Prøve eksamen

1. 1. Synlighet: funksjoner ved et system er synlige, tilstede og vise hvordan de brukes. Burde være enkelt å forstå hvordan det skal brukes.  
      Begrensninger: legge begrensninger på hvilke former for interaksjon som er mulig i et øyeblikk sånn at det ikke oppstår feil. Hindrer brukeren i å gjøre feil.  
      Gi eksempler på definisjonene.  
      Det er mindre «ting» på skjermen samtidig noe som gir en mer oversiktlig å mer ryddig design som resulterer i at brukeren vil oppleve det som mer behagelig.  
      2020 er bedre.
   2. Testene vil da ikke være representative for normale bruker, grunnet at både eksperter og utviklere vil ha mer kjennskap til disse type produkter, enn et annen individ. Eksperter og utviklere har ikke idiotfaktoren til vanlig andre mennesker. Fordeler er at de har mer erfaring med å finne feil i systemer og kan lettere finne disse enn en annen bruker.  
      Vanlige brukertester viser ofte hva som intuitivt vil bli forstått og tolket basert på det de får representert. Derimot evalueringer som: a/b-tester, feltstudier, heuristiske gjennomganger, pluralistiske gjennomganger, predikative modeller, osv.
   3. Personaer representerer en typisk brukermotivasjon, nyttig for å teste ut hva typiske brukere vil gjøre med/i systemet og vil gi innsikt i hva som kanskje må tilpasses og endres på. Scenarioer er en tenkt spesifikk handling som en personaer gjør og hvordan systemet skal respondere.
2. 1. 1. Det første som må tas i betraktning er det frivillige aspektet av en slik applikasjon. Med andre ord, hver bruker bør selv få velge om de vil delta eller ikke.
      2. Brukerne bør informeres eksplisitt om hvilke typer data som samles inn om dem.
      3. Applikasjonene må ikke gå lov til å bruke data smalet inn om hver bruker i en annen kontekst enn det den ble designet for (og som ble kommunisert til brukeren).
      4. Applikasjonen må gi brukeren muligheten til å avinstallere applikasjonen og gjerne all data smalet inn om dem på ethvert tidspunkt.
      5. Applikasjonene bør ha en policy om å slette all innsamlet data om en bruker etter en gitt tidsperiode, og dette må tydelig kommuniseres til brukeren.
      6. Applikasjonen bør gi brukeren mulighet til å temporært deaktivere applikasjonen og sporings muligheten.
      7. Applikasjonen natur tatt i betrakting, all annen informasjon om brukeren, som navn, alder, kjønn, osv., bør ikke samles inn.
      8. Det bør gis ekstra oppmerksomhet til at det ikke forekommer sikkerhetsbrudd eller datalekasje, applikasjonens natur tatt i betrakting. Det bør legges planer med forhåndsbestemte handlinger/prosedyrer dersom et sikkerhetsbrudd eller en datalekasje skulle på noe tidspunkt inntreffe.
   2. Tar for meg punkt for punkt, respektivt til oppgaven.
      1. Et enkelt og intuitivt design gjør at brukere vil komme tilbake og ønske å bruke det igjen, de vil også føle mer på nytteverdien av det.
      2. Forståelig informasjon er viktig for at brukeren skal forstå informasjonen på et effektivt og lett nivå.
      3. Toleranse for feil er viktig for å redusere sjansen for at uønskede hendelser finner sted.
      4. Ved å følge design krav som Universal Design og liknende vil man oppnå et design som gir like muligheter for alle.
      5. Det må være fleksibelt i bruk, altså tilpasse seg den plattformen det er på og endre seg om brukeren trenger det.
      6. Det fysiske aspektet at uansett brukerens fysiske tilstand så skal vedkommende kunne bruke designet.
   3. Alle utviklere har et etisk ansvar ovenfor produktet de lager. Ikke bare lover, men også moralske lover. Derfor konsulterer man gjerne med eksperter på feltet som har mer innsikt i andre kulturer og samfunn enn det de fleste utviklere har. Slik at en evt. applikasjon kan tilpasses et geografisk område grunnet f. eks. kulturelle forskjeller.
   4. En sprint review gjennomføres på slutten av alle iterasjonene i et Scrum prosjekt. Her vil progresjon vises fram til kunden og utviklerne vil få veldig nyttig feedback fra kunden. Dette sikrer at kunden får det de vil ha å gjør at utviklerne lettere kan gå tilbake å endre ting som kunden vil, siden det nylig ble lagt til. Dette fører også til et bedre sluttprodukt da kunden vil kunne kontinuerlig under utviklingen kunne komme med konstruktive tilbakemeldinger som sikrer at produktet blir et de er fornøyd med.
   5. Et issue-board er et system ofte brukt av utviklere for å holde orden på arbeidsoppgavene som skal gjøres og hvem som gjøre de. Det er ofte en TODO, Doing og Done kolonne fordeling. Hvor man vil henge lapper med beskrivelser for hva som skal gjøres på. Et Kanban-board er et issue-board med noen restriksjoner. Et Kanban-board har et maks antall tilatte lapper per kolonne, dette kan være et viktig verktøy i noen prosjekter, for det kan sikre framgang og mer konsentrert arbeidskraft. Hvis det er veldig mange lapper så kan det lett bli uoversiktlig igjen og ting blir gjort saktere og ikke i en optimal rekkefølge.  
      Et fysisk brett er på sett og hvis lettere, siden man fysisk tar på det og det kan være større og gi mer oversikt. Derimot så er de veldig upraktiske hvis man ikke har et kontor hvor alle jobber under hele prosjektet, da er et digitalt brett bedre, da kan alle legge inn, oppdatere og se det til enhver tid så lenge de kan komme seg på nettet.
   6. Hoved grunnen til at man dokumenterer prosjekter innenfor systemutvikling er at systemer kommer nesten garantert til å trege oppdateringer i framtiden og da er det enten lenge siden man jobbet på det eller så er det kanskje et helt nytt team som skal jobbe med det. Da blir det ekstremt mye lettere for dem om det er godt dokumentert så de kan lese seg opp på hvordan systemet henger sammen og hva alt gjør. Jeg mener at dokumentering av koden i form av kommentarer og JavaDoc (eller liknende for andre språk) gir stort utbytte, da vet vedkommende ofte hva de skal se etter og slipper å gå gjennom å analysere metoder for å finne ut hva som skjer. Men nesten viktigere er diagrammer (Domenemodell, klassediagram, use-case (med beskrivelse), aktivitetsdiagram, sekvensdiagram, osv.) som viser et mer helthetlig bilde av hvordan systemets komponenter henger sammen og gir da en grunnleggende forståelse for strukturen til systemet som man må ha for å kunne jobbe effektivt videre med det.
3. 1. Tester bør være deterministiske og repeterbare slik at vi kan være sikre på hva vi får ved en gitt input, og at vi alltid får det samme. Dette gjør at man også kan kjøre automatiske tester som er veldig kjappe og effektive.
   2. Kontinuerlig integrasjon gjør at man kan hele tiden legge inn nye deler av programmet. Dette gjør at programmet alltid vil være så oppdatert som det kan være og en kunde vil kunne se forbedringene og endringene og komme med kontinuerlig tilbakemeldinger underveis for å sikre et best mulig produkt.
   3. Automatiske tester er ekstremt kjappe og effektive, men disse krever at man angir spesifikt hva man ser etter, men funker flott for å sjekke store deler av systemet veldig fort. Derimot hvis man er ute etter å teste ting som man ikke eksplisitt leter etter må man bytte over til manuell testing, som er mye tregere, men kan finne veldig viktige ting. Her spiller kreativiteten til testeren veldig inn da man må prøve å finne ting som man ikke vet hva er.
4. 1. Arkitekturen til systemet burde planlegges veldig tidlig da dette vil kreve enrom arbeidskraft å endre på senere i prosjektet. Det vil da også koste mye mer penger. Det er også stor sannsynlighet for at man må skrive nesten all koden på nytt, eller ha en blandet og da dårlig struktur i systemet.
   2. MVC er et viktig verktøy for å lage en kobling mellom deler av prosjektet som: modellen, forretingslogikken og presentasjonen. Dette gjør at de tre partene står mer fritt og kan lettere gjøre endringer på sitt felt og vite at det ikke vil påvirke koblingen til dem andre feltene, og at når alt skal kobles sammen så fungerer det.