DTU Informatik
Informatik og Matematiks Modellering

Spring 2025
Hubert Baumeister

02161 Software Engineering 1

Obligatorisk Opgave

10. marts 2025, Version 1

1 Opgaven

I afsnit 5 findes projektbeskrivelsen, der vedrører udviklingen af et planlægningsværktøj til brug i forbindelse med styring og planlægning af softwareudviklingsprojekter, herunder registrering af arbejdstidsforbrug på de forskellige projekter.

På baggrund af dette oplæg skal der udarbejdes:

1. kravspecifikation og programdesign (rapport 1)
2. Feedback til en anden gruppes rapport 1 ved brug af FeedbackFruits
3. det endelige program og test (rapport 2 plus kildekode som Maven projekt)

Således at de væsentlige begreber fra oplægget kan spores i det udviklede program.

Det er ikke alle aspekter, der omtales i oplægget, der skal medtages i det system, der specificeres og udvikles. En del af denne øvelse består i at afgrænse opgaven, således at det er muligt at udvikle et sammenhængende produkt (rapport og tilhørende program) af høj kvalitet inden for den afsatte tid. Men det er også vigtigt, at der tages stilling til, hvordan systemet kan udvides med de aspekter, man har bortset fra i forbindelse med afgrænsningen.

Programmets applikationslag skal udvikles testdrevet. Til hver funktion i applikationslaget skal der findes en eller flere svarende Cucumber-features, som sikrer, at funktionen gør det, den skal (ligesom vi har gjort i øvelserne). Brugergrænsefladen behøver ikke at blive udviklet testdrevet (hvorfor ikke? Husk Robert C. Martins video[1]). Ved projekt-demonstrationen skal I vise testene.

Det er ikke et krav, at data kan gemmes på en fil eller i en database. Imidlertid giver det god mening at bruge dependency inversion-princippet til at tilgå en "in-memory-database", ligesom vi har gjort i forelæsningen.

Programmet skal have en brugergrænseflade. Det kan enten være en tekstbaseret brugergrænseflade (konsolgrænseflade) eller en grafisk brugergrænseflade.

Ved bedømmelsen af rapporterne vil der blive lagt vægt på:

- fuldstændighed, dvs. at de valgte funktioner bliver komplet behandlet, og
- sporbarhed, herunder konsistens igennem alle abstraktionsniveauer.

[1] `https://youtu.be/hG4LH6P8Syk?t=2397`

Det er bedre at sikre disse kvalitetsegenskaber gennem en bevidst afgrænsning af problemet, end at brede sig for meget ud over problemstillingen.

2 Opgave Del 1: Kravspecifikation og Programdesign

Rapport 1 skal indeholde følgende afsnit.

1. *Kravspecifikation*: Afsnittet har følgende underafsnit:

   (a) *Indledning*: Dette afsnit skal indeholde en kort indledning og opsummering af besvarelsen.

   (b) *Væsentlige begreber (ordliste, glossar)*: Dette afsnit skal indeholde en opremsning af de væsentlige begreber fra problemdomænet. Hvert begreb gives en kort definition på nogle få linjer. OBS: Der kræves ingen domæneklassediagram. Dvs. klassediagram, som viser problemdomænet.

   (c) *Use case-diagram*: Dette afsnit skal give en beskrivelse af de væsentlige operationer i form af et use case-diagram. Hvis et use case-diagram bliver uoverskueligt, kan der også bruges flere use case-diagrammer. Husk, at formålet med use case-diagrammer er at skabe en oversigt over programmets funktionalitet.

   (d) *Detaljerede use cases*: Der skal også være detaljerede use cases med hoved- og alternative/fejl-scenarier som **Cucumber features**. Der skal være 2 detaljerede use cases/Cucumber features til hver medlem i gruppen (dvs. 8 for firmandsgrupper og 10 for femmandsgrupper).

   Afsnittet afsluttes med en kort diskussion, hvor f.eks. uklarheder i oplægget diskuteres, og hvor der gøres rede for valg og afgrænsninger.

2. *Programdesign*: Afsnittet har følgende underafsnit:

   (a) *Klassediagram af programdesign (solution domain)*: Dette klassediagram skal være bindeledet mellem kravspecifikation og program. Klasserne skal indeholde væsentlige attributter og operationer med typer, således at det fremgår, hvordan de væsentlige begreber modelleres, og samtidig skal diagrammet give et overblik over programdesignet.

   OBS: Klassediagrammet må ikke genereres ud fra Java-kode. Klassediagrammet skal modelleres ved hjælp af et af de værktøjer nævnt i forelæsningen, fx UMLet[2], VisualParadigm[3] osv.

   (b) *Sekvensdiagrammer*: Dette afsnit indeholder sekvensdiagrammer for de use cases fra afsnit 1d, således at læseren får et overblik over, hvordan programmet bør

[2] `https://umlet.com`
[3] `https://online.visual-paradigm.com/`

udføre disse scenarier, som tilhører de valgte use cases. Der skal være sekvensdiagrammer til to use cases per gruppemedlem (et sekvensdiagram til hver hoved- og alternativt/fejl-scenarie af hver use case).

3. Afsnittet afsluttes med en kort diskussion, hvor der f.eks. gøres rede for de valg, der er truffet. Afsnittet kan også omtale valg af algoritmer og datastrukturer.

Slidserne i uge 6 anbefaler en process som kan bruges til at lave programdesign.

3 Opgave Del 2: Peer Review af Kravspecifikation og Programdesign

Inden starten af kursusuge 8 får I rapport 1 fra en anden gruppe.

Feedback skal være konstruktiv og tage stilling til følgende punkter og afleveres gennem FeedbackFruits-modulet i DTU Learn.

1. Korrekthed: Er notationen brugt korrekt? For eksempel, bliver use cases i use case-diagrammet vist som ovaler? I kan bruge `http://uml-diagrams.org` og "UML User Guide", hvis I er i tvivl om notationerne, og I kan spørge mig. Til "UML User Guide"findes der et link til den online version på kursets hjemmeside.

2. Konsistens: Er notationen brugt konsistent? For eksempel har use case-diagrammet de samme navne som de detaljerede use cases? Svarer use-case scenarier til sekvensdiagrammer? Svarer sekvensdiagrammer til de trin i Cucumber-scenarier? Er metodenavne i sekvensdiagrammer operationer i klassediagrammet?

3. Fuldstændighed: Er notationen fuldstændig? For eksempel forklares alle domænekoncepter, der bruges i use case-beskrivelsen?

4. Læsbarhed: Er kravspecifikationen og programdesignet nemme at forstå? Ville det være muligt for jer at implementere programmet ud fra designet? Er kravspecifikationen nem at forstå for kunden?

4 Opgave Del 3: Rapport 2, Implementering og Test

Der skal afleveres den endelige version af kildekoden, testene (Cucumber), det kørende program samt en instruktion i, hvordan man kører programmet som et Eclipse-, Visual Studio Code- eller IntelliJ-projekt. Instruktionen skal indeholde loginoplysninger (brugernavn og password) til applikationen, hvis login er nødvendigt (hvilket ikke er et krav). Der skal mindst være én bruger med initialerne "huba".

Projektet skal pakkes i en ZIP-fil. Efter udpakning skal det være et Maven-projekt, der kan importeres i Eclipse. Det er vigtigt, at I sikrer, at jeg kan køre både programmet og alle testene, samt at der ikke mangler nogen filer.

Desuden skal rapport 2 indeholde følgende afsnit:

1. *Systematiske tests (white-box tests)*: Dette afsnit skal indeholde planer for systematiske white-box tests af udvalgte metoder. Der skal vælges én metode per gruppemedlem. Vælg metoder, der udfører handlinger. Getter- og setter-metoder skal ikke vælges. Vis metoderne i rapporten. *Når I bruger skærmbilleder i rapporten, skal de være i lys tilstand (Light mode) med sort eller farvet tekst på hvid baggrund.* Benyt samme skabelon til systematiske tests som anvendt i forelæsningerne (dvs. to tabeller: den ene med inputegenskaber og den anden med konkrete værdier). De planlagte systematiske tests skal implementeres og udføres ved hjælp af JUnit eller Cucumber og skal afleveres som en del af Eclipse/IntelliJ-projektet. **OBS**: Testene erstatter ikke testene, der er oprettet gennem TDD/BDD. Det vil sige, at der bør være to test-sæt: det første sæt kommer fra white-box testtabellerne, og det andet sæt stammer fra TDD/BDD.

2. *Design by contract*: Dette afsnit skal indeholde kontrakterne for de metoder, I har valgt i afsnittet om systematiske white-box tests. Husk at vise metoderne i rapporten, og denne gang skal de også være præsenteret med assert-statements. Kontrakterne, dvs. pre- og postbetingelserne for de metoder, skal vises som en blanding af matematiske formler og programmeringssprog, som bedst udtrykker betingelserne. Desuden skal der tilføjes assert-statements i metoderne, som implementerer pre- og postbetingelserne i jeres kode. Disse skal også beskrives og forklares i rapporten. Der kan være én eller flere assert-statements for pre- og post-betingelserne af metoderne. Hvis der ikke er nogen særlige pre-condition, skrives `assert true;`.

3. Derudover skal rapport 2 indeholde et diagram, der viser code coverage af de enkelte Java-filer efter alle testene er kørt, samt en forklaring af resultaterne af code coverage.

4. *Designmønstre/SOLID-principper*: I dette afsnit skal I redegøre for de designmønstre og SOLID-principper, I har anvendt i jeres design og kode. Hvilke designmønstre og SOLID-principper har I brugt, og hvorfor? Hvis I ikke har brugt nogen designmønstre eller SOLID-principper, skal I forklare, hvorfor I mener, at det ikke var nødvendigt at bruge dem. Bemærk, at der ikke er krav om at bruge designmønstre i jeres design og kode. Designmønstre er løsninger på specifikke designproblemer. Hvis I ikke har stødt på disse problemer, eller der findes bedre løsninger, skal der heller ikke bruges designmønstre.

5. *Konklusion*: Der skal gives en kort status og vurdering af det afleverede produkt. Der skal desuden være et kort afsnit, der reflekterer over projektets forløb og sammenligner det aktuelle programdesign efter implementeringen med programdesignet fra rapport 1.

   I konklusionen kan I også nævne eventuelle bemærkninger om jeres samarbejde. Hvordan har I opdelt projektet? Har I lavet pair programming, mob programming eller noget andet? Har nogen bidraget mere eller mindre end andre? osv.

   Der skal også tilføjes en tabel, hvor I beskriver vægtningen af jeres bidrag til projektet. Dette er nødvendigt for at hjælpe os med at kvantificere jeres bidrag i forhold til de øvrige gruppemedlemmer. I denne sammenhæng skal I ikke kun overveje mængden

af tid og indsats, men også den generelle personlige indvirkning på at opnå resultaterne i projektet. Bidragene skal rapporteres i en tabel (se eksemplet nedenfor), hvor studentkolonnerne er navngivet ved hjælp af student-ID'erne for de respektive gruppemedlemmer.

Vægten er et reelt tal fra [0.0, 2.0], hvor 0.0 betyder et ikke-eksisterende bidrag, 1.0 er et forventet bidrag, og alt, der går ud over 1.0, bør falde under kategorien "bemærkelsesværdigt". For eksempel, i tabellen nedenfor, har gruppen af fem medlemmer følgende bidrag:

	student-1	student-2	student-3	student-4	student-5
Bidrag	1.0	1.5	0.5	0.0	0.5

Der skal bruges Git som versionskontrolsystem. Ved aflevering skal URL'en til Git-repositoriet vedlægges i rapport 2. Git giver mulighed for at tilknytte flere remote repositories til jeres projekt. Det nemmeste er, at I har et remote repository hos GBar (`https://gitlab.gbar.dtu.dk`), hvor I giver mig adgang til repository'et. Vær opmærksom på, at jeg skal have ret til at se indholdet af jeres repository.

Hvis Git-repositoriet ligger på `https://gitlab.gbar.dtu.dk`, kan I ved oprettelsen af Git-repositoriet vælge "Anonymous read-only access". Bemærk, at dette giver read-only adgang og ikke en URL, der starter med "git:".

Ved bedømmelsen af rapporterne vil der blive lagt vægt på:

- Fuldstændighed, dvs. at de valgte funktioner bliver komplet behandlet, og
- Sporbarhed, herunder konsistens igennem alle abstraktionsniveauer.

Det er bedre at sikre disse kvalitetsegenskaber gennem en bevidst afgrænsning af problemet, end at brede sig for meget ud over problemstillingen.

Slidserne i uge 6 anbefaler en proces, som kan bruges til at implementere systemet ud fra programdesignet i rapport 1.

5 Projekt Beskrivelsen: Timeregistrering og projektstyring

Softwarehuset A/S bruger i dag et ældre system til timeregistrering og projektstyring. Systemet er på flere måder tungt at arbejde med. Specielt klager mange medarbejdere over besværet med at registrere, hvor mange timer de har arbejdet på forskellige aktiviteter. Projektlederne klager over, at det er svært at bemande et projekt, fordi man ikke kan se, hvem der har tid hvornår.

Softwarehuset A/S har omkring 50 udviklingsmedarbejdere og omkring 30 igangværende projekter. Mange projekter varer blot et par måneder, men nogle varer flere år. Hvert projekt har en projektleder (valgt blandt udviklingsmedarbejderne). En af projektlederens opgaver

er at opdele projektet i aktiviteter, f.eks. kravspecifikation, projektledelse, analyse, design, programmering, test med mere.

Et projekt har typisk 30 aktiviteter, men nogle har næsten 100. Et projekt oprettes, når der skal arbejdes på en opgave. Det kan være en mulig kunde, som ønsker en udvikling, eller det kan være et "internt"projekt, der går ud på at lave noget for Softwarehuset A/S selv. På oprettelsestidspunktet er information om aktiviteter og starttidspunkt ikke fuldstændige, men dog kendt i et vist omfang. Projektlederen er ikke nødvendigvis udpeget fra starten.

Projektlederen eller medarbejderne (hvis projektlederen endnu ikke er udpeget) skal gradvist opdele projektet i aktiviteter. Aktiviteterne skal kunne planlægges længe inden arbejdet på dem begynder, men planerne skal kunne ændres hyppigt. For hver aktivitet skal projektlederen kunne anføre det forventede antal arbejdstimer til løsning af opgaven (budgetteret tid). Endvidere skal der kunne anføres start- og sluttidspunkt for aktiviteten. Der ønskes en opløsning på ugenummerniveau. For eksempel er aktiviteten "kravspecifikation"budgetteret til 100 timer fordelt over 3 uger, dvs. arbejdet starter i uge 10 i år 2022 og slutter i uge 12 i år 2022. Projektlederne eller medarbejderne skal kunne bruge systemet til at bemande projekter i god tid, dvs. til at tilføje medarbejdere til aktiviteter. Der kan være behov for, at mere end én medarbejder deltager i udførelsen af en bestemt aktivitet.

Systemet skal tillade, at en medarbejder kan registrere den forbrugte tid på en aktivitet, selvom vedkommende ikke er direkte tilordnet den af projektlederen, f.eks. for at hjælpe en anden medarbejder.

Softwarehuset A/S ønsker et system, der kan hjælpe projektlederne i forbindelse med bemanding af projekter. Det skal således give overblik over, hvilke medarbejdere der burde være ledige, dvs. ikke allerede optaget af andre opgaver eller aktiviteter, på det tidspunkt, hvor aktiviteten skal udføres.

En medarbejder arbejder typisk på mindre end 10 aktiviteter ad gangen, og varigheden af den enkelte aktivitet er typisk 2-3 uger. Systemet skal tillade enkelte medarbejdere at arbejde på op til 20 aktiviteter i løbet af en uge. Der skal være faste aktiviteter for registrering af ting som ferie, sygdom, kurser med videre, som ikke kan pålægges det enkelte projekt. Faste aktiviteter registreres med start- og slutdato.

Medarbejderne skal dagligt registrere, hvor meget tid de har brugt på de forskellige aktiviteter (1/2 times nøjagtighed er tilfredsstillende). Det skal være enkelt at foretage denne registrering, så medarbejderen ikke føler, at det er en byrde. Medarbejderen skal nemt kunne se, om han/hun har registreret alle timer, der er arbejdet i dag. Det skal være muligt at rette i allerede registrerede data, ligesom det skal være muligt at registrere fremtidige aktiviteter, f.eks. ferie.

Projektlederen skal kunne bruge systemet til opfølgning. Det betyder, at projektlederen let skal kunne se, hvordan timeforbruget udvikler sig per aktivitet og for hele projektet. Der skal kunne laves rapporter til brug ved Softwarehuset A/S' ugentlige projektmøder. Ved disse møder skal projektlederen endvidere kunne se det forventede restarbejde på projektet.

5.1 Lidt data

Medarbejdere identificeres med initialer på op til 4 bogstaver, fx huba, aha, ekki etc. Projektnumre skal tildeles af systemet og have formen årstal og et løbenummer, f.eks. 22001, 22002 etc. Desuden skal projektet kunne have et navn.

Systemet er til internt brug, så det er ikke nødvendigt med adgangskontrol. Det betyder, at der kun skal bruges initialer, men ikke passwords, for at tilgå systemet.

5.2 Afgrænsning

Kunden ønsker som minimum, at en medarbejder skal kunne oprette et projekt samt oprette og tilføje en ny aktivitet til projektet. Derefter skal det være muligt at tilføje en medarbejder til aktiviteten og tilknytte en projektleder til projektet. Medarbejderen skal kunne registrere tid på aktiviteten. Derudover skal det være muligt at generere en rapport, der viser, hvor meget tid der allerede er brugt på projektet sammenlignet med den budgetterede tid.

Angående programmet er det ikke et krav, at data kan gemmes på en fil eller i en database. Systemet skal have en brugergrænseflade, men det er heller ikke et krav, at systemet kan benyttes via en grafisk brugergrænseflade.

Med tak til Michael R. Hansen og Søren Lauesen for lån af ideen til system og for store dele af beskrivelsen.

6 Generelle Regler

- Opgaven skal løses i grupper på 4-5 studerende.
- Rapporterne skal skrives på dansk eller engelsk og afleveres som PDF-fil på DTU Learn. Rapporterne skal have følgende navngivning: `rapport_xx_1.pdf` og `rapport_xx_2.pdf`, hvor `xx` er jeres gruppenummer skrevet med to cifre (f.eks. 01, 02, 03, ... 10, 11, ...). Vær opmærksom på, at filnavnene skal være `rapport_xx_1.pdf` og `rapport_xx_2.pdf`, og ikke `Rapport_xx_1.pdf` eller `report_xx_1.pdf`. Det er vigtigt, at filnavnene er præcise, da det hjælper mig med hurtigt at se, om jeg har alle rapporter og projekter.
- Programmet skal afleveres som et Maven-projekt, der indeholder kildekoden, testene, det kørende program og en instruktion i, hvordan man kører programmet og testene, inklusive eventuelle loginoplysninger. Projektet skal være pakket som en ZIP-fil, som efter udpakning kan importeres som et Maven-projekt i Eclipse, således at programmet og testene kan køres fra IDE'en.
- Det er en god idé at tjekke, om jeres ZIP-fil (efter udpakning) kan importeres som et Maven-projekt i Eclipse, og at programmet fungerer, inden aflevering.
- ZIP-filen skal have navnet `projekt_xx.zip`, hvor `xx` er jeres gruppenummer skrevet med to cifre. **Programmet må ikke være afhængigt af særlige pakker eller**

jar-filer, der ikke er nævnt i kurset. Hvis I har behov for at bruge særlige pakker eller jar-filer, skal I spørge mig først.

- Hver rapport skal have en forside med titel (f.eks. "Rapport 1"), gruppenummer, dato samt studienummer og navne på alle gruppens medlemmer.
- Hvert afsnit, use case, diagram osv. skal have navnet på den person, der har lavet det. Der skal kun være ét navn på hvert afsnit, use case eller diagram. Sørg for, at hvert medlem af gruppen har bidraget lige meget til alle rapporter og de enkelte dele af rapporterne.
- Hver klasse og metode i kildekoden skal have navnet på den person, der har skrevet den. Der skal kun være ét navn på hver klasse og metode. Sørg for, at alle medlemmer af gruppen har bidraget lige meget til kildekoden.
- *Diskussion af den obligatoriske opgave er tilladt med andre gruppers medlemmer og kursusdeltagere fra tidligere år, men anvendelse af andres program eller rapportdele er ikke tilladt. Hver besvarelse skal være selvstændig.* **Afskrift uden kildeangivelse betragtes som snyd.**
- **Mandag den 12. maj** giver hver gruppe mulighed for at præsentere en kort demo (15 minutter) af deres program in-person for en hjælpelærer. Demoen skal mindst vise testresultaterne fra afsnit 4 i rapport 2.
- Tidsfristerne for de enkelte delopgaver findes i opgave-modulet og på FeedbackFruits på DTU Learn. Der er en egen opgave for hver delopgave på DTU Learn.
- Sammen med rapport 2 og projektet skal I også aflevere rapport 1 igen. Dette giver jer mulighed for at aflevere en opdateret version af rapport 1, hvis I har fået en bedre forståelse af notationerne, f.eks. baseret på peer-review. Hvis dette er tilfældet, bedømmes kun den opdaterede version af rapport 1, og ikke den oprindelige version. Det er ikke et krav at opdatere rapport 1.

  OBS: Det er helt i orden (og jeg forventer det) hvis modellen af jeres program og kravene har ændret sig fra rapport 1 til den endelige version af programmet. Opdateringen giver dog kun mening, hvis f.eks. de anvendte notationer i rapport 1 var forkerte. *Det er ikke et krav at dokumentere den endelige applikation som et klassediagram.*
- Yderligere informationer vil blive oplyst via DTU Learn.

7 Evalueringskriterier

Karakteren i kurset gives som en samlet vurdering af rapport 1, feedback på andres rapport 1, rapport 2 og kildekoden (inkl. test). Fokus er på den korrekte anvendelse af de teknikker, der er blevet gennemgået i kurset, f.eks. korrekt brug af notation i use case-, sekvens- og klassediagrammer, samt om teknikkerne er forstået og korrekt implementeret.

I programkoden vil der blive lagt vægt på en god arkitektur, og det er vigtigt, at kildekoden er velstruktureret, let at forstå og læse. Eksempler på dette inkluderer brug af korte metoder, de rette abstraktioner og organisering af klasser i pakker. Det er også vigtigt, at det fremgår tydeligt i rapporten og kildekoden, at alle projektmedlemmer har anvendt de teknikker, der er blevet undervist i, og at alle har bidraget til koden.

Det er ikke afgørende, hvor meget af funktionaliteten i problemstillingen, der er implementeret, men snarere kvaliteten af implementeringen og rapporterne. Det er også vigtigt, at I er i stand til at afgrænse problemet korrekt.