BGPP Integreret Projekt - 2013

Emne for projektopgaven

Projektets mål er at udvikle et softwaresystem, som skal holde styr på fly reservationer for et mindre flyselskab. Flyselskabet har forskellige flytyper og flyafgange (destinationer og tidspunkter).
En kunde skal f.eks kunne reservere sæder til bestemte afgange eller tidsrum. For eksempel at man gerne vil bestille 5 billetter ved siden af hinanden på afgang DW901 fra København til Rønne, kl 12.42 Onsdag d 19. December 2012.
Det er meningen at det udviklede system skal anvendes af en ansat hos flyselskabet og skal kun betjenes af en ansat ad gangen. Kunder kan ringe til flyselskabet og foretage reservationer, men det er ikke muligt at reservere over internettet. Det er således kun nødvendigt at lave et selvstændigt program, som kører på en enkelt maskine og ikke tager hensyn til samtidighedsproblemer, applets m.v. Det forventes kun at I beskæftiger jer med emner præsenteret på kurset.

Formål med projektopgaven

I projektet skal i analysere et givet problem og programmere et softwaresystem til at løse det. I skal i den tilhørende raport præsentere systemets formål, opbygning og virkemåde. I skal desuden afprøve systemet, redegøre for om det virker som ønsket, og vurdere i hvilken grad jeres afprøvning understøtter en sådan konklusion.
Projektet involverer udvikling, test og dokumentation af et Java-program med grafisk brugergrænseflade og brug af en relationsdatabase.

Formalia for projektarbejdet

- Projektet udføres i grupper af 3 personer.
- Projektrapporten skal afleveres i **tre eksemplarer** senest **16. december kl 14:00 i studieadministrationen**. Det er jeres ansvar at aflevere til tiden. Hvis i afleverer for sent kan i ikke gå til eksamen. Se på mit.ITU (Kursusbasen) for detaljer for afleverings tid og andre formalia. Der er som regel mange der aflevere samme dag så i kan med fordel planlægge at i skal udskrive rapporten aftenen før deadline for at sikre at i afleverer rettidig.
- I skal selv danne grupper og meddele Tobias NAVN og EMAIL på gruppens medlemmer senest torsdag 22. November kl 12:00.
- Systemet skal programmeres i Java. Man kan frit vælge databaseserver; Microsoft Access og MySQL er to oplagte muligheder.
- Projektrapporten (eksklusive bilag) skal have et omfang på 15-18 (max) normal sider (dvs sider uden figurer).
- Rapportens forside skal oplyse kursus, projekttitel, forfatternes navne og ITU-email, dato.
- Der er instruktorhjælp til projektet tirsdage 9-12 og torsdage 10-12 i projekt perioden.
- Brug meget gerne `LearnIT` til at stille spørgsmål af almen interesse, fx om fortolkning af denne opgavetekst.
- Plagiat er (naturligvis) forbudt. I må ikke bruge andres tekst, illustrationer eller programkode uden udtrykkelig kildeangivelse.

Arbejdsopgaver og krav til systemet

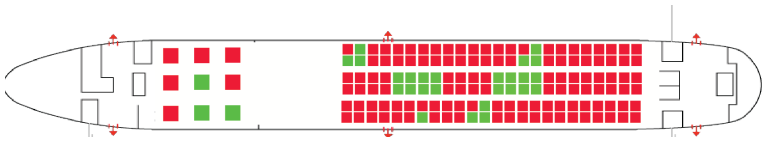
Systemet skal tillade brugeren at oprette reservationer, at rette i disse samt at give overblik over reservationer i en bestemt periode, på bestemte afgange eller på bestemte ruter /destinationer.

Typiske arbejdsgange som systemet kan understøtte er:

- En kunde ringer ind og vil bestille to sæder fra fra København til Rønne d 19. December.
- En kunde ønsker at ændre sine plads reservationer til andre på samme fly.
- En kunde ønsker at ændre en reservation til en anden afgang.
- En kunde ønsker at slette en reservation
- Den ansatte vil gerne have et overblik over hvilke afgange mellem d 17. og 21. December 2012 hvor der ledige pladser mellem Rønne og Timbuktu
- osv...

Det er op til den enkelte gruppe at definere og implementere og argumentere for de enkelte arbejdsopgaver, som systemet skal (og ikke skal) kunne håndtere, men følgende punkter skal være opfyldt:

- For at systemet skal kunne hjælpe ekspedienten med at få overblik over reservationerne er det nødvendigt at kunne repræsentere disse grafisk. Det skal være muligt på en hensigtsmæssig måde at kunne vælge og redigere reservationer vha musen (f.eks. ved at klikke på dem i GUI'en).
- I den grafiske brugergrænseflade skal det være muligt at få et overblik over reservationerne af flere afgange på samme tid såvel som at se hvilke pladser på en bestemt fly afgang som er frie og reserverede. Figur 1 viser et muligt design, men i må naturligvis gerne lave jeres eget.
- Det skal være muligt at søge efter afgange hvor der er ledige pladser i en bestemt periode.
- Det skal være muligt at lave en ny plads reservation (et eller flere sæder) på en bestemt afgang.
- Det skal være muligt at slette en reservation.
- Data skal gemmes i en database, men det er ikke et krav at systemet skal give brugeren et værktøj til at redigere, tilføje eller slette flytyper eller flyruter.



Figur 1: En mulig grafisk repræsentation af reservationerne.

Vurdering af projektarbejdet

Hovedvægten i vurderingen lægges på projektrapporten. Husk at underviserne og censor har begrænset tid til at vurdere jeres arbejde; rapporten skal være velskrevet og hurtigt give læseren overblik over hovedtrækkene i jeres løsning og jeres grunde til at løse problemet på netop den måde. Kode i hovedteksten bør undgås og kan i de fleste tilfælde løses ved at give referencer til appendiks. Der lægges vægt på at brugergrænsefladen understøtter hyppigt forekommende arbejdsopgaver godt og at den programmeringsmæssige løsning er velstruktureret, veldokumenteret og vedligeholdsvenlig og ikke indeholder overflødige dele og "smarte"hacks.

I skal kunne argumentere for at jeres program virker og dermed kunne overbevise en potentiel køber om at dette program virker efter hensigten. Det vil sige at jeres rapport skal afspejle udførlig automatisk og manuel afprøvning af programmet. Den overordnede struktur for afprøvningen kan skrives i hovedteksten men med specifikke detaljer i appendiks

Rapportens form

Det vigtige er at rapporten er velskrevet. I rapporten er det vigtigt at huske på hvem målgruppen er i de enkelte afsnit. Skriv generelt til andre på jeres niveau så de (i princippet) kan genbruge jeres overvejelser og struktur til at reproducere hvad i har lavet (uden at kende til jeres programkode). I kan naturligvis bruge andre studerende til at verificere om det i har skrevet er klart og tydeligt. Det burde være muligt at beskrive strukturer og tests uden at læse kode (java/Junit). I skal derfor huske a skabe overblik over stukturen af rapporten og jeres produkt. Ofte er det en fordel at beskrive de overordnede strukturer før detalje.

I skal yderemere sikre at læseren med overbevisning kan se at jeres produkt virker. Dvs hvordan har i testett produktet og hvilke dele. Giv overblik over hvad i tester for og hvordan, men uden at vise de enkelte unittests.

Det kan ofte være en fordel at bruge figurer / billeder i teksten, da det ofte bliver nemmere at beskrive strukturer og sammenhænge

For yderligere information om rapportens form og indhold, henviser vi til Peter Sestofts note *Udformning af rapporter* der findes på kursushjemmesiden.

Rapporten bør indeholde følgende afsnit men ikke nødvendigvis i nedenstående rækkefølge.

Et forord og en indledning

Baggrund og problemstilling. Bør indeholde en præsentation, præcisering og afgrænsning af krav til systemet, f.eks i form en liste af de arbejdsopgaver (tasks) og de data som systemet skal håndtere. Det er vigtigt at rapporten kan læses selvstændigt men lad være med at gentage hele problemformuleringen (det er tilladt at henvise til den). På den måde sparer i også plads uden at tabe vigtige informationer.

Brugervejledning og eksempel. Dette afsnit har til formål at hjælpe brugeren til at anvende programmet korrekt og skabe at overblik over programmets funktionalitet. Ofte kan brugervejledning med fordel placeres i begyndelsen af raporten for på den måde at skabe et overblik over jeres program og give læseren et overblik over systemet inden en detaljeret beskrivelse af objekt/klasse model. Husk god brug screen shots for at illustrere essentielle dele af af brugergrænsefladen.

Problemanalyse. Problemanalysen er det centrale afsnit i rapporten. Den har til formål at vise hvad i har tænkt inden i kodede. Analysen bør ikke blot beskrive jeres løsning men i skal også argumentere for jeres valg . En god analyse indeholder sædvanligvis en *nedbrydning af problemet* og afgrænsninger / fravalg . *Giv overblik*. Detaljer kan jo findes i kode som kommentarer eller uddybet i appendix. I kan evt. skrive om alternative design beslutninger, men pas på da der er mange alternative designvalg. Se nedenfor for eksempler på gode spørgsmål at diskutere. I bør dog undgå at remse de forskellige alternativer som i ikke brugte op. Nedenfor er en liste af mulige spørgsmål i kan diskutere i analysen. Listen er ment til inspiration, og til at illustrere hvilket abstraktionsniveau analysen skal foregå på. Hvilke spørgsmål I vælger at diskutere afhænger af jeres egen løsning, og kan meget vel være spørgsmål der ikke er på listen. Det er jeres opgave at finde de rette spørgsmål at diskutere.

- Hvad vil det f.eks. sige at redigere en reservation? Er det nok at kunne slette og oprette en ny?
- Hvilke(n) klasse(r) skal have til ansvar at
  - lave den grafiske brugergrænseflade?

  - styre navigationen i brugergrænsefladen, dvs. finde ud af hvad der skal ske, når brugeren klikker på en knap
  - generere SQL udtryk og kommunikere med databasen? Hvis dette er en speciel klasse, på hvilket abstraktionsniveau skal man så kommunikere med den?
  - Hvordan kan design patterns sikre at i får en bedre struktur.
- I den grafiske præsentation af reservationerne, hvordan finder i ud af hvilken aftale brugeren ønsker at redigere? Har i brug for at kalde databasen? Skal i bruge en særlig datastruktur til at holde styr på reservationerne? Svaret på det sidste spørgsmål kunne f.eks komme i afsnittet med programbeskrivelse.
- Hvordan sikres at programmet opfører sig korrekt og hvilke tests bør udføres for at sikre dette.

Beskrivelse af Design. Her er målgruppen vedligeholdelses programmøren. Hvad skal han/hun vide (de vigtigste dele) for at få et overblik over programmet. I kan derfor give en oversigt over klasser med en kort beskrivelse af klassens primære funktionalitet. En detaljeret beskrivelse af metoder og felter er ikke relevant i hovedteksten, men bør som minimum findes i koden som kommentarer. Det kan f.eks. betyde at i starter med at analysere brugergrænsefladen, derefter det overordnede objekt orienterede design, og til slut beskæftiger jer med mere detaljerede problemer. Database design hører også til i analysen. Generelt bør i skrive rapporten således at andre studerende på BGPP (f.eks til næste års studerende) kan forstå jeres design / program og kan re-implementere jeres løsning uden at se jeres kode. I bør undgå at skrive kode eksempler i rapporten - den kan jo ses i appendiks. Figurer over designet kan ofte lette teksten og gøre jeres design mere klart. Detaljerede informationer om tabeller og typer kan evt lægges i appendiks (med reference i hovedteksten)

Afprøvning. I dette afsnit skal i sandsynliggøre over for en evt. køber at jeres program virker efter hensigten dvs i bør overbevise læseren om at i har testet alle kroge af systemet på en systematisk måde. Det er obligatorisk at lave fyldestgørende JUnit tests på (som minimum) centrale klasser Afprøvningen bør omhandle både unit test af klasserne, system test og evt. usability test. Her kan f.eks tænke på at jeres værste fjende (f.eks politimesteren i Rønne) skal kunne forsøge alt for at få jeres program til at gå ned. Da der er begrænset med plads i hovedteksten kan i overordnet skrive om hvordan i har testet de enkelte dele, dvs hvilke klasser og hvilke overordnede cases i har valgt at teste. De enkelte tests or resultater bør lige som programkode kunne ses i appendix. Her skal i kunne overbevise os om at jeres program virker og give et overblik over hvordan og hvilke test i har lavet. De specifikke unittest (koden) kan lægges i appendiks.

En kort konklusion om i hvilken grad produktet opfylder de opstillede krav, herunder en mangelliste.

Bilag:

- kondenseret projektdagbog, og endelige versioner af arbejdsblade
- udskrift af Java-kildeteksten til det udarbejdede softwaresystem (overskueligt, med en ret lille font eller i to spalter på tværs af papiret)
- Test cases og detaljere test resultater
- Reflekterende beskrivelse af arbejdsprocessen.
- evt nødvendige informationer for at køre programmet.

Læg mærke til at projektrapporten både omhandler *produktet*, altså softwaresystemet og dets opbygning, og evt. en smule om *processen* - altså hvordan i har arbejdet. Mest vægt bør lægges på omtalen af produktet. Process beskrivelsen kan evt lægges i et appendiks med reference fra hovedteksten.

Gode råd

Før i starter med at programmere så bør i bestemme centrale klasser og deres sammenhænge samt identificere tekniske problemer, der skal løses og hvilke der er sekundære. Hvis noget virker meget uoverskueligt, så *Solve a Simpler Problem First*. Lav en løsning der er åbenlyst utilstrækkelig, men dog et skridt i den rigtige retning. Derefter er i meget klogere, og kan tage et nyt skridt. Start f.eks med at lave et skelet til brugergrænsefladen, hvor knapperne ikke gør noget, og implementer derefter funktionaliteten.

Det er meget bedre udtrykkeligt at beslutte at et specielt delproblem (f.eks. retning af reservationer) slet ikke håndteres i systemet, og eventuelt skrive det på en mangelliste, end at programmere en uigennemtænkt, uafprøvet og udokumenteret løsning til delproblemet. Sådan en delløsning kan komplicere den samlede softwareløsning, reducere dens brugbarhed og vanskeliggøre videreudvikling.

Lad være med at opfinde komplicerede softwareløsninger på opgaver som brugeren meget bedre kan løse manuelt. Brug hellere opfindsomheden på at lave et godt programdesign eller en god visuel understøttelse til brugeren. omend i skal passe på at iikke at bruger for meget energi på brugergræsefladen hvis andre dele som dokumentation, afprøvning mv halter.

Lad være med at lave alt på en gang. Forsøg at se på hvert delproblem for sig, og evt lav arbejdsblade til hvert delproblem, så i kan huske hvad i har overvejet og besluttet og hvorfor.

- Eksempel: Lav listen af ekspedientens arbejdsopgaver færdig.
- Eksempel: Undersøg om jeres brugergrænsefladedesign kan understøtte alle arbejdsopgaverne.
- Eksempel: Gennemtænk databasedesignet, opdigt nogle eksempeldata, og se om alle nødvendige data er til stede for at den tænkte brugergrænseflade ville kunne fungere.
- Bestem jer for hvilke dele der skal implementeres og en deadline. Uddelegér gerne opgaverne så alle ved hvad de har ansvar for, men brug hinanden løbende til afstemning og til at hjælpe hinanden.
- Selve rapporten vægtes højt så det er vigtigt i hele tiden arbejder på den og ikke undervuderer dens betydning for det samlede resultat.
- For at få god raport kan det være en fordel at gruppen aftaler et tidpunkt hvor der stoppes med programmere (”Code Freeze”) og lægge fokus på raport skrivning (mere end 1 uge før aflevering, men gerne tidligere).
- Husk at bruge tegninger og tabeller, ikke kun tekst. Det gør raporten nemmere at læse for andre og nemmere for jer at skrive.
- Husk på hvad formålet og modtager med hvert afsnit er og skriv det.
- Som en hovedregel bør det ikke være nødvendigt at have kildekode med i hovedteksten. Hvis der er specifikke detaljer (kode) i vil beskrive så gøres det nemmest ved en reference til et appendix. Hvis der skal (virkelig, virkelig..) være kode i teksten bør det være i den tekniske bskrivelse af programmet.

Inden aflevering

Inden aflevering bør i sikre at alle krav, som er beskrevet i dette dokument, er opfyldt. Gennemgå gerne dokumentet flere gange,

I kan bruge følgende ukomplette checkliste til at sikre at jeres aflevering har:

- □ En velskrevet rapport i papir form inklusiv programkode (appendix).
- □ Beskrivelse af hvordan programmet (JAR fil) skal køres af andre. Det er jeres opgave at sikre at programmet kan køres af lærere og censor. Det skal være muligt for eksaminator og censor at køre projektet som en del af bedømmelsen af projektet. Hvis det er nødvendigt at have login og password til databasen for at køre programmet skal disse fremgå tydeligt af rapportens forord.
- □ Digital medium (USB, DVD) indeholdende:
  - □ Rapport i pdf format.
  - □ Kode filer.
  - □ JAR fil til jeres program.