# Sys Eksamen Spørgsmål

**1.**

**A - Outsourcing**

- **Generelt**

  Outsourcing er en process man kan bruge når man hyrer en virksomhed eller person udefra til at løse opgaver og betaler dem for deres arbejdskraft. Dette kan for eksempel være hvis man ikke selv har de skills der skal til for at effektivt kunne nå i mål.

  - **Fordele / ulemper**

    **Fordele:**

    - Spare Tid
    - Fokusere på kernekompetencer
    - Større effektivitet
    - Lavere “Overhead”
    - Koncentrere og maksimer menneskelige ressourcer
    - Fleksibilitet i ansættelse
    - Et mere retfærdigt spillebræt
    - Hurtigere acceleration for nye projekter

    **Ulemper:**

    - Kan man regne med den man har kontaktet?
    - Kvalitet af det man får igen
    - Sprog barriere
    - Kode standard af det returnerede
    - Kan blive dyrt
    - Faste deadlines
    - Ikke afleveret til tiden
    - Giver adgang til projektets kode base og / eller følsomme endpoints

  - **I relation til dit projekt**

    **i. Hvordan har i kommunikeret?**

    - Brugte 2 websites til at finde en potentiel outsourcer
      - fiverr.com
      - freelancer.com
    - Gruppen havde lavet et dokument med alt dokumentation som vi tænkte en outsourcer skulle få brug for
    - I dokumentet var der API endpoints og illustrationer til at beskrive vores designforslag

    **ii. Hvordan fik det?**

    - Til at starte med havde vi fået kontakt til 1 udvikler på fiverr.com.

- Han havde et opslag hvor der stod at han kunne udvikle en mobile app med 2 sider til 50 usd.
- Vi kontaktede ham med vores dokument og han besvarede tilbage med en række spørgsmål.
- Efter omkring 1 dags kommunikationen frem og tilbage fik vi et tilbud på 500 usd.
- Vi prøvede at at forhandle prisen ned med ham, men vi endte med at ikke få noget respons tilbage fra udvikleren.
- Herefter begyndte vi at lede efter en ny outsourcer, og kom i kontakt med 5 forskellige personer der potentielt kunne udvikle vores mobile app til en rimelig pris.
- Vi fandt hurtig ud af hvem af disse 5 udviklere der forstod vores ide, og kunne herefter sortere de andre fra der ikke havde forstået opgaven.
- Efter at have sendt flere beskeder frem og tilbage endte vi med at vælge den billigste.
- Vi endte med at betale 100 usd og fik returneret en kode base indenfor 2 dage som aftalt.
- Vi var meget tilfredse med det vi havde fået tilbage.

**iii. Er der noget du vil gøre anderledes?**

- Til en anden gang skal der nok lægges større fokus på at finde en outsourcer tidligere i processen.
- Det kan være svært at kommunikere ens ide videre til en tredje part, så der skal nok lægges mere tid i at gøre design dokumentet klart og tydelig.
- Det blev lidt et stressmoment at vi først i sidste uge af projektet havde fundet en potentiel outsourcer til en god pris, dette kunne hurtigt have gået galt og endt i at vi ikke fik noget igen.

**B - Forklar kort følgende koncepter**

- **Continuous integration**
  Continuous integration er en udviklingspraksis fra XP hvor man som udviklingsteam arbejder ud fra en ideologi om at opbygge et projekt brik for brik. Hver udvikler har sin egen udvikling branch hvor man arbejder på et stykke af projektet og når man er færdig merger man sin egen branch in i en mainbranch som indeholder projektet i sin helhed. Automatiserede test og builds kører på main branch og sikrer at der ikke er fejl i kodebasen før noget bliver deployed.
- **Spike**
  Spikes er en særling type user story som søger at gøre det enklere at forstå et opfyldeskritereie og øge opgaveestimatets pålideliged. Spikes bruges f.eks til at bryde en stor user story ned i mindre enklere tasks. På samme måde som user stories skrives de ind i scrum boardet. Der kan være tekniske og funktionelle risici ved en story user story og spikes kan minimere risici ved at opfordre til

research og prototyping. Der findes 2 typer spikes. Tekniske spikes og funktionelle spikes.

- **Tekniske spikes**
  Bruges til at undersøge forskellige tekniske tiltag i løsning domænet og f.eks til at undersøge en teknisk detalje af produktet
- **Funktionel spikes**
  Bruges når der er usikkerheder i forhold til brugerens interaktion med produktet. F.eks at lave en prototype til at kunne få noget feedback.

**2.**

**A - Hvad er Agil udvikling**

- **Generelt**
  Agil udvikling er en iterativ tilgang til projektledelse og software udvikling, der hjælper tems med at levere værdi til deres kunder hurtigere og med færre hovedpiner. I stedet for at satse alt på en "big bang" lancering, leverer et agilt team arbejde i små, men forbrugsvenlige interballer. Krav, planer, og resultater evalueres løbende, så teams har en naturlig mekanisme til at reagere hurtigt på ændringer.

**i. Baggrund**

Agil udvikling er et udtryk man bruger til at beskrive en iterativ udviklingsproces. Man udvikler i et inkrementelt flow inddelt i flere mindre sprints. Et sprint er typisk 1 til 4 arbejdsdage. Et sprint indeholder arbejdsopgaver i en backlog med størrelses estimat og acceptance criteria. Alle sprint backlog items skal gennemføres i sprintet igennem kode test og validering. Der er fokus på samarbejdet mellem udviklingsteamet og product owneren som har ejerskab over produktet.

**ii. Principper**

Der er 12 kerneprincipper i agil udvikling for arbejdseffektivitet.

1. **Kundetilfredshed:** Højeste prioritet i agil udvikling er kundetilfredshed. Ofte er der flere stakeholder som er interesseret i projektet, og i agil udvikling er det deres tilfredshed som er absolut vigtigst.
2. **Dynamisk forandring:** Man er velkommen overfor forandring i agil udvikling, ergo navnet agil. Det er ikke statisk men dynamisk og bøjeligt. Selv sent i udviklingsprocessen skal man i agil udvikling være klar på forandring hvis det bliver nødvendigt.
3. **Jævne resultater:** Man levere resultater med jævne mellemrum, gerne oftere end sjældenere.
4. **Samarbejde:** Samarbejde er essentielt og et sundt og tillidsfuldt team er noget som agil udvikling påskønner
5. **Motiveret team:** Byg projekter omkring et motiveret team. Hjælp hinanden og hav tillid.

6. **Face to face kommunikation:** Ansigt til ansigt kommunikation er den mest effektive form for informationsudveksling
7. **Virkende software:** Virkende software er den primære måleenhed for agil udvikling
8. **Bæredygtig udvikling:** Agil udvikling påskønner bæredygtig udvikling. Teamet skal være i stand til at holde en konstant hast.
9. **Kontinuert opmærksomhed:** Kontinuert opmærksomhed på teknisk og designmæssig udmærkelse fremmer agil udvikling.
10. **Simplicitet:** Agil udvikling påskønner simplicitet. Kunsten at minimere arbejdsbyrden så meget som muligt er vigtig og effektivitet er en hoved søjle i udviklingen.
11. **Selvorganisering:** De bedste løsninger udspringer fra selvorganisering.
12. **Reflektion:** Teamet skal være istand til at reflektere over eget arbejde og med regulære intervaller taler man sammen om hvordan vi arbejder mere effektivt. Således finjusteres arbejdet konstant i agil udvikling.

- **I relation til dit projekt**

**i. Metode**

I vores projekt arbejde vi ud fra frameworke scrum og med udviklingspraksis fra XP-verdenen. Projektet blev indelt i 3 sprints, hver på 1 uges varighed. Her begyndte vi med at lave et product backlog, med user stories som havde størrelseestimatet og acceptance criteria. Vi afholdte daglige standup møder og delte rollen som scrum master ud imellem os. Vi holdt sprint planning, review, og retrospective møder. Derudover fulgte vi også en række udviklingspraksis fra XP. Vi benyttede os af continuous integration, collective ownership, pair programming og test af kode.

**ii. Fordele / Ulemper**

Det var tydeligt en fordel at vi havde brugt agil udvikling. Vores arbejde var meget mere effektivt og ved at havde lagt så stort fokus på planlægning kommer der også gode og effektive resultater. At arbejde agilt tvinger teamet til i høj grad at reflektere over det arbejde man har foran sig inden man går i gang med at arbejde. I et lille projekt som dette med kun 3 mand i teamet er fordelene mindre tidligere at mærke men f.eks i et større team kan man sikkert understrege vigtigheden i den struktur agil udvikling medbringer.

En ulempe kan være at man måske føler at der ikke bliver produceret nok, og der bliver brugt for meget tid på at snakke om problemer fremfor at løse dem.

**iii. Processen**

Hver uge var sit eget sprint. Første uge byggede vi selve quiz oplevelsen, anden uge var det statistics, og sidste uge var det leaderboard og highscore, deployment og outsourcing. Vi holdt sprint planning i starten af sprintet hvor det i 2. Og 3 sprint også blev til review møder. I midten af ugen havde vi teknisk review med retrospektive indenover.

**B - Forklar kort følgenede koncepter**

- **Outsourcing**
  Outsourcing er når en virksomhed henter arbejdskraft fra eksterne aktører. Den eksterne aktør hyres til at løse en opgave som virksomheden enten ikke selv er i stand til at løse eller ikke har tid til at løse.
- **Prototyping**
  En prototype er en lille udgave af noget software eller en bid af noget software. Som udvikler kan man udnytte prototyping til at danne overblik over et produkt eller præsentere et stykke arbejde for en interessant uden at bruge den samme tid på at bugge det som hvis det var et færdigt produkt.

**3.**

**A - Beskriv XP og Scrum**

- **Generelt**
  **XP** er en software udviklings metodologi beregnet til at forbedre software kvaliteten overfor skiftende kundekrav. Som en form for agil udvikling går XP ind for hyppige udgivelser i korte udviklingscyklusser, beregnet til at forbedre produktiviteten og indføre kontrol punkter hvor nye kundekrav kan vedtages.
  **Scrum** er et framework indenfor agil udvikling der har fokus på udvikling, levering, og opretholdelse af software produkter. Det er designet til arbejde der udføres i teams som bryder mål ned i mindre opgaver kaldet user stories som har tidshorisont og acceptance kriteria. Teamet mødes ofte og snakker om hvor langt de er i processen og hvad de næste opgaver de arbejder på består af.

**i. Lighedspunkter / Forskelle**

**Lighedspunkter**

- Begge går under agil udvikling
- Iterativ udvikling
- Fungerende software
- Release planning
- Iteration planning
- Daily meetings
- Retrospective
- God kommunikation mellem holdet

**Forskelle**

- XP motivere pair programming
- XP lægger fokus på unit tests

- XP kræver at kode bliver integreret ofte med continuous integration
- XP bruger refactoring af kode så ofte som muligt
- XP ligger vægt på collective ownership
- Scrum har en scrum master der sørger for at Scrum frameworket bliver fulgt ordentligt
- Scrum har en product owner der sørger for at fokus ligger på product backlog
- Scrum er meget sprint baseret
- Scrum bruger sprint planning i starten af et nyt sprint
- Scrum bruger sprint review i slutningen af hvert sprint
- Scrum bruger sprint retrospective hvor udviklerne reflektere over hvordan sprintet er gået og laver eventuelle justeringer

**ii. Fordele og ulemper ved at kombinere XP og Scrum**

**Fordele**

- Komplimentere hinanden godt
- Går hånd i hånd ved at lægge fokus på planlægning og udvikling
- Stor fokus på team kommunikation under alle dele af udviklingsprocessen
- Simplificere arbejdsflowet
- Fokus på feedback
- Arbejder målrettet mod et resultat

**Ulemper**

- Team størrelse er normalt mindre i hold der benytter sig af XP.
- XP er en mere disciplineret tilgang til udvikling, mens Scrum er mere fleksibel
- XP er mere fokuseret på at levere et fungerende software produkt, mens Scrum er mere fokuseret på at levere et færdigt produkt.

**iii. I relation til dit projekt**

Vi havde lagt væsentligt større fokus på XP og Scrum i dette projekt, end vi havde gjort i de andre projekter. Dette var blandt andet fordi at vi havde tænkt os at arbejde mere separat end vi havde gjort de andre gange. Har gjorde processen med at holde stand up review og planning fra Scrum det meget nemme at holde gruppen fokuseret på det samme end goal. Vi brugte continuous integration, collective ownership, pair programming når der var mulighed, og testede ofte vores arbejde. Vi fik det til at fungere rigtig godt med at kombinere de 2 verdener.

**B - Forklar kort følgenede koncepter**

- **Outsourcing**
  Outsourcing er når en virksomhed henter arbejdskraft fra eksterne aktører. Den eksterne aktør hyres til at løse en opgave som virksomheden enten ikke selv er i stand til at løse eller ikke har tid til at løse.
- **Test**
  Testing af ens kode er når man bruger unit testing til at sikre sig at hver enkelt metode man har kodet, gør det som man forventer. Man kan både teste før og efter man har skrevet sine metoder, og på denne måde kan man sørge for at man får det forventede resultat tilbage når et stykke kode bliver kørt. Test driven development er en god from gangs måde da det minimere muligheden for bugs i kode basen, da det tvinger udvikleren til at tænke fremad og sørge for at koden bliver produceret uden fejl.

**4.**

**A - Beskriv dit arbejde med User Stories**

- **Generelt**
  I software udvikling og projektstyring er en user story en uformel og naturlig sproglig beskrivelse af funktioner i et softwaresystem. De er skrevet fra en slutbrugers perspektiv og bliver oftest skrevet ind i et kanban board. User stories kan skrives af klienter, brugere, ledere, eller udviklingsteamet.

  **i. Hvorfor benyttes formen "As a…, I want to…, so that…"?**

  En user story er en general forklaring af en software egenskab beskrevet ud fra en brugeres perspektiv, ergo user story. Den har til formål at beskrive hvilken værdi software egenskaben vil skabe for brugeren. En hoved sten i den agil arbejdsmetode er at sætte mennesket først og brugeren i centrum. User stories bruger ikke teknisk sprog og skaber kontekst for udviklingsteamet. En user story beskriver hvad der skal bygges, hvorfor det skal bygges, og hvilken værdi det skaber. "As a" giver kontekst i forhold til hvem software egenskaben skaber værdi for. "I want to" beskriver hvad der skal bygges. "So that" beskriver hvorfor det skal bygges og hvorfor det er væsentligt for projektet.

  **ii. INVEST kriterier**

  INVEST kriterier er et acronym for 6 kriterier.

  1. **Independent:** Man skal skal sørge for at der ikke er afhængigheder i mellem user stories. Afhængigheder kan skabe planlægnings og prioriterings vanskeligheder. Derudover gør det estimereing vanskeligere.
  2. **Negotiable:** Stories er forhandlings dygtige. De er ikke kontrakter eller krav som software må opfylde. Det skal være muligt for udviklingsteamet og kunden at diskutere user stories og ændre dem hvis det er nødvendigt.
  3. **Valuable:** Hver user story skal skabe værdi for kunden eller brugeren.

4. **Estimable:** Det er vigtig at man er i stand til at estimere størrelsen på user stories.
5. **Small:** User stories skal ikke være for store, så man skal oftest prøve at bryde dem ned i mindre opgaver før de kommer på et kanban board. Ellers kan man også benytte sig af spikes ved større stories.
6. **Testable:** User stories skal skrives så de er testable. Hvis en user story er skrevet med non funktionelle krav, bliver det svært at se om den opfylder acceptance criteria.

- **I relation til dit projekt**

**i. Product backlog / Sprint backlog**

Vi benyttede os både af product backlog og sprint backlog idet vi inddelte projektet i 3 sprints. Vi havde et overordnet product backlog hvor vi indskrev alle de user stories vi kunne finde på til at starte med, og benyttede os også af mulighed for tilføjelse af flere under hele projektperioden. Når et nyt sprint begyndte, valgte vi så de user stories fra product backloggen der passede bedst sammen og til den kommende uges sprint.

**ii. Estimering**

Vi havde en tendens til at skrive meget få user stories hvilket gjorde det svært at estimere den given story korrekt. Har afveg vi lidt fra det 5. invest kriterie og havde haft nemmere ved tids estimat hvis vi havde brudt dem ned i mindre user stories.

**iii. Acceptance Criteria**

Vi benyttede os også af acceptance criteria til at klargøre hvad der skal til for at opfylde den givne story. Har lagde vi stor fokus på at den acceptance criteria vi skrev under hver user story havde en god beskrivelse af hvad vi forventede at det færdige system tilsvarende den givne user story. På denne måde kunne sikre os at alle var enige om hvad den givne user story skulle løse af problemer.

**B - Forklar kort følgenede koncepter**

- **Outsourcing**
  Outsourcing er når en virksomhed henter arbejdskraft fra eksterne aktører. Den eksterne aktør hyres til at løse en opgave som virksomheden enten ikke selv er i stand til at løse eller ikke har tid til at løse.
- **GitHub repository**
  Et GitHub repo er et system man kan bruge til source control af ens kode base. Det er her udviklingsteamet "committer" deres ændringer, så andre på holdet kan hente de nye ændringer ned og så alle kan sidde med en kode base der er identitisk. Man kan oprette "branches" i et repo, så hvert medlem af udviklingsteamet kan arbejde på forskellige systemer, uden at være afhængige af hinanden.

**5.**

**A - Beskriv prototyping**

- **Generelt**

  En prototype er en tidlig prøve, model. Eller udgivelse af et produkt bygget til at teste et koncept eller en proces. Det er et udtryk der bruges til at beskrive en nedskåret systemet under design fasen og / eller software programmerings fasen. En prototype bruges normalt to at evalurere et nyt design for at forbedre præcisionen af forventningerne af det nye system.

  **i. Typer af prototyper**

  - **Exploratory prototype:** Denne prototype fokusere på de grundlæggende problemer ved kommunikationen mellem udviklere og brugere, særligt i de tidlige faser af projektet. Her har udviklerne ikke det store kendskab til udviklingsområdet og har måske ikke en klar ide til produktet endnu. Her kan en exploratory prototype komme til gavn ved at demonstrere flere mulige funktionelter og flere forslag til bedre design.
  - **Experimental prototype:** I denne tilgang søger man en løsning på et problem gennem eksperimentel brug. Denne type er tættest på den oprindelige prototype.
  - **Evolutionary prototype:** Denne tilgang mener nogen at man slet ikke burde kalde en prototype. Et andet ord for det er “versioning”. Evolutionary prototyping springer ud fra en ide om at nye krav dukker løbende op og produktet skal ændres jævnligt. Denne dysnamiske strategi anser produktet som en seria af versioner hvor hver nye version en en prototype til den næste version.

  - **Low fidelity:** Disse prototyper ligner ikke det endelige design. De er ikke lavet af samme materialer eller i samme programmer og de har ikke samme funktionalitet som det endelige produkt.
  - **High fidelity:** Omvendt minder disse mere om det endelige produkt, har måske nogle af de samme funktionaliteter og kan derved teste interaktioner bedre. Dog tager de længere tid at udvikle.

- **I relation til dit projekt**

  **i. Har i brugt nogle prototyper**

  I selve projektet har vi ikke anvendt prototyping, men man kan godt argumentere for at det vi fik outsourcern til at designe kan anses som værende en prototype til en android app – version af vores quiz. Denne prototype var af kategorien low fidelity hvor design ikke nødvendigvis var tilsvarende vores tanker om det endelige design Prototypen demonstrerede udelukkende programmets hovedfunktion, quizzen, og ikke statistikker,

highscores og andre funktionaliteter. Skal man klassificere prototypen indenfor en af de 3 typer, vil jeg mene den er af typen experimental prototype. Vi havde en klar ide om vores produkt og søgte at udforske kundens interaktion med hoved funktionen i vores projekt.

**B - Forklar kort de følgende koncepter**

- **User Stories**
  I software udvikling og projektstyring er en user story en uformel og naturlig sproglig beskrivelse af funktioner i et softwaresystem. De er skrevet fra en slutbrugers perspektiv og bliver oftest skrevet ind i et kanban board. User stories kan skrives af klienter, brugere, ledere, eller udviklingsteamet.
- **Continuous Integration**
  Continuous integration er en udviklingspraksis fra XP hvor man som udviklingsteam arbejder ud fra en ideologi om at opbygge et projekt brik for brik. Hver udvikler vil have sin egen udviklings branch hvor man arbejder på et stykke af projektet og når man er færdig merger man ind i en 'main branch' som indeholder projektet i sin helhed. Automatiserede tests og builds kører på main branchen for at tjekke for fejl, bugs, og mangler.

**6.**

**A - XP Practices**

- **Forklar nogle forskellige practices i XP**
  1. **Planning game:** Her handler det om hurtigt at identificere næste releases scope. Dette er igen en agil plan som skal være opdaterings dygtig og klar til forandring.
  2. **Small release:** Sæt hurtigt små releases i produktion og udgiv nye versioner med korte intervaller.
  3. **Metaphor:** Hav fælles forståelse for projektet og lad denne historie være guide for hvordan hele systemet fungere.
  4. **Simple design:** Systemet skal designs så simpelt som overhovedet muligt. Redundans og over kompleks kode fjernes så snart det opdages.
  5. **Test:** Kode skal testes hele tiden og test skal køre fejlfrit før at udviklingen kan fortsætte.
  6. **Refactoring:** Konstant restrukturering af kode uden at ændre systemets adfærd for at løfte kode kvaliteten, simplificere og fjerne redundans.
  7. **Pair programming:** Programmer sammen i par med driver / navigator roller.
  8. **Continuous integration:** Integrer ny kode og byg ovenpå projektet mange gange dagligt.
  9. **40 hour work week:** XP fokusere på produktivitet og påskønner en længere arbejdsuge end normalt.
  10. **On site customer:** En live bruger som er en del af teamet og udviklingen.

11. **Coding standards:** Skriv en kodestandardt som alli i teamet følger, så man undgår at bruge tid på at rette formalia.

- **Hvorfor og hvordan?**
  XP står for extreme programming og er en del af den agile arbejdsmetode. XP indebærer en række udviklingspraksis. XP søger at gøre op med de risici der er ved software udvikling, såsom skridende tidsplaner, aflyste projekter og forretnings ændringer. Det er en 'code first' tilgang til arbejdet og den understreger 4 hovedaktiviteter.
  1. **Coding:** Man koder for hvis man ikke koder har man ikke noget arbejde.
  2. **Testing:** Man tester for at vide hvornår man er færdige med at kode.
  3. **Listening:** Man lytter fordi du skal vide hvad du skal kode eller teste.
  4. **Designing:** Man designer så du kan blive ved med at kode, teste, og lytte.

- **Redegør for hvilke i har brugt af disse i jeres projekt**
  1. **Planning game:** Vi var gode til at finde user stories til projektet.
  2. **Test:** Vi testede vores kode igennem unit-testing.
  3. **Pair programming:** Vi skiftede til at arbejde i par af 2.
  4. **Continuous integration:** Vi havde hver vores branch og dagligt mergede vi ind i en main branch.

**B - Forklar kort følgenede koncepter**

- **Prototyping**
  En prototype er en lille udgave af noget software eller en bid af noget software. Som udvikler kan man udnytte prototyping til at danne overblik over et produkt eller præsentere et stykke arbejde for en interessant uden at bruge den samme tid på at bugge det som hvis det var et færdigt produkt.
- **Branching**
  Branching handler om at hver udvikler i teamet har egen udvklingsbrach. Herved undgår man at flere udviklere arbejder i den samme udgave af projektet med stor risiko for konflikter mellem kode, afhængigheder, komme til at slette noget som en anden arbejder på osv. Udviklere merger sit arbejde ind i en main-branch som indeholder det samlede system.

**7.**

**A - Domænemodel og klassediagram**

- **Beskriv formålet med hhv. Domænemodel og klassediagram**
  **Domænemodellen** er noget af det første man laver i projektforløbet. Det er en konceptuel model som beskriver et system ud fra dets identiteter. Formålet er at drage opmærksomhed på de overordnede egenskaber og funktionaliteter som systemet skal rumme. Domænemodellen repræsenterer koncepter fra den virkelige verden og deres sammenhænge, ikke teknikaliteter som klassenavne, metoder og attributter. Domænemodellen bruger altså naturligt sprog.

Domænemodellen skaber således en god grundlæggende forståelse for systemet

**Klassediagrammet** er mere detaljeret og teknisk. Her bruges ikke udelukkende naturligt sprog men også tekniske betegnelser som attribut typer osv. Klassediagrammet som navnet indikerer, beskriver systemets klasser, attributter, metoder og relationer mellem objekter. Det er altså længere i processen at man opretter klassediagram da det kræver en grundig teknisk forståelse af systemet at lave det. Disse relationer kan være af flere forskellige typer.

1. **Dependency:** Disse relationer findes mellem objekter som er afhængige af hinanden på en måde. Ændres den ene ændres den anden også.
2. **Association:** Denne beskriver den statiske relation meller to objekter, som ikke nødvendigvis er afhængige af hinanden.
3. **Aggregation:** Denne relation beskriver en “has a” relation. Den kan ikke involvere mere end 2 klasser. Den opstår når der er en relation mellem to klasser men den ene klasse godt kan eksistere uafhængigt af den anden. F.eks en professor kan godt eksistere uden en klasse.
4. **Composition:** Omvendt af aggregation kan disse relationer ikke eksistere uden hinanden. F.eks en bil uden en motor

**i. Forskelle / ligheder / sammenhæng**

De er begge med til at beskrive et system. Begge diagrammer beskriver entiteter og relationer. Klassediagrammet er mere teknisk og detaljeret og siger mere om hvordan systemet skal kodes, hvor domænemodellen er mere konceptuel og udelukkende siger noget om hvordan systemet ser ud i sin helhed, uden at sige noget om hvordan der skal kodes.

**ii. Fordele / ulemper**

Fordelene ved at bygge domæne og klassediagram er at det giver et grundlag for en effektiv start og en god konceptuel og teknisk forståelse af systemet. Ulempen er at man kan komme til at låse sig fast på en idé om hvordan systemet skal bygges, når arbejdet går i gang.

- **I relation til dit projekt**

  **i. Brugte i den ene og / eller begge diagrammer(ne) - Hvordan gik det?**

  Inden vi begyndte at kode oprettede vi en domænemodel for at danne et grundlæggende overblik over hvordan systemet ville komme til at se ud. Domænemodellen blev meget simpel men gav alligevel en god forståelse for systemet. Vi lavede ikke klassediagram, hvilket nok kunne gavnet os hvis vi invisterede tiden i dette.

  **ii. Hvis du skulle bruge de 2 diagrammer, hvor ville du så gøre det?**

Domænemodel ville jeg lave i begyndelsen af projektet som vi også gjorde. Og klasse diagrammet ville jeg løbende opdatere mens jeg koder projektet. På denne måde kan jeg også nemt sætte nye udviklere ind i hvordan systemet fungere.

**B - Forklar kort følgende koncepter**

- **Prototyping**
  En prototype er en lille udgave af noget software eller en bid af noget software. Som udvikler kan man udnytte prototyping til at danne overblik over et produkt eller præsentere et stykke arbejde for en interessant uden at bruge den samme tid på at bugge det som hvis det var et færdigt produkt..
- **Spike**
  Spikes er en særling type user story som søger at gøre det enklere at forstå et opfyldeskritereie og øge opgaveestimatets pålideliged. Spikes bruges f.eks til at bryde en stor user story ned i mindre enklere tasks. På samme måde som user stories skrives de ind i scrum boardet. Der kan være tekniske og funktionelle risici ved en story user story og spikes kan minimere risici ved at opfordre til research og prototyping. Der findes 2 typer spikes. Tekniske spikes og funktionelle spikes.
  - **Tekniske spikes**
    Bruges til at undersøge forskellige tekniske tiltag i løsning domænet og f.eks til at undersøge en teknisk detalje af produktet
  - **Funktionel spikes**
    Bruges når der er usikkerheder i forhold til brugerens interaktion med produktet. F.eks at lave en prototype til at kunne få noget feedback.

**8.**

**A - Scrum**

- **Beskriv de forskellige cæremonier / møder i Scrum - Generelt**
  **i. Kom herunder ind på flowet i et sprint**

1. **Sprint planning meeting:** Dette møde bruges til at fastlægge hvad et Scrum team ønsker at opnå for et given sprint, og vurdere den mængde ressourcer teamet har til rådighed. Man planlægger spurten, tideler opgaver, og sætter deadlines. Man sørger for at hver udvikler forstår de “ins and outs” af de opgaver de bliver tildelt. Product owner er som regel med til dette møde så de kan afklare eventuelle uklarheder og hjælpe med at skabe forventninger.
2. **Daily standup meeting:** Dette møde er det hyppigste afholdte Scrum møde. De er korte og præcise og afholdes hver dag. De er typisk det første møde på arbejdsdagen. Hver udvikler går normalt fra mødet når der er blevet besvaret 2 kritiske spørgsmål: Hvad nåede jeg i går? Og hvad skal jeg nå i dag? Der bliver normalt også snakket om hvilke

problemer en udvikler er stødt på, og eventuelle løsninger deles i åbent forum.

3. **Sprint review meeting:** Dette møde afholdes i slutningen af hver sprint. Her er der mulighed for teamet at demonstrere hvad der er blevet opnået i løbet af et sprint over for hinanden og for product owner. Målet ved dette møde er at få feedback på løsningerne der er fundet i løbet af sprintet.
4. **Sprint retrospective meeting:** Ligesom i sprint review meetings, afholdes sprint retrospective meetings i slutningen af hvert sprint. Dette møde er primært beregnet til udviklings teamet, og holdet besvarer følgende spørgsmål sammen: Hvad gik godt i løbet af sprintet? Hvad gik ikke så godt? Og hvad kan vi gøre anderledes næste gang?
5. **Product backlog refinement meeting:** Disse møder opstår mere dynamisk end de andre gør, dog afholdes de oftests en gang per sprint. Her er der mulighed for at gennemgå de user stories der er i backloggen og redigere / rydde op i dem. Det er med til at sørge for at produktet går i den rigtige retning og der er overensstemmelse for hvad de kommende uger kommer til at bestå af.

- **Hvordan har de forskellige Scrum møder været i forhold til jeres projekt?**
  Vi holdt daglige standup møder på 15 minutters tid. Vi holdt planning i starten af hvert sprint. Review holdt vi med vores PO Kim og retrospective holdt vi med Jörg. Vi valgte at se det på denne måde da vi fokuserede mere på produkets features og det bagved liggende sprint til review med Kim, hvor vi kunne gå mere teknisk til værks med Jörg. Dog kan man godt argumentere for at review/retrospective blev slået lidt sammen i et møde både med Kim og Jörg.

**B - Forklar kort følgenede koncepter**

- **Branching**
  Branching handler om at hver udvikler i teamet har egen udvklingsbrach. Herved undgår man at flere udviklere arbejder i den samme udgave af projektet med stor risiko for konflikter mellem kode, afhængigheder, komme til at slette noget som en anden arbejder på osv. Udviklere merger sit arbejde ind i en main-branch som indeholder det samlede system.
- **Prototyping**
  En prototype er en lille udgave af noget software eller en bid af noget software. Som udvikler kan man udnytte prototyping til at danne overblik over et produkt eller præsentere et stykke arbejde for en interessant uden at bruge den samme tid på at bugge det som hvis det var et færdigt produkt.

**9.**

**A - Prototyping og Spikes**

- **Hvad er prototyping?**

En prototype er en tidlig prøve, model. Eller udgivelse af et produkt bygget til at teste et koncept eller en proces. Det er et udtryk der bruges til at beskrive en nedskåret systemet under design fasen og / eller software programmerings fasen. En prototype bruges normalt to at evalurere et nyt design for at forbedre præcisionen af forventningerne af det nye system.

**i. Typer af prototyper**

- **Exploratory prototype:** Denne prototype fokusere på de grundlæggende problemer ved kommunikationen mellem udviklere og brugere, særligt i de tidlige faser af projektet. Her har udviklerne ikke det store kendskab til udviklingsområdet og har måske ikke en klar ide til produktet endnu. Her kan en exploratory prototype komme til gavn ved at demonstrere flere mulige funktionelter og flere forslag til bedre design.
- **Experimental prototype:** I denne tilgang søger man en løsning på et problem gennem eksperimentel brug. Denne type er tættest på den oprindelige prototype.
- **Evolutionary prototype:** Denne tilgang mener nogen at man slet ikke burde kalde en prototype. Et andet ord for det er “versioning”. Evolutionary prototyping springer ud fra en ide om at nye krav dukker løbende op og produktet skal ændres jævnligt. Denne dysnamiske strategi anser produktet som en seria af versioner hvor hver nye version en en prototype til den næste version.

- **Low fidelity:** Disse prototyper ligner ikke det endelige design. De er ikke lavet af samme materialer eller i samme programmer og de har ikke samme funktionalitet som det endelige produkt.
- **High fidelity:** Omvendt minder disse mere om det endelige produkt, har måske nogle af de samme funktionaliteter og kan derved teste interaktioner bedre. Dog tager de længere tid at udvikle.

- **Hvad er en Spike?**
  Spikes er en særling type user story som søger at gøre det enklere at forstå et opfyldeskritereie og øge opgaveestimatets pålideliged. Spikes bruges f.eks til at bryde en stor user story ned i mindre enklere tasks. På samme måde som user stories skrives de ind i scrum boardet. Der kan være tekniske og funktionelle risici ved en story user story og spikes kan minimere risici ved at opfordre til research og prototyping. Der findes 2 typer spikes. Tekniske spikes og funktionelle spikes.

  **i. Typer af Spikes**

  - **Tekniske spikes**
    Bruges til at undersøge forskellige tekniske tiltag i løsning domænet og f.eks til at undersøge en teknisk detalje af produktet
  - **Funktionel spikes**

Bruges når der er usikkerheder i forhold til brugerens interaktion med produktet. F.eks at lave en prototype til at kunne få noget feedback.

- **Ligheder / forskelle mellem Prototype / Spike**
  Formålet med prototyper og spikes er at blive klogere på et specifikt område. Der undersøges og skæres ud i pap mht. hvordan en opgave kan / skal løses. Spikes er mere målrettet efter nedbrydningen af store user stories, og en prototype kan bruges til at bedre visualiser en user story.

- **Har i brugt en prototype og / eller Spikes i jeres projekt?**
  I selve projektet har vi ikke anvendt prototyping, men man kan godt argumentere for at det vi fik outsourceren til at designe kan anses som værende en prototype til en android app – version af vores quiz. Denne prototype var af kategorien low fidelity hvor design ikke nødvendigvis var tilsvarende vores tanker om det endelige design Prototypen demonstrerede udelukkende programmets hovedfunktion, quizzen, og ikke statistikker, highscores og andre funktionaliteter. Skal man klassificere prototypen indenfor en af de 3 typer, vil jeg mene den er af typen experimental prototype. Vi havde en klar ide om vores produkt og søgte at udforske kundens interaktion med hoved funktionen i vores projekt. Vi brugte ikke rigtig spikes.

**B - Forklar kort følgenede koncepter**

- **Branching**
  Branching handler om at hver udvikler i teamet har egen udvklingsbrach. Herved undgår man at flere udviklere arbejder i den samme udgave af projektet med stor risiko for konflikter mellem kode, afhængigheder, komme til at slette noget som en anden arbejder på osv. Udviklere merger sit arbejde ind i en main-branch som indeholder det samlede system.
- **Continuous integration**
  Continuous integration er en udviklingspraksis fra XP hvor man som udviklingsteam arbejder ud fra en ideologi om at opbygge et projekt brik for brik. Hver udvikler har sin egen udvikling branch hvor man arbejder på et stykke af projektet og når man er færdig merger man sin egen branch in i en mainbranch som indeholder projektet i sin helhed. Automatiserede test og builds kører på main branch og sikrer at der ikke er fejl i kodebasen før noget bliver deployed.

**10.**

**A - Redegør for hvordan et Scrum team arbejder**

- **Kom herunder ind på de forskellige Scrum roller**
  Et scrum team arbejder agilt. Der arbejdes i korte intervaller kaldet sprints som indeholder opgaver teamet skal løse. Planning, review, retrospective og stand-up.
  Et scrum team består grundlæggende af 3 roller.

Først og fremmest skal der være en "**Scrum Master**". Scrum masteren er ikke leder over teamet, men mere en leder over processen. Det er scrum masteren der er med til at sætte møder og events op - han er med til at skabe hele fundamentet for scrum-processen.

Den anden rolle er "**Udviklingsteamet**", som er den enhed, der arbejder med den udviklingsopgave, som teamet er sat i verden for at løse. Teamet skulle gerne indeholde så mange kompetencer som muligt.

Den sidste rolle er "**Product Owneren**", som er den person, som sætter retningen og vælger de opgaver ud, der er viktigst. Han kan også ses som "kunderepræsentanten", der sætter visionen og sørger for, at den rigtige værdi bliver leveret i hvert sprint og har stakeholders interesser i sin varetægt.

- **Forklar om estimereing og Burn Down Chart**
  Estimering bruges til at estimere ens user stories. Normalt vil man estimere opgaver med abstrakte enheder som t-shirt størrelser eller arbitrere tal i stedet for faktiske timer. Estimering er en udfordrende opgave og der er forskellige værktøjer som kan tages i brug når der skal estimeres.

  **Relative sizing:** Her estimerer man opgaver ud fra viden om andre opgaver.

  **Planning poker:** Her præsenteres alle user stories og efter en periode med diskussion skriver hver medlem af teamet sit estimat på kortet med user stories. Man udvælger kort og diskuterer igen

  **Burndown chart** er et diagram der illustrerer hvor meget arbejde der er tilbage i sprintet. På y-aksen ses det totale antal estimerede timer tilbage. På x-aksen ses dagene fra start til slut. Derefter tegnes en streg fra toppen af y-aksen til enden af x-aksen. Diagrammet opdateres dagligt og kurven skal helst ligge under den originale streg. Man opdaterer grafen ved at smide user stories i 'done'.

- **Hvorledes har i arbejdet med dette i jeres projekt?**
  Vi estimerede med tal hvor 1 svarer til 1 dags arbejde. Vi havde få store user stories og havde måske haft gavn af at nedbryde dem i mindre stories. Vi estimerede stories i starten af hvert sprint. I første sprint overestimerede vi og i andet sprint underestimerede vi. Vi brugte burndown chart men glemte at opdatere backloggen så fik ikke taget den i brug som vi burde have.

**B - Forklar kort følgenede koncepter**

- **Simple Design**

Simple design er en af udviklings praksisserne i XP. Den lyder på at systemet skal designes så simpelt som overhovedet muligt. Redundans og over kompleks kode fjernes så snart det opdages.

- **Kravspecificering**
  Kravspecificering handler om at stille de krav som skal til for at det færdige produkt er fyldestgørende. Kravspecificering beskriver hvad produktet skal kunne og ikke hvordan det skal kunne det. Det er kravspecificeringerne som skaber product backloggen og som afføder user stories.

**11.**

**A - Redegør for hvordan man arbejder med kravene i Scrum**

- **Kom herunder ind på Product Backlog og Sprint Backlog**
  I scrum ridses kravene op i starten af et projektforløb. Scrum teamet opretter user stories og estimerer og prioriterer dem. Med scrum er det user stories med højest prioritering som man laver først. Scrum bruger en strategi, progressive refinement hvor man bryder større mindre detaljerede stories ned i mindre mere detaljerede stories, også kendt som spikes. Når stories er estimeret og har acceptance criteria ryger de ind i product backloggen som user stories, product backlog items. Denne backlog har typisk 5 kollonner. To do, in progress, test, pair review, done. Det er de 5 steps som en PBI må igennem for at blive færdiggjort fyldestgørende. Product backloggen indeholder user stories på et helt projekt som deles op i sprints. Her har hvert sprint sin egen 'sprint backlog'. I scrum er man åben over for at stories kan ændres, omprioriteres eller slettes. Alt er agilt og flydende og intet er hugget i sten.

- **Hvorledes har i arbejdet med krav i jeres projekt?**
  Vi oprettede user stories i starten af hvert sprint under sprint planning. Vi brugte få store user stories og havde haft gavn af noget mere seriøs refinement. Vi brugte Taiga som backlog og brugte også burn-down chart.

**B - Forklar kort følgenede koncepter**

- **Branching**
  Branching handler om at hver udvikler i teamet har egen udvklingsbrach. Herved undgår man at flere udviklere arbejder i den samme udgave af projektet med stor risiko for konflikter mellem kode, afhængigheder, komme til at slette noget som en anden arbejder på osv. Udviklere merger sit arbejde ind i en main-branch som indeholder det samlede system.
- **Prototyping**
  En prototype er en lille udgave af noget software eller en bid af noget software. Som udvikler kan man udnytte prototyping til at danne overblik over et produkt eller præsentere et stykke arbejde for en interessant uden at bruge den samme tid på at bugge det som hvis det var et færdigt produkt.

**12.**

**A - Gør rede for hvordan de forskellige systemudviklingsmetoder arbejder med krav**

Waterfall bliver alt sat i sten, altså bliver alle krav stillet fra starten og de kan ikke ændres.

Agil udvikling er mere flydende, altså er alle krav fra starten ikke stillet, men kommer løbende i samarbejde med kunden selv.

- **Gør herunder rede for hvordan i arbejde med krav i jeres projekt**
  Vi oprettede user stories i starten af hvert sprint under sprint planning. Vi brugte få store user stories og havde haft gavn af noget mere seriøs refinement.

**B - Forklar kort følgenede koncepter**

- **Small releases**
  Small releases kommer fra XP-verdenen og er en tanke om at udgive små versioner af dit software til offentligheden i korte intervaller, og efter analysere den feedback man får tilbage og skabe forbedringer til den næste release. Således bliver hver release en slags prototype til dens efterkommer
- **Simple Design**
  Simple design er en af udviklings praksisserne i XP. Den lyder på at systemet skal designes så simpelt som overhovedet muligt. Redundans og over kompleks kode fjernes så snart det opdages.