

02161 Software Engineering 1

Obligatorisk Opgave

5. marts 2023, Version 1

1 Opgaven

I afsnit 5 findes projektbeskrivelsen, der vedrører udvikling 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 Peergrade
3. det endelige program og test, (rapport 2 plus kildekode som Eclipse eller IntelliJ 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 indenfor den afsatte tid. Men det er også vigtigt at der tages stilling til, hvordan systemet kan udvides med de aspekter, man så bort fra i forbindelse med afgrænsningen.

Programmets applikationslag skal udvikles test-dreven. Til hver funktion i applikationslag skal der findes en eller fler svarende Cucumber features som sikrer at funktionen gøre det den skal (ligesom vi har gjort i øvelsener.) Brugergrænsefladen behøves ikke udvikles test-dreven. Ved projekt demonstration skal I vise testene.

Ved bedømmelse 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 igennem 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 problem domænet. Hvert begreb gives en kort definition på nogle få liner. OBS: der kræves ingen domæneklassediagram. Dvs. klassediagram som viser problem domain.
- (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 et formålet af use case diagrammer er et skabe en oversigt over programmernes funktionalitet.
- (d) *Detaljeret use cases* Der skal også være detaljeret use cases med hoved- og alternative/fejl scenarier som **Cucumber features**. Der skal være 2 detaljeret use cases/Cucumber features til hver medlem i gruppen (dvs. 8 for firmandsgrupper og 10 for femmandsgrupper).

Afsnittet afsluttes med en kort diskussion, hvor, for eksempel, 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 samtidigt skal diagrammet give et overblik over programdesign.
- (b) *Sekvensdiagrammer*: Dette afsnit indeholder sekvensdiagrammer for de use cases fra afsnit 1d, således at læseren få et overblik over hvordan programmet bør udføre disse scenarier som tilhører til disse use cases. Der skal være sekvensdiagrammer til to use cases til hver gruppemedlem (et sekvensdiagram til hver hoved- og alternative scenarier).

Afsnittet afsluttes med en kort diskussion hvor der, for eksempel, 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 start af kursusuge 8 får I rapport 1 fra en anden gruppe. Feedback skal være konstruktive og skal tage stilling til følgende punkter og afleveres gennem Peergrade modul i DTU Learn.

1. Korrekthed: Er notationen brugt korrekte? For eksempel, bliver use caserne i use case diagram vist som oval? I kan bruge <http://uml-diagrams.org> og “UML User Guide” hvis I er i tvivle om notationerne og I kan spørge mig. Til “UML User Guide” findes der et link til online version på kursets Web side.
2. Konsistens: Er notationen brugt konsistent? For eksempel har use caseren i use case diagram samme navn som de detaljerede use cases? Svarer use-case scenarier til sekvensdiagram? Svarer sekvensdiagram til de trin i Cucumber scenarioer? Er metodenavn brugt i sekvensdiagrammer operationer i klassediagram?
3. Fuldstændighed: Er notationen fuldstændig? For eksempel forklares der alle domænekoncepter som bruges i use case beskrivelsen?
4. Læsarbarhed: Er kravspecifikation og programdesign nemt at forestår? Ville det være muligt for jer at implementere program ud fra design? Er kravspecifikation nemt at forstå af kunden?

4 Opgave Del 3: Implementering og Test

Der skal afleveres jeres endelige version af kildekoden, testene (Cucumber), det kørende program og en instruktion hvordan man kører programmet som enten Eclipse eller IntelliJ projekt. Instruktionen skal også indeholde login og password til applikationen, hvis der skal logges in (*som er ikke et krav*).

Desuden skal rapport 2 indeholde følgende afsnit.

1. *Systematiske test (white box test)*: Dette afsnit skal indeholde planer for systematiske white box tests til udvalgte metoder. Der skal vælges en metode til hver gruppemedlem. Vælger metoder som gør noget. Der skal ikke vælges getter og setter metoder. Vis metoderne i rapporten. *Når I bruger skærbilleder i rapporten, så skal de være sort eller farvet tekst på hvid baggrund. Dvs. I skal bruge Light mode til at lave skærbilleder.* Benyt samme skabelon til systematiske tests som er brugt i forbindelse med forelæsnings (dvs. to tabeller; den ene med input egenskaber og den anden med konkrete værdier). De planlagte systematiske test skal implementeres og udføres ved brug af JUnit eller Cucumber og skal afleveres som en del af Eclipse/IntelliJ projektet. **OBS:** testene erstatter ikke testene i opretter gennem TDD/BDD. Det vil sige, at der bør være to test-sets som tester de metoder i har brugt til at aflede white-box test.

2. Ydeligere skal rapport 2 have et diagram som viser code coverage af de enkelte Java filer efter alle testene er kørt, sammen med en forklaring af code coverages resultater.
3. *Design by contract*: Dette afsnit skal indeholde kontrakterne til de metoder I har valgt i afsnit om systematiske white box tests. Husk, at I igen skal vise metoderene i rapporten og den gang med assert statements. Kontrakterne, dvs. pre- og post-betingelserne af de metoder. De skal vises som blanding af matematiske formler og programmeringssprog som udtrykker betingelserne bedst. Desuden skal der tilføjes assert statements i metoderne, som implementerer pre- og post-betingelserne i jeres kode, og som er også beskrevet og forklaret i rapporten. Der kan være en eller flere assert statements til pre- og post-betingelserne af metoderne. Hvis der er ingen særligt pre-condition så skrives `assert true;`.
4. *Design mønstre/SOLID principper*: I dette afsnit skal der redegøres af de brugte design mønstre og SOLID principper i jeres design/kode. Hvilke design mønstre/SOLID principper har I brugt og hvorfor? Hvis I har ikke brugt nogen design mønstre/SOLID principper så skal der skrives hvorfor I synes det giver ikke mening at bruge design mønstre eller SOLID principper.
5. *Konklusion*: Der skal gives en kort status og vurdering af det afleverede produkt. Desuden skal der være et kort afsnit, der reflekterer omkring projektets forløb, og sammenligner det aktuelle programdesign efter implementeringen med programdesign fra rapport 1.

Her kan der også nævnes hvis I har nogle bemærkninger vedrørende jeres samarbejde. Hvordan har I opdelt projektet? Har I lavet pair programming eller mob programming eller noget andet? Er der nogen som har bidraget mere/mindre end andre, osv.

Programmets applikationslag skal udvikles test-dreven. Til hver funktion i applikationslag skal der findes en eller flere svarende Cucumber tests som sikrer at funktionen gør det den skal (ligesom vi har gjort i øvelserne.) Brugergrænsefladen behøves ikke udvikles test-dreven. Ved projekt demonstration skal I vise testene.

Der skal bruges Git som versions kontrol system. Og ved afleveringen skal der vedlægges URL til adgang til Git repository i rapport 2. Git har muligheden for at tilknytte flere remote repositories til jeres projekt. Det nemmeste er at I har også et remote repository hos GBar (<https://gitlab.gbar.dtu.dk>) hvor I giver mig adgang til repository. Være opmærksom på at jeg skal have ret til at se indholdet af jeres repository.

Ved bedømmelse 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 igennem en bevidst afgrænsning af problemet, end at brede sig for meget ud over problemstillingen.

Slidserne i uge 6 anbefaler en process som kan bruges til at implementere systemet ud fra programdesign 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 man 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 medarbejdere (hvis projektlederen er ikke endnu udpeget) skal gradvis 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 forventet antal arbejdstimer til løsning af opgaven (budgetteret tid). Endvidere skal der kunne anføres start- og sluttid på aktiviteten. Der ønskes en opløsning på ugenumre niveau. For eksempel, aktivitet "kravspecifikation" er budgetteret til 100 timer fordelt over 3 uger, dvs. arbejde starter i uge 10 og slutter i uge 12. Projektlederne skal kunne bruge systemet til at bemande projekter i god tid. Der kan være behov for, at mere end en medarbejder deltager i at udføre en bestemt aktivitet.

Systemet skal tillade, at en medarbejder kan registrere den forbrugte tid på aktiviteten, selvom han er ikke direkte tilordnet den af projektlederen. For eksempel 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 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 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 pålignes det enkelte projekt.

Medarbejderne skal dagligt registrere hvor meget tid de har brugt på de forskellige ak-

tiviteter (1/2 times nøjagtighed er tilfredsstillende). Det skal være enkelt at foretage denne registrering, så medarbejderen ikke føler det er en byrde. Medarbejderen skal enkelt kunne se, om han/hun har registreret alle timer man har 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 se det forventede restarbejde på projektet.

5.1 Lidt data

Medarbejdere identificeres med initialer på indtil 4 bogstaver, fx. huba, aha, ekki, etc. Projektnumre skal tildeles af systemet og have formen årstal og et løbenummer, f.eks. 22001, 220002, etc.. Desuden skal projektet kunne have et navn.

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

5.2 Afgrænsning

Kunden prioriterer projekt planlægning før registrering af arbejdstidsforbrug. Til sidst kommer rapporter til brug ved projektmøder.

Kunden ønsker at det er muligt at oprette et projekt og tilføje en medarbejder til dette projekt. Desuden skal det være muligt at tilknytte en projektleder til projektet og opretter en aktivitet. Medarbejderen skal være i stand til at registrere tid til den aktivitet.

Angående programmet, så er det ikke et krav, at data kan gemmes på en fil eller 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 hedde `rapport_xx.1.pdf` og `rapport_xx.2.pdf` hvor `xx` er jeres gruppe nummer skrevet med 2 cifre, dvs. 01, 02, 03, ..., 10, 11, Være opmærksom på at det skal være `rapport_xx.1.pdf` og ikke `Rapport_xx.1.pdf` eller `report_xx.1.pdf` osv. Navne er meget vigtig fordi det hjælper mig at se med det samme om jeg har alle rapporter og Eclipse/IntelliJ projekter.

- Program skal afleveres på DTULearn som Eclipse/IntelliJ projekt som indeholder kildekoden, testene, det kørende program og en instruktion hvordan man kører programmet og testene, især login informationer. Projektet skal være i en ZIP fil som (efter udpakning) kan importeres som Maven projekt i Eclipse eller IntelliJ så et man kan køre programmet og testene fra IDEen.

Det er en god ide at tjekke om jeres ZIP (efter udpakning) kan importeres i Eclipse og IntelliJ som Maven projekt og at programmet kører i Eclipse eller IntelliJ inden aflevering.

Navnet skal være **projekt_xx.zip** hvor xx igen er jeres grupper nummer skrevet med 2 cifre. **Program må ikke være afhængig af nogle særlige pakker eller jar-filer som er ikke nævnet i kurset. Hvis I har behov for at bruge særlige pakker eller jar-filer, så skal I spørge mig først.**

- Hver rapport skal have en forside med titel, fx. "Rapport 1", gruppe nummer, dato og studienummer og navn for alle gruppens medlemmer.
 - Hvert afsnit eller use case, diagram o.s.v. skal have navnet på fra personen, som har lavet afsnittet, use case, diagrammet o.s.v. Der kan være kun et navn på hvert afsnit, use case, diagram, o.s.v. Sikre jer at hver medlem af gruppen har bidraget lige til alle rapporter og dele i rapporterne.
 - Hver klasse og method skal have navnet på fra personen, som har skrevet klasse/metode. Der kan være kun et navn på hvert klasse/metode. Sikre jer at hver medlem af gruppen har bidraget lige til kildekoden.
 - *Diskussion af den obligatoriske opgave er tilladt også med andre gruppens medlemmer og kursusdeltager fra tidligere år, men ingen form for anvendelse af andres program eller rapportdele er tilladt. Hver besvarelse skal være selvstændig. Afskrift uden kildeangivelse betragtes som snyd.*
 - **Mandag den 8 maj** giver hver gruppe en kort demo (15 min) af deres program in-person eller via MS Teams til en hjælpelærer. Demoen skal (som minimum) vise testresultaterne fra afsnit 4 i rapport 2.
 - Tidsfristerne til de enkelte delopgaver findes i opgave modul og på Peergrade på DTU Learn. Til hver delopgave findes der en egen opgave på DTULearn.
 - Det er muligt, men ikke krav, at aflevere en opdateret version af rapport 1 ved aflevering af rapport 2 og kildekoden hvis I nu har en bedre forståelse vedr notationerne. I den tilfælde bedømmes der kun den opdaterede version af rapport 1 og ikke den oprindelige version.
- OBS — det er helt okay (og jeg forventer det også) hvis modellen af jeres program og kravene har ændret sig fra rapport 1 til det endelige prgram. Dog opdateringen giver kun mening hvis I har, fx, brugt forkerte notationer i rapport 1.

- Yderligere informationer vil blive oplyst via DTU Learn.

7 Evalueringskriterier

Karakteren i kurset gives som helheds vurdering af rapport 1, feedback til andres rapport 1, rapport 2 og kildekoden (inkl. test). Fokus er på den korrekte brug af teknikerne der bliver vist i kurset, fx. brug af den korrekte notation ved use case-, sekvens- og klassediagrammer, og om teknikerne er forstået eller ej.

I programkoden bliver der lagt vagt på en god arkitektur, og at kildekoden er velstruktureret og let at forstå og læse, fx. ved brug af korte metoder, de retter abstraktioner osv. Det er også vigtigt at jeg kan se i rapporten og kildekoden at hver projektmedlemmer har brugt alle teknikerne og har bidraget til koden.

Det drejer sig ikke om hvor meget af den funktionalitet i problem beskrivelsen er implementeret, men om kvaliteten af implementeringen og rapporterne. Det er også vigtigt at I er i stand til at afgrænse problemet.