INF112 – Ting som er viktig for eksamen

Om eksamen

- Vi kjører «åpen bok»-eksamen (dvs. alle skrevne og trykte hjelpemidler tillatt). Det vil si at spørsmålene i liten grad vil handle om konkrete fakta, men heller at du skal vurdere, reflektere over eller analysere noe.
- Tema for oppgavene vil falle innenfor:
    - Team-arbeid, kommunikasjon og sosiale ting
    - Utviklingsmetodikk, krav, planlegging, osv
    - Teknisk – abstraksjon, objekt-orientering, testing, design patterns, osv
    - Verktøy – versjonskontroll, bygging, testing, kvalitetssjekk, deployment
- Typisk form på oppgavene:
    - Forklar hvorfor / vurder i forhold til noe («hva er forskjellen på Scrum og Kanban? Hvilke fordeler/ulemper har de?»)
    - Gi eksempel på nytten av, eller riktig/feil bruk av noe («Hva er vitsen med *Factory* design pattern? Gi et eksempel som illustrerer dette»)
    - Gi tilbakemelding på noe – hva er «feil» i denne koden? («Hvordan vil det være å utvide/endre denne klassen til å også støtte var-zombier? Hvordan vil du evt. forbedre designet?»)
    - Tenk deg denne situasjonen (f.eks. team med kommunikasjonsproblemer), hva slags råd ville du gitt?
- For noen av oppgavene forventer vi korte svar, f.eks. noen setninger – dette står i oppgaveteksten

De viktigste tingene har du allerede lært når du jobbet med prosjektet! Bruk gjerne tid til å tenke gjennom erfaringene, og evt diskutere dem med andre.

Begrunnelser

Det er viktig å gi gode begrunnelser, f.eks.

- «I morgen skal dere ha møte med den nye sprint-eksperten på Scrum-teamet. Du har ansvar for å skaffe snacks, og Petter er kanin. Hva gjør du?»

    - ok svar: «Jeg vil gjerne at han skal føle seg velkommen og at vi alle får et positivt førsteinntrykk, så jeg kjøper inn gulrøtter, for jeg vet kaniner liker gulrøtter.» – Akkurat hva slags snacks du skaffer er ikke viktig, men du forklarer *hvorfor*, hva du prøver og oppnå og hvorfor du mener det er viktig for teamarbeidet.
    - dårlig svar: «Jeg kjøper gulrøtter.» – Kan være gulrøtter er den beste løsningen, men vi vil vite hva du har tenkt.

Valgoppgaver

På noen av oppgavene skal du velge én av deloppgavene og besvare *kun* denne, mens på andre skal du svare på alle deloppgavene – pass på å lese oppgaven nøye så du ikke gjør unødig arbeid.

Kilder

Vi har ikke sett så mye på formell litteratur om f.eks. møter og kommunikasjon, så det er ikke forventet at dere refererer til kilder på f.eks. hvordan man håndterer en uenighet i teamet.

Men:

- Hvis du henter informasjon fra skriftlige kilder du har med, er det naturlig at du refererer til kilden i besvarelsen.
- Hvis du siterer fra en kilde – f.eks. besvarer eksempelspørsmålet om Scrum og Kanban med å (blant annet) kopiere eller omskrive fra side 49/50 i Scrum/Kanban-boken, *må* du gjøre det tydelig hvor du har hentet ting fra og om det er et direkte sitat eller en omskriving – noe annet er fusk
- Dette gjelder selvfølgelig også om du kopierer eksempelkode. Tenk også på at hvis spørsmålet er «lag et eksempel» så imponerer du ikke sensor ved å bruke samme eksempel som i boken!

Les mer på UiBs sider om kilder

Pensumoversikt

Krav, ønsker og behov

- Funksjonelle og ikke-funksjonelle krav – «kan importere fra CSV» vs. «skal være robust mot feil i input-filer»

- Brukerhistorier – «As a *goat* I want *more bushes and shrubs* so that I can *ruminate about the quality of my code*.»
- Akseptansekriterier – «Gitt X, så skjer Y», testbart (akseptansetesting) og klart definert – brukerhistorie handler om behov, akseptansekriterie handler om å definere hvordan vi måler at vi når behovet. Når akseptansekriteriene er oppnådd, er vi ferdig med brukerhistorien
- Arbeidsoppgaver – Hva må gjøres for å oppfylle akseptansekriteriene og kunne gjøre brukerhistorien til virkelighet?
- Minimum Viable Product – Det minste som må være på plass for at vi skal kunne gå i produksjon. Viktige krav kan gjerne løses manuelt eller på andre måter enn å lage kode for det – det ABSOLUTTE minimum, ikke ferdig system.

Planlegging

- Project board
- Prosjekt og arbeidsmetodikk
    - The Agile Manifesto
    - XP, Scrum, Kanban

Kommunikasjon

- Kunnskapsdeling og -overføring
- Forstå hverandre, vite hva vi holder på med, løse de rette problemene
- Være trygge og yte sitt beste
- Epost, chat, issue tracking, ansikt-til-ansikt, møter, telefon, video, figur, tekst – og *kode*
- Hva er best i hvilken situasjon? Hvordan håndtere vanskeligheter/ubehageligheter

Versjonskontroll

- Hensikt? spore og distribuere endringer
- Git – teknisk: working directory, staging, repositorium (lokalt og remote/server); merge og konflikt; branching

- Git vs. git-tjenester (GitLab, GitHub, etc)
- Utviklingsmodeller («git-flow): trunk-based, branching

Bygg og deploy

- Hvorfor? Håndtere teknisk rundt kompilering, testing og distribuering

Kvalitet

Programvarekvalitet

- Pålitelighet – gir *korrekt* resultat (gyldig input gir gyldig resultat) – («jeg er ikke sikker på om dette vil funke» / «jeg må sjekke resultatet for hånd for å stole på det» → programmet er upålitelig)
- Robusthet – håndterer feil, ugyldig input oguforutsette situasjoner – både brukerfeil, programmeringsfeil og eksterne feil (hardware, natur). («editoren kræsjer av og til når jeg prøver å lagre» / «VIKTIG! ikke trykk på knappen mens det røde lyset lyser» → programmet er lite robust)
- Fleksibilitet – kan tilpasses / brukes annerledes når kravene endrer seg («jeg implementerte spill-logikken i Excel» → regneark er veldig fleksible!)
- Vedlikeholdbarhet – hvor lett det er å finne og rette feil, eller endre oppførselen («hver gang jeg fikser en bug kommer det to nye» – koden er *skjør* slik at én endring/bugfix lett fører til feil andre steder)
- Sikkerhet – nært knyttet til punktene over (er programmet så «fleksibelt» at det kan misbrukes? robusthetsfeil er ofte sikkerhetsproblemer, i det minste i form at *denial of service*)
- Effektivitet – «får jeg svar raskt nok til at det er vits i å vente?» / «denne appen spiser batteriet» – responstid og jevn respons er ofte viktig for brukere, ok å vente på svar så lenge du vet at det skjer noe
- *Kodekvalitet* – formattering, lesbarhet/navn osv

Arbeid med kvalitet:

- Code reviews
- Statiske analyseverktøy – f.eks. SpotBugs eller SonarQube
- Kodekvalitet kan måles (bla. med SonarQube)

Programmering

(Faller kanskje inn under 'kvalitet'? eller omvendt.)

Mye av dette er dekket i INF101, så det blir ikke spurt om direkte, men vi regner med at du kan det – sjekk evt. denne oppsummeringen

- Abstraksjon – interfaces, klasser, metoder, generics – forenkle, se bort fra uvesentlige detaljer, fokusere på de essensielle detaljene
    - **Abstraksjon i programmering er like viktig som kommunikasjon i teamarbeid – det gir deg et *språk* som gjør at du kan kommunisere tydelig med maskinen og med andre programmører!**
- Objektorientering
- Feilhåndtering og exceptions
- Modellering – se f.eks. INF112 tips
- Separation of concerns – skill forskjellige aspekter av funksjonaliteten fra hverande. F.eks., *model-view-controller* sørger for å skille modellen (informasjonen som prosesseres) fra viewet (hvordan det vises til brukeren) – kan f.eks. gjøre det enklere å lage flerspillerklient i et spill eller gjøre programvaren tilgjengelig for blinde eller på forskjellig plattformer eller forskjellige språk

Gode prinsipper for å skrive gjenbrukbar, vedlikeholdbar kode:

- SOLID, GRASP
- KISS (keep it simple), DRY (don't repeat yourself)

Testing

- Hvorfor og hvordan
- Standard bruk av JUnit
- Mocking – hvorfor? Verktøy som f.eks. Mockito

- Parametriserte tester, generering av testdata

(Hvis det er spesifikke ting dere skal bruke på eksamen vil det følge med dokumentasjon.)

Sikkerhet

- Ser vi nøyere på i INF226
- Pålitelighet og robusthet er viktig for sikkerhet – hackere utnytter ofte bugs, persondatalekkasjer skjer ofte pga kodefeil
- GDPR – man er ansvarlig for å oppbevare og behandle persondata sikkert
- Viktig å tenke på underveis, mye vanskeligere å gjøre noe sikkert i ettertid
- Særlig viktig å sjekke / feilhåndtere grundig når man tar imot data fra internett – hackere bruker automatiserte verktøy og prøver *alt*
- **Veldig viktig:** det er gjerne veldig mye som kan gå galt / mange feil man kan gjøre – ting man gjerne aldri ville tenkt på på egenhånd. Det er viktig å orientere seg om hvilke problemer som er aktuelle for den type applikasjon man utvikler – f.eks. OWASPs Top Ten liste for webapplikasjoner – eller for programmeringsspråket/plattformen man bruker (se f.eks. fjorårets feil i log4j)

Lover, regler og personvern

(Spør ikke om dette på eksamen, men dere må sørge for at dere er generelt orientert om slike ting.)

Immaterielle rettigheter *(Intellectual property)*:

- **Kort oversikt over det viktigste mtp. opphavsrett, og hvordan du velger lisens til egen kode osv.**
- *Opphavsrett* (copyright) handler om åndsverk (åndsverksloven – produktet av kreativt arbeid (tekst, musikk, programvare, bilder, kunst). Åndsverk er automatisk beskyttet mot mangfoldiggjøring og bearbeidelse. Beskyttelsen varer lenge (i Norge, 70 år etter forfatters død); for programvare tilfaller opphavsrettet arbeidsgiver om ikke annet er avtalt, ellers kan den evt. overføres mot (rimelig) vederlag.
    - Aktuelt ved kopiering/gjenbruk av kode, og med grafikk/lyd. I Norge er det generelt straffbart å reprodusere andres åndsverk uten tillatelse.

    - Gjenbruk av Mario-sprites fra Nintendo-spill vil f.eks. rammes – antakelig også spillfiguren «Kurt-Mario» illustrert som er en liten nord-norsk rørlegger med bart og rød kjeledress
    - «Kopiering» er ikke bare klipp/lim av kode, men også f.eks. bruk av biblioteker – når du legger inn en dependency i Maven kopierer du i praksis et åndsverk (og evt distribuerer det om du distribuerer programmet ditt). Du kan regne med å ha rett til selve nedlastingen, men så bør du sjekke at lisensen tillater redistribusjon og evt. modifisering.
    - Det er likevel begrensninger på opphavsretten, f.eks. kan man ikke nekte folk rett til å sitere, og til å kopiere til privat bruk, og det er regler for bruk i undervisning, for personer med nedsatt funksjonsevne osv.
    - I tillegg til å beskytte eierskap/økonomiske interesser, omfatter loven også *ideelle rettigheter* – forfatteren/opphaveren har rett til å bli navngitt og til å verne verket mot «krenkende bruk».
- *Varemerker* o.l (trademark) handler om gjenkjennbar produktidentitet (Varemerkeloven) – bedriftsnavn, logoer, produktnavn, produktutforming, osv. Må som regel registreres, og må forsvares (f.eks. saksøke om noen misbruker logo eller produktnavn) – handler blant annet om at forbrukere ikke skal villedes av at folk lager forvirrende like produkter. Aktuelt ifm. domenenavn, logoer, design/fargevalg på programvare
    - Spillet «Kurt-Mario Kart» (fra INF101 eksamen 2018) ville nok resultert i varemerkesøksmål fra Nintendo om det faktisk ble lansert
- *Patenter* handler om å beskytte ideer/oppfinnelser (patentloven) – Etter søknad får man en tidsbegrenset enerett til å utnytte en spesifikk oppfinnelse. I mindre grad aktuelt for programvare i Norge, men mye brukt i USA, ofte på relativt trivielle ting. (Dere har sikkert alle brutt minst en amerikansk programvarepatent i INF112-prosjektet.)
    - Komprimeringsalgoritmen i GIF-bilder var patentert frem til 2004, så man måtte betale for lisens for å implementere den (*koden* og *teksten* til spesifikasjonen blir dekket av opphavsrett, *ideen*/*algoritmen* dekkes av patentet, i tillegg kunne formatnavnet GIF eventuelt vært et varemerke – og alle tre tingene være eid av forskjellige eiere).

- **Som programutvikler:** ***du er selv ansvarlig for å følge loven*** – du kan ikke nødvendigvis skylde på sjefen (eller foreleser) om du har brutt loven ved å kopiere kode eller bruke et bibliotek uten lov.
    - For INF112: bibliotekene vi har brukt er fri programvare – LibGDX bruker Apache 2.0-lisens, som gir rett til å bruke/kopiere/modifisere biblioteket, også i ikke-fri (proprietær) programvare ; JavaFX (OpenJFX) bruker GPL v2-lisens (krever normalt at din kode også er fri programvare), med et spesielt unntak for bruk som Java-bibliotek. Grafikk/lyd vi har tipset om har som oftest hatt en Creative Commons-lisens – de krever som regel at du krediterer forfatteren, og av og til gir de ikke rett til å modifisere (f.eks. endre på sprites).
- **Som student:** ***du er selv ansvarlig for å følge universitetets regler om plagiat*** – dette er separat fra opphavsrett og reglene er ganske strenge (f.eks. med annulering av eksamen eller utestengelse).

Personvern:

- GDPR (General Data Protection Regulation) er den viktigste tingen å kjenne til. I Norge inngår GDPR ipersonopplysningsloven og Datatilsynet har ansvar for å følge opp regelbrudd.
- Generelle prinsipper for GDPR:
    - *Du* eier dine egne data
    - Gjelder så lenger den som samler opplysningene eller prosesserer dem, eller de opplysningene handler om er basert i EU/EØS
    - Persondata kan bare behandles hvis man har en gyldig grunn – f.eks. ved samtykke, ifm. med avtaler/kundeforhold, lovkrav, osv. Dvs. du må alltid ha en *legitim grunn* for å samle og behandle personopplysninger, og du kan ikke lagre opplysninger du ikke har bruk for. Hensynet/interessene til personen det gjelder veier (som regel) tyngre enn bedriftens interesser («trenger jeg å lagre dette for å gjøre applikasjonen bedre for personen? eller er dette bare for å gjøre applikasjonen bedre for reklame-kundene våre?»)
    - Samtykke må være informert, og man må eksplisitt velge å samtykke, man skal kunne trekke samtykke like enkelt som man ga det

    - Man har (som regel) rett til å vite hvilke opplysninger som er lagret, og få dem utlevert, og til å opplysningene korrigert eller slettet
    - Den som samler/behandler opplysningene er pålagt å ivarta sikkerhert og behandle opplysninger i henhold til reguleringen. Det er rimelig strenge straffer (store bøter for selskaper, evt. også politisak)
- Generelle krav, bla. for oss som programutviklere:
    - personopplysninger skal kun oppbevares så lenge det er nødvendig
    - personopplysninger skal kun brukes til sitt opprinnelige formål (det vil si at hvis du har samlet inn informasjon får du ikke lov å bruke informasjonen til noe annet enn det du først har bedt om lov til/informert om)
    - løsninger vi lager skal ha innebygd personvern
    - løsninger skal oppfylle krav til dokumentasjon
    - alle som behandler personopplysninger kalles databehandlere og skal ha databehandleravtale
    - en *databehandleravtale* er en avtale som regulerer hensikt med behandling av informasjon, varighet, formål, rettigheter og plikter osv. Skal finnes i alle ledd som behandler informasjon (alle underleverandører)
- Selv om det er databehandleren som har hovedansvaret for å ivareta personvernet, har du som programmør også et selvstendig moralsk ansvar for å ikke bidra til uetiske eller ulovlige systemer