Bachelor

Bachelor

Bachelor in Software Development,
IT-University of Copenhagen

Jakob Melnyk, jmel@itu.dk
Frederik Lysgaard, frly@itu.dk

May 22st, 2012

Indholdsfortegnelse

1 Indledning

Det har længe været på tale at få lavet et integreret system til IT-Universitetets booking af lokaler for interne og eksterne brugere. Den nuværende metode er besværlig og uoverskuelig, da planlægningen og håndteringen af bookinger foregår i flere forskellige systemer med Facility Management (FM) som centralt bindepunkt[KravSpec, Kap. A].
Det integrerede systems primære formål er at lette arbejdsbyrden for FM samt minimere fejl.

Denne rapport er udarbejdet i forbindelse med et projekt løbende fra 15. februar til 22. maj, 2013. Målet med projektet var at udvikle en løsning til IT-Universitetets ønske om et bookingsystem.

Projektet resulterede i en første udgave, som opfylder nogle af kravene fra en kravspefikation (beskrevet i 1.1). Den første udgave af løsningen består af det grafiske design og implementationen.
Vi evaluerer løsningen ud fra antal problemer i usability tests samt hvor stor en del af kravspecifikationen, løsningen opfylder. Desuden har vi lavet en kort undersøgelse af, hvilke alternativer IT-Universitetet har i forbindelse med anskaffelsen af et booking-system.
Vores konklusion består af en evalueringen af projektet samt en beskrivelse af de erfaringer, vi har gjort igennem processen.

1.1 Kravspecifikationen

Systemet er baseret på en kravspecifikation udarbejdet på kurset "Anskaffelse og kravspecifikation" i efteråret 2012. Kravspecifikationen er udarbejdet af Stig Larsen (stla@itu.dk), Miki Ipsen (mikc@itu.dk), Garwun Jeffrey Lai (gjel@itu.dk) og Merete Larsen (mnol@itu.dk)[1].

1.2 Versionsstyring, database og service info

Vi har anvendt Git (på github.com) til versionsstyring. Derudover har vi brugt en virtuel server opsat af ITU til at hoste vores service og database.

GitHub repository https://github.com/esfdk/itubs

Service URL http://rentit.itu.dk/RentIt12/Services/Service.svc

Database RentIt12 på http://rentit.itu.dk (brugernavn: RentIt12Db – adgangskode: Zaq12wsx)

Test Administrator Brugernavn="Admin@BookIt.dk" Adganskode="zAq12wSx"

Test Bruger Brugernavn="test@bookit.dk" Adgangskode="qwerty123"

[1] Vi anskaffede kravspecifikationen hos Jeffrey Lai.

2 Baggrund for projektet

2.1 Udfordringen ved et bookingsystem

I lang tid har lokale bookning på ITU været et uoverskueligt og problemfyldt område, der har forhindret at ITU's lokaler kunne bruges optimalt. Vi vurderer derfor, at det er vigtigt at udvikle et ordentligt lokale bookingsystem således, at ITU kan udnytte deres mange lokaler optimalt. Det udfordrende ved et bookingsystem er, at det er et servicesystem der det både skal understøtte servicer til booking af lokaler samt opdatering og vedligeholdelse af systemet. Der er derfor mange aktører og mange arbejdsområder der skal tages hensyn til. For at have overblik over, hvad der skulle understøttes og hvad der var vigtigst fulgte vi en kravspecifikation som blev udarbejdet i kurset "Anskaffelse og kravspecifikation" på ITU i efteråret 2012. Kravspecifikationen er lavet med fokus specifikt på ITU og hvad et bookingsystem til ITU kræver.

2.2 Kravspecifikationen og ændringer

I kravspecifikationen beskrives hvordan den nuværende booking situation er på ITU, i øjeblikket foregår bookingen af lokaler i mange forskellige systemer, hvor der er en aktør som fungerer som bindeled for administrationen af lokaler. For at forbedre den løsning foreslår kravspecifikationen, at der laves et system som alle aktørerne arbejder op imod. Det er så vidt muligt den løsning vi vil lave en begrænset implementering af. Figur **??** viser den nuværende situation.

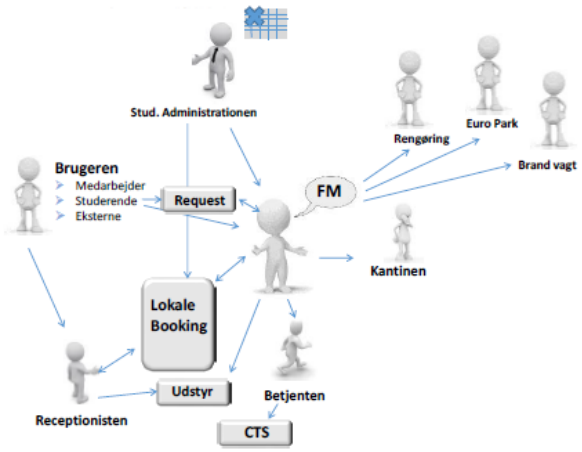

Figure 2.1: Nuværende system flow for booking systemet på ITU

2.2.1 Ændringer til kravspecifikationen

Kravspecifikationen præsenterer en datamodel, der beskriver hvilke informationer systemet skal bruge. Vi har lavet nogle ændringer så den passer til vores implementering af bookingsystemet. Vi har fjernet lokale status og lavet en "mange til én" forbindelse mellem booking og lokale da vi gerne ville have, at en booking kun kunne have et lokale. Vi fjernede også "Lokale egenskaber" da vi vurderede, at det gav mest mening hvis "Inventar" repræsenterede et unikt stykke inventar og ikke havde antal i "Lokale egenskaber". Vi slettede også alle "pris" værdier fra modellen, da vi ikke havde nogen intention om at understøtte fakturering.

2.3 Arbejdsopgaver

Denne sektion beskriver hvilke arbejdsopgaver vi understøtter i første release samt giver en prioritering af, hvilke arbejdsopgaver der bør fokuseres på i følgende releases.

2.3.1 Arbejdsområde 1: Booking

C1: Administrer booking

Arbejdesopgave	
C1.1	Understøttet
C1.1a	Understøttet
C1.2	Understøtet
C1.2a	Understøttet
C1.3	Understøttet
C1.4	Understøttet
C1.5	Understøttet

C2: Forespørg på booking

Arbejdesopgave	
C2.1	Understøttet
C2.1a	Understøttet
C2.2	Understøttet
C2.3	Understøttet

2.3.2 Arbejdsområde 2: Booking behandling

C3: Modtag forespørgsel fra Eksern

Arbejdesopgave	
C3.1	Understøttet
C3.2	Understøttet
C3.2a	Understøttet
C3.3	Ikke understøttet
C3.4	Understøttet
C3.5	Understøttet
C3.6	Ikke understøttet

C4: Klargør lokaler

Arbejdesopgave	
C4.1	Ikke understøttet
C4.2	Ikke understøttet
C4.3	Ikke understøttet

C5: Udlever nøgle

Arbejdesopgave	
C5.1	Ikke understøttet
C5.2	Ikke understøttet

2.3.3 Arbejdsområde 3: Forplejning

C6: Håndter forplejning

Arbejdesopgave	
C6.1	Ikke understøttet
C6.2	Ikke understøttet
C6.3	Ikke understøttet
C6.4	Ikke understøttet

C7: Fakturer forplejning

Arbejdesopgave	
C7.1	Ikke understøttet
C7.5	Ikke understøttet

C8: Opadter menukort

Arbejdesopgave	
C8.1	Ikke understøttet

2.3.4 Arbejdsområde 3: Systemadministration

C9: Opdater listen for ekstra udstyr

Arbejdesopgave	
C9.1	Understøttet

C10: Opdater lokaler

Arbejdesopgave	
C10.1	Understøttet
C10.2	Understøttet

C11: Administrer bruger

Arbejdesopgave	
C11.1	Ikke understøttet
C11.1a	Ikke understøttet

C12: Håndter statistik

Arbejdesopgave	
C12.1	Ikke understøttet
C12.2	Ikke understøttet

2.3.5 Arbejdsområde 4: Finans

C13: Behandel faktura

Arbejdesopgave	
C13.1	Ikke understøttet
C13.2	Ikke understøttet

2.3.6 Prioritering af arbejdsopgaver

Til den første release af systemet valgte vi at fokusere på at understøtte systemadministration og booking af lokaler og udstyr, hvilket tydeligt kan ses i afsnit**??** da alle arbejdsopgaver i arbejdsområderne C1,C2,C3,C9 og C10 er understøttet. I kravspecifikationen er de arbejdsområder også blandt dem som er vægtet højest og det er samtidig de arbejdsområder som indeholder kernefunktionerne i systemet. Det som ikke blev prioriteret højt nok til at komme med i første release var de arbejdsopgaver som fokuserede på integrationen med kantinen, arbejdsopgaverne C6.1- C6.4 er eksempler på sådanne opgaver.
Grunden til at vi ikke tog de arbejdsopgaver med var, at vi vurderede, at de ikke var nødvendige for at have et system der understøttede basis funktioner, derudover lagde arbejdsopgaverne op til, at der skulle implementeres et interface specifikt til kantinen hvilket vi vurderede ville tage lang tid at implementere og vi ville også have tre brugergrupper at tage hensyn til ift. usability og testing. Fokus for næste release vil derfor være at få udarbejdet et interface som kantinen kan bruge til at integrere med resten af systemet og få finpudset de allerede eksisterende funktioner i kravspecifikationen. Behandling af faktura har også en høj vægtning, så det vil der også blive lagt fokus på.

3 Design af brugergrænsefladen

Dette kapitel beskriver det generelle design af brugergrænsefladen samt de beslutninger, som ligger bag.

3.1 Generelle mål

Vi har valgt at designe vores brugergrænseflade ud fra reglerne om design af virtuelle vinduer[SL, s. 169] samt Ease Of Use principperne[SL, s. 9]. I forbindelse med dette valg har vi sat følgende mål for designet:

- Strømlinet brugergrænseflade
- Få forskellige skærmbilleder
- Overblik
- Effektivt

3.1.1 Strømlinet brugergrænseflade

Vi har valgt at designe skærmbillederne med samme grundstruktur. Denne lighed bør gøre det intuitivt at gå fra et skærmbillede til et andet i forbindelse med udførsel af opgaver. Desuden følger det designregel 1[1] om få vindueskabeloner.

3.1.2 Kort vej fra en opgave til en anden

Brugergrænsefladen skal gøre det hurtigt og nemt for brugeren at komme fra en opgave til en anden. Dette skal gøres ved at have få skærmbilleder involvereret i en enkelt task (designregel 2[2]).

3.1.3 Overblik

Brugeren skal have mulighed for nemt at danne sig overblik over bookinger, udstyr og forplejning (regel 6[3]). Derfor skal vi have seperate skærmbilleder, som giver overblik over hver type.

3.1.4 Effektivt

Det skal være effektivt at udføre opgaver for brugere, som anvender systemet ofte.

3.2 Brugergrænsefladens udvikling og udseende

Vores skærmbilleder er opdelt i tre typer: Gitter, Almindelig og Pop-ups.
Gitterskærmbillederne bruger vi til booking af lokale, forplejning og udstyr samt administration af udstyr og lokaleinventar.
Almindelige vinduer anvender vi, hvis man skal ændre noget på et stykke udstyr/inventar eller et lokale.
Pop-up skærmbilleder er generelt advarsler eller fejlbeskeder.

[1] Few window templates
[2] Few window instances per task.
[3] Necessary overview of data

For at få et overblik over hvordan man navigere i brugergrænsefladen og hvordan de forskellige skærmbilleder ser ud, vil vi tage udgangspunkt i et eksemepel på hvordan work flowet ville være, hvis man skulle booke et lokale og tilføje forplejning til det. Vi vil derefter gennemgå de resterende interessante skærmbilleder individuelt.

Det første brugeren ser når de starter hjemmesiden op er en liste over lokale og hvornår de er ledige/booket på den givene dag se figur3.2 her man i stand til at markere de tider man gerne vil booke et lokale, for at kunne booke skal man dog først logge ind via login knappen i øverste højre hjørne. Vores første mockup af gitterskærmbilledet se figur 3.1 havde et gitter, hvor hver række var et lokale og tiderne var kolonner. Man skulle klikke i et felt for at vise, at man ønskede at booke på et bestemt tidspunkt. Når man havde valgt de tider og lokaler, man gerne ville booke, skulle man trykke på en "Book" knap. Som man kan se på Figur 3.2 valgte vi at beholde den struktur, men vi tilføjede checkboxses til gitteret. Dette blev gjort efter første runde af usability tests hvor vi observerede, at brugeren var i tvivl om, hvor man skulle klikke for at vælge tider for en booking, derudover tilføjede vi også, at udstyr m.m., man har booket, ligger øverst i gitteret, når man får oversigten over fx udstyr. Dette gør det nemt at finde det element, man har booket/bestilt.

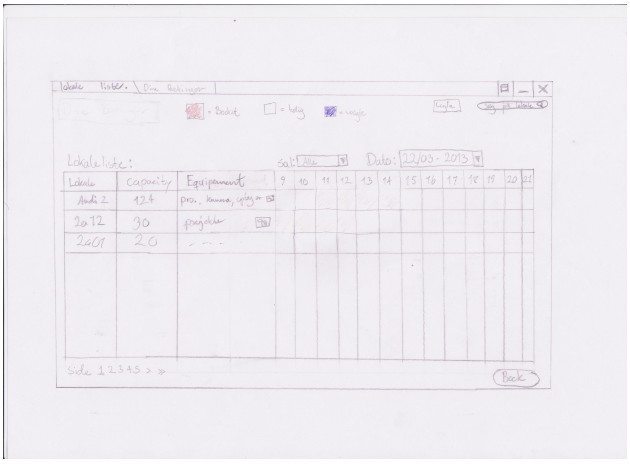

Figure 3.1: Første udgave af gitter layoutet

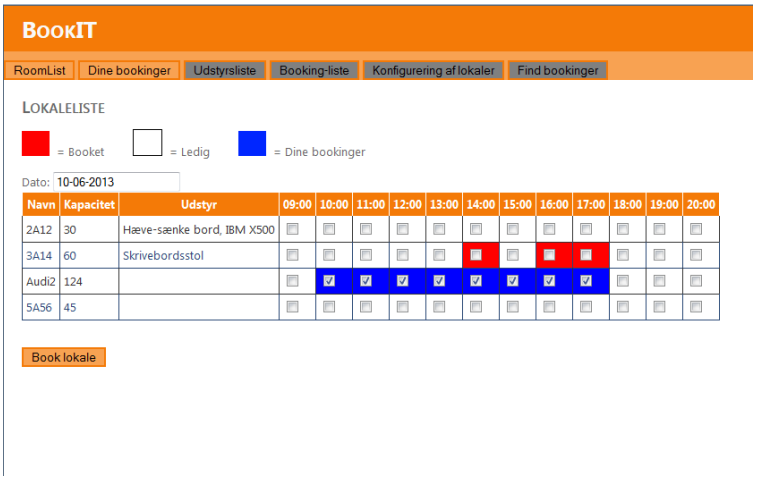

Figure 3.2: Den endelige udgave af gitteret

Hvis man markere et lokale i det tidsrum man gerne vil booke det og trykker book vil bookingen blive oprettet og man vil blive navigeret videre til en side der visser en oversigt over alle dine bookinger, se figur3.3. Til designet af det skærmbillede fokuserede vi på at overholde Gestalt-lovene[SL, s. 68], specifikt Law of Proximity[4], så det virkede naturligt at knapperne til højre hørte til gitteret.

Figure 3.3: Skærmbilledet til visning af bookinger

Hvis man markere en booking og trykker på "tilføj forplejning" vil man blive derigeret videre til en side der i udsenet minder meget om startesiden, se figur3.4. Til designet af det skærmbillede fokuserede vi, at det grafisk skulle minde om listen over lokaler således, at vi overholdte designregel 1 så brugeren intuitivt ville vide hvordan man skulle tilføje forplejning. De samme design beslutninger gælder for tilføjelse af udstyr til en booking, skærmbilledet kan ses i **??** på side **??**.

Figure 3.4: Skærmbilledet til booking af forplejning

3.2.1 Bookingliste

For at en bruger kan tilføje udstyr eller forplejning til sin booking skal den først godkendes. Dette gøres af recepsionisten, som har en liste over alle bookinger som brugerne har oprettet, se figur 3.5. Til det endelig design af skærmbilledet valgt vi at flytte godkend og afvis knapperne fra inde i gitteret som de var i papermockupen se figur 3.6 til at være ved siden af gitteret således at vi ligesom i figur3.3 benytter "Law of proximity". Desuden havde ingen af de andre gittre knapper i sig, så vi vurderede, at det burde være ens på alle billeder således at vi overholdt designregel 1.

3.2.2 Udstyrsliste

Figuren 3.5 viser det endelige skærmbillede af listen over udstyr og tilføjelse af nyt udstyr til systemet. Til designet af skærmbilledet har vi brugt "Law of proximity" så det er tydeligt, at de to funktioner i skærmbilledet ikke hører sammen. Den eneste ændring der er blevet lavet til det endelige skærmbillede i forhold til den originale papermockup er, at der er blevet tilføjet en checkbox, hvor man kan vælge om

[4] "Pieces that are close together are perceived as belonging together".

Figure 3.5: Skærmbilledet af recepsionistens liste af bruger bookinger.

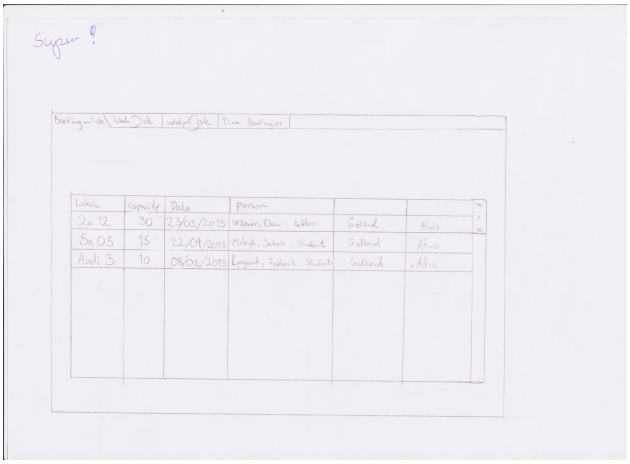

Figure 3.6: Papermockup af recepsionistens liste af bruger bookinger.

udstyret skal kunne udlånes. Hvis den ikke udfyldes vil udstyret blive kvalificeret som inventar og det vil derfor ikke blive vist under muligt udstyr, der kan tilføjes en booking.

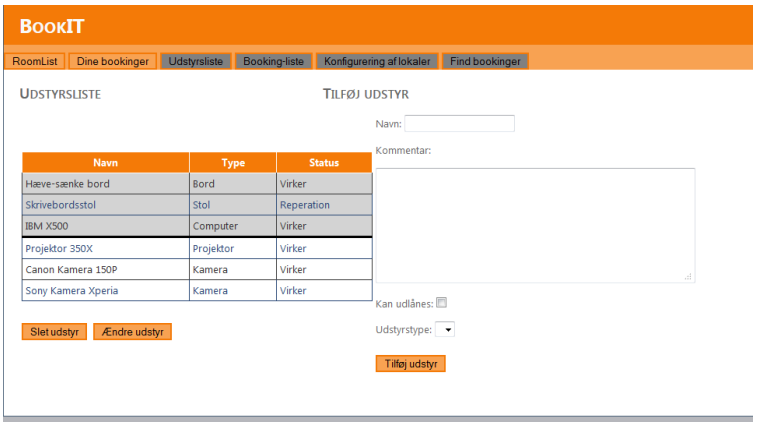

Figure 3.7: Skærmbilledet af recepsionistens liste over udstyr i ITUs system.

3.2.3 Lokale liste

I systemet er der også mulighed for ændre i listen over lokaler, se figur 3.8. Til designet af skærmbilledet fokuserede vi på, at der skulle være klart overblik, vi sørgede derfor for at overholde law of proximity ved at ligge knapperne tæt på gitteret, hvilket også gør, at det kun er det vigtige information der bliver præsenteret i gitteret.

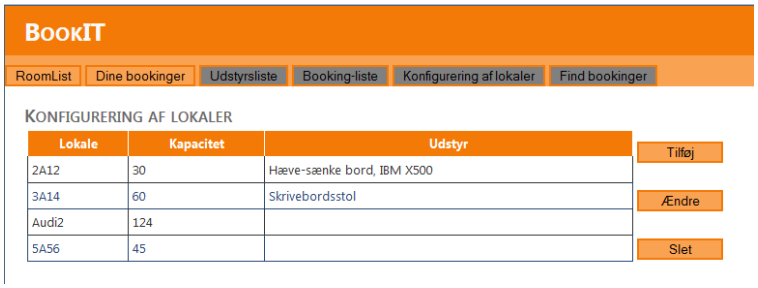

Figure 3.8: Skærmbilledet af recepsionistens liste over lokaler i ITUs system.

3.2.4 Ændring af lokale

Til ændring af lokaler i systemet fokuserede vi på, at man skulle være i stand til kunne ændre de basale ting som navn og kapacitet, og samtidig være i stand til at se en liste over inventar i lokalet hvor man skulle være i stand til at tilføje eller fjerne udstyr. Til papermockupen, se figur 3.10 havde vi skærmbilledet delt op i tre afsnit, et til at skifte kapcitet og navn, et til at tilføje inventar, og et til at slette inventar, dette var en meget klumpet og uoverskuelige løsning og den var ikke strømlignet i forhold til resten af GUI'en. Vi endte derfor med at lave en løsningen som vi i stil med de andre skærmbilleder se figur 3.9, vi valgte, at skærmbilledet kun skulle være fokuseret på en gitterløsning hvor den øverste del ville være det udstyr der allerede var i lokalet og resten af gitteret ville så være fyldt med udstyr som man kunne vælge at tilføje til lokalet.

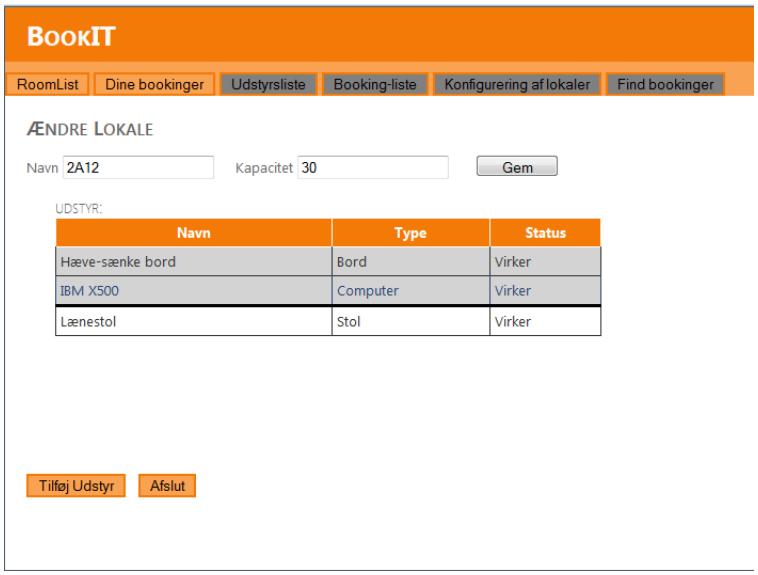

Figure 3.9: Skærmbilledet til ændring af lokale.

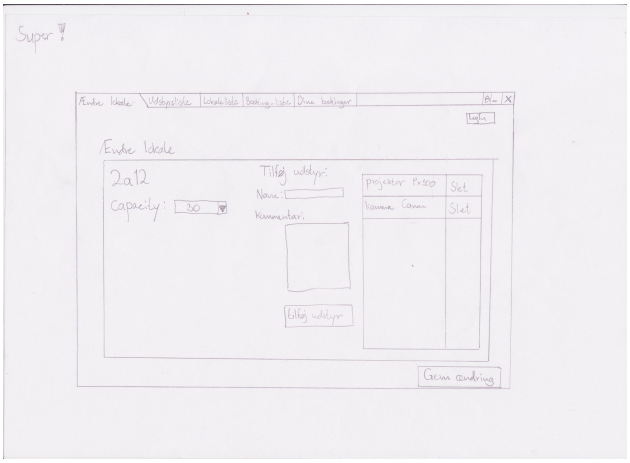

Figure 3.10: Papermockup til ændring af lokale.

4 Usability testing

Dette kapitel beskriver, hvad vores strategi for usability testing har været samt resultaterne af vores tests.

4.1 Test strategi

Kravspecifikationen beskriver, at det er nødvendigt at lave usability tests tidligt i processen for at bevise proof of concept. Vi lavede papermockups af vores skærmbilleder for at opfylde dette krav.

Vi har valgt at lave to runder af usability tests.
Runde 1 bestod af tests med en papermockup på en enkelt bruger.
Runde 2 består af tests af det endelige produkt på flere brugere.

Vi valgte kun at teste én bruger i runde 1, da flere testpersoner tidligt i udviklingsfasen kan føre til en overvældende liste af problemer[SL, s. 416]. Derudover er en enkelt testperson ofte nok til at afsløre de mest seriøse problemer i brugergrænsefladen.

Vi mener, at det havde været optimalt, hvis vi kunne have udført en usability test midtvejs i processen (mellem runde 1 og runde 2), men vi vurderede, at vi ikke ville få nok ud af testen, da det ville være svært at finde tid til at teste nok personer samt implementere de mulige ændringsforslag.

4.1.1 Test cases

Vi har to brugergrupper i vores usability tests. Der er almindelige brugere (studerende/undervisere) og administration (Facility Management/Receptionister). Til hver af disse brugergrupper har vi defineret syv test cases.

Test cases til almindelige brugere:

Test Case 1: Du har booket et lokale fra 9-10 til et møde med din vejleder og du vil gerne have en diktafon med, så du kan optage mødet.

Test Case 2: Du skal holde et møde på tre timer om et par dage. Du skal i den sammenhæng booke et lokale til formålet.

Test Case 3: Et af dine gruppemedlemmer har meldt tilbage, at han skal aflevere sin datter i børnehaven. Mødet kan derfor først holdes fra klokken 10.

Test Case 4: Da mødet ligger om morgen, tænker du, at det vil være fornuftigt, hvis der var noget kaffe og morgenbrød klar.

Test Case 5: Du skal til dit projekt optage en kort reklamefilm og har derfor brug for et kamera.

Test Case 6: Et af dine gruppemedlemmer har kamera med, så du behøver ikke længere booke et hos ITU.

Test Case 7: Deltagerne til dit møde i 2a12 har desværre aflyst.

Test cases til administrationsbrugere:

Test Case 1: En elev har kontaktet dig og gjort dig opmærksom på, at der ikke længere er en projektor i 2A12.

Test Case 2: ITU har til sin udstyrssamling erhvervet sig to nye kameraer.

Test Case 3: En student henvender sig til dig ved skranken og spørger om hans booking er blevet godkendt.

Test Case 4: Du har et møde i de kommende dage og vil helst gerne holde det på 4. sal.

Test Case 5: Du skal afholde et fordrag for 30 mennesker den 24, tidspunktet er irelevant men der skal være plads til min. 30 mennesker.

Test Case 6: Lokalet 5a12 er ikke længere et privat lokale, men skal istedet registreres som et mødelokale.

Test Case 7: Lokalet 2a12 er ikke længere til rådighed i forbindelse med booking.

Vi har designet vores test cases således, at testpersonen ikke får at vide hvor i systemet, de skal udføre deres arbejdsopgave. Dette har vi gjort for at finde ud af, hvor intuitivt vores system er.
De 14 test dækker til sammen størstedelen af systemets workflows. Vi udfører testene som "think aloud" tests[SL, s. 421]. "Think-aloud" tests består af at læse test casen op for testpersonen og bede dem tænke højt og forklare, hvad de gør under testen.

Efter testen får brugeren mulighed for at give input til, hvad de synes er godt og hvad der kan forbedres. Dette giver os mulighed for at få et indtryk af, hvad vi har overset i vores design og hvilke elementer, vi bør bruge flere steder i systemet.

I første runde af tests havde vi fokus på de store problemer i systemet, hvor vi i anden runde havde fokus på at følge op på de ændringer, vi havde lavet i forhold til resultaterne fra første runde. I anden runde var vi også interesserede i at finde ud af, om vores system er intuitivt og effektivt nok.

4.2 Runde 1: Resultater og konsekvenser

Efter den første runde af tests var der fem problemer, problemerne er klassifikeret i forhold til skalaen fra [SL, s. 439]:

Punkt 1: Medium problem - Usikker på hvordan man vælger booking tidspunkter.

Punkt 2: Cumbersome - Brugeren synes det var besværligt at navigere imellem skærmbillederne.

Punkt 3: Minor problem - Brugeren fandt det ikke naturligt, at man skulle bruge lokalelisten til at ændre sin booking.

Punkt 4: Minor problem - Problem med at tilføje udstyr/forplejning.

Punkt 5: Positiv feedback på den generelle struktur af systemet.

Punkt 1 Test personen havde problemer med at finde ud af, hvordan vores gitter til booking af lokaler fungerede. Personen troede ikke, man kunne klikke på felterne i gitteret, så der skulle lidt hjælp til, før personen forstod princippet.

Konsekvenser og løsning Vi blev enige om, at dette problem kunne løses ved at indsætte en checkbox i felterne. Det vil gøre det mere intuitivt, når man skal trykke på tiderne.

Punkt 2 Testpersonen følte, at der manglede navigeringsmuligheder mellem de forskellige skærmbilleder. Det var ikke intuitivt, at man skulle bruges tabsne til at skifte skærmbillede.

Konsekvenser og løsning Vi løste problemet ved at lave en menubar, som var placeret nærmere midten af billedet, så den blev mere synlig.

Punkt 3 Testpersonen fandt det ikke naturligt, at man skulle bruge lokalelisten til at ændre sin booking.

Konsekvenser og løsning Da vi kun testede på én person og det er et minor problem, valgte vi ikke at gøre noget ved problemet. Hvis problemer opstår i yderligere usability tests, bør en ændring overvejes.

Punkt 4 Testpersonen nævnte, at hun ikke mente, det burde være muligt at tilføje udstyr/forplejning til en booking før, den er blevet godkendt.

Konsekvenser og løsning Vi valgte at imødekomme dette ved at implementere logik, så man ikke kan trykke på knapperne til at tilføje forplejning/udstyr, hvis man ikke har valgt en godkendt booking.

Punkt 5 Testpersonen kunne lide den overordnede struktur af systemet og var positiv overfor booking funktionaliteten (gitteret). Personen skulle dog lære at bruge booking funktionaliteten.

Konsekvenser og løsning Denne feedback havde stor indflydelse på, at vi valgte at beholde vores grundstruktur.

4.3 Usability Test: Runde 2

Vi fik ikke mulighed for at udføre runde 2 af vores usability tests i tide. Da vi planlægger at arbejde videre på systemet efter denne rapport er afleveret, så vil vi følge op på vores strategi ved at lave runde 2 efter at de programmingstekniske fejl er rettet.

Som beskrevet i afsnit 4.1 består runde 2 af test med flere brugere. Der skal laves tests med 2-3 personer fra hver af de to testgrupper. Disse brugere skal udføre "think-aloud" tests som beskrevet i vores usability test strategi. Testcases fra runde 1 skal genanvendes. Desuden vil vi tage en feedback-session efter testen sammen med brugeren på samme måde, som gjort i forbindelse med runde 1.

Målet med runde 2 er, at problemerne fra runde 1 er blevet løst samt at der ikke er nogen major problem eller task failures.

4.3.1 Mulige konsekvenser af usabilitytest runde 2

Konsekvenserne af testresultaterne i runde 2 ville være forslag til ændringer til næste udgave af systemet. Hvis ingen af problemerne fra runde 1 gentager sig, går vi ud fra at det overordnede design/layout er i orden.

I tilfælde af at der kun er cumbersome problemer i forbindelse med brugergrænsefladen, mener vi, at det vil være sandsynligt at systemet er være klar til brug i forhold til usability.

5 Teknisk Design

5.1 Cost-benefit analyse af arkitektur og platform

Da vi skulle implementere vores system var det vigtigt for os, at vi valgte den korrekte implementationsstrategi i forhold til, hvor meget gavn brugeren ville få ud af de forskellige løsninger. Vi overvejede derfor hvilken nytte brugerne ville få hvis vi brugte en implementationsmulighed frem for en anden og hvor meget det ville koste os at implementere det. For at få et overblik over, hvordan disse ting forholdt sig til hinanden valgte vi at lave et skema der viste nytten af en implementation kontra omkostninger.

	Java	.NET	WCF (.NET)	Webservices (Java)	Restful (.NET)	Restful (Java)
	XX	XX				
Thick browser client	O	OO				
	X	X				
Thick application client	O½	O				
			XXX	XXXX	XXX½	XXXX
Webservice (browser client)			OOOO	OOO	OOOO½	OOO½
			XX	XXX	XX½	XXX
Webservice (app. client)			OO½	OOO½	OOO	OOOO

Figure 5.1: Tabel over cost/benefit i forhold til de forskellige implementations muligheder

Figur 5.1 viser vores cost benefit skema over de forskellige implementationsmuligheder. De røde X'er repræsenterer omkostningerne for at implementere den tilsvarende kombination. For at komme frem til omkostningerne af de forskellige implementationsmuligheder tog vi udgangspunkt i hvorvidt vi først skulle tilegne os viden eller om vi allerede havde den fornødende forståelse og således kunne gå i gang med det samme. eksempelvis ville en "Java, Thick browser client" løsning være dyrere end en "Java, Thick application client"løsning da vi ikke ved, hvordan man laver en browser klient i java. Derefter kiggede vi på, hvor meget kode der skulle skrives til de forskellige implementationsmodeller for at kunne få et fungerende system. Eksempelvis er alle Webservice løsningerne dyrere end Thick client løsningerne, da der skal kodes mere, hvis der skal laves en webservice.

De grønne O'er repræsenterer den gavn som brugeren får af den givne implementationsmodel. En af måderne, hvor vi fandt ud af hvor meget gavn de forskellige implementationsmuligheder gav brugeren var at se på, hvor mange platforme der var understøttet.

Eksmepelvis synes vi, at det gav brugeren mere nytte, hvis deres system kunne køre på alle computerere i en browser end hvis man skulle downloade en klient og tjekke om den kunne køre på computeren. Derudover kiggede vi også på, hvor nemt det ville være for brugeren at udvikle videre på systemet, eksempelvis giver implementationsmuligheden "WCF(.NET), Webservice(browser client)" brugeren mulighed for at udvikle en applikation til windows phone da der er en service at kode op imod.

Udfra de måder at vurderer cost benefit på vurderede vi, at den løsning hvor kunden ville få mest gavn var "WCF(.NET), Webservice (browser client)" da det ville give brugeren mulighed for at kunne køre systemet på stort set alle computerere med internet connection og, at de ville være i stand til at lave en mobilapplikation op imod webservicen.

5.2 Databasen

5.3 Servicen

5.4 Klienten

6 Alternative systemer

Udover at udvikle vores udgave af et booking-system til IT-Universitetet, vil vi i dette kapitel give en kort beskrivelse af to alternative systemer og sammenligne dem med vores løsning.

6.1 Room Booking System

"Room Booking System"[1] er et web-baseret booking system designet til at opfylde både skoler og virksomheders behov.

Interfacet er designet som en kalender, der viser eksisterende bookinger i farvekoder. Desuden er login til systemet seperat og dermed er der ikke mulighed for at integrere det med hverken Active Directory (AD) eller Where Are You From (WAYF). Et abonnement, der understøtter 200 lokaler/udstyr og 500 brugere, koster 434 kr. om måneden.

En fordel ved "Room Booking System" er, at det giver mulighed for at lave statistikker. Desuden kan det ses som en fordel, at det er hostet eksternt.

Ulempen ved "Room Booking System" i forhold til vores løsning er, at der er en grænse for hvor mange lokaler og udstyrselementer, som kan bookes i systemet. Desuden er det ikke muligt at integrere med et eksisterende ITU login system. Da der kun er mulighed for 500 brugere i systemet, vil nogle brugere skulle slettes for, at der kan oprettes nye i forbindelse med nye studerende/ansatte.

6.2 School Booking

"School Booking"[2] er et andet web-baseret system, som primært er designet til brug af skoler.

Systemet giver mulighed for at booke både lokaler og udstyr[3]. Man kan betale ekstra for at tillade eksterne bookere. Statetisk er også muligt i dette system. Der er dog ikke mulighed for at integrere det med AD eller WAYF.

Et system, der understøtter 1000 brugere, 300 lokaler/resurser, eksterne bookere og statestik koster årligt 11.500 kr.

Fordelen ved dette system er det samme som ved "Room Booking System": muligheden for at lave statestik.

Ulempen ved dette system er, at der er en begrænset mængde af brugere, som kan anvende systemet, samt at det ikke er muligt at integrere det med AD eller lignende, fordi det er hostet eksternt.

Systemet er det, som opfylder kravspecifikationen i højest grad, men det er samtidig det dyreste af alternativerne.

[1] http://www.roombookingsystem.co.uk/
[2] http://www.schoolbooking.com/
[3] Det bookede udstyr kan være forplejning, hvis man sætter rigtigt op.

7 Konklusion

7.1 Lessons learned

I løbet af projektforløbet var der ting som vi gerne ville have gjort anderledes og ting vi burde have overvejet bedre før vi gav os i kast med dem. Vi vil i dette afsnit gennemgå de vigtigste ting vi har lært og hvordan det har påvirket projektet.

7.1.1 Kravspecifikationen

Da vi startede projektet var vi alt for sikre på, at kravspecifikationen var en fejlfri vejledning som, hvis man fulgte ville sikre, at man endte ud med et færdigt produkt som opfyldte de krav som ITU har til et booking system. Vi fandt ud af, at det ikke var helt rigtig da der var visse ting der skulle ændres i kravspecifikationen for at kunne implementere et godt system, vi endte derfor med at bruge tid på design ting fra kravspecifikationen som vi troede skulle bruges men som vi blev nød til at skrotte da der skulle ændres i kravspecifikationen.

Et andet problem vi havde med kravspecifikationen var at vi så den som en vejledning til hvad der skulle inkluderes i første release og ikke at den i virkeligheden dækkede over alt hvad der skulle være i det endelig bookingsystem. Det faktum gjorde at vi prøvede at inkludere alle kravne i vores første release i stedet for kun at vælge de mest nødvendige, hvilket gjorde at vi kom i tidsnød da vi prøvede at udvikle for mange funktionaliteter på for kort tid.

Det vi har lært er at man skal være kritisk når man læser kravspecifikationen således, at hvis der er ting der virker forkert kan man tidligt i processen få konfimeret om det er en fejl. Derudover har vi også lært at man skal dele kravspecifikationen op således, at man kan priotere hvad der er vigtigst at få implementeret først så man ikke prøver at udvikle et fuldt produkt i en iteration.

7.1.2 Projekter af samme natur er ikke nødvendigvis ens

Et stort problem som vi lærte meget af var forventningen til hvad det krævede at lave et bookingsystem. Vi havde tidligere begge været med til at udvikle et bookingsystem til film som også var skrevet i c# som havde en bagved liggende service. Vi følte derfor vi havde en god basic for hvad sådan et projekt krævede og hvor meget tid der skulle bruges til at implementere det. Det vi dog overså var at vi havde været fem mennesker til at udvikle systemet til booking af film, og der var derfor nogen af arbejdsområderne vi ikke havde den fornødende ekspertise til at implementere i vores eget system uden problemer. Vi undervuderede også størrelsen af projektet i og med vi troede vi kunne genbrug en stor del af arkitekturen fra vores forrig projekt og på den måde gøre op for det mindre antal af gruppemedlemer, det viste sig dog, at selvom grundstruktur var ens for de to projekter var meget af funktionaliteten ikke ens og vi skulle derfor bruge mere tid på udviklingen end regnet med.

7.1.3 En serviceløsning er ikke altid bedst

Da vi satte os for at vælge hvilken implementationsstrategi vi skulle benytte os af var vi beggen enige om, at vi skulle have en bagvedligende service da det ville give brugeren mulighed for at arbejde videre med systemet, eksmepelvis hvis de gerne ville udvikle en mobil applikation. Det vi ikke tænkte over var, at projekt periode var kort og vi derfor ikke ville have særlig lang tid til at udvikle funktionaliteter hvis der også skulle laves en service, vi endte derfor ud med en service og et system som ikke havde brug for en service

for at fungerer. Vi ville derfor hvis vi skulle lave et ligne projekt vælge at lave et system uden service som fokuserede mere på funktionalitet og proof of concept end at fokuser på godt design.

7.1.4 Projekt styring

En ting som vi mener ville have gjort, at vi kunne have opnået mere i løbet af projekt perioden var projekt styring. Da vi begyndt at implementere systemet satte vi ingen mål for hvad der skulle gøres færdig til hvornår, vi forsøgte i stedet at lave det hele i en iteration, derudover havde vi også en seperation af arbejdsopgaver hvilket gjorde, at visse problemer ikke blev opdaget før sent i processen, i og med at vi ikke havde nogen fast styring blev mange små beslutninger taget løbende og blev derfor ikke dokumenteret. Derfor hvis vi skulle lave projektet igen ville vi nok lave det over flere iterationer og sørge for, at alle beslutninger blev dokumenteret således at der ikke ville være forvirring om hvilke beslutninger der var blevet taget.

Bibliography

[SL] Lauesen, Søren. User Interface Design, A Software Engineering Perspective. Great Britain: Pearson Educated Limited, 2005. Print.

[KravSpec] Miki Ipsen, Garwun Jeffrey Lai, Merete Larsen, Stig Larsen. Kravspecifikation til ITU Booking System. December, 2012.

Appendices

A Udvalgte Dele Af Koden

A.1 SQL-kode til oprettelse af database

A.2 BookItModel.cs

A.3 RoomListViewModel.cs

A.4 BookItContext.cs

A.5 BookItContext.cs

A.6 Configuration.cs