle tilstand ved problemer
- Reproduserbarhet
    - Containeren er identisk i produksjon, test og lokalt - avhengighetene og miljøet er det samme
- Lett å distribuere, flytte og skalere horisontalt