Interaktive bridgetjenester for Internett og mobil

Remy Monsen og Eirik Tenold

Masteroppgave

Institutt for informatikk
Universitetet i Bergen
2. juni 2008

Forord

Denne rapporten markerer avslutningen på ca halvannet års arbeid for oss. Det har vært en lærerik tid, og vi har begge gjort oss en del erfaringer i løpet av dette prosjektet, både i forhold til nye teknologier, og til praktisk prosjektarbeid.

Vi vil gjerne i denne anledningen rette en stor takk til vår veileder, Sven-Olai Høyland, både for selve oppgaven, og for hans hjelp og støtte gjennom hele perioden. Ingen av oss hadde nevneverdige bridgekunnskaper da vi startet, så det er takket være ham at vi har fått tilført nødvendig domenekunnskap for å takle dette prosjektet.

Vi vil også rette en takk til Norges Brigdeforbund, og da spesielt Harald Skjæran, for deres interesse og hjelp med turneringsapplikasjonen. Videre vil vi takke Høgskolen i Bergen og Universitetet i Bergen for arbeidsplasser og økonomisk støtte til prosjektet. Andre bidragsytere som fortjener en takk er Bjarte Kileng, for serverplass til turneringssystemet under utviklingen, og Bergen Akademiske Bridgeklubb for å stille med medlemmer til å teste mobilapplikasjonen. Til slutt vil vi også sende en takk til alle som har vært med å benytte turneringssystemet, og spesielt til de som har sendt oss tilbakemeldinger som vi har kunnet benytte for å gjøre systemet bedre.

Remy Monsen og Eirik Tenold

Bergen, 2. juni 2008

Sammendrag

Denne rapporten presenterer utviklingen av to forskjellige systemer til bruk i forbindelse med kortspillet bridge. Det første systemet er et web-basert administrasjonssystem for bridge-turneringer, mens det andre systemet er en bridgeklient som gjør det mulig å spille bridge via mobiltelefoner i et nettverk.

Det webbaserte turneringssystemet ble testet i reell bruk og ble aktivt brukt i en turnering i regi av Norsk Bridgeforbund. Nettsiden er tilgjengelig på http://eple.hib.no/bts/ (midlertidig).

En analyse av ytelsen til turneringssystem er også utført, samt en gjennomgang av noen sikkerhetskonsepter.

Mobilapplikasjonen gjorde det mulig å spille bridge på mobiltelefoner som kommuniserer via radiosignaler. Denne applikasjonen fungerer som et "proof of concept" og vil kunne gi grunnlag for eventuelle videre prosjekter.

De to delene ble utviklet med forskjellige utviklingsmetoder. Dette ble gjort for å høste erfaringer ved bruk av disse to metodene. Konklusjonen er basert på erfaringene til prosjektets to deltakere.

Hvert av disse systemene presenteres i en separat del av rapporten, med en felles avsluttende konklusjon.

Det blir også gitt en kort introduksjon til spillet bridge slik at leseren får en viss forståelse av grunnleggende konsepter.

Innholdsfortegnelse

Figuroversikt

1 Innledning

1.1 Bakgrunn for oppgaven

Bridge er et populært kortspill i Norge, og ellers i verden. Spillet spilles av et stort antall nordmenn, hvorav mange av disse er medlemmer i Norsk Bridgeforbund (NBF) [1]. Spillet er populært både som sosial aktivitet, og også som en seriøs idrett hvor det arrangeres flere store turneringer hvert år. De tilgjengelige støttesystemene som finnes i dag er ikke alltid helt ideelle for oppgaven, og det er både behov og ønsker om å lage bedre løsninger. Det gjelder for systemer innenfor mange områder, men man har i denne omgang valgt å se på to spesifikke, men veldig forskjellige behov. På den ene siden finner man behovet for å ha tilgjengelig et solid og pålitelig verktøy for å kunne administrere bridgeturneringer, spesielt større turneringer som krever mye administrasjon. I motsatt ende av spekteret finner man ønsket fra spillere om å kunne spille bridge når og hvor som helst, uten å være avhengig av å ha spillere og plass tilgjengelig. Det finnes muligheter for å spille bridge på Internett i dag, men dette krever en PC med Internettforbindelse. Dette ønsket kan lett innfris ved å benytte den elektroniske enheten som nesten alle har med seg til enhver tid i dagens moderne samfunn, mobiltelefonen. De fleste moderne mobiltelefoner har alle de egenskapene som behøves for å kunne tilby brukeren tilgang til bridge nesten uansett hvor man befinner seg i verden.

1.2 Oppgavens mål

Målsetningen med denne oppgaven er å utvikle to fullstendig separate interaktive tjenester for bridge. Hver av disse tjenestene vil bli behandlet som et eget delprosjekt.

1.2.1 Administrasjonssystem for bridgeturneringer

Første deloppgave har som formål å utvikle et webbasert administrasjonssystem for bridgeturneringer. Flere store bridgeturneringer i Norge går over en lengre tidsperiode, noen over perioder på mer enn seks måneder. Her foregår turneringene asynkront, det vil si at spillerne ikke samles på et sentralt sted for å spille kampene samtidig, men at kampene spilles innen en viss tidsfrist. For å gjøre jobben lettest mulig for turneringsarrangøren trengs det derfor et system hvor spillerne selv kan gjøre store deler av jobben direkte, og hvor all informasjon som angår turneringen kan samles, slik som trekninger, resultater, poeng og fakturagrunnlag.

1.2.2 Bridgespill for mobile klienter

Andre deloppgave har som målsetning å utvikle et bridgespill til bruk på mobile klienter. Ved å tilby et slikt spill vil man gi bridgespillere muligheten til å spille bridge når og hvor som helst, så lenge man har dekning eller andre spillere i nærheten. Man er derfor ikke avhengig av en datamaskin.

1.3 Oversikt over rapportens oppbygning

Denne rapporten er delt opp i fire hoveddeler. Første del av rapporten beskriver oppgaven i grove trekk, og fokuserer på de overordnende elementene som ikke direkte tilhører noen av deloppgavene. Del to og tre av rapporten tar for seg hver av de to deloppgavene, henholdsvis administrasjonssystemet for bridgeturneringer og bridge for mobile klienter. Hver av disse to

delene har innbyrdes samme struktur, og beskriver først problemstillingen, etterfulgt av en beskrivelse av løsningen, analyse og til slutt en oppsummering og konklusjon. Siste del av rapporten inneholder konklusjonen av arbeidet som en helhet, og presenterer også en sammenligning av de forskjellige arbeidsmetodikkene som ble benyttet i de to delprosjektene.

2 Hva er bridge?

Bridge er et kortspill for fire personer. To og to av disse danner et makkerpar som spiller mot hverandre. Spillerne fordeler seg rundt et bord, slik at hvert makkerpar sitter rett overfor hverandre, og da med en motstander på hver side. Når man skriver om bridge er det er vanlig å omtale spillerne som himmelretninger; nord, øst, sør og vest. Ett makkerpar blir da nord/sør, mens det andre blir øst/vest. Illustrasjonen under viser hvordan spillerne sitter fordelt rundt bordet.

Figur 2.1: Visuell fremstilling av et bridgebord med posisjoner

2.1 Hvordan spilles bridge

Bridge spilles med en vanlig kortstokk med 52 kort. Innenfor hver farge rangeres kortene fra 2 til ess, hvor 2 er lavest og ess er høyest. De fire høyeste kortene i hver farge (ess, konge, dame og knekt) blir kalt honnører. Alle kortene deles ut, 13 til hver spiller. Kortene skal holdes skjult, også for makkeren. Spillet deles inn i to deler, først meldedelen, og deretter selve spilldelen.

I meldedelen bestemmer hvert makkerpar seg for hvor mange stikk de tror de kan ta. Spillerne har ikke lov å kommunisere direkte med hverandre, men kan kun formidle informasjon ved hjelp av strukturerte meldinger. I denne delen av spillet er det derfor veldig viktig å kunne forstå omtrent hva makkeren sitter med på hånden basert på meldingene til vedkommende, og så komme med en fornuftig melding selv. Det er også viktig å kunne forstå motstanderens meldinger korrekt. Det er alltid giveren som starter meldedelen. Giver går på rundgang mellom spillerne. En viktig del av strategien i bridge ligger i meldedelen.

En melding består av et tall fra og med en til og med syv som forteller hvor mange stikk utover seks vedkommende tror at makkerparet klarer å ta, pluss en farge som forteller hva vedkommende vil ha som trumf. Det er også mulig å spille uten trumf. Man bruker da benevnelsen grand for å si at man ikke ønsker trumf. Meldingene er rangert etter verdi og

farge, hvor fargene rangeres grand, spar, hjerter, ruter og kløver. En melding på 1 spar er derfor høyere enn en melding på 1 ruter, mens en melding på 2 ruter er høyere enn 1 spar. Når man melder må meldingen alltid overgå den forrige meldingen. Som et alternativ kan man melde pass. Dersom ingen har meldt enda, brukes pass hvis man ikke har gode nok kort til å melde noe. Andre spesielle meldinger er dobling, som sier at man ikke tror motstanderen klarer meldingen de har avgitt. En doblet melding kan også redobles, som er et svar fra motstanderen om at de tror de klarer denne allikevel.

Meldinger blir gitt i stigende rekkefølge, helt til ingen ønsker å melde høyere, som blir signalisert ved at tre spillere på rad melder pass. Den høyeste meldingen blir da kalt kontrakten, og man går videre til selve spilldelen. Bridge har også et begrep som heter faresone. I hvert spill er enten ingen, nord/sør, øst/vest eller alle i faresonen. Hvem som er i faresonen varierer etter et fast mønster. Faresone har mindre betydning for selve spillet, men poengberegningen avhenger av dette. Man melder derfor gjerne litt annerledes dersom man er i faresonen, siden mer poeng står på spill.

Figuren under viser meldingsforløpet i et tenkt spill. Meldingene blir vist i et tabellformat og ligger fra venstre mot høyre i den rekkefølgen budet ble avgitt. I dette eksempelet er nord giver. Som vi ser er meldedelen over siden vest, nord og øst alle har meldt pass etter tur. Meldingen til sør blir dermed stående siden ingen av de andre spillerne velger å melde høyere, og kontrakten blir da 3 grand, altså ingen trumf. Dette betyr at nord/sør som fikk kontrakten må klare ni stikk. Klarer de dette får de poeng, hvis ikke får motstanderne poeng.

Figur 2.2: Meldedel i bridge

Når meldedelen er over, blir en av spillerne i makkerparet som vant kontrakten spillefører. Dette er den spilleren som kom med første meldingen i samme farge som kontrakten. Makkeren blir det som kalles blindemann, som betyr at han ikke selv får spille, men at han etter første utspill må legge ned kortene sine på bordet, synlig for alle. Deretter er det spilleføreren som bestemmer hvilke kort som skal spilles fra blindemannen.

Selve spilldelen i bridge minner mye om kortspillet “amerikaner”. Hvert makkerpar prøver å vinne så mange stikk som mulig. Ett stikk består av ett kort fra hver spiller. Spillet starter med at spilleren til venstre for spillefører spiller ut et kort. Spillet går så rundt bordet med klokken, der hver spiller legger på et kort. Den som vinner et stikk, spiller først ut til neste stikk.

Spillerne er nødt til å følge samme farge som første kortet i hvert stikk. Har ikke spilleren denne fargen på hånd, kan han legge på en hvilken som helst farge. Dersom trumf blir spilt vinner den som spilte den høyeste trumfen, ellers vinner spilleren som spilte ut det høyeste kortet i åpningsfargen. Trumf kan kun spilles dersom første kortet i stikket er en trumf, eller man er tom for fargen det spilles.

Illustrasjonen under viser et spill sett fra synsvinkelen til en utenforstående observatør som kan se alle kortene. Kortene er vist slik de ofte presenteres i bridgespalter i aviser. Først et fargesymbol, og så liste over alle kort spilleren har i denne fargen. Spillerne sitter rundt bordet på samme posisjoner som på illustrasjonen som viser bordfordelingen på forrige side. Illustrasjonen viser en sen fase i spillet, hvor det bare er noen få kort igjen å spille. Øst og sør har allerede lagt på kort i dette stikket, og det er nå vest sin tur.

Figur 2.3: Spilldelen i bridge

Etter at alle kortene er spilt beregnes poengene. Dersom makkerparet som har kontrakten klarte denne får de poeng, hvis ikke får motstanderparet poeng. Dersom man tar flere stikk en nødvendig, eller vinner en kontrakt som var doblet eller redoblet får man ekstra poeng. Poengene er også avhengig av hvilken farge som var trumf. Klarer man ikke kontrakten får motstanderne poeng basert på hvor mange stikk man manglet, igjen med bonus dersom kontrakten var doblet eller redoblet. Det har også betydning for poengene om makkerparet

som fikk kontrakten var i faresonen eller ikke. Dersom disse var i faresonen, kan poengene øke.

Når man spiller bridge, spiller man alltid flere spill. Ett spill er èn meldedel, med den tilhørende spilldelen, og tar vanligvis fem til ti minutter for erfarne spillere. I sosiale anledninger spiller man ofte så mange spill som det blir tid til, mens i turneringer er det et forhåndsbestemt antall spill.

For mer informasjon om bridge henvises det til en lærebok i bridge, for eksempel [2].

2.2 Bridge i Norge

Bridge er et meget populært kortspill i Norge. Statistikk fra Norsk Bridgeforbunds (NBF) database viser at det er ca 30 000 registrerte medlemmer i landet. Av disse er ca 10 000 med i en av de 454 bridgeklubbene i Norge. Utenom disse finnes det et stort antall uregistrerte spillere, hovedsakelig rekreasjonsspillere som ikke deltar i turneringer og lignende. Norsk Bridgeforbund er en av seks organisasjoner som er med i Norsk Tankesportforbund, som ble etablert i november 2005.

Bridge er et kortspill som passer for personer i alle aldre, og man kan finne spillere helt fra barneskolealder og opp til 80-årene. Man finner regelmessige bridgespalter i en god del av nasjonens aviser og magasiner.

Årlig holdes det et stort antall turneringer både nasjonalt og internasjonalt. Norge hevder seg sterkt internasjonalt, og i 2007 ble det norske laget verdensmestere [3].

2.3 Bridge på Internett

Som mange andre ting, har også bridge funnet veien til Internett. Her finner man alt fra regler og bridgespalter, til spilling av bridge. En av de mest populære tjenestene for bridge på Internett i dag er Bridge Base Online (BBO), som tilbyr spilling av bridge via Internett. Dette tilbys gjennom et program man kan laste ned fra deres nettsider [4]. Selve programmet er gratis, og det er koster heller ikke noe å spille bridge. På BBO spiller tusenvis av spillere hver dag fra alle land. Man har også muligheten til å være tilskuer i stedet for å spille selv. Sistnevnte er spesielt populært når viktige kamper spilles, og en enkelt kamp kan tiltrekke seg tusenvis av tilskuere. Det finnes også andre populære tjenester på nettet, slik som OKbridge [5] og Topbridge [6], men spesielt i Norge er BBO den desidert mest populære.

3 Teknologi

Som beskrevet i forrige seksjon, angående rapportens oppbygning, ble det bestemt å dele rapporten inn i seksjoner. Hver seksjon vil derfor ha ett eget kapittel om teknologi, der komponentene brukt i den delen av oppgaven blir beskrevet. Men det er også en del overordnede systemer som understøtter alle delprosjektene. Disse vil bli beskrevet nærmere her.

3.1 Mantis

Mantis [7] ble valgt som feilhåndteringssystem for prosjektet. Mantis er fritt tilgjengelig, og lisensiert under GPL-lisensen[1]. Ved å bruke et slikt feilhåndteringssystem, gjør man det lett å holde orden på feil som må fikses. Mantis gir muligheten til å registrere alle feil/problemer i en database, slik at man har full oversikt over disse. Feil kan så tildeles en utvikler, som har ansvaret for å følge opp og rette disse. Systemet tillater å registrere mer informasjon om en feil underveis, slik at det er veldig lett å få en oversikt over hvordan status er med hvert av de rapporterte problemene. Her har man også muligheten til å angi hvor alvorlig hver feil er, og prioriteringen på rettingen, slik at man sikrer seg at feil blir prioritert i en fornuftig rekkefølge.

I tillegg til å håndtere feil, kan systemet også registrere ønsker om produktfunksjoner og forbedringer. Ved å ha disse registrert et sentralt sted, kan nye funksjoner lett prioriteres mot retting av feil.

Figur 3.1: Skjermbilde fra Mantis

3.2 WordPress

WordPress [8] er et system for blogging. For de som er ukjent med dette begrepet, så er dette rett og slett en dagbok på nettet, som vanligvis alle kan lese. Leserne kan også skrive

[1] Se avsnittet om Kildekodelisens senere i kapittelet.

kommentarer til innleggene i denne dagboken. Kun eierne av dagboken kan legge ut nye innlegg. WordPress er gratis tilgjengelig, og baserer seg på GPL-lisensen.

WordPress ble brukt som en prosjektdagbok for utviklingen. Her ble informasjonen lagt ut etter hvert som prosjektet skred fremover. Det ble også lagt ut filer, slik som relevante dokumenter som databaseskjemaer, dokumentmaler og annen nyttig informasjon. Dette ble også brukt som en sentral portal for prosjektet, og det var lagt opp lenker herifra til alle de andre nettbaserte verktøyene som ble benyttet av prosjektet, slik som MediaWiki, dotProject og Mantis.

Figur 3.2: Skjermbilde fra WordPress

3.3 dotProject

For prosjektplanlegging ble programmet dotProject [9] valgt. Dette er gratis tilgjengelig under GPL-lisensen.

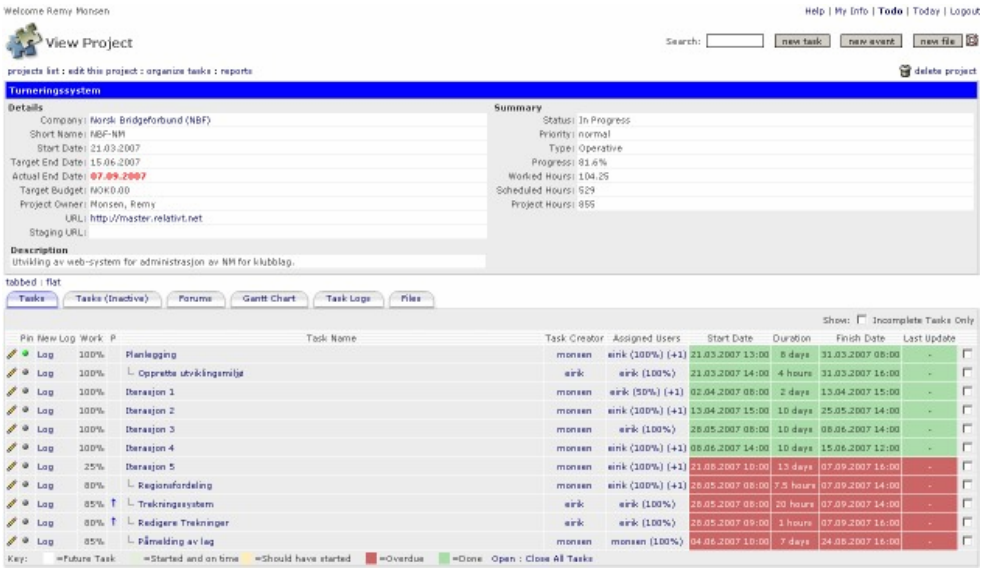
Her ble dotProject brukt for å få både en overordnet og detaljert oversikt over prosjektene. Programvaren gir mulighet til å dele prosjektene inn i oppgaver, og videre inn i deloppgaver. Hver oppgave/deloppgave kan da tilordnes bl.a. forventet tidsforbruk, planlagt start- og slutt-dato, personellressurser og detaljert beskrivelse av oppgaven.

Basert på disse opplysningene kan programvaren lage oversiktlige lister og grafer, slik at man visuelt kan se hvordan fremdriften er i prosjektet, og om ting har brukt mer eller mindre tid enn planlagt, og hvilke oppgaver som venter. Programvaren bruker også fargekoder for blant annet å vise hvilke oppgaver som man allerede burde ha begynt på, og hvilke oppgaver som burde vært ferdig.

Denne applikasjonen komplementerer godt Mantis. I dotProject skapes et mer helhetlig bilde av hele prosessen, og man planlegger på et mer overordnet nivå. Mantis lar en beskrive ting på detaljnivå, slik som enkeltfunksjoner. I dotProject finner man for eksempel en deloppgave som heter “Innlogging”, som omhandler implementering av innloggingssystem i BTS, men uten å gå i detaljer på hvordan dette skal implementeres rent praktisk. I Mantis finnes detaljer, slik som “send nytt passord virker ikke”, og “må opprette side for å skifte passord”.

Her ligger også detaljerte beskrivelser om hva som er det eventuelle problemet, og hva som må gjøres.

Etter at et prosjekt er fullført, kan da dotProject også benyttes til å hente ut grafer og data, slik at estimatene og planleggingsprosessen kan analysers, og avvik i prosessen identifiseres.



Figur 3.3: Skjermbilde fra dotProject

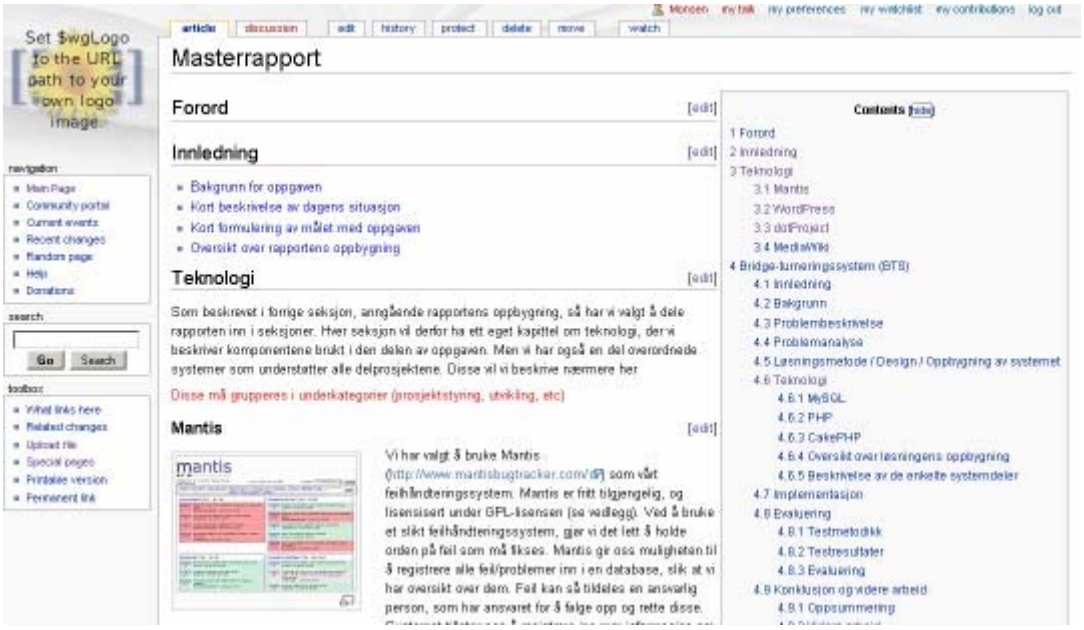
3.4 MediaWiki

MediaWiki [10] er fritt tilgjengelig under GPL-lisensen. MediaWiki er hovedsakelig programvare for å drive et leksikon på nett, og ble opprinnelig utviklet for å drive Wikipedia [11]. Ideen med en wiki er at hvem som helst skal kunne oppdatere informasjonen som ligger her. Tanken er at når brukerne selv kan oppdatere leksikonet sitt, så blir feil fortere rettet, og nye artikler kommer fortere inn.

Her ble det valgt å sette opp en wiki for å skrive selve masterrapporten. Dette gir den fordelen at masterrapporten er tilgjengelig online, og alle prosjektdeltagerne kan gå inn å redigere denne direkte, uten å være avhengig av å sende dokumenter frem og tilbake, og passe på å holde disse oppdatert til enhver tid. Veilederen for prosjektet fikk også tilgang her, og kunne komme med kommentarer til oppgaven fortløpende uten å måtte få tilsendt siste versjon av dokumentet hele tiden.

Som nevnt over er tankegangen at alle skal kunne redigere fritt, men programvaren har også muligheten til å begrense hvem som kan gjøre endringer, slik at ikke utenforstående får tilgang til å endre på rapporten.

MediaWiki lagrer også alle endringer som blir gjort, slik at ved behov kan man til gå tilbake til en tidligere versjon av dokumentet, og det er også lett å se forskjellen mellom forskjellige versjoner, slik at det er mulig å se hva som har blitt endret, og når dette er gjort.



Figur 3.4: Skjermbilde fra MediaWiki

3.5 Subversion

Subversion [12] er et versjonskontrollsystem som kjører på en tjener. Ved å benytte seg av en slik tjeneste forenkler man situasjonen betraktelig når flere utviklere jobber på samme kode. Systemet fungerer ved at Subversion-tjeneren fungerer som en sentral lagringsplass for koden, og den enkelte utvikler kjører en lokal Subversion-klient for å synkronisere sin lokale kode mot tjeneren. Tjeneren har rutiner som automatisk fletter inn endringer, dersom disse ikke skaper konflikter. Dersom konflikter oppstår, blir utvikleren bedt om å gjøre de relevante endringene for å fjerne konfliktene.

Dette sørger for at utviklerne hele tiden jobber på samme kode, og minsker risikoen for å støte på store inkompatibiliteter når de skal flette sammen koden sin. Systemer som Subversion fungerer definitivt best når utviklerne synkroniserer koden sin ofte mot tjeneren, noe som reduserer risikoen for at man manuelt må håndtere en konflikt.

Subversion lagrer også tidligere versjoner av endrede filer. Skulle man ha behov for dette, kan man da gå tilbake til en tidligere versjon av koden. Programvaren støtter også forgreninger, slik at når for eksempel versjon 2.0 av systemet skal utvikles, lager man en forgrening, slik at man fortsetter å utvikle versjon 1 langs en gren (for bugfiksing, sikkerhetsoppdateringer, etc.), mens utviklingen av versjon 2 foregår langs den andre grenen. Begge grenene tok utgangspunkt i den samme koden, men utvikler seg videre i hver sin retning.

Subversion lar også utviklerne skrive kommentarer hver gang de sender oppdatert kode til tjeneren, og på denne måten får man også en god og enkel endringslogg. Bruken av et versjonskontrollsystem medfører også at man har en sentral kopi av koden liggende, slik at det er lett å ta sikkerhetskopi. Man trenger ikke være avhengig av den enkelte utviklers håndtering av dette.

4 Kildekodelisens

Et hvert program som skal distribueres eller publiseres bør ha en lisens knyttet til seg. En lisens sier hva som kan gjøres med et program og hvilke begrensninger som gjelder for brukerne. Alle datamaskinprogrammer som skapes, dekkes automatisk av åndsverkloven [13]. Denne tildeler opphavsretten til den som skaper åndsverket. Denne retten gir opphavsmannen enerett til å råde over åndsverket. Opphavsmannen har mulighet til å gjøre det tilgjengelig for allmennheten hvis vedkomne ønsker dette. En lisens vil gi opphavsmannen rett til å legge føringer for hvordan åndsverket kan benyttes og hvilke rettigheter brukeren eventuelt får.

Produktene som er utviklet i forbindelse med denne masteroppgaven, ble av utviklerne ønsket frigitt under en lisens som har en åpen holdning til hvordan programkoden kan benyttes av andre og til videre utvikling. Utviklerne valgte derfor å gjøre disse tilgjengelig under GPL-lisensen. I dette kapittelet vil det bli gitt en gjennomgang av kjente åpne lisenser og hvordan disse kan brukes av andre.

4.1 Åpen versus lukket kildekode

Begrepet "åpen kildekode" er blitt en definisjon av hva som kreves av en lisens i tillegg til å få tilgang til kildekoden. Open Source Initiative (OSI) er en organisasjon som arbeider for å fremme programvare basert på åpen kildekode. De har en definisjon av åpen kildekode som setter en standard for hva som kreves for å kunne kalle et program for åpen kildekode [14]. Denne definisjonen krever blant annet at kildekoden skal følge med programmet, at kildekoden skal være fri til å distribueres videre og at lisensen skal følge kildekoden. Full oversikt over definisjonen og kravene finnes i filene som ligger vedlagt rapporten. Eksempler på programvare som følger denne definisjonen er nettleseren Mozilla Firefox [15], OpenOffice.org [16] og operativsystemet GNU[1]/Linux [17].

Lukket kildekode er navnet på proprietær programkode som ikke følger den ovennevnte definisjon. Vanligvis har disse programmene en restriktiv brukerlisens som begrenser hvilke muligheter brukeren har til å modifisere programvaren. Eksempler på slik programvare er Microsoft Windows [18], Adobe Photoshop [19] og Apple iTunes [20].

Produsentenes forretningsmodell spiller en veldig stor rolle for hvilken lisenstype som blir valgt for sine produkter. For noen bedrifter vil en åpen lisens for programvare være en mulighet for å øke markedsandeler. Sun Microsystems [21] er et eksempel på en slik leverandør; de gir ut mye programvare gratis og med åpne lisenser. OpenOffice.org og Java [22] er to produkter de gir ut gratis i håp om at flere skal konvertere til deres maskinvareportefølje.

4.2 Åpne kildekodelisenser

Det finnes flere lisenser som faller inn under OSIs definisjon av åpen kildekode og det vil bli gjort en enkel gjennomgang av fire av de mest populære.

[1] GNU er et rekursivt akronym for "GNU's Not Unix"

4.2.1 BSD Licence og MIT Licence

BSD- [23] og MIT-lisensene [24] er de mest kjente lisensene som er tilnærmet lik prinsippet "offentlig eiendom". Juridisk vil dette si at råderetten over verket har falt bort. I praksis vil dette bety at det er fritt frem for å gjøre hva man vil med åndsverket. Samtidig frasier opphavsmennene seg alt ansvar for skade som kan oppstå i forbindelse med bruk av programmet.

Disse to lisensene setter ikke noe krav til å videreføre lisensen til nye verk som bygger på det opprinnelige verket.

4.2.2 GNU General Public License (GPL)

GPL-lisensen [25] er en lisens produsert av GNU-prosjektet. Denne lisensen er del av en familie lisenser som populært kalles "copyleft"-lisenser. Copyleft er et konsept som sikrer at modifikasjoner i applikasjonen, og dens mulige redistribusjon, vil bli gjort tilgjengelig under samme betingelser som den opprinnelige versjonen. Dette kan best illustreres med et eksempel: Person A har utviklet et program og gjør dette tilgjengelig under en GPL-lisens. Person B laster ned denne programvaren og gjør endringer i koden slik at den får ekstra funksjonalitet. Person B vil så publisere sin versjon av programmet, men det må lisensieres med samme lisens som person A brukte. Dette sikrer at programvaren vil forbli åpen og tilgjengelig selv etter flere "generasjoner" med modifikasjoner.

Den som har mottatt et program som er GPL-lisensiert har fått en del rettigheter og noen forpliktelser. Ved å bruke GPL-lisensen gir man mottakeren en rett til å få kildekoden til programvaren, gjøre modifikasjoner på programvaren og fritt distribuere programvaren og/eller modifikasjonen videre etter eget ønske. Det er derimot et krav at videre distribusjon ikke skal begrense rettighetene og pliktene som opprinnelig er gitt. Dette medfører at også denne programvaren må lisensieres under samme lisens som det opprinnelige verk.

Det som er viktig å merke seg er at ovennevnte krav kun gjelder distribusjon av selve programvaren. Data som produseres av programvaren (filer, utdata, osv) vil ikke bli dekket av GPL. For eksempel vil en tekstbehandler som er GPL-lisensiert, ikke produsere tekstfiler som også dekkes av GPL-lisensiering. Hvis selve programmet (tekstbehandleren) distribueres, dekkes denne av GPL-lisensen. Et annet eksempel på dette vil være et webbasert program som brukes på en hjemmeside. De besøkende har ikke krav på å se selve kildekoden til websiden med mindre selve programvaren gjøres tilgjengelig for nedlasting.

At et program er GPL-lisensiert betyr ikke nødvendigvis at programmet er gratis. Det er fullt mulig å selge et program med denne lisensen; men kjøperen har da krav på å få kildekoden til programmet og kan distribuere denne videre uten økonomisk kompensasjon.

Kommersiell bruk av GPL-lisensiert programvare er fullt mulig. Men videredistribusjon faller under samme vilkår som ved ikke-kommersiell bruk. Det er derimot mulig for en opphavsperson til å frigi koden under flere forskjellige lisenser. Noen av disse lisensene kan være "proprietære" og brukes i salg til bedrifter. Trolltechs Qt [26] frigis under to forskjellige lisenser; GPL-lisens for "Open source" (åpen kildekode) utviklere og en kommersiell lisens til bedrifter som ikke ønsker å frigi kildekoden til sine kunder.

4.2.3 GNU Lesser General Public License (LGPL)

I noen tilfeller vil det være ønskelig å frigi et bibliotek under en GPL-lisens, men samtidig tillate at biblioteket kan benyttes i proprietær programvare. LGPL [27] kan i disse tilfeller

benyttes. Denne lisensen gir produsenten av proprietær programvare mulighet til å inkludere en umodifisert versjon av et LGPL-lisensiert bibliotek under produsentens ønskede lisenstype. Kravet om videreføring av GPL-lisensen er ikke tilstedet i dette tilfellet.

Hvis et modifisert LGPL-lisensiert bibliotek blir inkludert i en proprietær programpakke stilles det ekstra krav til lisensen som brukes for programpakken; brukeren må få lov til å bruke "reverse engineering" på koden for å kunne utforske modifikasjonene så lenge dette er til eget bruk.

OpenOffice.org er et eksempel på en programpakke som benytter seg av en LGPL-lisens [28].

Del 1: Bridge Turneringssystem

5 Innledning

Bridge turneringssystem (BTS) er den første av de to delene i prosjektet. BTS er et web-basert administrasjonssystem for bridgeturneringer. Norsk Bridgeforbunds (NBF) turnering NM for klubblag er primærmålet for dette systemet.

NM for klubblag er en stor bridgeturnering som blir avholdt hvert år, med deltagere fra hele landet. Denne turneringen arrangeres som en cup. Lagene som møtes arranger selv møtested og tid for kampen. Selve turneringen er desentralisert, med unntak av finalen, som holdes i fellesskap.

5.1 Bakgrunn for oppgaven

NBF er organisasjonen som står bak organisert bridge i Norge, og avholder flere forskjellige typer landsdekkende turneringer hvert år. Alle disse turneringene medfører mye arbeid for forbundet i forbindelse med administrering av disse.

NBF har derfor behov for et system som kan forenkle administrasjonen av sine turneringer.

5.2 Dagens situasjon

Før turneringen startet måtte påmeldingene håndteres. Disse ble fakset til NBF, eller sendt som brev eller e-post. NBF måtte så behandle disse, og manuelt registrere hver påmelding inn i sine interne systemer.

Ved begynnelsen av hver runde i turneringen måtte det så foretas en trekning. Denne ble utført ved at det ble laget til trekningslapper i Excel, som ble skrevet ut, og trukket. Hver kamp ble så registrert i interne systemer, og ble så manuelt kopiert til en statisk nettside på NBF sin server.

Etter hvert som hver kamp ble spilt, sendte vinnerlaget inn et resultatregistreringsskjema via telefaks, e-post eller vanlig brevpost. Dette ble så manuelt registrert i NBFs systemer, og etter alle kampene var spilt ble resultatlistene manuelt kopiert over til en statisk nettside igjen. Vinnerne av hver kamp ble så gjort klar for trekking til neste runde.

Spillerne på vinnerlagene fra hver runde får også utdelt et antall forbundspoeng. Disse varierer med hvilken runde man er i. Disse poengene følger hver spiller gjennom livet, og gir en viss indikasjon på hvor god bridgespiller man er.

I tillegg til det rent bridgetekniske rundt turneringen, blir også klubbene fakturert for hver runde hvert av lagene deres deltok i. Denne rundeavgiften kan variere fra runde til runde. Lag blir også fakturert ekstra for spillere som ikke har en såkalt årslisens.

Alle disse prosessene som nevnt over krever en god del manuell behandling, og derfor mye arbeid fra NBFs side slik systemet er i dag. Et system som kan lette dette arbeidet er derfor svært ønskelig.

5.3 Målet med oppgaven

Målsetningen med denne oppgaven var å lage et helhetlig system som ikke bare ville lette administrasjonsarbeidet for NBF i forbindelse med avvikling av turneringer, men som også vil gjøre det lettere for deltagerne i turneringen. Servicen for andre interesserte som er ute etter informasjon om trekninger og resultater vil også forbedres. Hovedfokuset var på

avviklingen av NM for klubblag som blir avholdt hvert år, men systemet bør gjerne også kunne håndtere andre turneringer.

Det ble derfor fokusert på å utvikle et system hvor det enkelte lag selv kan registrere seg direkte i systemet, og hvor lagene selv kan registrere resultater etter spilt kamp. Dette vil medføre at NBF sentralt slipper mye registrering, og det blir utviklet et system som er dynamisk, der resultatene er tilgjengelig for interesserte fortløpende etter hvert som de blir registrert. Systemet skal også la turneringsansvarlige enkelt ta ut relevante rapporter, slik som hvor mange forbundspoeng hver spiller har opptjent i turneringen, og hvor mye hver klubb skal faktureres for sin deltagelse.

6 Bakgrunn

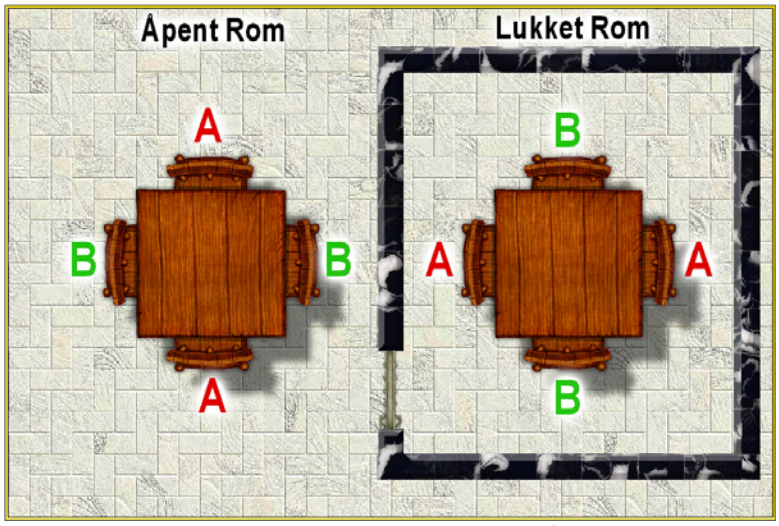
6.1 NM for klubblag

NM for klubblag er en bridgeturnering som blir avholdt hvert år, i perioden september til mai.

Første del av turneringen spilles som utslagsrunder (cup). Det vil si at to lag møter hverandre og vinneren går videre til neste runde. Når det gjenstår åtte lag, spiller disse en finale. Den arrangeres som en serieturnering, hvor alle møter alle.

6.1.1 Kampene

En kamp gjennomføres ved at det spilles på to bord samtidig. Ett bord er åpent for tilskuere, det andre er i et lukket rom. Samme spill spilles ved begge bord, men med motsatte posisjoner. Dette blir gjort for å gjøre kampen rettferdig, og eliminere tilfeldigheter ved kortfordeling. Ved halvtid sammenlignes resultatet og bortelagets spillere skifter bord. Det er kun tillatt å bytte spillere ved halvtid. Figur 6.1 viser hvordan spillerne fordeles ved bordene. Spillerne merket A er hjemmelaget, mens B er bortelag.



Figur 6.1: Fordeling av spillerne på bord

Det arrangerende lag stiller spillested. Lagene forplikter seg til å arrangere og spille kampen innen den oppgitte tidsfristen for hver runde. Det er kun i finalen at alle lagene møtes på et felles sted.

6.1.2 Arrangør av finalen

En relativt ny ordning er at arrangøren av finalen har rett til å sende ett lag direkte til finalen. Det er derfor kun syv plasser i finalen som de deltagende lagene må kjempe om. Hvorvidt dette er en ordning som kommer til å fortsette er ukjent for utviklerne, men det er en faktor systemet må håndtere.

6.1.3 Beste tapere

Første runde skiller seg fra de andre rundene ved at noen tapende lag blir med til runde to for å få et passende antall lag. Hvor mange lag som går videre, avhenger av antallet lag som deltar i turneringen, men under normale norske forhold vil runde to normalt inneholde 224

lag (runde to må inneholde nøyaktig $7*2^k$ {k>0} lag for at syv lag skal gå videre til finalen). Disse ekstra lagene blir valgt etter prinsippet "beste taper". Taperne rangeres etter hvor mye de har tapt. De som har tapt minst rangeres først. Det vil si at en del av lagene som tapte kampen i første runde allikevel går videre til andre runde, og får dermed en ekstra sjanse. Fra og med runde to er det vanlig cup.

6.1.4 Walkover

Som i de fleste turneringer, så tildeles en walkover dersom motstanderen ikke kan møte. Det vil at laget går direkte videre til neste runde. I NM for klubblag er det også andre muligheter for walkover. Dersom det er et ujevnt antall lag i første runde, vil også ett av lagene få walk-over til andre runde. Det er vinnerlaget fra forrige år som vanligvis får tildelt dette. Tidligere var det vanlig at i stedet for å bruke prinsippet med beste tapere som beskrevet over, fikk en del lag walkover til runde to. Systemet med beste tapere stammer fra bare få år tilbake.

6.1.5 Rundeavgifter

Klubben faktureres for hvor mange runder hvert av klubbens lag har deltatt i. Prisen for hver runde øker frem mot finalen. Totalprisen for en klubb vil da variere alt etter hvor mange lag klubben har i turneringen, og hvor godt disse lagene gjorde det. Dersom spillerne ikke har årslisens, må de også betale en tilleggsavgift for hver runde de deltar i. I dag gjelder dette ikke første runde.

6.1.6 Forbundspoeng

Deltagerne på det vinnende laget vil få tildelt forbundspoeng. Forbundspoeng er en personlig score for hver spiller, som samler seg opp etter hvert som vedkommende deltar i turneringer. Antallet forbundspoeng varier med hvilken runde i turneringen de er i. Dersom en spiller bare har spilt i en halvrunde, vil vedkommende da bare få halvparten av forbundspoengene.

6.1.7 Resultatrapportering

Siden lagene selv arrangerer og spiller sine kamper, må de selv rapportere resultatene sine tilbake til NBF. Lagene rapporterer da hvilke spillere som deltok, om spillerne har lisens eller ikke og hva resultatet av kampen ble.

7 Utviklingsmetode

Som beskrevet i innledningen til rapporten, er oppgaven todelt, og det ble derfor bestemt at forskjellige utviklingsmetoder skulle benyttes for de forskjellige delene. Dette ble valgt for å opparbeide erfaringer med de forskjellige metodene. Unified Process (UP) ble derfor valgt som utviklingsmetode for arbeidet med denne delen av oppgaven.

UP finnes i et stort antall varianter, som blant annet Rational Unified Process (RUP), som er utviklet av Rational Software, og nå tilgjengelig fra IBM [29]. I relasjon til denne oppgaven, så betyr UP den versjonen av UP [30] som ble undervist i faget LOD058[1] ved Høgskolen i Bergen [31] i 2005. Siden hensikten her ikke er en dyptgående analyse av UP, bestemte utviklerne seg for å ikke se på noen av de andre versjonene av UP, med unntak av noen korte blikk på RUP som er en forholdsvis kjent metodikk i dag.

7.1 Oversikt

UP er en utviklingsmetode som legger stor vekt på dokumentasjon og planlegging av systemet før den faktiske implementeringen finner sted. Man planlegger først hvilke oppgaver systemet skal utføre, og lager detaljerte beskrivelser på hvordan hver oppgave skal utføres, ofte med fokus på hvordan oppgaven skal utføres av systemets operatør. Disse beskrivelsene av enkeltoppgaver blir benevnt som brukstilfeller, og inneholder detaljert informasjon om hvordan oppgaven utføres steg for steg, eventuelle variasjonsmuligheter og unntakstilfeller, og hvilke andre brukstilfeller dette brukstilfellet avhenger av. Det blir også skrevet en serie med andre dokumenter, slik som en detaljert kravspesifikasjon, begreps- og risikoliste.

UP er en iterativ utviklingsmetode som følger fossefallsmodellen. For hver iterasjon oppdateres dokumentene med de detaljene som skal implementeres i denne iterasjonen, og ved slutten av iterasjonen har man ett fungerende produkt, som inneholder den funksjonaliteten man har planlagt så langt.

7.2 Dokumentasjon

Når en utvikler etter UP-prinsippene er god dokumentasjon veldig viktig. I motsetning til XP (eXtreme Programming) [2], hvor det er forventet å ha en høy grad av verbal kommunikasjon innad i teamet, vektlegger UP skriftlig dokumentasjon. UP fokuserer på at dokumentasjonen skal være god nok til at en utvikler hovedsakelig skal kunne lese seg frem til det han trenger, slik at teamet drar i samme retning uten å være avhengig av konstant kommunikasjon mellom utviklerne på samme måte som i XP. Dokumentene og diagrammene man produserer under UP er kjent som artefakter.

Under UP produseres derfor alle artefaktene i hver iterasjon før man begynner på selve implementeringen. Følgende liste inneholder de vanligste artefaktene man produserer under UP:

Domenemodell. Dette er et Unified Modeling Language (UML)-diagram som beskriver den helhetlige oppbyggingen av systemet.

[1] Fagbeskrivelsen for dette faget er ikke lengre tilgjengelig på nettsidene til Høgskolen i Bergen. Disse er tilgjengelig ved henvendelse til Høgskolen i Bergen.

[2] XP er gjennomgått i nærmere detalj i rapportens del 2.

Visjonsdokument. Dette dokumentet beskriver systemets overordnede plass i organisasjonen, hvilke arbeidsoppgaver systemet er tiltenkt å håndtere, og behov på et høyt nivå.

Kravspesifikasjon. Dette dokumentet beskriver de detaljerte kravene til systemet. I dette dokumentet finner man blant annet brukstilfellene, som er detaljerte beskrivelser av hvordan hver enkelt deloppgave i systemet skal utføres. Brukstilfellene inneholder følgende komponenter:

- Detaljert beskrivelse av hvordan oppgaven skal gjennomføres.
- Detaljert beskrivelse av alternative måter oppgaven kan gjennomføres, og eventuelle unntak.
- Oppgavens forventede ressurskrav til systemet, på en relativ skala fra lav til høy.
- Oppgavens sammenheng med andre oppgaver i systemet.
- Oppgavens forhold til interessenter og aktører.
- Oppgavens prioritet i utviklingssyklusen.
- UML-diagram over oppgavens interaksjon med aktører og andre oppgaver.

Videre så inneholder kravspesifikasjonen en overordnet oversikt over systemet, og også detaljerte ikke-funksjonelle krav, slik som krav til ytelse, brukervennlighet, etc.

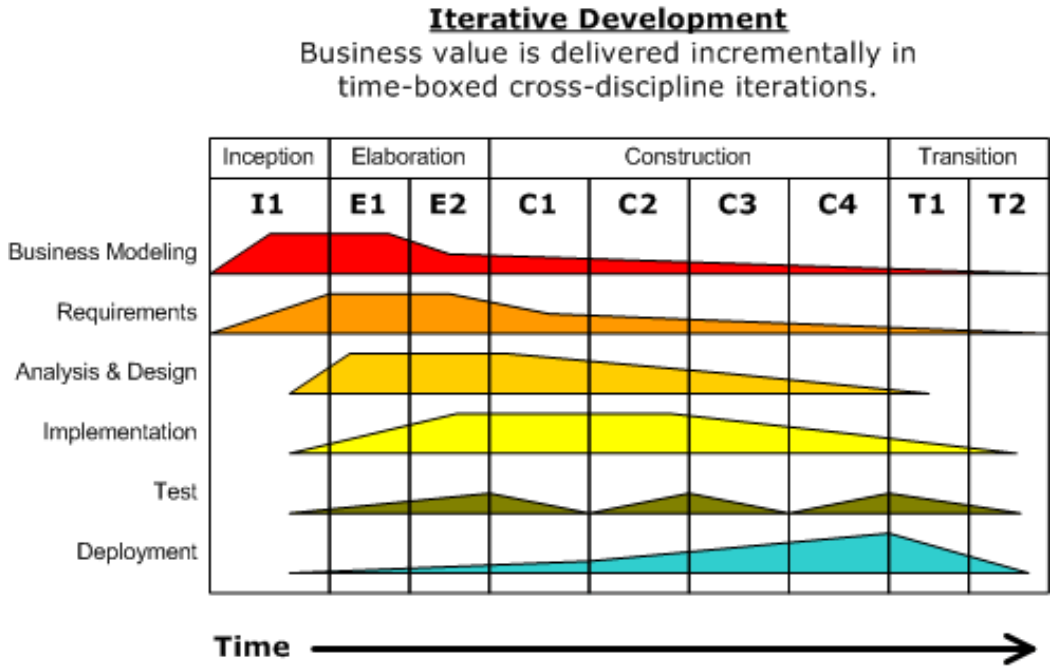
Risikoliste. Dette er et dokument som beskriver forutsette risikoer som man kan møte under utviklingen.

Dette kan for eksempel være en underleverandør som muligens ikke kan levere komponentene innen tidsfristen, eller at den planlagte reorganiseringen av bedriften kan medføre at kravene til systemet endrer seg.

Det finnes en hel del andre artefakter som kan produseres. Felles for de fleste artefaktene innen UP er at de er detaljerte, og inneholder mange UML-diagrammer for å beskrive systemet.

7.3 Iterativ utvikling

Under UP foregår utviklingen iterativt etter fossefallsmodellen. Før prosjektet startes, bestemmes lengden av hver iterasjon, ofte en til fire uker, avhengig av totallengden for prosjektet.



Figur 7.1: Fordeling av fokus på oppgaver i de forskjellige iterasjonene i et UP-drevet prosjekt

I hver iterasjon begynner man med å oppdatere de forskjellige artefaktene, som for eksempel å legge til i kravspesifikasjonen de brukstilfellene som man planlegger å. Man endrer også på artefaktene der dette trengs, for eksempel dersom kravene har endret seg. Ved å oppdatere artefaktene for hver iterasjon forsikrer man seg om at disse er fullstendig oppdaterte og relevante til enhver tid.

Etter at dokumentene er oppdaterte begynnes implementeringen for gjeldende iterasjon. Her implementeres de brukstilfellene som er planlagt. Ved slutten av hver iterasjon er målet å ha et fungerende og testbart system, som tilbyr den funksjonaliteten som er blitt implementert så langt. Systemet blir dermed bygget opp bit for bit, og får mer og mer funksjonalitet etter hvert som prosjektet skrider frem.

Figur 7.1 viser hvordan tidsfordelingen til de enkelte oppgavene varierer med hvor langt prosjektet er kommet. For eksempel, så er det mye fokus på modellering og design i starten av et prosjekt, men siden flytter dette fokuset seg til implementering, og til slutt til utrulling. Figuren er hentet fra [32].

7.4 Fordeler med Unified Process

UP er en veldig strukturert utviklingsprosess. Ved å legge fokus på produksjon av artefakter før implementeringen finner sted, og oppdatering av disse i begynnelsen av hver iterasjon, sikrer man at prosjektet er godt dokumentert til enhver tid. Herunder også at dokumentene beskriver den faktiske tilstanden til systemet. Planleggingen sørger for at det til enhver tid vites hva som skal implementeres i den gjeldende iterasjonen, og de detaljerte brukstilfellene

sørger for at det er lett for utviklerne å ha en felles forståelse av hvordan systemet skal utvikles. Artefaktene er også nyttige dokumenter når systemet eventuelt skal videreutvikles på et senere tidspunkt.

7.5 Ulemper med Unified Process

UP bruker mye tid på dokumentasjon, og den krever at utviklerne har en forståelse for hvordan prosessen fungerer. Artefaktene som produseres blir også fort utdatert, hvis det gjøres systemendringer uten at disse oppdateres. Dette er en virkelighet som oppleves i de fleste systemers vedlikeholdssyklus.

7.6 Utviklernes erfaringer med Unified Process

Da denne delen av prosjektet ble startet, ble utviklingsmetoden fulgt og de vanligste UP-artefaktene ble produsert. Det ble fort oppdaget at dette var en tidkrevende prosess, men et fullstendig sett med UP-dokumenter ble etter hvert produsert. Ett av de første "problemene" som oppstod, var at artefaktene var til veldig liten nytte i forholdt til tiden som ble brukt på å produsere dem. Dette skyldes hovedsakelig at med et så lite utviklingsteam, så hadde utviklerne allerede god nok kontroll på hva som måtte gjøres uten å måtte konsultere artefaktene. Det ble også erfart at det er vanskelig å planlegge brukstilfellene godt nok, slik at man endte ofte opp med å måtte implementere dem annerledes enn man først hadde planlagt. All planleggingen i forkant av selve implementeringen medførte også at den resulterende domenemodellen ble unødvendig kompleks, noe som førte til at også databaseskjemaet var mer komplekst enn strengt nødvendig.

Prosjektet ble også en del forsinket i forhold til opprinnelig tidsplan. Dette medførte at implementeringen av en del funksjoner begynte å haste. Teamet opplevde da at det var svært vanskelig å holde seg korrekt til UP. NBF var avhengig av å ha systemet i drift til en bestemt dato, så det ble svært viktig å ha de rette funksjonene på plass til rett tid. Grunnet tidspresset, ble det også nødvendig å implementere delvis funksjonalitet for enkelte brukstilfeller, slik at de kunne brukes nå, og heller implementere resterende senere. Alt dette medførte at teamet hadde problemer med å klare å forholde seg korrekt til UP. Det ble ikke tid til å oppdatere dokumentene skikkelig, og det ble veldig vanskelig å planlegge hver iterasjon skikkelig på forhånd. UP ble derfor opplevd som en prosess som er vanskelig å forholde seg korrekt til når man opplever stort tidspress.

I dette tilfelle så var heller ikke utsettelse av tidsfristen et alternativ, siden det var enten å ta systemet i bruk nå, eller neste høst. Det ble derfor bestemt i samråd med NBF, å ta systemet i bruk nå, siden dette ville gi utviklingsteamet flere verdifulle erfaringer i forhold til det å utvikle et system som man ikke ville få testet i reell drift i full skala. Denne avgjørelsen må teamet konkludere med var den beste for dette prosjektet i sin helhet. I tillegg medførte beslutningen at ekstra erfaringer ble gjort i forbindelse med tidspress.

7.7 Konklusjon

UP er nok en utviklingsmetode som er egnet for større prosjekter enn to utviklere. UP fungerer heller ikke spesielt bra når prosjektet kommer i tidsnød, og trenger personellressursene til å drive utvikling for i det hele tatt få ting klart i tide. I større prosjekter vil nok tidsnød bli sett på med andre øyne, siden det kan være vanskelig å få utført noe som helst nyttig hvis man har 50 utviklere, og plutselig slutter man å produsere dokumenter som

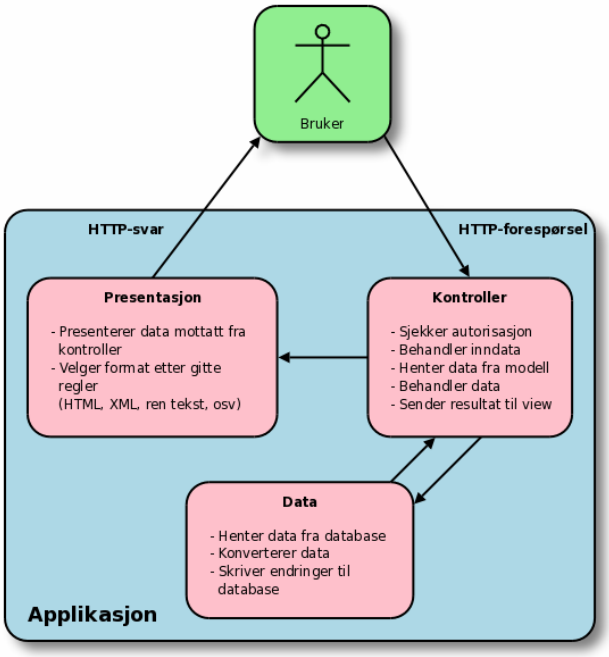
forteller dem hva som skal gjøres. I større prosjekter vil heller ikke alle utviklerne være en del av artefaktproduseringen, så effekten vil heller ikke være den samme.

Prosjektmedlemmene vil her påpeke som tidligere nevnt at det finnes flere versjoner av UP tilgjengelig, og at konklusjonen i denne sammenheng ikke nødvendigvis er gyldig for alle varianter av UP. En fullstendig analyse av alle disse ligger utenfor omfanget til dette prosjektet.

8 Design og oppbygning av systemet

8.1 Teknologi

Som en helhetlig arkitektur ble Data-Presentasjon-Kontroller-modellen valgt (eng: Model-View-Controller; MVC [33]). Denne arkitekturen er delt i tre forskjellige nivåer som har hver sin oppgave. Datalaget er ansvarlig for representasjon av data. Dette skjer som regel i form av objekter som representerer informasjon som er persistent lagret, dette kan for eksempel være tabeller i en database (ett spiller objekt, en turnering, en kamp, osv). Presentasjonslaget er ansvarlig for visning av data til brukeren. I vårt prosjekt vil framstillingen være webbasert og presentasjonslaget vil dermed generere HTML-kode [34] som sendes til brukerens nettleser. Kontrolleren inneholder "forretningslogikken" som behandler forespørsler som kommer fra brukeren. Det er en del oppgaver som er plassert i kontrolleren; tilgangskontroll, oppdatering av modellene og behandle inndata som kommer fra brukeren. Figuren under illustrer dette prinsippet.



Figur 8.1: Oppbygning av en tredelt applikasjon og hvordan denne behandler en HTTP-forespørsel

Som et eksempel på hvordan arkitekturen fungerer i praksis, kan det bli tatt utgangspunkt i en sluttbruker som klikker på en lenke i sin nettleser. Nettleseren vil sende en forespørsel til webtjeneren, som igjen videresender til riktig skript (kontrolleren i dette tilfellet). Den respektive kontrolleren vil så sjekke om brukeren har tilgang til å se denne informasjonen. Hvis brukeren er autorisert til informasjonen, vil kontrolleren finne frem relevante data og gi disse til presentasjonslaget. Presentasjonslaget vil gå gjennom informasjonen den får tilsendt og fremstille denne i form av et HTML-dokument.

En slik tredeling av ansvarsområdene bidrar til en mer oversiktlig systemarkitektur. I tillegg er dette noe som har vært brukt i pensum, og er ganske utbredt og godt akseptert i industrien [33]. Denne arkitekturen anbefales blant annet brukt i Suns Java Enterprise Edition [35].

Til utviklingen av BTS ble PHP [36] og MySQL [37] valgt som teknisk plattform. Det finnes mange andre alternativ av tekniske løsninger, men det er denne kombinasjonen som kommer til å være Norges Bridgeforbunds plattform på sine servere. Valget falt derfor naturlig på denne kombinasjonen.

8.2 HTML, CSS og JavaScript

BTS skal være en webbasert tjeneste. Dette medfører at HTML [34] (HyperText Markup Language) er et naturlig valg for presentasjon av data. Tilnærmet alle datamaskiner har i dag en nettleser som kan presentere HTML-dokumenter. HTML er et "markeringsspråk" som definerer hvordan en webside skal se ut. Den definerer blant annet hva som skal være uthevet skrift, hvor bilder skal være plassert og har lenker videre til andre dokumenter eller filer på Internett. Det finnes forskjellige versjoner av HTML standarden. De fleste er fullt kompatible med hverandre. Med flere års varierende grad av manglende etterfølgelse av standardene, har nettleserprodusentene utviklet nettleserne på en slik måte at de tillater veldig mye feil og fremdeles viser websiden korrekt i de fleste tilfeller. De siste trendene i miljøet beveger seg nå mot en retning som følger standarden for å sikre at websider blir presentert korrekt i forskjellige nettlesere. Dette er en trend som ble etterfulgt i denne oppgaven. XHTML 1.0 Transitional [38] (Extensible Hypertext Markup Language) er HTML standarden som ble valgt for BTS. XHTML er kompatibel med tradisjonell HTML, men setter større krav til korrekt syntaks i dokumentene. XHTML holder samme dybde og bredde som HTML, men beveger seg i retning av XML (Extensible Markup Language) [39] syntaksmessig.

CSS [40] (Cascading Style Sheets) er et stilsett som brukes til å definere hvordan et HTML-dokument skal presenteres. HTML definerer hvordan et dokument er semantisk oppbygget; hva som skal være et avsnitt, hvor et bilde skal være, hva som skal stå i en tabell, osv. CSS sier hvordan et dokument skal se ut; hvilken font skal benyttes, tekststørrelse, farge, fet tekst, kursiv tekst, osv. Dette setter et skille mellom innhold og presentasjon. Dette separerer ansvarsområdet og gjør oppbygningen av en webside mer strukturert og oversiktlig.

JavaScript er et skriptspråk som kjøres i nettleseren. Dette skriptspråket gir mulighet til å manipulere HTML-dokumentet lokalt hos brukeren og kan gjøre HTML-dokumenter mer interaktive. Programvare på webtjeneren kan generere innhold som er dynamisk generert for den enkelte bruker, men dette blir sendt tilbake til brukeren som en statisk side (selv om innholdet ble generert automatisk). Ved å benytte JavaScript kan dette HTML-dokumentet reagere på brukerens interaksjon og tilpasse seg selv for å gi et dynamisk inntrykk.

8.3 MySQL

MySQL [37] er det mest populære databasesystemet som blir brukt i webløsninger. Databasen er rask, enkel å administrere og har god ytelse. MySQL er en relasjonsdatabase som sikter seg inn på små og mellomstore løsninger. I senere versjoner har den fått støtte for funksjonalitet som det er behov for i større løsninger. Dette inkluderer mulighet for automatisk synkronisering av data mellom tjenere; noe som bidra til økt driftssikkerhet og distribuert lastbalansering.

LAMP (Linux, Apache [41], MySQL and PHP) er en sammensetning av operativsystem, webtjener, databasetjener og skriptspråk til websider som er fritt tilgjengelig og gratis til alt bruk. Denne kombinasjonen av programvare har rykte på seg for å være rask og pålitelig.

Sammenlignet med MySQL versjon 4.0 har siste versjon støtte for lagrede spørringer (eng: views); som er "virtuelle" tabeller som blir bygget opp av egendefinerte spørringer. På denne

måten kan man behandle data fra forskjellige tabeller og kombinere disse dataene sammen til en virtuell tabell. Dette er en praktisk funksjon som gjør det mulig å bygge opp avanserte virtuelle tabeller som er sammensatte og kan i en del tilfeller gi et ekstra abstraksjonsnivå for å skjule underliggende logikk. Det er fullt mulig å kjøre vanlige spørringer mot en virtuell tabell for å hente ut den informasjonen som er ønskelig.

I BTS er produksjonsversjonen av MySQL, som er versjon 5.0, valgt som plattform for database. Dette valget og ønsket å bruke lagrede spørringer medførte at applikasjonen ikke er bakoverkompatibel med eldre versjoner av MySQL.

8.4 PHP

PHP [36] er et skriptspråk som er ganske utbredt blant web-utviklere. I følge Netcraft var det over 20 millioner domener som brukte PHP eller som hadde støtte for det. Totalt var det litt i overkant av 50 millioner domener som var aktive i samme periode. [42][43]

Det finnes to hovedversjoner av PHP; versjon 4 og 5. PHP 4 ble lansert mai 2000 og har siden lansering blitt mer og mer utbredt. I juli 2004 ble PHP 5 lansert med en rekke forbedringer i forhold til PHP 4. Noen av de største forbedringene var bedre ytelse, bedre støtte for objektorientert programmering, økt støtte for databaser og bedre feilhåndtering. Overgangen fra PHP 4 til PHP 5 har pågått over lengre tid, da en del skript må skrives om for å fungere skikkelig under PHP 5. Utviklerne av PHP har annonsert at støtte for versjon 4 kun vil være tilgjengelig ut 2007 og alle anbefales til å oppgradere til PHP 5.2 eller nyere. PHP 5.2 er regnet som “produksjonsversjonen”, som bør benyttes i driftsmiljøer.

På bakgrunn av dette, falt valget på PHP 5.2 og nyere i utviklingsarbeidet. En stor del av systemet vil kunne kjøre i PHP 4-miljø, men med noen begrensninger.

8.5 Apache

Apache [41] er en webtjener basert på åpen kildekode, som har ca 50% markedsandel [43]. Apache kan kjøres på de fleste operativsystemer (Windows, BSD-versjoner [44], Unix, Linux, Novell [45], Mac OS X [46] og så videre). Denne webtjeneren er en populær kandidat til å jobbe sammen med PHP, Perl [47] og Python [48]. Valget falt på denne tjeneren på grunn av støtten for forskjellige operativsystem, god funksjonalitet og tidligere gode erfaringer med den fra andre prosjekt.

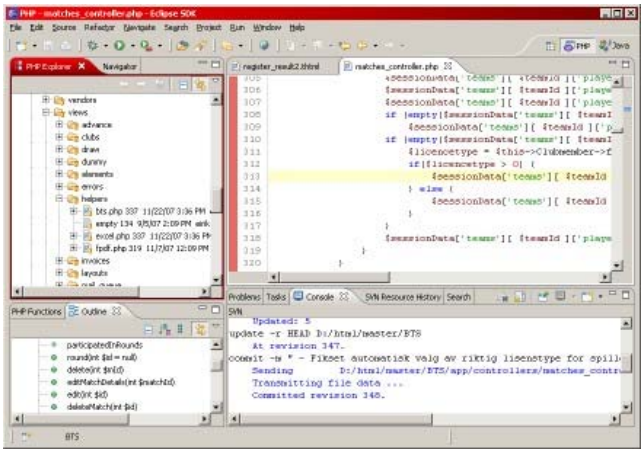
8.6 CakePHP

I valget om hvilket rammeverk som skulle benyttes, var kravet om MVC-støtte avgjørende. CakePHP [49] støttet dette og ble valgt fremfor andre liknende rammeverk på bakgrunn av godt miljø rundt rammeverket og god dokumentasjon. CakePHP gjør det enkelt og raskt å utvikle websider som presenterer informasjonen som er lagret i en database. Det er en god del funksjonalitet som er under panseret; enkel håndtering av lesing/skriving til databasen, enkel skjemahåndtering av brukersendte data og et abstraksjonsnivå til databasetabellene, som blir kalt modeller.

CakePHP har litt av sin inspirasjon fra Ruby on Rails som er et ”raskt-å-bruke”, ”enkelt-å-lære” og ”kort-tid-fra-idé-til-produksjon” rammeverk som er skrevet i skriptspråket Ruby. CakePHP har en økende tilhengerskare og fokuserer på enkelhet og rask utvikling. I tillegg til en del innebygd funksjonalitet som fjerner behovet for å skrive rutinepregede “standard kodesnutter”.

8.7 Verktøy

8.7.1 Eclipse



Figur 8.2: Skjermbilde fra Eclipse

Eclipse [50] er et integrert utviklingsmiljø, eller Integrated Development Environment (IDE) på engelsk. Et IDE er en samling verktøy som er slått sammen til et program, med den hensikt å lage et enkelt miljø for å utvikle programvare. Et IDE tilbyr vanligvis funksjoner som:

Kodeeditor: Editor for å skrive selve kildekoden. Denne har vanligvis fargekoding av syntakselementer i kildekoden, og automatisk kodesjekk som varsler om feil syntaks, manglende tegn, etc. Man kan også ha flere vinduer åpne samtidig slik at man kan ha flere kildefiler fremme på skjermen samtidig.

Prosjekthåndtering: Holder orden på alle kildefiler og støttefiler til programvaren i et felles prosjekt, slik at det er lett å organisere og holde orden på kildefilene.

Støtte for versjonskontroll: Dette gjør det lettere når flere utviklere skal jobbe på samme kode. Bruk av versjonskontroll er beskrevet i avsnittet om Subversion nedenfor.

Integrert støtte for feilsøk: Dette gjør at det meget lett å følge kodeutførelsen linje for linje, sette bruddpunkt, sjekke verdier på variabler med mer.

Eclipse er et IDE primært utviklet for utvikling av Java-programmer, men det finnes utvidelser for Eclipse som gjør det mulig å også benytte dette som utviklingsverktøy for andre språk. Eclipse er basert på åpen kildekode, på lik linje med de fleste andre verktøyene som ble brukt under prosjektet. Eclipse er gratis tilgjengelig fra [50].

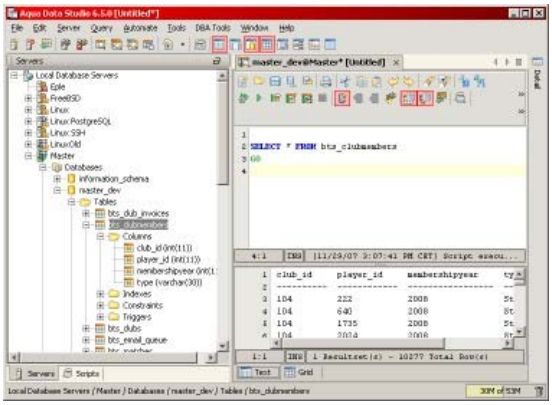
Subclipse

Subclipse [51] er en utvidelse til Eclipse som integrerer en Subversion-klient i IDEet. Dette gjør at kildekoden enkelt kan synkroniseres mot Subversion-tjeneren uten å måtte starte eksterne programmer. Dette gjør det enkelt og praktisk å benytte Subversion i utviklingen. Subclipse kan lastes ned gratis.

PHP Development Tools

PHP Development Tools (PDT) [52] er en utvidelse for Eclipse, som setter Eclipse i stand til å utvikle programmer i PHP. PDT er et prosjekt underlagt "Eclipse-paraplyen", som omfatter et stort antall prosjekter relatert til Eclipse. PDT er som Eclipse selv basert på åpen kildekode, og er fritt tilgjengelig fra hjemmesiden.

8.7.2 Aqua Data Studio



Figur 8.3: Skjermbilde fra Aqua Data Studio

Aqua Data Studio (ADS) [53] er et program for å behandle databaser. Programmet gir brukeren en god oversikt over databasens struktur, og lar brukeren opprette, endre og slette tabeller og andre databaseelementer via dialogbokser, uten at brukeren trenger å ha god kjennskap til SQL-syntaks.

Som ethvert profesjonelt databaseverktøy, så tilbyr også ADS muligheten for å skrive inn SQL-setninger direkte. Her har ADS en editor som gir fargekoding av spørrestrukuren, slik at det blir mye enklere å lese SQL-spørringer og sørge for at disse er korrekte.

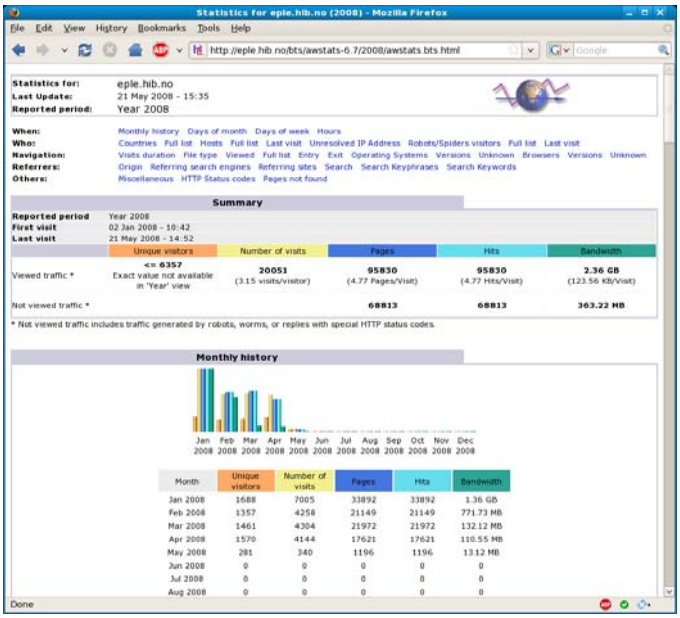
ADS har også mange andre gode verktøy, blant annet generering av ER-diagrammer og databaseskjema basert på databasen.

I tillegg til å kunne gjøre endringer på tabeller via enkle dialogbokser, gir også ADS muligheten til å enkelt redigere dataene som ligger i en tabell direkte, uten å måtte gå omveien om SQL-spørringer (sett fra en brukers ståsted, SQL-spørringer blir selvsagt kjørt skjult i bakgrunnen).

ADS kan på mange måter sammenlignes med Microsoft Access [54] når det gjelder måten ting utføres på i forholdt til databaseadministrasjon, da mange av oppgavene utføres på samme måte, slik som å opprette tabeller og spørringer, og direkteredigering av dataene i tabellene. Det presiseres at ADS og Access er allikevel to veldig forskjellige programmer, og at likhetene her ligger i det grafiske grensesnittet, og ikke måten programmene fungerer internt.

ADS er i motsetning til de fleste andre programmene som ble benyttet, ikke basert på åpen kildekode.

8.7.3 AWStats



Figur 8.4: Skjermbilde fra AWStats

AWStats [55] kan generere statistikk om besøksdata som er samlet inn i loggene til forskjellige webtjenere. Denne statistikken kan brukes til å få et innblikk av brukerne som benytter webtjenesten. Den kan bryte statistikken opp i år, måneder, dager og timer. Hvilke nettlesere og operativsystem som benyttes kan også vises. AWStats er gratis i bruk og er skrevet i skriptspråket Perl.

8.8 Andre bibliotek

8.8.1 Prototype

Prototype [56] er et JavaScript-rammeverk som byr på ekstra funksjonalitet for å gjøre web-sider mer levende. Rammeverket sikter inn på å forenkle arbeidet med å lage dynamiske websider. En del lange repeterende oppgaver blir enklere og rammeverket arbeider for å være mest mulig kompatibel med de største nettleserne.

8.8.2 script.aculo.us

JavaScript-biblioteket script.aculo.us [57] bygger på Prototype og tilbyr utvikleren en rekke visuelle effekter og kontrollere som kan benyttes på websider. De visuelle effektene inkluderer animasjoner, gjennomsiktighet, endring i størrelse og fremheving. I tillegg tilbys et enkelt grensesnitt for å utvikle dra-og-slipp funksjonalitet i helt vanlige websider. En ekstra mulighet blir gitt til inndatafelt som kan få en “autofullfør” funksjon som søker etter mulige treff alt etter hva brukeren skrev inn i feltet.

8.9 Valg av løsninger

8.9.1 Mellomlagring

BTS er et system hvor så godt som alt av innhold blir dynamisk generert. Dette medfører en vesentlig større last på tjeneren i forhold til å servere statiske sider. Det er derfor viktig å vurdere tiltak for å redusere ressursbruken. En av måtene dette kan oppnås på er ved å innføre mellomlagring (eng: caching) av ofte besøkte sider. For eksempel, så blir innholdet på resultatsidene kun oppdatert hver gang det er spilt en ny kamp, så det er egentlig ikke nødvendig at denne siden blir generert på nytt igjen for hver visning, da det er godt nok at den blir generert på nytt når nye data blir rapportert inn. Sider som denne er derfor perfekte kandidater til mellomlagring.

Mellomlagring foregår ved at første gang siden blir generert, så blir det lagret en statisk kopi av siden på webtjeneren. Neste besøkende som så ber om samme siden, får servert denne kopien av den statiske websiden i stedet for at den blir generert dynamisk. Dersom tjeneren oppdager at det har vært endringer siden den statiske siden ble generert, vil den i stedet generere en ny side dynamisk og lagre denne.

Et av problemene som ofte oppstår med mellomlagring, er at tjeneren ikke alltid klarer å følge med når data har blitt endret, slik at den besøkende får servert en utdatert statisk side. Dette problemet er ofte forårsaket av at tjeneren sjekker for sjelden om datagrunnlaget for en side er oppdatert.

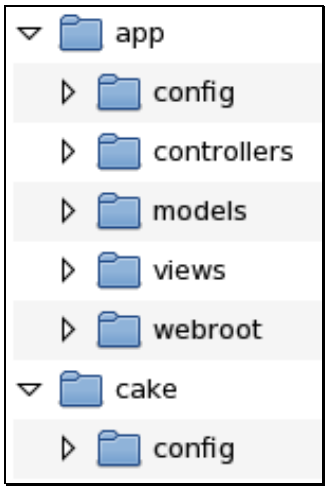
Rammeverket som ble benyttet, CakePHP, har innebygd mulighet for mellomlagring. Dette kan derfor enkelt aktiveres for BTS dersom dette blir nødvendig. For å unngå problemet med utdaterte mellomlagringssider har CakePHP to mekanismer for å håndtere dette. Dersom CakePHP selv lagrer data, vet den at alle sidene som baserer seg på den datamodellen som nettopp ble lagret data til er utdatert, og må regenereres. I tillegg utløper sidene etter en definert tid, slik at endringer som kommer via andre kilder også blir tatt hensyn til. Hvor lenge mellomlagringssidene bør være gyldige vil derfor være avhengig av hvor ofte man forventer at andre applikasjoner vil skrive til databasen. I BTS' tilfelle skjer dette kun hver gang spillertabellen synkroniseres mot NBFs medlemsdatabase, noe som gjøres sjeldnere enn en gang per dag. Så for bruk mot BTS kan utløpstiden for mellomlagringssider være relativt høy, gjerne en time eller mer, da websidene automatisk blir regenerert hvis CakePHP selv oppdaterer dataene.

Etter en analyse av trafikken mot webtjeneren, ble det observert at antallet brukere er så lavt, at ressursbruken disse forårsaker på tjeneren var minimal. Det ble konkludert med at mellomlagring ikke vil føre til noe vesentlig forbedring slik systemet er nå. Siden dette også medfører et ekstra nivå av kompleksitet, ble dermed ikke mellomlagring aktivert i denne omgang. Dersom systemet senere blir tatt i bruk på en slik måte at antallet daglige besøkende øker drastisk, kan det da være verdt å aktivere disse funksjonene. Siden dette ligger innebygd i rammeverket, kan dette utføres uten å måtte gjøre store endringer i systemet.

9 Implementering

I dette kapittelet vil det bli gitt en oversikt over hvordan BTS er bygget opp og hvilke viktige valg ble foretatt under implementeringen.

9.1 Oversikt over CakePHPs filstruktur og grunnleggende funksjonalitet



Figur 9.1: CakePHPs katalogoppbygning

CakePHP har ordnet filsystemet i en trestruktur som først deler applikasjonskoden ("app") og rammeverkets interne kode ("cake"), i hver sine kataloger. I "cake" underkatalogen finnes rammeverkets interne filer og standardkonfigurasjon. BTS' kildefiler finnes i "app" katalogen. Disse kildefilene er videre sortert i sine respektive underkataloger, avhengig av hvilken funksjon de har i MVC-modellen. Figur 9.1 viser katalogstrukturen benyttet i CakePHP.

Rammeverket gir standardkomponenter som det er mulig å utvide. I BTS er denne muligheten brukt til logging, komprimering av output og sikkerhet. Denne endringen er plassert i applikasjonsmappen etter den interne navnekonvensjonen. Ved en oppgradering av rammeverket til en nyere versjon, vil ikke BTS-spesifikke endringer bli overskrevet, da ingen av endringene ble gjort i kjernekoden til CakePHP.

Modellene i CakePHP er en representasjon av tabellene i databasen. I modellene er det mulig å definere relasjoner til andre modeller. Dette gir et fordelaktig separasjonslag mellom databasen og modellene brukt i applikasjonen. For eksempel ved et navnebytte av en tabell er det kun behov for å endre referansen i modellen. I applikasjoner som ikke innfører et slikt abstraksjonslag, vil alle spørringer måtte skrives om i nevnte eksempel. I tillegg er det mulig å få modellen til å utføre bestemte handlinger før lesing og skriving av data til database-tabellen finner sted. I BTS benyttes denne funksjonaliteten enkelte steder til å konvertere datoer mellom forskjellige format (fra ISO til "norsk" format)[1].

1 ISO er et standardisert format (YYYY-MM-DD HH:MM:SS). Den "norske standarden" bytter om på datofeltene (DD.MM.YYYY HH:MM.SS)

Kontrollerne er bindeleddet mellom datamodellene og presentasjonslaget. Både tilgangskontroll og prosessering av innkomne og utgående data blir behandlet av en kontroller. Det finnes en kontroller for hver logiske del av BTS. Et eksempel på dette er kamp-kontrolleren som er ansvarlig for å registrere kampresultat som kommer fra sluttbrukeren, vise kampinformasjon og lignende. Hvis en bestemt side er definert til å kun vises til en begrenset mengde brukere (for eksempel kun innloggede brukere, kun administratorer, kun klubbkontaktpersoner og så videre) vil den første oppgaven kontrolleren utfører være å sjekke om sluttbrukeren har tilgang til den aktuelle funksjonen. Hvis tilgang er autorisert vil kontrollerens neste oppgave være å hente nødvendige data fra modellene. Kontrolleren kan hente data fra forskjellige modeller og behandle/koble sammen data ved behov. Etter denne jobben er ferdig sendes det ferdige datasettet videre til presentasjonslaget.

Presentasjonslaget er ansvarlig for framstilling av data til sluttbrukeren. Dette strukturerer dataene som kommer fra kontrolleren på en hensiktsmessig måte for visning. I BTS benyttes HTML/CSS som presentasjonsmedium. I de fleste tilfeller blir dataene presentert i form av en tabell eller lignende struktur. Hvis det er behov for å sende informasjon tilbake til systemet, vil presentasjonslaget opprette et skjema hvor brukeren angir ønsket informasjon og denne sendes videre til kontrolleren som er ansvarlig for videre behandling.

I CakePHP er det mulig å opprette egne hjelpeklasser som kan utføre repeterende oppgaver i presentasjonslaget. Dette ble benyttet i BTS for å gi en sentral samling av ofte benyttede funksjoner. Ved å samle slike kodesnutter i en sentral klasse, oppnår man den fordelen at det kun er behov for å gjøre endringer ett sted når koden skal vedlikeholdes.

Det finnes også en mulighet for å vise ekstra debug-informasjon på skjermbildet når CakePHP er satt i utviklingsmodus. Da kan ekstra informasjon om genereringen av en side presenteres. For eksempel vises alle spørringene som er kjørt mot databasen, i tillegg til hvor lang tid den brukte på hver spørring. Dette har vært en viktig kilde for informasjon når enkeltsider har blitt analysert for å bedre lastetiden.

9.2 Brukertilganger og roller

Det er implementert tre forskjellige brukernivåer; gjest, registrert bruker og administrator. En gjest har lesetilgang til det meste av informasjonen som er tilgjengelig. I tillegg har gjester mulighet til å registrere kampresultat, dersom vedkommende representerer lagene som deltok i kampen. Registrerte brukere har mulighet til å melde på lag til en turnering og endre kontaktinformasjon for et lag. En administrator er en bruker som har full tilgang til alle funksjoner og utvidede rapporteringsverktøy.

Registrerte brukere kan tildeles roller for sin tilknytning til et lag eller klubb. Kontaktpersonen for en klubb vil ha rollen som klubbansvarlig. Vedkomne har tilgang til funksjonalitet som gjør det mulig å endre informasjon om en klubb og klubbens lag i en turnering. Brukeren som er ansvarlig for en turnering blir tildelt rollen som turneringsansvarlig. Vedkomne vil ha tilnærmet samme tilgang som en administrator, men begrenset til turneringen rollen er tildelt.

9.3 Interaktivitet mellom nettleseren og applikasjonen

BTS har primært benyttet HMTL, CSS og JavaScript som presentasjon mot sluttbrukeren. Eksport til et Excel-ark [58] har blitt brukt i noen få tilfeller hvor det er ønskelig for turneringsansvarlig/administrator å behandle dataene videre lokalt. Eksport av laglisten og

trekklistene er implementert ved hjelp av eksternt bibliotek som sørger for generering av en Excel-fil med spesifisert data [59].

Ajax (asynkrone JavaScript kall) er blitt brukt i noen av veiviserne som et hjelpemiddel for brukeren. Ved registrering av lag er dette benyttet for å gi brukeren en indikasjon på at en gitt klubb ID tilhører ønsket klubb. Ved å bruke Ajax, sendes kun IDen tilhørende klubben til webtjeneren, som så sender klubbens navn tilbake til nettleseren. Dette blir presentert til brukeren ved å vise navnet på klubben ved siden av feltet som spør etter IDen til klubben. Dette sparer både båndbredde og prosesseringskraft på både klient og tjener. Alternativet til å implementere samme funksjonalitet uten bruk av Ajax, ville vært å oppdatere hele siden hver gang et tekstfelt var endret. Eventuelt kan man sende hele listen med klubbnavn til nettleseren, og brukt JavaScript til å hente riktig element. Ved å benytte asynkrone kall i bakgrunnen, fremstår applikasjonen mer interaktiv og gir assosiasjoner til vanlige tradisjonelle skrivebordsprogrammer, hvor slik interaktivitet har vært benyttet i flere tiår. Ved å benytte Ajax-kall er implementert funksjonalitet i tekstfeltet hvor brukeren skal skrive en klubb-ID, slik at brukeren også kan søke etter en klubb her. Dette tilbyr brukere som ikke vet klubbens ID, en rask mulighet til å søke opp sin klubb uten å forlate det aktive skjermbildet.

Ved å bruke Ajax som en del av interaktiviteten, vil det i enkelte tilfeller oppleves en liten forsinkelse fra brukerens handling til resultatet er synlig på skjermen. Dette vil i de fleste tilfeller skyldes en forsinkelse på Internettforbindelsen til brukeren eller tjeneren. Typiske situasjoner vil være tilfeller hvor forbindelsen til brukeren benyttes til andre ting enn surfing. Da vil Ajax-kallet vente i en kø før det blir sendt av gårde til tjeneren. I praktisk bruk pleier denne forsinkelsen til å være godt under ett sekund. I implementeringen av BTS er det blitt fokusert på å bruke Ajax som et tillegg i brukeropplevelsen, og ikke som et krav for å benytte funksjonaliteten i BTS. På denne måten vil brukere som ikke har støtte for JavaScript i sin nettleser, også ha mulighet til å benytte tjenestene.

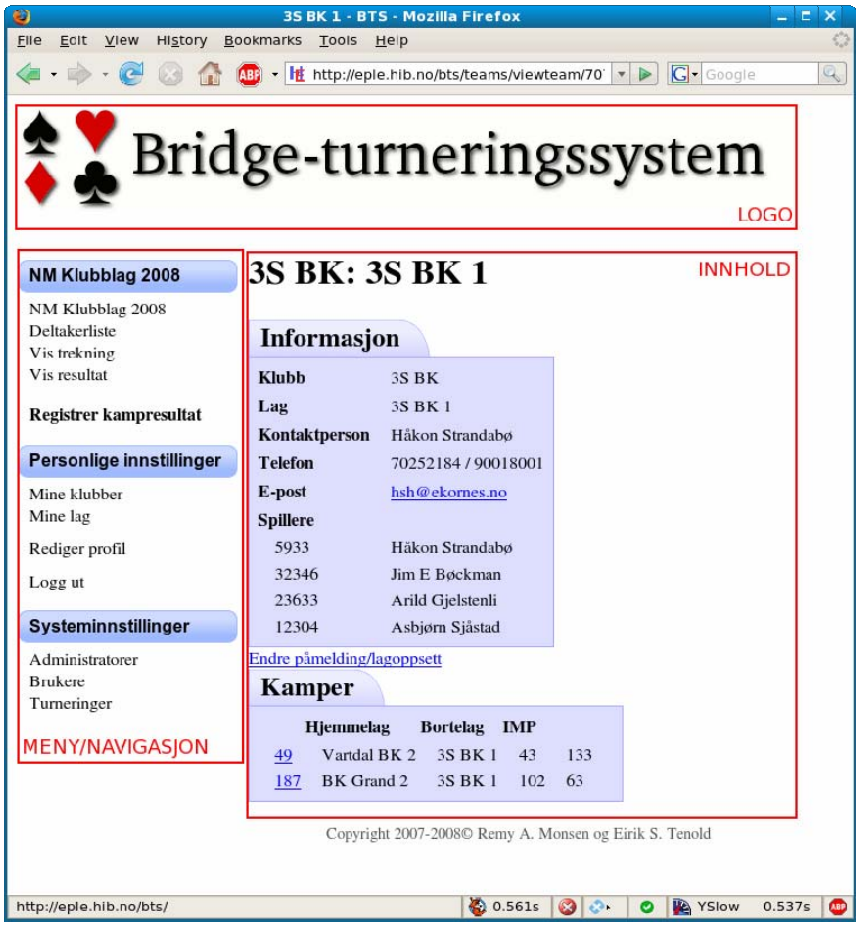
9.4 Oppbygning av websiden

Websiden er bygget opp som en tradisjonell fordeling mellom logo/tittel, meny og innhold. Området øverst på skjermen kalles tittelfeltet. Dette området strekker seg over sidens bredde. Her er logoen til applikasjonen plassert.

Navigasjonsområdet/menyen er lokalisert til venstre på skjermen og er rett under logoen. Her blir hovedvalgene til brukeren presentert. I BTS er menyen gruppert inn i tre logiske seksjoner; *Turnering*, *Personlige innstillinger* og *Systeminnstillinger*. Hvilke menyvalg som vises avhenger om brukeren er innlogget og hvilket tilgangnivå brukeren har. Turneringsmenyen vises til alle besøkende uavhengig av innlogging og tilgangsnivå. Her presenteres lenker til all offentlig tilgjengelig.

Personlige innstillinger vises kun til brukere som er innlogget. Denne menyen gir tilgang til innstillinger og informasjon som er begrenset til innloggede brukere. Systeminnstillinger blir kun vist til brukere som er innlogget og har administratortilgang. Her kan brukeren sette innstillinger som gjelder alle brukere og turneringer.

Figur 9.2 viser denne oppbygningen.



Figur 9.2: Skjermbilde som viser oppbygningen av BTS. Skjermen er delt opp i tre deler; meny/navigasjon, logo og innhold

Selve innholdet på hver enkelt side bestemmes av hvilken metode i kontrolleren som blir kjørt. Oppbygningen av adressen følger dette formatet:

```
http://<navn på tjener>/<katalogstruktur>/<kontroller>/<kontrollermetode>/
<evt. parametere>
```

Et konkret eksempel på denne oppbygningen er adressen til visningen av et lag (Figur 9.2):

```
http://eple.hib.no/bts/teams/viewteam/7070
```

Her er *eple.hib.no* referansen til webtjeneren hvor BTS er plassert. Filstien "bts" viser til hvilken katalog BTS finnes i. Argumentet "teams" forteller CakePHP hvilken kontroller som skal benyttes. Siste argument, "viewteam", forteller hvilken metode i kontrolleren som skal kalles. Verdien "7070" er en parameter til metoden som i dette tilfellet forteller hvilket lag som skal vises.

9.5 Synkronisering av medlemsdata mellom BTS og Norsk Bridgeforbund

BTS har en egen liste over medlemmer for å enkelt koble disse mot lag og resultater. Dette medfører at BTS' medlemsliste regelmessig bør synkroniseres mot NBFs medlemslister. Dermed vil BTS være oppdatert med siste informasjon om medlemmer (dette inkluderer nye medlemmer, klubbmedlemskap og lisens).

NBF har sitt medlemsregister i en Microsoft Access-database [54], som er tilgjengelig over Internett til bruk i prosjektet (noen felt blir automatisk sensurert). For å automatisere prosessen med å overføre data, ble en liten applikasjon utviklet. Den tok seg av synkroniseringen mellom MySQL-databasen, som BTS benyttet, og Microsoft Access-databasen med medlemsinformasjon. Denne applikasjonen åpnet en forbindelse til en lokal versjon av Microsoft Access-databasen ved hjelp av ODBC (Open Database Connectivity) [61], og sammenlignet medlemslisten opp mot hva som var registrert i MySQL-databasen. Nye medlemmer ble lagt til, og eksisterende medlemmer fikk sin informasjon oppdatert. Medlemmer som var slettet fra NBFs medlemsliste, ville fremdeles beholde sin spiller i BTS for å vise korrekt informasjon på historiske data.

Synkroniseringen er foreløpig enveis. BTS-databasen skal alltid oppdateres med endringer gjort i Access-databasen. En mulig fremtidig endring ville kunne gi mulighet for brukere å endre egen informasjon direkte. Dette er utenfor omfanget til denne masteroppgaven.

9.6 Gjennomgang av BTS

I dette underkapitlet vil det bli gjort en gjennomgang av vesentlig funksjonalitet i BTS og kommentarer til implementeringsvalg som ble gjort underveis.

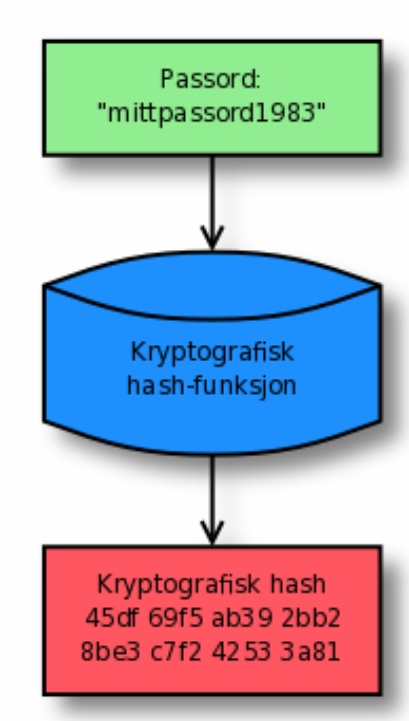
9.6.1 Registrering av bruker

En del av funksjonaliteten til BTS er begrenset til registrerte brukere. En del områder krever å vite hvem brukeren er for å gi personlig tilpasning av innhold. Andre steder er det behov for å vite hvilket tilgangsnivå en bestemt bruker har.

Første steg for brukeren er å velge seg et brukernavn og passord. Brukernavnet må være unikt og kan ikke være brukt av andre. Passordet er selvvalgt av brukeren og lagres sikkert i databasen i form av en kryptografisk hash av passordet. Det er satt en minimumsgrense på fire tegn for at passordet skal være gyldig.

En del personer benytter samme passord til forskjellige tjenester på Internett. Dette medfører at databaser med brukernavn og passord er et yndet mål for kriminelle. Det er ganske vanskelig å lage en helt sikker løsning, men målet er å gjøre dataene så unyttig som mulig for en eventuell angriper. For å hindre konsekvensene ved et innbrudd i databasen bli alle passordene lagret i form av en kryptografisk hash. Ved å benytte seg av denne metoden vil det fremdeles være mulig å kontrollere om en bruker oppgir riktig passord, men for en angriper vil han ha problemer med å hente ut brukernes passord.

En kryptografisk hash er et "elektronisk fingeravtrykk" av en gitt mengde data. Ved å ta en tekststreng og sendes denne gjennom en kryptografisk funksjon vil man få ut en "signatur" på en fast lengde. Denne signaturen er ganske unik og små endringer i dataene som blir hashet, vil resultere i store forandringer i signaturen. Mengden forskjellige signaturer er enorm i forhold til antall tegn normalt brukt i selvvalgte passord av brukere. De mest brukte kryptografiske hashfunksjoner har minimum 2^{128} unike signaturer. Til sammenligning trengs det ca 20 tegn bestående av tilfeldige små og store bokstaver, tall og 20 spesialtegn for lage passord som har like stor variasjon i unike kombinasjoner som signaturene fra hash-funksjonene. Dette gjør at antallet mulige forskjellige signaturer er mer enn godt nok, og kan brukes på selvvalgte passordene til brukerne.



Figur 9.3: Eksempel på hvordan "signaturen" til et passord kan genereres

Ved å kun lagre denne signaturen til brukerens passord vil man ved et eventuelt innbrudd i databasen, ikke røpe brukerens passord. Det er heller ikke mulig for administratorene å se passordet til en bruker. I BTS har MD5 [62] (Message-Digest algorithm) blitt benyttet til å sikre passordene. Figur 9.3 viser hvordan et passord forandres til en kryptografisk hash som kan lagres i databasen. Se *Appendiks C: Kryptografisk hash* for mer informasjon om kryptografiske hash-funksjoner.

I tillegg til å velge et brukernavn og passord blir brukeren bedt om å skrive inn noen tegn som er angitt på et bilde (Captcha - Completely Automated Public Turing test to tell Computers and Humans Apart [63]). Dette gjøres for å hindre automatisk registrering gjort av datamaskiner. Selve bildet består av tre alfanumeriske tegn samt litt bakgrunnsstøy. Hvis brukeren ikke klarer å lese tegnene er det mulig å få generert et nytt tilfeldig bilde.

I neste steg av veiviseren vil brukeren bli spurt om sitt NBF-medlemsnummer. Dette brukes for å knytte opp en bruker mot en registrert spiller. Det er ikke et krav om å ha et NBF-medlemsnummer for å registrere seg.

Steg tre ber brukeren om personlig informasjon som navn, adresse og telefonnumre. Hvis brukeren skrev inn sitt NBF-medlemsnummer vil navn og adresse automatisk fylles ut hvis denne informasjonen er tilgjengelig fra NBFs medlemsdatabase.

Siste steg i veiviseren tilbyr brukeren å registrere sitt medlemskap opp mot en klubb. Dette er ikke påkrevd å registrere, da ikke alle spillere er tilknyttet en klubb. Hvis et spillernummer ble oppgitt i forrige steg, vil klubbmedlemskap automatisk være fylt ut. Ved fullført registrering sendes en e-post til den oppgitte e-postadressen som en bekreftelse på vellykket registrering.

9.6.2 Registrering av lag

Noe av det første som utføres i begynnelsen av en turnering er å motta påmeldinger. Blant informasjonen som trengs er hvilken klubb et lag er tilknyttet og hvem som er kontaktpersonen for laget. For å gjøre påmeldingen enkel, ble det opprettet en veiviser som består av fem steg. I hvert steg blir det gitt en kort beskrivelse av hva som trengs av informasjon. For å hindre problematikken rundt "spam-boter" som sender inn reklame i alle tilgjengelige skjema, er det kun mulig for registrerte brukere å melde på et lag.

Det første steget i veiviseren er en enkel velkomstbeskjed som forteller kort om hva som skjer i de neste stegene. Steg to omhandler klubben som påmeldingen gjelder. Her kan det angis både ved hjelp av klubbens NBF-nummer eller klubbens navn.

Steg tre ber om kontaktinformasjonen for laget. Her er det mulig å søke både på spillernummer og spillernavnet på vedkomne som skal være kontaktpersonen. Brukeren blir spurt om e-postadresse, mobiltelefonnummer og det er et eget tekstfelt som kan brukes til å spesifisere spesielle hensyn som må bli tatt hensyn til av andre. Et konkret eksempel på dette er et lag som hadde en døv kontaktperson. I dette tilfellet var det kun mulig å kontakte vedkomne ved hjelp av SMS eller e-post.

Fjerde steg gir brukeren mulighet til å registrere hvilke spillere som skal delta på laget. Det er ikke nødvendig å registrere alle spillerne i denne omgang.

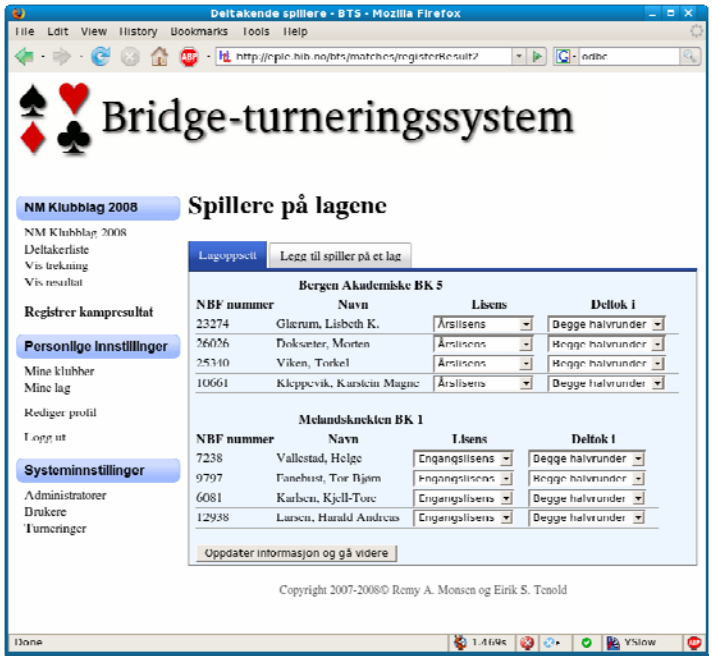
Den siste delen av veiviseren er en oppsummeringsside hvor brukeren får opp en oversikt over informasjonen som vil bli registrert. Etter brukeren har lest gjennom informasjonen vil han/hun bekrefte at dette stemmer. Deretter vil det bli sendt en bekreftelse til den registrerte e-postadressen til kontaktpersonen.

9.6.3 Registrering av kampresultat

Registrering av resultatet etter en kamp er en av de sentrale komponentene som benyttes i gjennomføringen av en turnering. For å sikre at kun autoriserte personer kan registrere resultatet etter en kamp, er det implementert en valideringskode som blir tildelt begge lag. Denne koden består av fire sifre, og er tilfeldig generert for hvert lag i en kamp. Fra tidligere turneringer i NM for klubblag har vinnerlaget vært ansvarlig for å sende inn resultatet til NBF. Dette ble gjort med et skjema som måtte signeres av begge lag og ble sendt via faks eller e-post til NBF.

Ved utsendelsen av e-post med trekningsinformasjon, får kontaktpersonen informasjon om hvem de møter og hvilken valideringskode de har for kampen. Begge valideringskodene trengs for å registrere et kampresultat i BTS. Valideringskoden til taperlaget blir overlevert vinnerlaget, som så er ansvarlig for å sende inn resultatet. I enkelte tilfeller hvor det ikke er ønskelig å registrere resultat via BTS, er det mulig å fakse inn resultatskjemaet på samme måte som tidligere. NBF vil så manuelt registrere resultatet i BTS. En turneringsansvarlig kan registrere et kampresultat uten å kjenne valideringskodene. Figur 9.4 viser ett av stegene i veiviseren for å registrere kampresultater.

Hvis valideringskoden ikke er mottatt av en part, er det mulig for lagets og klubbens kontaktperson til å få vist valideringskoden, hvis vedkomne er innlogget i BTS og går inn på kampsiden. Utenom dette er det mulig for turneringsansvarlig å hente ut en liste over alle kamper og respektive valideringskoder. Disse kan brukes til å opplyse om valideringskoder over telefon.



Figur 9.4: Skjermbilde som viser et steg i veiviseren som registrer kampresultat

Det er ikke et krav om at brukeren skal være innlogget for å registrere resultat. I praksis vil valideringskodene tilsvare en PIN-kode på åtte siffer. Etter brukeren er verifisert til å registrere et resultat, blir brukeren presentert et skjermbilde hvor deltakerne på lagene blir registrert. Denne informasjon er viktig i forbindelse med fakturering av spillere uten årslisens og tildeling av forbundspoeng til den individuelle spiller.

Ved initial visning av deltakerregistrering blir alle spillere som tidligere har deltatt på laget listet opp. I de fleste tilfeller vil det være de samme spillerne som deltar på et lag gjennom turneringen. Enkelte ganger byttes spillere ut underveis og det vil derfor være behov for å registrere flere spillere som deltakere på en lag. Brukeren som registrerer resultatet kan da velge mellom å hente spilleren fra en liste over spillere registrert som medlemmer av den aktuelle klubben, eller om det er mest hensiktsmessig å angi et NBF-medlemsnummer.

Når deltakerlisten er komplett, slik at det i hver halvrunde deltar fire spillere på hvert lag, vil brukeren få frem et spørsmål om hvor mange poeng hvert lag klarte å oppnå i kampen. Til slutt vil brukeren få presentert en oppsummering hvor deltakere og poengfordeling er listet opp. Denne må brukeren bekrefte er korrekt før kampresultatet blir lagret og gjort offentlig tilgjengelig på BTS. Hvis resultat feilregistreres, er det mulig å gjenta registreringsprosessen for å registrere korrekt resultat.

Siden resultatregistreringen i BTS er et supplement til den tradisjonelle resultatrapporteringen via brev eller faks, er det kun knyttet fordeler opp mot denne tjenesten. Det er raskt for vinnerlaget å registrere resultat i forhold til å sende en faks eller brev. Resultatet blir også tilgjengelig umiddelbart etter bekreftelsen. Turneringsansvarlig vil også spare tid brukt til manuelt arbeid i forbindelse med behandling av faks/brev. Dette vil resultere i bedre resultatservice for alle interesserte i turneringen.

9.6.4 Trekning av runder

Trekning av runder blir foretatt av turneringsansvarlig. Denne prosessen har tre steg; velge hvilke lag som gikk videre, sette opp kamper og vise trekning. Dagens rutiner for trekning av

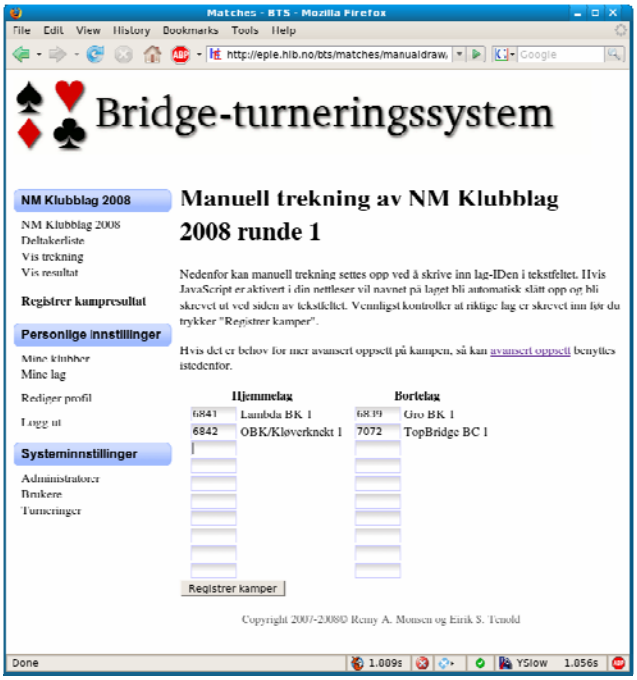
runder foregår ved å ha alle lag på hver sin lapp og kampene blir satt opp på bakgrunn av trekningen av disse lappene. I de innledende rundene blir det også tatt hensyn til regioner og kamper blir satt opp innad i sine respektive regioner.

I BTS er første del av trekningen import av lag som skal delta i runden. Dette gjøres semi-automatisk ved at BTS viser en liste over alle lag i forrige runde og forhåndsmarkerer de lagene som vant kampen. Dette må da turneringsansvarlig ta en rask kontroll på, for å sikre at datagrunnlaget ser korrekt ut. Deretter importeres valgte lag til neste runde. I første runde blir en del "beste tapere" tatt med videre for å få korrekt antall lag i runden. Her kan turneringsansvarlig markere hvilke lag som skal få delta i runden.

Når alle lag som skal delta i runden er importert vil turneringsansvarlig skrive ut trekningslapper. Trekningslappene er A4-ark som er delt inn i to kolonner og fem rader. Hver rute har lagets navn, region, og tidligere spilte kamper i den aktuelle turneringen. Disse arkene blir skrevet ut på perforerte ark som gjør det enkelt å dele lappene i like store deler.

Automatisk trekning ble vurdert, men på grunn av praktiske begrensninger ble dette punktet for stort for denne oppgaven. Noen av momentene som må bli tatt hensyn til ved trekning er geografiske begrensninger for hvilke lag som kan spille mot hverandre. For eksempel kan et luftlinjemål kunne gi problemer hvis en fjellovergang er stengt. Med bakgrunn i dette ble det besluttet at denne versjonen av BTS kun skulle støtte manuelle trekninger. En mulig fremtidig utvidelse vil kunne tilfredsstille slike spesielle behov ved trekninger.

Etter turneringsansvarlig har trukket kampene, brukes IDen til laget (blir skrevet på trekkelappene) til å registrere hvilke lag som skal spille mot hverandre. Lag IDene som blir angitt går gjennom noen enkle sjekker for å sikre at elementære krav er tilfredsstilt. Blant annet sjekkes det om laget er med i den aktuelle turneringen og om de allerede har en kamp i runden. Etter hvert som kamper registreres blir disse synlige på trekningslisten i BTS. Figur 9.5 viser skjermbildet hvor trekningen blir registrert.



Figur 9.5: Skjermbilde som viser registreringssiden som benyttes for å sette opp kamper etter en manuell trekning

9.6.5 Rapport for visning av forbundspoeng

BTS inneholder også en rapport som viser hvor mange forbundspoeng en spiller er tildelt basert på hvor mange vinnende kamper vedkomne har deltatt i. Rapporten finner lagene som er slått ut i en runde og regner ut hvor mange poeng deltakerne skal ha. Dette baseres på om spilleren deltok i begge halvrunder av en kamp og laget var seirende.

9.7 Sikkerhetskopier

For å sikre mot uønsket datatap ble testtjeneren satt opp til å regelmessig ta sikkerhetskopier av databasen med alle data brukt i produksjonsversjonen av BTS. Skriptet som var ansvarlig for å kjøre sikkerhetskopieringen ble kjørt hver tredje time. Dette medførte at åtte daglige "stillbilder" av dataene i databasen var tilgjengelig hvis det oppstod behov for å gjenopprette tilstanden til et tidligere tidspunkt. Ved et datatap ville dette ikke medført store konsekvenser. For eksempel ville et eventuelt tap av noen få kampresultater medført at turneringsansvarlig hadde etterlyst resultatet fra de aktuelle kampene, og resultatet ville kunne bli registrert på nytt. Et system som BTS er ikke underlagt strenge krav og lovgivning slik som banker og finansinstitusjoner.

Ved å automatisk overføre sikkerhetskopiene til en annen fysisk plassering bidro dette til å minske konsekvensene ved eventuell harddiskkrasj, brann, tyveri eller lignende.

Heldigvis var det aldri behov for å gjenopprette databasen. Ved en eventuell gjenoppretting ville alle endringer forsvinne som var gjort etter tidspunktet sikkerhetskopien ble tatt. Dette var en av grunnene til et tre timers intervall ble valgt. Ved et minsket intervall mellom sikkerhetskopiene ville det vært mulig å minske omfanget av "tapt data" ved en eventuell gjenoppretting. Dette ville også medført økt mengde med sikkerhetskopier.

Selve sikkerhetskopien var opprettet som en tekstfil med komplette SQL-spørringer som trengtes for å gjenopprette hele databasen både i struktur og innhold. Denne tekstfilen ble så komprimert med bzip2-algoritmen [64], som er en gratis åpen kildekode algoritme som har god ytelse i forhold til de tradisjonelle gzip- [65] og zip- [66] algoritmene. Den komprimerte sikkerhetskopien var rundt 1,2 MB stor. Dette var en grei størrelse for å raskt overføre denne over Internett til den eksterne testtjeneren.

I tillegg til å ha sikkerhetskopier av databasen var det også behov for å ha en sikker kopi av kildekoden. Ved å bruke Subversion som et verktøy for versjonshåndtering av koden, fungerte dette i seg selv som en slags sikkerhetskopi. I Subversion er det fullt mulig å gjenopprette en tidligere versjon av filen, selv om den er slettet av en bruker. Dette i seg selv er ikke en god nok løsning for sikkerhetskopiering av kildekoden hvis tjeneren mister sine data. Utenom dette var det en komplett versjon av koden på de to utviklingsmaskinene, produksjonstjeneren og en tredje maskin som var satt opp til å automatisk synkronisere kildekoden. I praksis medførte dette at det var minst fem komplette versjoner av kildekoden fordelt på minst tre fysisk forskjellige steder.

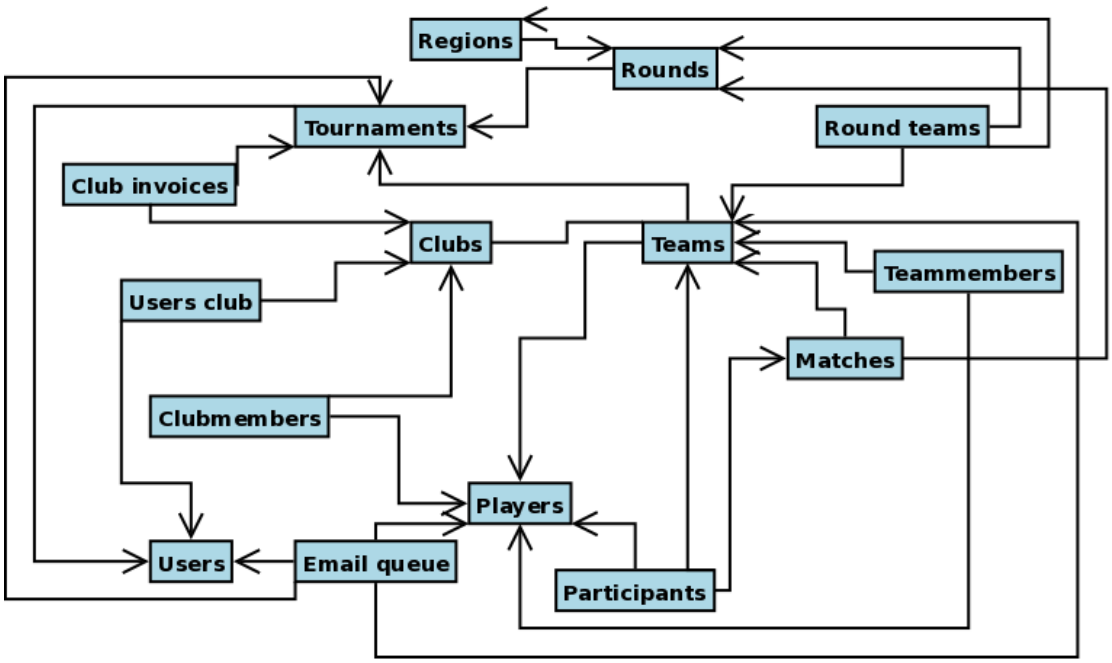
9.8 Konseptuell datamodell

All informasjon som BTS inneholder er lagret i tabeller i en relasjonsdatabase. En relasjonsdatabase kan bestå av en eller flere tabeller som logisk samler informasjon som er relatert til hverandre. En tabell består av en eller flere kolonner som spesifiserer hva slags informasjon som blir lagret, og hva slags format som blir brukt (tall, tekst, osv). Selve dataene i tabellen blir spesifisert som rader, hvor hver enkelt rad har mulighet til å lagre

informasjon som er definert i kolonnene. Et eksempel på dette kan være medlemsregisteret til Norsk Bridgeforbund. Selve medlemsregisteret (tabellen) inneholder en eller flere attributter; navn, adresse, telefonnummer og kontaktinformasjon (kolonner). Hvert enkelt medlem (rader) har et sett med verdier som defineres av kolonnene. Dette betyr at hvert medlem har registrert sitt navn, adresse, telefonnummer og kontaktinformasjon.

Det er også mulig å koble flere tabeller sammen slik at data fra mange tabeller kan bli slått sammen til en stor logisk struktur. Eksempelet kan utvides til å inkludere en oversikt over lisensinnbetalinger (tabell) hvert enkelt medlem har gjort. Denne tabellen kan inneholde informasjon om hvilket medlem som har betalt, når innbetalingen skjedde og beløpet som ble innbetalt (kolonner). For hver innbetaling mottatt (rader) vil nødvendig informasjon registreres. Ved å koble disse to tabellene sammen vil man få oversikt over et medlem og alle innbetalinger som er gjort av vedkomne.

I dette avsnittet vil det bli gjort en konseptuell gjennomgang av informasjonen som lagres i BTS og hvordan disse er koblet sammen. Attributtene til den enkelte tabell blir ikke nevnt med mindre dette er signifikant for å gi forståelse av hvordan disse brukes for å relatere til andre tabeller. Tabeller som kun har en hjelpefunksjon eller brukes til å lagre midlertidige data er ikke tatt med i denne oversikten. Figur 9.6 viser et grafisk diagram av modellen



Figur 9.6: En overordnet konseptuell datamodell som viser relasjoner mellom tabeller i databasen

Turneringsinformasjonen (Tournaments) er det øverste av elementene i hierarkiet og inneholder informasjon om hver enkelt turnering som brukes i BTS. Med et unntak er alle andre data er relatert direkte eller indirekte til en turnering. Unntaket er brukeren (Users) som er registrert som turneringsansvarlig.

Hver turnering består av en eller flere runder (Rounds). Hver runde inneholder informasjon om hvilket tidsrom runden ble avholdt og deltakeravgiften i hver enkelt runde. Hver runde kan bestå av en eller flere regioner (Regions) som kan brukes til å dele deltakende lag inn i logiske områder (for eksempel etter fylke). Hver runde består av en eller flere kamper som er

koblet mot de to deltakende lagene. Kampen kan også merkes som walkover hvis laget automatisk skal gå direkte til neste runde.

Klubbene (Clubs) er representert i en tabell som ikke er avhengig av en turnering. En klubb er den samme gjennom flere turneringer. Hver klubb kan delta med null eller flere lag (Teams) i hver turnering. Lagene er knyttet opp mot en klubb og en turnering. Dette betyr at et lag kun er gyldig en spesifikk turnering, i motsetning til klubben.

Spillere (Players) er, i likhet med klubber, uavhengige av en spesifikk turnering. Spillere blir registrert som lagmedlemmer (Teammembers) og som deltakere i en kamp (Participants). I tillegg kan en spiller være satt opp som kontaktperson for et lag. Hvis en spiller er medlem i en klubb kan dette også registreres (Clubmembers).

Brukere (Users) er entiteter som representerer personer som har opprettet en konto i BTS. Denne er skilt vekk fra spillere da en bruker av systemet ikke nødvendigvis er en spiller. En bruker som er tilknyttet en klubb (Users club) kan bli gitt ekstra tilganger for å administrere egen klubb.

All e-postutsending gjøres via en egen tabell (Email queue) som brukes til å lagre nye e-poster som skal sendes ut. Herfra vil et eget skript hente ut alle usendte e-poster og sende de til sine respektive mottakere. Sendte e-poster blir også lagret som en referanse til senere bruk. Det finnes også en referanse til turnering, lag, spiller og bruker som kan benyttes til å gi en indikasjon på en avhengighet til andre entiteter. For eksempel vil e-poster som sendes til en bruker med informasjon om en bestemt turnering inneholde en peker til både brukeren og turneringen. På denne måten vil det være enkelt i etterkant å hente ut alle sendte e-poster som er relatert til en bruker, spiller, lag eller turnering (eller en undermengde av disse kriteriene). Forslag til utvidelse av denne funksjonaliteten er beskrevet i oppsummeringen av BTS.

Som et alternativ til å differensiere mellom brukere og spillere ville det vært mulig å slå disse to sammen til en ny felles entitet. Dette ville gitt en enklere modell av databasen, men ville også økt kompleksiteten i oppbygningen av spilleren. En del av funksjonaliteten i BTS tilbyr en mulighet til å synkronisere medlemsdatabasen til Norsk Bridgeforbund opp mot spillertabellen i BTS. Ved å sørge for å skille mest mulig mellom brukere og spillere blir det mindre komplisert å synkronisere de to spillerdatabasene. En mulighet til å foreta et skille i en kombinert tabell ville vært muligheten til å benytte et “flagg” eller lignende felt som indikerte om raden var en spiller, bruker eller begge deler. Begge alternativene er fullverdige løsninger, men “separasjonsmodellen” ble valgt i denne oppgaven.

9.9 Sikkerhet

Dagens Internett har et ganske stort omfang, som dekker de fleste aspekter av den virkelige verdenen. Det finnes både farer og gleder på nettet som brukes av mange hundre tusen nordmenn hver eneste dag. De som gjør tjenester tilgjengelig på Internett må fokusere på begge aspektene; både sikkerhet og en god opplevelse for sine besøkende. I dette kapitlet blir forskjellige momenter som er viktig å fokusere på for tjenestetilbyderne. Det vil bli gitt en introduksjon til en del “vanlige” angrepsformer og hva som konkret er gjort for å sikre BTS mot denne type angrep.

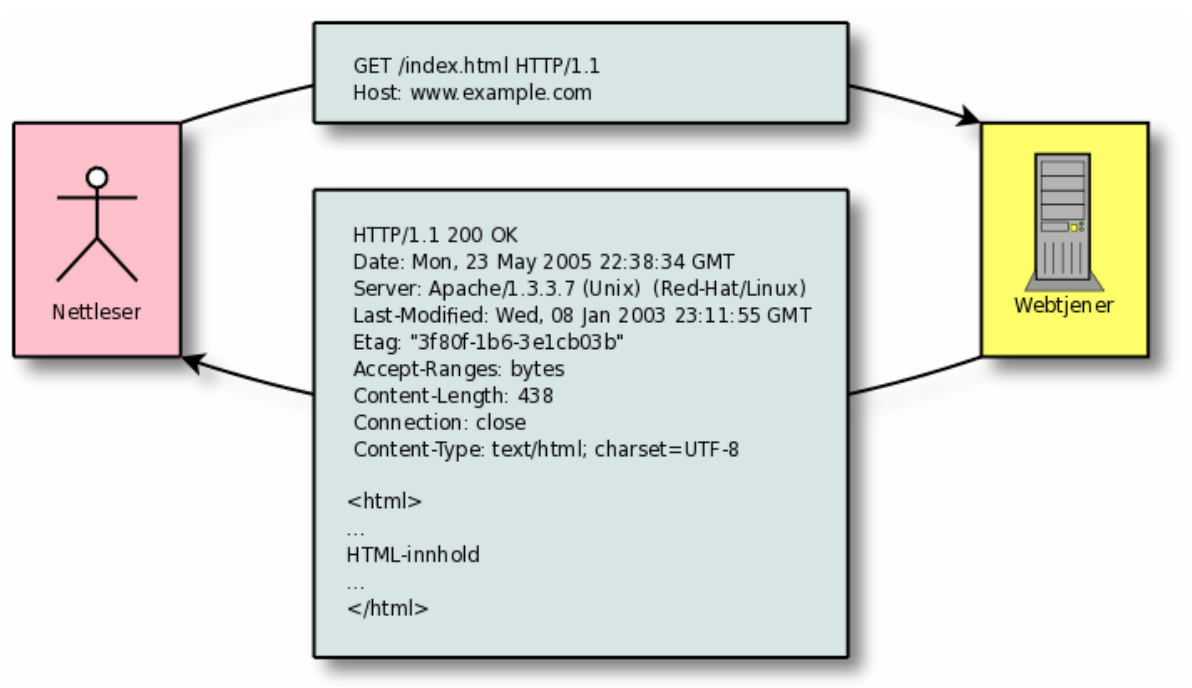
Bak et virtuelt angrep kan det stå et menneske eller en orm/trojaner/virus som forsøker å finne og utnytte svakheter. Et menneske har begrenset kapasitet til å angripe mange tjenester samtidig og vil som regel sikte inn på en eller få tjenester og kjøre spesialiserte angrep mot disse tjenestene. Ondsinnet kode som ormer, trojanere og virus er utstyrt med en

verktøykasse av sin skaper som inneholder bestemte metoder for å finne og systematisk utnytte feil. De fleste virus vil heller satse på kvantitet fremfor kvalitet. Disse vil tråle nettet for kjente svakheter og utnytte disse. Sammenlignet med mennesker, vil et virus være mindre intelligent, men har fordelen med at den kan raskt formere seg og spre sine angrep og håpe på statistiske lykketreff.

9.9.1 Hvordan data sendes mellom nettleseren og webapplikasjonen

De fleste websider av en viss størrelse er bundet opp mot en database i en eller annen form. Det kan være innholdet er lagret der, brukerautentisering, logging, brukergenerert innhold eller annen systematisert informasjon. Enhver webapplikasjon, hvor innholdet er lagret i en database, vil ha behov for å spesifisere hvilke data som hentes ut fra databasen og hvordan disse skal presenteres til sluttbrukeren. Det finner flere måter å sende denne identifikatoren mellom nettleseren til brukeren og webtjeneren. I noen tilfeller er det mest hensiktsmessig å sende denne identifikatoren som et argument i adressen. Vanligvis er dette i form av et tall eller en kombinasjon av tall/bokstaver.

Webtjeneren mottar en forespørsel fra nettleseren og behandler denne ved å sende en beskjed til webapplikasjonen om at brukeren ønsker å vise en bestemt side og angir hvilke parametere som er sendt av nettleseren (Figur 9.7). Disse parametrene brukes av webapplikasjonen til å for eksempel gjøre et oppslag i databasen for å finne ønsket innhold og sender dette tilbake til webtjeneren, som igjen sender resultatet tilbake til nettleseren. Parametrene som sendes er essensielle for at webapplikasjonen skal ha noen mulighet til å presentere riktig data til riktig bruker.



Figur 9.7: Eksempel på en HTTP-forespørsel som blir sendt en webtjener og svaret som sendes tilbake til brukeren

Selve forespørselen som sendes fra nettleseren inneholder en del informasjon; hvilken side nettleseren ønsker, hvilken protokoll som brukes, hvilke språk nettleseren støtter, eventuelle parametere sendt fra et skjema og annen informasjon som kan være relevant for tjeneren. [67]

Svaret som kommer fra tjeneren er delt i to; hodet og kroppen til svaret. Hodet inneholder informasjon om resultatet i form av statuskode, total lengde på innhold, hva slags programvare tjeneren bruker, hva slags type innhold osv. Selve innholdet vil i dette tilfellet være et ferdig HTML-dokument som er generert av webapplikasjonen. Det som skiller hodet fra kroppen er en tom linje. Se Figur 9.7 for eksempel.

For å vise informasjon om et enkelt lag behøver BTS en måte å identifisere det unike laget. Det samme er nødvendig for å identifisere en klubb. Registrering av et kampresultat trenger også en unik identifikator som peker til akkurat den ene kampen. Informasjon som dette er plassert i adressene til de enkelte sidene som en bruker får opp i nettleseren sin. For å vise informasjonen om for eksempel Arendals BK1 må nettleseren gå inn på http://eple.hib.no/bts/teams/viewteam/6906. Her er tallet 6909 en unik identifikator som identifiserer Arendals BK 1 i databasen. Ved å endre tallet 6909 til noe annet vil et annet lag bli vist på skjermen, hvis det finnes i databasen. Denne identifikatoren er lett synlig for brukeren og er enkel å endre.

9.9.2 SQL-injection

Under normal bruk vil programmereren som utviklet webapplikasjonen, vite hva slags data som blir sendt frem og tilbake. For eksempel vil en innloggingsside spørre brukeren om et brukernavn og passord. Dette vil så webapplikasjonen slå opp i databasen for å kontrollere om kombinasjonen er korrekt og vil utføre forhåndsbestemte oppgaver (hvis gyldig brukernavn/passord, vis velkomstside. Hvis ikke; vis påloggingsskjerm på nytt, sammen med en feilmelding). Et brukernavn består som regel av en samling bokstaver og muligens noen tall. Hvis programmereren implisitt går ut i fra at dette alltid er tilfellet, uten å kontrollere dette, vil en eventuell angriper ha en mulighet til å angripe applikasjonen. Dette kan gjøres ved å sende inn data som vil få applikasjonen til å oppføre seg annerledes enn det som er tiltenkt av utvikleren.

En utbredt, og potensiell veldig farlig metode, er SQL-injection [68] hvor angriperen plasserer egen kode i spørringen som kjøres mot databasen. Ved å sette inn sin egen modifikasjon av spørringen vil angriperen kunne utføre handlinger som utvikleren aldri hadde åpnet opp for under normal bruk i applikasjonen.

Et praktisk eksempel på hvordan dette kan misbrukes er påloggingseksemplet. Hvis man tar utgangspunkt i at webapplikasjonen kontrollerer en pålogging ved å sjekke om det finnes et brukernavn og et passord som er identisk med det brukeren har oppgitt, så tillates tilgang, hvis ikke nektes brukeren tilgang. En SQL-spørring som sjekker et brukernavn og passord mot databasen kan for eksempel se slik ut:

```
SELECT * FROM Users WHERE username = 'olsen' AND password = 'makrell'
```

Brukernavnet og passordet er hentet direkte fra forespørselen som er sendt fra nettleseren til webapplikasjonen. Hva hadde skjedd hvis angriperen visste strukturen på spørringen som ble kjørt og kunne manipulere denne? Angriperen kunne ført opp olsen' -- (med en enkel apostrof og to bindestreker) som brukernavn i innloggingsfeltet på websiden.

Enkle apostrofer (') brukes til å skille mellom SQL nøkkelord og vanlige tekststrenger. Doble bindestreker brukes til å indikere at resten av linjen skal ignoreres. Hvis angriperens

"brukernavn" kommer frem til databasen, blir følgende SQL-spørring kjørt for å verifisere brukerens identitet:

```
SELECT * FROM Users WHERE username = 'olsen' --' AND password = 'tilfeldige tegn'
```

Blir i utføring til:

```
SELECT * FROM Users WHERE username = 'olsen'
```

Ved å blindt stole på data fra brukeren vil en angriper lett kunne omgå passordsjekken i den tenkte webapplikasjonen. SQL-injection er ikke begrenset til innloggingskontroller, men alle typer spørringer som kjøres mot databasen. I tillegg kan det sendes spørringer som sletter, endrer eller oppretter informasjon i databasen som webapplikasjonen har tilgang til.

Det finnes to forskjellige måter å sikre seg mot slike angrep; den første, og anbefalte, metoden er å innføre preparerte spørringer (eng: prepared statements). Dette medfører at man først lager en spørring som inneholder referanser til variabler i stedet for å sende verdien av variablene direkte i spørringen. Et eksempel på en modifisert spørring som tar i bruk preparert spørring:

```
SELECT * FROM Users WHERE username = ? AND password = ?
```

Denne spørringen blir så sendt til databasen samt en liste over hvilke verdier som skal assosieres med hvert spørsmålstegn. På denne måten får databasen vite hva som er spørringen og hva som er inndata. Hvis en angriper hadde forsøkt å benytte olsen' -- som brukernavn i den nye spørringen ville databasen søkt etter en bruker med brukernavnet olsen' – i stedet for å tolke ' -- som en del av selve strukturen i spørringen. Preparerte spørringer har også den fordelen av at databasen kan optimalisere spørringen slik at den blir utført raskere. Dette gjelder spesielt i applikasjoner hvor en spørring kjøres flere ganger, men med forskjellige parametere. Preparerte spørringer blir opprettet på en annen måte enn vanlige spørringer.

Den andre metoden for å sikre spørringer er å filtrere inndata from brukeren, slik at alle spesialtegn blir beskyttet ved å plassere en \ før hvert spesialtegn. Ved å beskytte et tegn, spesifiseres det at tegnet skal behandles som inndata og ikke være en del av strukturen i spørringen. I ovenstående eksempel, vil angriperens forsøk resultere i følgende spørring:

```
SELECT * FROM Users WHERE username = 'olsen\' --' AND password = 'tilfeldige tegn'
```

Denne spørringen vil resultere i at databasen behandler olsen' – som en selvstendig variabel og ikke som en del av strukturen til spørringen. På samme måte bør passordet, som brukeren oppgir, bli beskyttet på samme måte som brukernavnet.

9.9.3 Inn- og utdatavalidering

All data som kommer fra brukeren bør sjekkes før bruk. Det finnes primært to måter å kontrollere inndata; hvit- og svartelisting. Med hvitlisting spesifiserer utvikleren hvilke tegn eller tallområder som er gyldig som inndata. Dette kan for eksempel være kun tegn og tall (alfanumerisk), kun tall (numerisk) eller hele alfabetet i tillegg til noen utvalgte tegn.

En svartelisting vil spesifisere hvilke tegn som ikke er lovlige og eventuelt fjerner disse fra dataene. Svartelisting av farlige data er ofte støttet i skriptspråk som brukes i forbindelse med webutvikling.

Et annet aspekt ved å validere data, er å sikre at data som sendes tilbake til brukeren ikke er skadelig. Dette kalles utdata-validering eller -filtrering. Dette foregår ved at data som for

eksempel er hentet fra databasen blir vasket, slik at HTML og/eller JavaScript blir uskadeliggjort før dataene sendes til brukeren. Dette kan for eksempel være en bruker som har fylt ut en kommentar for et produkt i en netthandel. Hvis brukeren sender inn HTML- eller JavaScript-kode i sin kommentar vil alle besøkende få denne koden, som kan gjøre alt mulig fra å sende brukeren til et annet nettsted, stjele sesjonen, bestille varer og lignende. Denne formen for angrep kalles Cross Site Scripting [69] (XSS). Ved å erstatte alle HTML-/JavaScript-kodesnutter med tilsvarende HTML-entiteter [70] vil man stoppe det meste av XSS-angrep som kommer fra en bruker.

9.9.4 Captcha

Captcha [71] er et akronym for *Completely Automated Public Turing test to tell Computers and Humans Apart.* Dette er en teknikk for å sile ut dataprogram fra mennesker. Dette gjøres ved å gi en visuell utfordring til brukeren, som et menneske noenlunde enkelt kan løse og som er veldig komplisert for en datamaskin. Slike teknikker brukes på mange websider, som tilbyr tjenester til mennesker og hvor det er uønsket at datamaskiner skal automatisk utføre handlinger. Et konkret eksempel på dette er e-posttjenestene Gmail [72] og Hotmail [73] som tilbyr e-post via nettleseren, og som må hindre misbruk av datamaskiner som sender ut spam. Under registreringen av en ny konto vil brukeren bli vist et bilde med tilfeldige tall og bokstaver som brukeren må skrive inn i et tekstfelt som sendes tilbake til websiden. Eksempler på hvordan en Captcha-utfordring ser ut er illustrert i Figur 9.8.

Figur 9.8: Illustrasjon som viser Captcha-utfordringer som blir brukt av tre ledende e-postleverandører

Det finnes en del variasjoner på hvordan en Captcha-utfordring kan være utformet; noen varianter tar for seg tekstgjenkjenning, andre er av litt annen visuell art: “Blant disse bildene av dyr, trykk på bildet av et pattedyr”. En del av tekstgjenkjennings-utfordringene bruker støy i bildene slik at OCR-gjenkjenning blir vanskelig, men for et menneske må det bare litt ekstra konsentrasjon til for å løse oppgaven.

En stor ulempe med visuelle Captcha er at de ikke er enkel for alle mennesker å løse. Disse kan være en utfordring for de som lider av forskjellige former for lesevansker, blindhet, nærsynthet, fargeblindet og lignende. Noen websider prøver å hjelpe på denne situasjonen ved å tilby en lydsnutt som skal skrives inn i et tekstfelt. Andre Captcha-utfordringer er så godt skjult/vridd at “normale” mennesker har store problemer med å tyde oppgaven korrekt. Hvis oppgaven er for vanskelig vil dette raskt kunne medføre at en del brukere ikke klarer å komme gjennom valideringsprosessen og sitter igjen med et dårlig inntrykk av tjenesten.

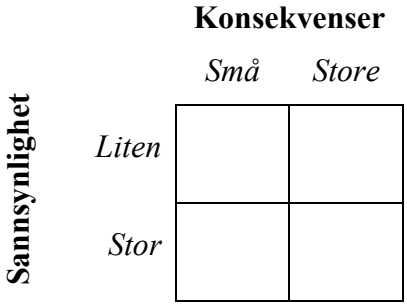
BTS har benyttet en visuell Captcha som presenterer en kombinasjon av tre alfanumeriske tegn. Disse skal repeteres i et tekstfelt under registreringsprosessen. Hvis oppgaven som brukeren ble presentert for var for uklar, har brukeren mulighet til å trykke på bildet for å få generert et nytt bilde med nye tilfeldige bokstaver og tall.

9.9.5 Risikoanalyse

Enhver utvikler som skal gjøre en tjeneste offentlig tilgjengelig bør tenke gjennom hvilke farer som truer integritet, tjenestekvalitet og personvern. Dette inkluderer en gjennomgang av hvilke trusler som blir møtt på det verdensomspennende Internett.

En risikoanalyse er en systematisk gjennomgang av hvilke farer som finnes, sannsynligheten for at de inntreffer og konsekvensene ved et inntreff. Ved å foreta en risikoanalyse ser man etter en positiv korrelasjon mellom fordeler og risikoene ved disse. En bevisstgjøring av risiko for et prosjekt vil kunne bidra til en nedgang av de negative konsekvensene. Selve risikovurderingen er en løpende prosess og kan sjeldent være komplett og endelig. I sin enkleste form kan risiko defineres som sannsynligheten for en gitt begivenhet multiplisert med konsekvensen av begivenheten.

Ved en systematisk gjennomgang av begivenheter som har negativ konsekvens, kan det være vanskelig å tallfeste konkret hvor stor sannsynlighet det er for at en begivenhet inntreffer. På samme måte kan det være vanskelig å kategorisere konsekvensene. Et konkret eksempel på hva som er enkelt å kvalifisere ved et større prosjekt er sannsynligheten for at en utvikler blir syk over lengre tid. Denne type statistikk kan være tilgjengelig på bakgrunn av sykefravær for avdelingen eller sykehistorien til den enkelte utvikler. Selv disse tallene kan ha usikkerhetsmomenter, men kan brukes som et utgangspunkt i en klassifisering. Konsekvensene ved et sykefravær vil avhenge av lengden til sykefraværet; noen dagers sykefravær til kunne medføre mindre forsinkelser i motsetning til en nøkkelutvikler som blir borte i flere måneder.



Figur 9.9: Eksempel på en enkel kategorisering av risiko. Sannsynlighet og konsekvenser er delt inn i to underkategorier

Som et hjelpemiddel i klassifisering av risiko kan en enkel målestokk brukes. For eksempel kan sannsynligheten deles i to kategorier; liten og stor sannsynlighet. Konsekvensene kan deles i to respektive kategorier; små og store konsekvenser. Dette kan presenteres som et tabell på to rader og to kolonner, hvor sannsynlighet og konsekvenser får hver sin akse (se Figur 9.9). Her er risikoen mer alvorlig jo lengre til høyre og lengre ned man kommer i diagrammet. Nederst til høyre finnes risikoer med stor sannsynlighet og store konsekvenser, noe man ønsker å unngå. I ovennevnte eksempel med sykefravær kan korttids sykefravær bli klassifisert som stor sannsynlighet med små konsekvenser. Langtids sykefravær vil kunne plasseres med liten sannsynlighet og stor konsekvens. I en reell risikoanalyse vil det nok være hensiktsmessig å benytte en mer detaljert klassifisering med flere nivåer.

I følge Computer Security Handbook [74] finnes det fire hovedgrupper for hvordan risiko kan håndteres; redusere konsekvenser, akseptere risiko, overføring av risiko og terminering av risiko. Ved å redusere konsekvensene endrer man miljøet som risikoen tilhører. Dette kan gjøres ved å endre sannsynligheten for begivenheten inntreffer eller minske skadeomfanget. Dette kan for eksempel være å innføre nye rutiner, bruke rammeverk eller definere ansvarsområder. Ved å akseptere risikoen foretar man et bevisst eller ubevisst valg om godene som risikoen medfører er verdt de eventuelle negative konsekvensene. Overføring av risiko kan brukes til å plassere et "ansvar" over på andre. Eksempler på dette inkluderer kjøp av forsikring, endre avtalevilkår, lovendringer osv. Den siste strategien fjerner sannsynligheten for at en gitt begivenhet skjer eller eliminerer de negative konsekvensene. Det er også mulig å kombinere disse fire forskjellige teknikkene for å oppnå ønsket resultat.

Ved utviklingen av BTS ble det foretatt en vurdering av miljøet som applikasjonen skulle kjøre i, hvilke data som skulle være tilgjengelig og om det var noe sensitiv informasjon som skulle lagres. Tilgjengeligheten over Internett medfører fare for angrep av både virus-/trojanere og mennesker. Det ble enighet om at det var liten sannsynlighet for et menneskelig angrep da BTS ikke er innenfor "sensitive" områder som politikk, religion, livssyn eller andre kontroversielle tema. "Pøbelfakter" kan ikke utelukkes, men blir ansett som liten sannsynlighet og middels konsekvens. Automatiserte program som ser etter svakheter tar ikke hensyn til ovennevnte kategorier og kan derfor ha større sannsynlighet for å inntreffe og har middels konsekvenser i form av opprydning.

De to største risikoene som ble vurdert var "spam-boter" og uriktige registreringer av resultat. Med spam-boter menes automatiserte program som tråler nettet for skjema som kan brukes til å sende uønsket reklame. Disse spam-botene er livlige besøkende på gjestebøker og blogger, hvor de sender inn linker til forskjellige "gode" tilbud på nettet. Noen av innleggene prøver å imitere et menneske som anbefaler produktene som er nevnt. I enkelte tilfeller blir fraser fra bloggposten inkludert i teksten for å øke troverdigheten for at teksten ble skrevet av et menneske. Slike boter vil i verste fall kunne registrere feilaktige kampresultat og melde på lag med navn som sikter til billige utenlandske medikamenter og lignende. Den andre risikoen som ble vurdert var tilfeller hvor noen ville registrert et feilaktig resultat etter en kamp.

I begge tilfellene ble det valgt å minske sannsynligheten for at begivenhetene inntraff. For å hindre sannsynligheten for at spam-boter sendte inn skjema, ble Captcha innført ved registrering av brukere. I tillegg er det kun mulig å registrere et lag for en bruker som er logget inn i systemet (og dermed indirekte har bestått Captcha-testen). Hvis en spam-bot skulle komme seg forbi denne hindringen blir det en liten manuell opprydningsjobb for administratoren av turneringen. En mulighet for å senke sannsynligheten ytterligere kan være å bytte ut Captchaen med en mer komplisert form (flere tegn, annen type Captcha osv).

All informasjon som lagres om spillere og klubber er allerede fritt tilgjengelig på NBF Data [75]. I sin natur skal BTS være et åpent system som viser statusen til NM for klubblag. Det er kun endring av data som krever en identifisering av bruker og tilhørende autorisasjon.

9.10 Ytelse

Ytelsen til websiden som brukeren besøker er en viktig del av den totale brukeropplevelsen. Det finnes en grense ved ca 1/10 sekund responstid som markerer hva brukeren opplever som umiddelbar reaksjon. Ved ca 1,0 sekund vil brukerens "mentale arbeidsflyt" bli forstyrret. Hvis responstiden passerer 10 sekunder begynner brukeren å bli utålmodig og vil gjerne se

etter andre oppgaver å gjøre i ventetiden [76]. Dette vil si at det er behov for å ha en grense på litt i underkant av ett sekund for hva som er akseptabel ventetid før siden blir presentert på brukerens skjerm.

Fra brukeren trykker på musen og tjeneren mottar forespørselen etter informasjon må dataene gjennom en del flaskehalser. I følge Jakob Nielsen [77] er første flaskehals ytelsen på tjeneren og databasetjeneren. Med dagens priser på maskinvare er ikke dette nødvendigvis den største flaskehalsen, selv om det er mulighet for at utviklerne av webapplikasjonen ikke har optimalisert databasespørringer eller bruker unødvendig tunge datastrukturer. Etter generering av utdata som skal sendes brukerens nettleser må disse sendes over Internett-forbindelsen til tjeneren. Hvis denne er underdimensjonert i forhold til tjenerens trafikk-mønster, vil brukeren oppleve treg responstid. Selve transporten gjennom Internett kan gå via mange tjenesteleverandører, land, kommunikasjonsformer og kontinenter. Her kan det være skjulte flaskehalser som ikke er lett å måle uten å ha spesialisert måleutstyr. Når datapakkene kommer frem til brukerens Internettforbindelse, vil denne også virke dempende på ytelsen hvis forbindelsen brukes til flere ting samtidig som for eksempel filnedlasting, streaming av video og sending av store vedlegg i e-poster. Til slutt vil hastigheten på brukerens nettleser avgjøre hvor raskt siden kan presenteres på brukerens skjerm. De fleste nyere nettlesere er ganske flinke på dette området, men hvis websiden inneholder store tabeller vil dette ha en negativ effekt på lastetiden.

Det er viktig å merke seg at forsinkelsen i de forskjellige ledd akkumulerer, og at det er den totale ventetiden som brukeren opplever i sin praktiske bruk av webapplikasjonen.

Som et ledd i utviklingen av BTS ble det foretatt en test av lastetiden på konkrete enkeltsider som brukes ofte i applikasjonen. Denne testen ble foretatt mot to forskjellige oppsett; en hvor både databasetjeneren og webapplikasjonen var på samme maskin og en versjon hvor databasetjeneren var plassert geografisk adskilt fra selve webapplikasjonen. En middels god Internettforbindelse bandt sammen de to delene. På denne forbindelsen var det en stabil forsinkelse på ca 40 millisekunder. Bakgrunnen for å dele testene i to forskjellige tilfeller var for å se hvor stor betydning det har for databasetjeneren å være på samme lokale nettverk som webtjeneren, eller aller helst på samme maskin. Større og tyngre applikasjoner vil ha fordel av å ha en dedikert databasetjener som kjører på sin egen tjener.

Seks forskjellige sider ble valgt som testgrunnlag. Disse ble valgt etter to kriterier; kompleksitet og "popularitet" blant brukerne. Både enkle sider med få spørringer og store sider med mange spørringer ble valgt. De valgte sidene ble brukt opp mot det samme datagrunnlaget i begge testgruppene. For å gi en mest mulig jevn måling ble hver side lastet inn på nytt raskt etter hverandre for å gi fordelene ved mellomlagring av metadata i CakePHP-rammeverket og eventuell optimalisering av spørringene i databasetjeneren.

De valgte sidene i utvalget var; førstesiden som møter brukeren, oversikten med påmeldte lag, trekningslisten, oversikten med siste resultat, listen med kretsvise lag som vises til turneringsansvarlig og en opplisting av alle kampene i første runde av NM for klubblag 2007/2008.

CakePHP vil som en del av rammeverket regelmessig kjøre spørringer hvor den ber databasetjeneren om å beskrive de aktuelle datatabellene som kommer til å bli brukt i utføringen av resterende spørringer. Disse spørringene lagres i CakePHP slik at den husker databasestrukturen og dermed ikke er behov for å kjøre ved hver sidelasting. Når CakePHP kjører i utviklingsmodus oppdateres denne etter ca fem sekunder. I utviklingsmodus skrives det ut en oversikt over hvilke spørringer mot databasen som kjøres, samt hvor lang tid hver

enkelt spørring tok. Det er denne modusen som er benyttet under testen. I produksjonsmodus blir denne informasjonen mellomlagret over lengre tid og det vil derfor ikke få noen negativ effekt på ytelsen.

Utenom dette var CakePHP satt opp til å spesifisere tegnsettet UTF-8 for hver database-forbindelse som opprettes. Dette gjøres for å sikre at tegnsettet som blir brukt til lagring av data holder seg konsist. Hvis forskjellige tegnsett brukes i databasen og webapplikasjonen, og hvor det ikke forekommer korrekt konvertering, vil særnorske tegn bli erstattet av kryptiske tegn. Dette vil raskt kunne gi brukeren et inntrykk av at noe er feil. Hvis BTS blir satt opp med en tjener som kun benytter UTF-8 som standard er det mulig å enkelt fjerne denne type spørringer for hver side.

		Fjern		Lokalt	
	# spørringer	Genereringstid Totalt	Genereringstid Innhold	Genereringstid Totalt	Genereringstid Innhold
Forsiden	2	437 ms	191 ms	14 ms	8 ms
Påmeldte lag	3	924 ms	391 ms	29 ms	16 ms
Trekningsliste	24	1 145 ms	448 ms	40 ms	23 ms
Siste resultat	7	2 135 ms	1 438 ms	104 ms	87 ms
Kretsvise lag	3	1 517 ms	820 ms	49 ms	32 ms
Kamper i runde	4	1 346 ms	772 ms	42 ms	28 ms

Figur 9.10: Antall spørringer som kjøres for å generere websidene

Målingene fra "lokalt" oppsett av database og webtjener er sannsynligvis mest relevant i reelle situasjoner hvor BTS skal implementeres. De fleste webhotell og profesjonelle hosting-firma plasserer database og webtjener så nær hverandre som mulig for å oppnå best mulig ytelse. Sammenligningen opp mot "fjernoppsett" vil derfor ikke være fullt så relevant i forhold til praktisk bruk, men heller som en dokumentasjon på fordelene ved å plassere database- og webtjener nær hverandre. Figur 9.10 Viser antall spørringer som kjøres for å generere websidene. Kolonnen "fjern" inneholder genereringstiden for et oppsett hvor web- og databasetjener er adskilt. "Lokalt" representerer et oppsett hvor disse to tjenerne er på samme datamaskin. Genereringstiden for innholdet forteller hvor lang tid det tok å hente innholdet til websiden.

Som vist i Figur 9.10, er genereringstiden for de respektive sidene godt under et sekund. Dette er ikke medregnet tid som brukes for å sende websiden gjennom nettverket. Kun tiden CakePHP bruker for å hente data fra databasen og presentere dette i HTML-form klart til transmisjon.

Ved bruk av separat lokalisering av database- og webtjener økte genereringstiden markant. Her var korteste genereringstid litt i overkant av 0,4 sekunder. Dette gav merkbar treg respons i nettleseren og i verste tilfelle tok det over 2 sekunder for å laste en side. Websiden som viser siste kampresultat var testens tyngste side målt i genereringstid. Med en nettverksforsinkelse på 40 ms ble ytelsen over 20 ganger dårligere for denne konkrete websiden i en situasjon hvor database- og webtjener er fysisk separert over lengre avstand.

Når det gjaldt antall spørringer som ble utført for hver side var trekningslisten den websiden som brukte flest spørringer; over tre ganger så mange spørringer som nummer to på listen. Basert på utføringstiden og struktur på spørringene kan disse spørringene klassifiseres som små og raske i utførelsen. Det er lite data som overføres etter hver spørring og det gjør det mulig å kjøre mange små spørringen på samme tid som få større spørringer.

CakePHP bruker et abstraksjonslag mellom modell og databasetabell. Dette gir en fordel ved at spørringer som brukes av BTS ikke er knyttet mot bestemte databasetabeller, men mot CakePHPs modell. Dette medfører at det er mulig å endre attributter på en tabell i databasen og gjøre tilsvarende endring i datamodellen så vil det gjøre BTS klar for den nye databasemodellen. På denne måten slipper utviklerne å hardkode SQL-spørringer direkte mot databasen. Selve spørringene som må opprettes av utviklerne vil relatere seg med modellene som CakePHP er satt opp med. CakePHP vil igjen ta denne spørringen og konvertere denne til en vanlig SQL-spørring basert på modellen som er registrert. Dette medfører en del konverteringer og til dels kunstige spørringer enkelte steder. I BTS er det fokusert på å sørge for at det skal være en best mulig kompatibilitet mellom BTS og CakePHP slik at en eventuell databaseendring ikke medfører omskrivning av alle spørringer.

10 Noen utvalgte problemstillinger

10.1 Internet Explorer og standarder

Microsofts Internet Explorer (IE) [78] er kjent for sin manglende støtte for standarder. De mest kjente feilene er feilberegning av bredde i objekter [79][80]. En oversikt over andre kjente feil finnes også som et hjelpemiddel for webutviklere [81][82]. IE er ikke den eneste nettleseren som inneholder feil i forhold til standarder, men er desidert den nettleseren som krever mest arbeid for tilpasninger. De fleste feil er relatert til HTML-standardene, CSS eller JavaScript.

I utviklingen av BTS ble Firefox benyttet som primær nettleser. Etter ny funksjonalitet var ferdig utviklet ble den testet i IE. Hvis funksjonaliteten ikke fungerte eller visningen av siden var feil, måtte "feilen" rettes slik at det ble korrekt vist i IE og andre nettlesere.

Det som er viktig å passe på under feilretting, er at standardene fremdeles følges, slik at ved fremtidig versjon av IE og andre nettlesere som følger standardene viser siden på korrekt måte.

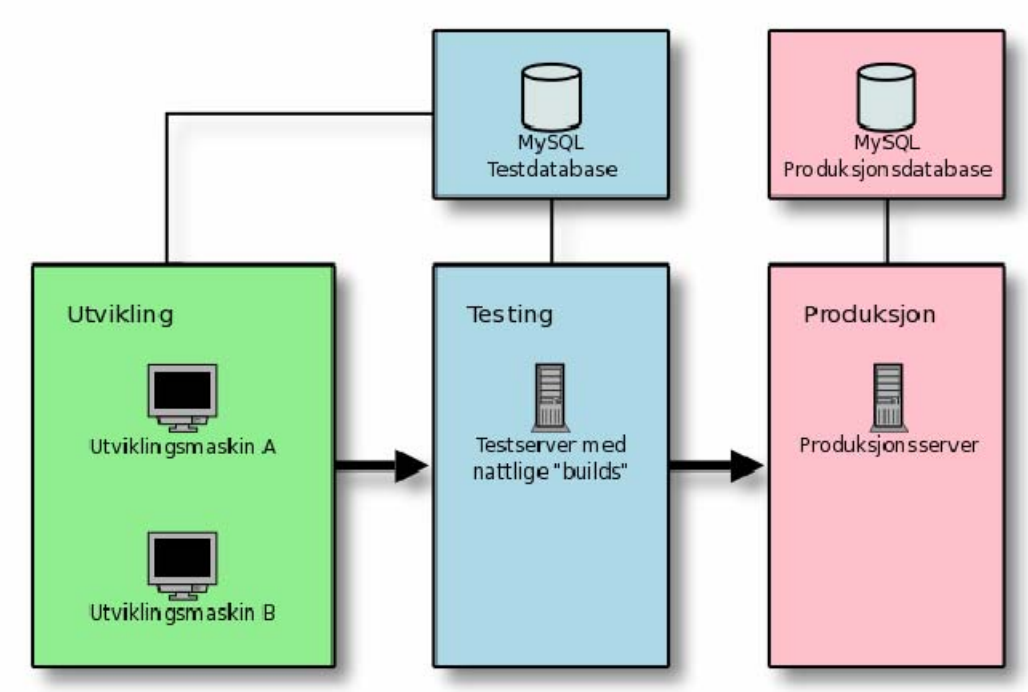
De mest populære nettleserne har to forskjellige modi for å sikre mest mulig kompatibilitet; "quirks mode" og "standards mode". *Standards mode* betyr at nettleseren skal følge standarden som er spesifisert i toppen av HTML-dokumentet. *Quirks mode* er en mellomting hvor nettleseren tillater en del feil og vil gjerne vise siden på samme måte som en tidligere versjon av samme nettleser. Det som er felles for de fleste nettlesere, er at de automatisk går i *quirks mode* hvis det ikke eksplisitt er definert hvilken standard dokumentet følger. For en utvikler som vil at nettleseren skal følge standarden er derfor nødt til å spesifisere hvilken standard som følges. Oppdager en nettleser flere feil, mens den opererer i *standards mode*, vil den automatisk hoppe over til *quirks mode*. Terskelen for å velge standards- og quirks mode er forskjellig mellom nettleserne [83]. I BTS er det spesifisert at nettleseren skal følge XHTML 1.0 Transitional-standarden.

Et konkret eksempel på en utfordring med IE var måten tekstmarkering blir gjort. Opprinnelig ble BTS designet til å bruke lag (eng: layers HTML: <DIV>) til å gruppere menyen for seg, innholdet for seg og tittelfeltet i et eget lag. CSS ble brukt til å plassere de forskjellige lagene korrekt på skjermen. Visuelt så dette fint ut i de fleste nettlesere, men hvis en bruker som brukte IE prøvde å markere noe tekst på skjermen ville hele dokumentet bli markert. Dette medførte at klipp-og-lim ikke fungerte som tiltenkt. For å løse problemet ble en tabell brukt, som plassere elementene der de hørte hjemme på skjermen, i stedet for å benytte lag. Denne løsningen følger standardene, men det er ikke ansett som en god løsning siden presentasjon og innhold blandes.

11 Evaluering

11.1 Testmetodikk

I utviklingen ble to forskjellige databaser benyttet til å separere testdata og de reelle dataene som ble brukt i selve turneringen som NBF holdt. Dette ble gjort for å hindre at test av systemet ikke medførte feilaktige oppføringer i systemet som var i drift og brukt av spillere som deltok i turneringen. I sluttfasen av utviklingen ble produksjonsdataene kopiert over til testdatabasen for mest mulig realistisk testing.



Figur 11.1: Viser tredelingen som ble brukt under utvikling av kildekoden

I programkoden ble tre nivåer benyttet for å skille systemet i produksjon fra koden som ble endret på daglig basis. Denne tredelingen er illustrert i Figur 11.1. På datamaskinene brukt til utvikling var det satt opp webtjenere som ble brukt lokalt og en tilkobling til databasen med testdata. Etter endringene gjort i koden lokalt på datamaskinene, ble disse testet og sendt til versjonskontrolltjeneren. Regelmessig ble lokal kode oppdatert opp mot den sentrale lagringen. På denne måten var alltid lokal kode oppdatert med siste versjon av koden.

Hvert døgn ble den komplette og oppdaterte koden automatisk plassert på en tjener benyttet til testing. Denne installasjonen ble brukt til å teste hvordan systemet fungerte utenfor utviklingsmiljøene. Det ble tilstrebet å la testtjeneren være konfigurert på en slik måte at den mest mulig simulerte hvordan oppsettet ville være på en produksjonstjener. Denne versjonen av programmet brukte også testdatabasen slik at tester kunne gjennomføres slik at eksempelvis endret kampresultat og lignende ikke ville være synlig for sluttbrukeren.

Etter en manuell test av versjonen som var plassert på testtjeneren, ble hele katalogstrukturen flyttet over på produksjonstjeneren. På dette stadiet var det begrenset for hvor dyptgående tester som kunne gjennomføres uten å manipulere reelle data. Det ble stort sett bare gjennomført visuelle tester og testing av alle linker førte frem til reelle sider. På dette stadiet

ble svært få feil funnet som skyldes feil i kildekoden. De fleste feil som ble funnet på dette stadiet var relatert til endringer i databaseskjemaet.

Siden BTS ble satt i produksjon mens det framdeles var under utvikling, oppstod det en ekstra "testmetode"; Brukerne av systemet. Helt fra systemet ble satt i drift, ble tilbakemeldinger fra brukere mottatt. Disse bidro med kommentarer til design, brukernes egne erfaringer og generelle tilbakemeldinger.

11.2 Erfaringer

11.2.1 Valg av tekniske løsninger

Da prosjektet ble startet var et av valgene hvilke tekniske løsninger som skulle benyttes. Etter å ha sjekket mot NBF ble det avklart at de brukte PHP og MySQL, så valget falt naturlig på disse plattformene. Etter en gjennomgang av hjemmesidene til de respektive løsningene ble det bestemt at produksjonsversjonen av PHP, versjon 5, skulle brukes som skriptspråk. Forrige versjon av PHP, versjon 4, var satt opp til å nå enden av sin livssyklus ved slutten av 2007. Likeledes falt valget på MySQLs produksjonsversjon, versjon 5.

Når tiden kom for å begynne planlegging for å flytte over systemet til NBF sin server, viste det seg at opprinnelige valg hadde vært litt optimistiske med tanke på hvor oppdaterte hosting-selskapet holdt serverne sine. Det viste seg her at serveren til NBF kjørte både PHP 4 og MySQL 4. Det var muligheter for oppgradering av serveren her, men ikke før 2. kvartal 2008.

Porteringen av BTS til PHP 4 var triviell, på grunn av god bakoverkompatibilitet fra versjon 5 til 4. MySQL versjon 5 var ikke bakoverkompatibel med versjon 4 hvis funksjonalitet kun tilgjengelig i versjon 5 var benyttet. "Lagrede spørringer"-funksjonaliteten gjorde det mulig å lage et abstraksjonsnivå over enkelte tabeller. For å gjøre BTS bakoverkompatibel med MySQL 4 krevde dette store omskrivninger av delmoduler til å ikke benytte disse. På grunn av dette store arbeidet ble det besluttet at BTS ikke skulle være kompatibel med MySQL 4. Etter planen skulle NBFs servere oppgraderes i løpet av andre kvartal 2008.

Som en mellomløsning bestemte utviklerne seg for å stille seg til disposisjon for NBF til å hjelpe med overflytting av BTS til NBFs servere etter oppgraderingen av tjeneren.

11.2.2 Utvikling under produksjon

Som tidligere beskrevet i rapporten, oppstod den situasjonen at programmet ikke var ferdigstilt til datoen det skulle igangsettes. Det var da to valg; enten måtte NBF gjennomføre turneringen på tradisjonell måte ett år til, noe som ville medført at mange verdifulle tilbakemeldinger fra brukerne ikke kunne brukes for å bedre systemet underveis. Eventuelt kunne systemet innføres i den tilstand det var i, og sette fokus på å utvikle de delene av systemet som trengtes først.

Etter å ha konferert med NBF, falt valget på å sette systemet i drift selv om dette kunne medføre ekstraarbeid. Dette medførte at konsentrasjonen ble på de funksjonene som det var behov for der og da.

11.3 Besøksstatistikk

BTS logger alle henvendelser i en loggfil som inneholder informasjon om hver enkelt forespørsel. Denne informasjonen kan så brukes til å identifisere feil og generere statistikk. Følgende informasjon lagres om hver eneste forespørsel:

- IP-adressen til den besøkende.
- Tidspunktet forespørsel ble mottatt.
- Hvilken side som ble etterspurt.
- Statuskoden for forespørselen (siden ikke funnet, en intern feil har oppstått, alt OK, osv).
- Hvilken side som brukeren kom i fra (hvis nettleseren sender denne informasjonen).
- Hvilken nettleser som ble brukt (eventuelt søkerobot).
- Størrelsen på siden ble generert.
- Tiden det tok for å generere siden.

Ved å benytte AWStats til å analysere loggfilen ble det generert interessant statistikk for hvor mange brukere som har besøkt websiden og et estimat for hvor mange unike besøkende som har sett på siden. Det er tre forskjellige begrep som brukes for å beskrive trafikkmønsteret. Et treff defineres som en enkelt forespørsel gjort mot tjeneren. Dette kan være en forespørsel om å hente en side, et bilde eller en annen fil. Et besøk består av flere treff gjort av en klient i et avgrenset tidsrom. Et unikt besøk er antallet unike besøkende uavhengig av hvor mange besøk hver bruker har utført.

Det finnes usikkerhetsmoment i generering av estimat av unike besøkende og "vanlig" besøk. For å skille mellom besøkende er IP-adressen den største identifikatoren som kan benyttes til å skille brukere fra hverandre. En IP-adresse peker ikke nødvendigvis mot en enkel bruker, men mer mot en datamaskin. Hvis en person besøker en bestemt side fra både hjemmedatamaskinen og fra en PC på arbeidsplassen, vil dette fremstå som to forskjellige "brukere" i loggen. På denne måten kan antall unike besøkende bli overrepresentert.

Men det finnes også tilfeller hvor flere forskjellige personer deler samme IP-adresse, dette kan for eksempel være gjennom en proxy eller en delt datamaskin. En offentlig datamaskin på et bibliotek vil for eksempel beholde samme IP-adresse selv om det er mange forskjellige mennesker som har benyttet datamaskinen i løpet av en dag. En proxy er en tjeneste som samler flere brukere bak en felles IP-adresse og tar seg av all kommunikasjon, som en slags mellommann som sørger for trafikken fra brukerne sine blir anonymisert. Hvis en proxy er blitt benyttet av en gruppe brukere, vil antallet besøkende og unike besøkende ikke representere det reelle antallet. Estimatene vil da være lavere enn det reelle tallet.

Dette er usikkerhetsmoment som er vanskelig å estimere. Tallene i dette kapittelet har ovennevnte usikkerhetsmomenter.

Måned	Unike besøkende	Antall besøk	Treff	Datamengde
november 2007	2 198	5 917	36 311	869 MB
desember 2007	1 407	4 756	26 966	944 MB
januar 2008	1 688	7 005	33 892	1 360 MB
februar 2008	1 357	4 258	21 149	772 MB
mars 2008	1 461	4 304	21 972	132 MB
april 2008	1 570	4 144	17 621	111 MB

Figur 11.2: Viser trafikkmengden BTS hadde under de mest aktive månedene i gjennomføringen av NM for klubblag

Figur 11.2 viser hvor mange besøkende som var innom BTS i løpet av november til april. Den økte mengden besøkende i november er relatert til overgangen mellom runde en og to. På dette tidspunktet ble det bestemt hvilke beste tapere som gikk videre til runde to. Fra desember og utover var det jevnt antall unike besøkende. Med dette datagrunnlaget, tilsvarer dette er dagsgjennomsnitt på ca 53 unike besøkende som sammen hadde ca 167 besøk.

Datamengden som ble overført var moderat i forhold til innholdet på BTS. Et gjennomsnittlig treff medførte en nedlastingsmengde for brukeren på ca 27 KB. I midten av februar 2008 ble det gjort optimaliseringer i systemet, blant annet komprimering av utdata, noe som medførte at datamengden gikk ned betraktelig, selv om innholdet som ble servert var det samme. Nedgangen i datamengde vises tydelig i Figur 11.2.

Et annen interessant observasjon er de tilnærmet daglige indekseringene gjort av søke-robotene til Google [84], Microsoft [85] og Yahoo [86]. Disse robotene søker gjennom Internett og oppdaterer sine databaser som brukes i de respektive søkemotorene. Besøkende som kommer via en søkemotor sender som regel en referanse til hvilke søkeord som er brukt for å finne resultatet. De mest brukte søkeordene som gav resultat med BTS i listen var *bk*, *bridge* og *nm*. I tillegg var det overraskende mange som hadde søkt etter navnene til privatpersoner.

Ved å bruke informasjonen som sendes av nettleseren er det mulig å se i enkelte tilfeller fra hvilken side en bruker kommer fra. Dette gav et overraskende overblikk av hvor mange eksterne sider som hadde lenket videre til BTS. I noen tilfeller var også oversikter kopiert over til klubbers private hjemmesider. Denne type viderebruk var en positiv overraskelse og viste en stor interesse for tjenestene som BTS tilbyr.

AWStats tilbyr også en mulighet for å estimere hvor lenge et besøk varte. Dette blir generert ved å ta summere lengden av hvert besøk. Dette kan regnes ut ved å inspisere tidspunktene treffene ble utført basert på hvert enkelt besøk.

Lengde	Antall besøkende	
0s-30s	10 511	52,4%
30s-2min	4 729	23,6%
2min-5min	2 419	12,1%
5min-15min	1 148	5,7%
Andre	1 244	6,2%

Figur 11.3: Viser gjennomsnittlig lengde en besøkende var på BTS

Som Figur 11.3 viser, så forlot størstedelen av besøkende (76 %) BTS før to minutter var gått siden første sidevisning. Besøk som varte lenger enn én time kan skyldes nettlesere som er satt opp til oppdatere en siden ved regelmessige intervall. Disse ville regnes som besøk som varer over et lengre tidsrom.

Nettleser	Treff	
MS Internet Explorer	82 140	85,7%
Firefox	10 900	11,4%
Safari	1 219	1,3%
Opera	806	0,8%

Figur 11.4: Viser andel nettlesere som ble brukt til å vise BTS i januar-april 2008

Operativsystem	Treff	
Windows	92 369	96,4%
Macintosh	1 768	1,8%
Linux	961	1,0%

Figur 11.5: Viser andel operativsystem brukt i perioden januar-april 2008

Hvilke operativsystem og nettleser avgjør hvilken støtte som finnes på brukerens datamaskin. Nettleseren er den primære kilden til feil visning av websider. Det er nettleseren som bestemmer hvordan presentasjonen på skjermen skal se ut. Hvilke standarder som støttes av nettleseren er også avgjørende for hvordan brukeropplevelsen er. Figur 11.4 og Figur 11.5 viser en oversikt over hvilke nettlesere og operativsystemer de besøkende benyttet seg av. I utviklingen ble Firefox benyttet som referanse-nettleser for å kontrollere at BTS fulgte standardene best mulig. En del tilpasninger måtte gjøres for å sikre god presentasjon i Microsoft Internet Explorer (IE). Internet Explorer 6 ble lansert i siste halvdel av 2001 og siden den gang er det kommet flere nye standarder. Arvtakeren, Internet Explorer 7, ble lansert i slutten av 2006. Fremdeles er det en stor andel brukere som benytter seg av IE 6.

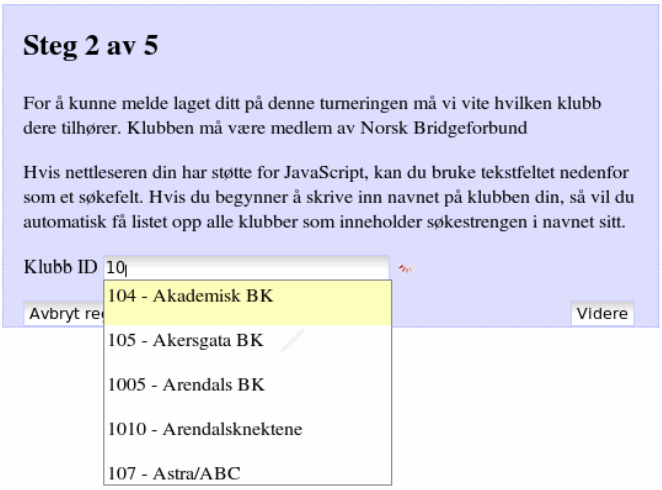
Andelen IE 6 og IE 7-brukere er fordelt i like store deler (respektiv 42,1 % og 43,3 % "markedsandel") basert på analyse av besøksloggen. Dette medførte at også IE 6 måtte testes for å sikre best mulig kompatibilitet blant de største nettleserne. Under utviklingen ble både Windows og Linux brukt som testplattformer, og nettleserne IE 6, Firefox [15] og Opera [87] ble benyttet til å kontrollere opptegningen av BTS i nettleserne. En visuell test av komponentene viste at det er minimal forskjell i visningen av BTS i de testede nettleserne. Tjenesten Browsershots [88] ble også benyttet for å teste den visuelle fremstillingen. Browsershots er en tjeneste som tilbyr en mulighet til å få skjermbilder av en spesifikk webside i forskjellige nettlesere (og versjoner av nettleserne), operativsystem, oppløsning og fargedybde.

11.4 Tilbakemelding fra brukerne

I løpet av de første innledende rundene av NM for klubblag ble litt i overkant av 100 e-poster mottatt fra brukerne av systemet. Dette vil si perioden september til desember. Utenom disse, var det kontakt med Fagsjef i Norsk Bridgeforbund, Harald Skjæran, som var prosjektets kontaktperson hos NBF. Skjæran hadde det administrative ansvaret for gjennomføringen av NM for klubblag. Han kom med både ønsker og tilbakemeldinger på hvordan systemet fungerer fra en turneringsansvarligs synspunkt.

En god del av henvendelsene fra vanlige brukere omhandlet bruk av valideringskoder ("PIN-koder" som brukes til resultatregistrering) og om man fremdeles skulle sende inn resultat via Posten; om valideringskodene kunne sendes ut på nytt; om hva som skulle gjøres hvis motstander ikke hadde mottatt valideringskoden. På de spørsmålene hvor svarene var kjent, ble svar sendt direkte til brukeren. De spørsmålene som omhandlet rutiner og generelle spørsmål om turneringen ble videresendt til NBF for behandling der.

Når det gjaldt tilbakemeldingene rundt opplevelsen av BTS, så varierte dette fra noen få negative tilbakemelder til positive og konstruktive e-poster. På forhånd ble det mistenkt at det kunne komme noen henvendelser på bruken av JavaScript og Ajax, på grunn av varierende støtte blant eldre nettlesere.



Figur 11.6: Skjermbilde som viser et steg i påmeldingsveiviseren. Søkefunksjonaliteten blir vist som et "nedtrekksvindu"

I den interaktive påmeldings-veiviseren (Figur 11.6) ble Ajax benyttet til å hente forslag til klubb når brukeren begynte å skrive inn klubbnummeret eller klubbnavnet. På denne måten fikk brukeren mulighet til å søke opp klubben sin hvis klubbnummeret var ukjent. Denne funksjonaliteten ble satt pris på av brukerne.

Det var tre tilbakemeldinger angående bruk av linjeskift-tasten sammen med skjemaene som ble brukt under registreringen. Her ville veiviseren gå videre hvis man benyttet linjeskift-tasten for å velge et forslag til innhold via bruk av Ajax. Dette var i følge tester bare et problem med Internet Explorer. Det ble ikke funnet en endelig løsning på dette problemet da JavaScript-implementeringen i Internet Explorer ikke følger en utstrakt standard og manglende dokumentasjon på dette området. Dette er nærmere beskrevet i kapittel *10.1 Internet Explorer og standarder*.

12 Konklusjon og videre arbeid

12.1 Oppsummering

BTS har under NM for klubblag 2007/2008 hele tiden vært i produksjon og fått ny funksjonalitet underveis. Helt fra starten hvor en begrenset gruppe klubber deltok i registreringen til turneringen har det vært jevn progresjon i implementeringen av ny funksjonalitet som det har vært behov for under turneringens fremdrift. Under hele utviklingsprosessen ble det mottatt tilbakemeldinger fra brukerne, og rapporterte feil ble rettet. Dette gav verdifull tilbakemelding på hvordan ting kunne implementeres på en annen måte i brukergrensesnittet.

De mest fundamentale funksjonene som det er behov for i en turneringsavvikling ble implementert; registrere lag, resultatregistrering og publisering av trekningsresultater. I tillegg til dette ble det lagt til ekstra funksjonalitet som er spesifikt for denne konkrete turneringen; forbundspoeng og deltakere i en kamp. Kontaktpersonen hos NBF har også kommet med ønsker til rapporter som er etter beste evne forsøkt å implementere. Noen av de opprinnelige ønskene fra oppdragsgiver (reiseregninger og lignende) falt utenfor tidsrammen som var tilgjengelig for denne delen av prosjektet. Disse punktene er nærmere spesifisert i avsnittet *Videre arbeid*.

Målet om å utvikle et system som kunne brukes som et administrativt hjelpemiddel i gjennomføringen av NM for klubblag føler vi ble vellykket. Tilbakemeldinger fra NBF gir uttrykk de har spart en del tid, til tross for at det var snakk om innføring av et nytt system. Utviklerne tror derfor at ved neste turnering vil dette systemet medføre en mye enklere administrering av turneringen for NBF.

12.2 Videre arbeid

12.2.1 Bedre rutiner for utsendelse av e-poster

Modulen som tar seg av utsendelse av e-post er ganske enkel og har få muligheter for kontroll og feilrapporter. En mulig utvidelse ville vært en feillogg som viste hvilke e-post som ikke var vellykket mottatt av e-posttjeneren. Her ville det vært naturlig å tilby brukeren en mulighet til å rette en e-postadresse og så prøve å sende e-posten på nytt.

Brukerne kunne også fått opp en side med en oversikt over hvilke e-post som var sendt ut og hvilken status hver enkelt e-post hadde (sent, feil adresse, kommet i retur og så videre). Ved å tilby en slik funksjonalitet vil brukere, som for eksempel mistenker en anti-spam løsning for å stoppe innkomne e-poster, kunne bla frem tidligere sendt e-post.

12.2.2 Hendelseslogger

For å bedre sporbarheten av handlinger kan det være ønskelig i en senere versjon av programvaren å inkludere bedre rutiner for å spore handlinger som er utført av brukerne. Dette kan gjerne inkludere en "versjonskontroll" av innstillinger slik at ved vandalisme, feilkonfigurasjon, eller lignende kan man på en enkel måte se hvem som gjorde endringen og få gjenopprettet en tidligere tilstand.

12.2.3 Varsling ved bestemte begivenheter

Å tilby ekstra varsletjenester kan nok være interessant for enkelte brukere. Dette kan være enkle abonnement som varsler om trekninger, resultater eller lignende for et lag man er interessert i. Dette kan sammenlignes med tjenester som tilbys fotballsupportere (følg med på ditt favorittlag med mer)

En naturlig utvidelse av varsling kan være å inkludere varsling til mobiltelefoner via SMS. I motsetning til e-post, medfører utsending av SMS en økonomisk utgift. Dette vil kunne medføre forskjellige ønsker på betalingsmodell avhengig av hvem som arrangerer turneringen. De tre mest realistiske betalingsmodellene er; overtaksert SMS, turneringsansvarlig blir fakturert (og tar dette inn via turneringsavgiften) eller hver enkelt bruker har sin egen konto som brukeren kan fylle på selv. Disse tre betalingsmodellene er grundigere gjennomgått i masteroppgaven til Bjørnar Pettersen, *Nettsted for Bridgetjenester* [89].

12.2.4 Forskjellige turneringstyper

Det finnes en del forskjellige turneringstyper som det muligens ville vært fordelaktig å få implementert. Seriemesterskap blir avholdt innen forskjellige sportsgrener, bridge inkludert. Å tilby slik funksjonalitet vil nok være en naturlig utvidelse av BTS. En mulig implementering ville vært å tilby et sett med moduler som hver innførte en turneringstype. I NM for klubblag blir alle de innledende rundene spilt etter utslagssystemet. I finalen spilles det seriespill mellom de deltakende lagene. På bakgrunn av dette kan det virke fordelsmessig å innføre en mulighet til å spesifisere turneringstype per runde.

12.2.5 Automatiske trekninger

En mulig fremtidig utvidelse av turneringsmodulen vil være å tilby funksjonalitet som automatisk trekning etter gitte regler. Fritrekning er en enkel utvidelse. I gjennomføringen av NM for klubblag er det flere små momenter som har betydning for hvilke lag som kan møte hverandre. Det kan for eksempel være spesielle forhold som må bli tatt hensyn til i noen regioner. Disse reglene kan implementeres som enkle funksjoner eller moduler som tilbyr spesialisert funksjonalitet. Ved å modularisere reglene vil det være mulig å kombinere moduler til en gitt trekning. For eksempel kan geografisk inndeling og erfaringsnivået på et lag bli inkludert i vurderingen av hvilke lag som kan møte hverandre. En mulighet for å automatisk bestemme hvilket lag som skal være hjemmelag/bortelag kan regnes ut på bakgrunn av tidligere kamper, slik at alle lag fikk en jevn fordeling av antall hjemmekamper. En fare med en slik utvidelse vil være tiden som trengs for å sette opp systemet kan overgå tiden som brukes til å gjennomføre trekning manuelt. En grundig gjennomgang bør derfor utføres før et slikt prosjekt startes.

12.2.6 Reiseregninger

En turnering slik som NM for klubblag medfører mye reising for en del av lagene. En del av dette skal dekkes av NBF. Det er derfor behov for en modul for å behandle dette. Dette vil være et naturlig tillegg til BTS, siden dette systemet allerede håndterer de andre elementene rundt denne turneringen.

Del 2: Bridge for mobile enheter

13 Innledning

13.1 Bakgrunn for oppgaven

Det har i det siste vært en kraftig utvikling innenfor mobile enheter, og det blir mer og mer vanlig at den gjennomsnittlige forbruker har en mobiltelefon som både kan laste ned nye programmer, og har muligheten til å operere på trådløse nett. Dette gjelder både nett som Wi-Fi [90], som er standarden for trådløse datanettverk, og Bluetooth[1] [91]. Dette er funksjonalitet som tidligere var reservert de største og dyreste modellene. Veldig mange som eier slike telefoner ønsker også å benytte seg de avanserte mulighetene slike telefoner tilbyr, noe som for eksempel viser seg i den utstrakte bruken av digitale kameraer i mobiltelefoner.

Bridgespillere er ikke noe unntak, og man kan se for seg flere måter en mobil enhet kan benyttes i sammenheng med bridge. Bridge er for eksempel veldig populært å spille via Internett. Bridge på mobil vil derfor være et utmerket alternativ når man ikke har en datamaskin tilgjengelig, men kan også være en erstatning i en mer tradisjonell situasjon der man har fire mennesker samlet på samme sted, men ikke har plass eller utstyr tilgjengelig til å sette opp et vanlig spill med bord og kortstokk.

13.2 Målsetning med oppgaven

Formålet med denne oppgaven er å utvikle et bridgespill for mobile enheter. Spillet skal kunne kjøres på flere mobile enheter i ett nettverk, slik at fire spillere kan spille bridge sammen fra hver sin enhet. Spillet må sørge for at reglene for bridge blir fulgt, og drive spillet fremover.

I tillegg til å kunne tilby spilling, er det også ønskelig med diverse relaterte tjenester. En slik mulighet er å kunne spille av tidligere spilte spill, da både spill man selv har spilt, eller interessante spill som lastes ned for eksempel fra Internett. Andre interessante tjenester er for eksempel meldetrening for to personer, enten i opplæringsøyemed for nye spillere, men også som trening for mer erfarne spillere.

Applikasjonen som blir utviklet bør også kunne kjøre på så mange forskjellige typer mobiltelefoner som mulig, med forskjellige skjermstørrelser og tekniske spesifikasjoner, så lenge disse støtter visse minstekrav.

[1] Også kjent som "Blåtann" på norsk. Det presiseres at Bluetooth er et varemerke som dermed ikke skal oversettes (se Bluetooth SIG [91]), og "Blåtann" er derfor ikke en korrekt benevnelse på denne teknologien.

14 Problemanalyse

14.1 Kravspesifikasjon

Programvaren som skulle utvikles ble definert til å ha følgende krav:

- Være plattformuavhengig (innenfor mobile enheter).
- Kunne brukes på enheter med forskjellige skjermstørrelser.
- Forholde seg korrekt til reglene for bridge.
- Gjennomføre et komplett bridgespill.
- Kunne kommunisere med andre enheter over minst en form for nettverk slik at fire spillere kunne spille sammen på hver sin enhet.

Videre bør applikasjonen ikke være avhengig av funksjoner som kun finnes på de dyreste og mest luksuriøse enhetene på markedet. Brukergrensesnittet burde også være så intuitivt som mulig, helst minne litt om de grensesnittene bridgespillere er vant til når de benytter seg av elektroniske bridgetjenester. Dette betydde hovedsakelig BBO [4], siden dette i dag er den mest populære tjenesten for elektronisk bridge i Norge.

14.2 Analyse

Det var umiddelbart klart at uansett hvor godt man prøver, kan ikke en slik applikasjon kjøre på alle mobiltelefoner. Det finnes for eksempel telefoner på markedet som ikke gir muligheten til å selv installere programmer, og slike telefoner produseres ennå, som for eksempel Apples iPhone, som kun tillater brukeren å laste ned programvare distribuert av Apple. Det er heller ikke alle telefoner som har mulighet til å ta del i ett nettverk, som er nødvendig for denne applikasjonen.

En nærmere analyse av kravene med overnevnte i tankene gir da følgende:

- Plattformuavhengighet: De fleste telefoner i dag støtter i utgangspunktet to hovedgrupper av programmeringsspråk; Java og C/C++. Begge disse to har sine fordeler, men også sine ulemper.

  C/C++ implementering er veldig leverandør- og modellspesifikk, og hver telefon har sin egen måte å bruke telefonens egenskaper, slik som nettverk. Ikke alle telefoner støtter C/C++, men bruker andre språk. En del telefoner er også begrenset til hvilke type applikasjoner som brukeren selv kan laste ned på telefonen, og ikke alle tillater applikasjoner i C/C++ selv om telefonens egne applikasjoner er skrevet i dette språket.

  Java på den annen side er rimelig standardisert på tvers av merke og modell. Dette medfører at det er mye mindre behov for å tilpasse applikasjonen til hver enkelt mobil enhet. Java tilbyr også standardiserte grensesnitt for å bruke telefonens funksjoner, slik som for eksempel nettverksfunksjonalitet. Java har også tradisjonelt vært det språket i mobilverdenen som har vært åpnet opp for at brukeren kan installere egne applikasjoner på enheten. Java viser seg derfor å være det beste valget i denne sammenhengen for å oppnå målet med å være plattformuavhengig og kunne installeres på et størst mulig antall mobile enheter.

- Skjermstørrelse: Mobiltelefoner på markedet i dag har svært forskjellig skjermstørrelser. Generelt sett vil en dyrere enhet ha større skjerm. Etter å ha sjekket med en del nettsteder hva som var tilgjengelig i dag, var det tydelig at skjermer gikk fra 128x160 punkter og oppover. Telefoner med skjermer mindre enn dette var stort sett billigtelefoner som ikke hadde de andre nødvendige egenskapene. 128x160 ble derfor definert som en minimumsstørrelse for denne applikasjonen.
- Forholde seg til bridgeregler: Reglene i bridge er ikke veldig kompliserte, og krever heller ikke store tunge kalkulasjoner. Dette kunne derfor implementeres i kode uten å stille spesifikke krav til enheten det kjørte på.
- Kunne gjennomføre et komplett spill: Igjen så setter ikke dette noen fysiske krav til enheten det kjøres på. For denne biten var det viktig at prosjektet ble styrt på en slik måte at dette lot seg gjennomføre.
- Nettverkskommunikasjon: De fleste mobile enheter i dag har en eller annen form for trådløs nettverkskommunikasjon tilgjengelig. De vanligste alternativene her er Bluetooth, WLAN og GPRS. Ønsket var å få implementert alle disse tre dersom tiden tillot dette. Alle metodene har sine fordeler og ulemper, men siden GPRS koster penger å benytte, og WLAN kun er tilgjengelig på litt dyrere mobile enheter, så var Bluetooth et godt startpunkt. En annen god grunn til å begynne med Bluetooth er at de fleste mobile enheter ikke tillater brukeren å være tjener i et TCP/IP-basert nettverk, noe som medførte at man da må sette opp en egen tjenermaskin på Internett for å støtte disse. Med Bluetooth slipper man dette.

Konklusjonen blir da at Java ble valgt som programmeringsspråk, med Bluetooth som første nettverksbærer. Etter planen ville dette sørge for en størst mulig plattformuavhengighet, samtidig som løsningen ville være kompatibel med et så stort antall mobile enheter som mulig. Applikasjonen ville heller ikke være avhengig av en ekstern tjener. Ulempene med dette valget er at Bluetooth kun er beregnet for kommunikasjon over korte distanser, og krever at alle spillerne er innefor ca 10 m fra hverandre. Med WLAN eller GPRS kunne spillerne ha befunnet seg i hver sin del av landet, og fremdeles spilt sammen. Det viktige her er selvsagt at Bluetooth bare er første steg. Dersom tiden tillot det ville prosjektet også implementere andre nettverksbærere. Hvis ikke kunne dette videreutvikles på et senere tidspunkt.

Java- og Bluetooth-teknologiene vil få en grundigere gjennomgang senere i rapporten.

15 Valg av verktøy

15.1 Rammeverk

Det har etter hvert dukket opp en del rammeverk på markedet for utvikling av Java-applikasjoner for mobiltelefoner. I sammenheng med dette prosjektet ble et par av disse vurdert.

For grafiske grensesnitt (GUI) var dette hovedsakelig J2ME Polish [92], som blant annet er lisensiert under GPL-lisensen. Dette ble fort forkastet da dette rammeverket, sammen med de fleste andre tilsvarende rammeverk, hovedsakelig var en utvidelse av høynivå-GUIet til Java ME (Micro Edition)[1]. Behovet i dette prosjektet gjorde at det hovedsakelig var lavnivå-GUIet som trengtes i applikasjonen, hvor de enkelte skjermelementer kunne tegnes direkte, og ikke bare plassere tekstfelter, inputbokser, dialoger og lignende. Dette valget medførte at utviklerne måtte selv få opptegningen av grensesnittet til å tilpasse seg varierende skjerm-størrelser. Siden de eksisterende rammeverkene ikke kunne oppfylle de kravene applik-asjonen hadde til å manipulere lavnivågrensesnittet direkte, var disse derfor ikke til noe hjelp i denne sammenhengen.

Et av de vanskeligste elementene med J2ME-programmering på mobil er når man skal få enheter til å kommunisere via Bluetooth. Det var derfor viktigere å finne et godt rammeverk for å minimere disse problemene. Dette viste seg vanskeligere enn først antatt. Selv om det finnes mange eksempler tilgjengelig for implementering av Bluetooth-funksjonalitet, var det få rammeverk for dette. Det beste alternativet som ble funnet var "Bluetooth Multiplayer Games Framework" [93]. Dette var basert på åpen kildekode. Etter en del testing av dette rammeverket ble det klart av rammeverket hadde nøyaktig de samme problemene som utviklerne selv hadde med å få Bluetooth-støtten til å fungere tilfredsstillende på alle mobile enheter. Siden dette rammeverket da ikke hadde noe fornuftig å tilby applikasjonen, ble det bestemt å ikke benytte seg av dette.

En annen av fordelene med bruk av rammeverk er lettere tilpasning til mange typer mobile enheter. Dette foregår vanligvis ved at rammeverkene sørger for at applikasjonen blir kompilert kun med de funksjoner hver enkelt mobil enhet støtter, slik at man får en tilpasset applikasjon til hver enhet. For dette prosjektet var denne egenskapen heller ikke spesielt relevant. Prosjektet hadde veldig spesifikke krav, nemlig at enheten støtter Bluetooth, og at dette kan bli aksessert fra en Java ME-applikasjon på enheten. Bluetooth-støtten for Java er implementert på alle telefoner som støtter dette gjennom en J2ME-profil som heter JSR-82 [94]. Profiler i Java ME blir forklart senere, men hver profil er i prinsippet støtte for en spesifikk type funksjonalitet, i dette tilfellet Bluetooth. For denne applikasjonen vil det grovt sett si at enten er denne tilstedet, og enheten kan kjøre applikasjonen, eller at JSR-82 ikke tilgjengelig, og applikasjonen kan ikke kjøres på enheten. Dersom enheten ikke støtter Bluetooth vil det være meningsløst å kompilere en versjon uten Bluetooth, siden applikasjonen da ikke vil ha noen reell funksjon. Slike fordeler er dermed best egnet for applikasjoner som har en del valgfrie komponenter, slik som for eksempel avspilling av lyd og bilde, og muligens dersom oppkobling til Internett er en fordel for applikasjonen, men ikke et krav. I slike tilfeller kan det være svært hensiktsmessig med et rammeverk som gjør det lett å lage forskjellige versjoner for forskjellige mobile enheter ved å fjerne funksjonalitet

[1] Java ME er grundigere gjennomgått senere. Se seksjonen om Java.

den enkelte enhet ikke støtter. Siden dette da ikke ville være noe hjelp for dette prosjektet, ble det avgjort at et slikt rammeverk bare ville forårsake unødvendig kompleksitet, og derfor ble heller ikke denne typen rammeverk vurdert.

Da det ikke var aktuelt å betale for et rammeverk, ble ingen kommersielle rammeverk vurdert. Java ME er også utstyrt med høynivå GUI-komponenter, og disse er tilgjengelige uten bruk av noe rammeverk. For denne applikasjonen var disse tilstrekkelig for å dekke det minimale behovet applikasjonen hadde for slike komponenter. J2ME Polish kunne nok ha gjort disse skjermene mer visuelt tiltalende, men siden hoveddelen av spillet foregår på en skjerm hvor J2ME Polish ikke har noe å bidra med, var ikke det hensiktsmessig å benytte verken dette eller tilsvarende rammeverk. Java ME klarer også å automatisk tilpasse disse høynivåskjermene til tilgjengelig skjermplass, uten behov for noe ekstra rammeverk.

Utviklerne fant det dermed mest hensiktsmessig å ikke benytte seg av noe rammeverk for mobil utvikling, da det ikke fantes gode, gratis rammeverk som kunne tilby nødvendig funksjonalitet til applikasjonen.

15.2 Utviklingsmiljø

Da det kom til valget av utviklingsmiljø ble det vurdert to programpakker, Eclipse [50] og NetBeans IDE [95]. Det finnes utvidelser til begge disse utviklingsmiljøene for å utvikle Java ME-applikasjoner. I all hovedsak er funksjonaliteten til disse to utviklingsmiljøene ganske like, spesielt med tanke på behovene for dette delprosjektet. I motsetning til første delprosjekt, der kun Eclipse hadde en god utvidelse for PHP-utvikling, kunne begge miljøene tilby den nødvendige funksjonaliteten til utvikling av mobilapplikasjoner. Begge utviklingsmiljøene ville dermed ha taklet denne jobben godt, og det var derfor ikke noen gode tekniske årsaker til å velge den ene fremfor den andre. Avgjørelsen ble da overlatt til personlige preferanser, og NetBeans ble dermed valgt.

15.3 Mobile enheter

For å kunne utvikle en applikasjon for mobile enheter er det svært viktig å utføre testing på faktiske mobile enheter. Tidligere erfaringer viste at det å få en applikasjon til å virke i JWT-emulatoren[1] er meget forskjellig fra å få den til å virke på en faktisk enhet. Det ble også bestemt at for å sikre at applikasjonen ikke bare var tilpasset en telefontype, trengtes det telefoner av forskjellige merker, slik at man oppnådde et heterogent testmiljø hvor man lettere kunne avdekke kompatibilitetsproblemer. Etter å ha undersøkt markedet og generell Java ME-støtte på moderne enheter, ble det bestemt at disse telefonene skulle støtte CLDC 1.1 og MIDP 2.0, som da også ble definert som minstekrav til selve applikasjonen[2]. I tillegg måtte enhetene støtte Bluetooth og WLAN for å kunne brukes i et nettverk. Prisen var også et element, da budsjettet for dette prosjektet var begrenset. Etter en nærmere undersøkelse av markedet i Norge i dag, viser det seg at dette i all hovedsak domineres av Nokia [96] og Sony Ericsson [97], derfor ble det bestemt å anskaffe en telefon fra hver av disse produsentene. Basert på de tekniske spesifikasjonene, prisen og lagerbeholdning i butikken, ble N95 fra Nokia og W960i fra Sony Ericsson valgt. I tillegg til disse to

[1] Mobiltelefonemulator for Java ME utvikling. JWT (Java Wireless Toolkit) er forklart i sin egen seksjon.

[2] CLDC (Connected Limited Device Profile) og MIDP (Mobile Information Device Profile) blir forklart i avsnittet om Java.

telefonene, hadde også utviklerteamet en Sony Ericsson K800i tilgjengelig, som oppfylte noen av kravene, og kunne derfor benyttes til en del testing.

Telefonene som ble brukt under dette prosjektet ligger nok i klassen for relativt dyre telefoner[1], men erfaringer viser at det som er standard på litt dyrere telefoner i dag kommer til å være standard på middelklassetelefoner neste år. Valget av mobiltelefoner er derfor ikke urealistisk, siden denne funksjonaliteten disse har kommer til å være mer tilgjengelig for folk flest innen kort tid.

	Nokia N95	**Sony Ericsson W960i**	**Sony Ericsson K800i**
MIDP	2.0F	2.0	2.0
CLDC	1.1	1.1	1.1
Wi-Fi	802.11 b/g	802.11 b	-
Bluetooth	2.0	2.0	2.0
GRPRS	Ja	Ja	Ja
Skjermoppløsning	240x320 (1)	240x320	240x320 (2)
OS	Symbian S60. 3. utgave	Symbian v9.1	?

(1) skjermen kan endres til liggende format, og blir da 320x240.

(2) mulighet til å dynamisk forandre skjermstørrelsen som en applikasjon har tilgjengelig.

Figur 15.1: Mobiltelefonspesifikasjoner

15.4 Verktøy for mobilprogrammering

For utvikling av Java ME-applikasjoner leverer Sun [21] en verktøykasse, Sun Java Wireless Toolkit (JWT). Denne inneholder de nødvendige verktøy for å verifisere og kompilere kildekode for mobil. Utvidelsene for mobil uvikling til både NetBeans og Eclipse er knyttet mot denne, så JWT er nødvendig for å utvikle for mobil.

I tillegg til JWT har stort sett de fleste mobilleverandører en egen verktøykasse tilpasset deres mobiltelefoner, som brukes i tillegg til JWT. Det ble diskutert blant utviklerne om disse skulle benyttes, men utviklerne ble enige om at applikasjonen burde være generisk, og ikke støtte spesialfunksjoner kun funnet på enkelte mobiler. Det ble derfor avgjort at det ikke var hensiktsmessig å benytte seg av disse verktøyene, og heller holde seg kun til den generiske JWT. Det var bedre å utvikle applikasjonen etter prinsippet om minste felles multiplum, men innenfor minimumsspesifikasjonene som ble bestemt.

[1] Ca 5 000 kroner per telefon første kvartal 2008

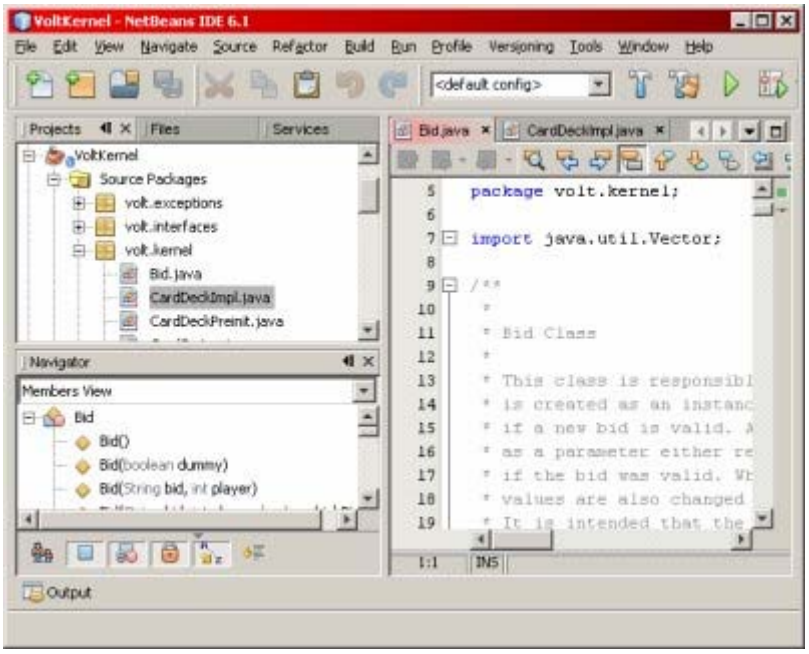
15.5 Programvare

Under utviklingen av denne applikasjonen ble følgende applikasjoner brukt til utviklingen. Grunnlaget for valget av disse er for det meste diskutert tidligere. Disse verktøyene kommer i tillegg til applikasjonene som står beskrevet i innledningen til rapporten.

15.5.1 NetBeans IDE

NetBeans IDE [95] (vanligvis omtalt som NetBeans) er, akkurat som Eclipse, et integrert utviklingsmiljø. Som med Eclipse, er også NetBeans primært utviklet for å skrive Java SE-programmer (Standard Edition), men det finnes et stort utvalg utvidelser som gir støtte for blant annet C++, Java ME (Micro Edition) og PHP. Da NetBeans ble brukt til å utvikle bridgespill for mobile klienter, er det her benyttet utvidelsen NetBeans Mobility Pack, som inneholder støtte for å utvikle Java ME-programmer.

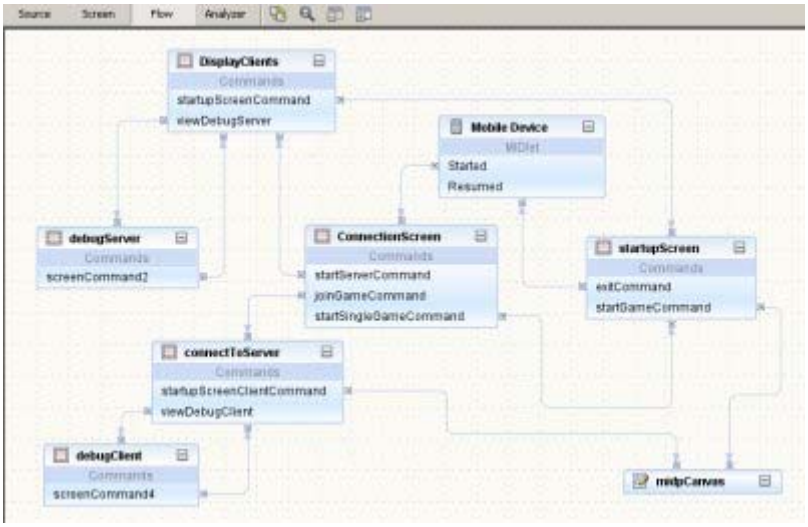
NetBeans har veldig mye funksjonalitet innebygd, blant annet støtte for testing av Javakode ved hjelp av JUnit. NetBeans er gratis, og basert på åpen kildekode.



Figur 15.2: Skjermbilde fra NetBeans IDE

NetBeans Mobility Pack

NetBeans Mobility Pack er en utvidelse for NetBeans, som er beregnet for utvikling av programvare i Java ME for mobile klienter. Denne utvidelsen inneholder blant annet en grafisk skjermdesigner, som gjør at det er meget enkelt å designe de forskjellige skjermene i applikasjonen grafisk, og koble disse sammen enkelt ved hjelp av dra-og-slipp. Denne utvidelsen er gratis, og lastes ned direkte via den innebygde utvidelsesbehandleren i NetBeans.



Figur 15.3: Skjermbilde fra NetBeans Mobility Pack

Code Coverage

Code Coverage er en utvidelse for NetBeans som sjekker hvilke deler av koden som blir testet av enhetstester[1]. Ved å bruke denne utvidelsen, er det lett å kontrollere at enhetstestene man har laget tester alle kodelinjene i applikasjonen. Dette er spesielt nyttig når koden inneholder løkker og valg, for å forsikre seg om at alle eventualiteter blir testet. Utvidelsen fungerer ved at den markerer alle linjer i koden som er blitt testet med en egen farge. Dette gjør at det blir enkelt å se hvilke kodelinjer som ikke er testet. Denne utvidelsen er gratis, og lastes ned direkte via den innebygde utvidelsesbehandleren i NetBeans.

JUnit

JUnit [98] er et rammeverk for å skrive og kjøre enhetstester for å teste Java-kode. En enhetstest er en test som tester en liten begrenset del av koden, vanligvis en enkelt metode. Fordelen med å bruke slike tester, er at man på et senere tidspunkt lett kan se om man har gjort endringer i koden som har introdusert feil. JUnit er gratis, og basert på åpen kildekode.

15.5.2 Sun Java Wireless Toolkit

Sun Wireless Toolkit (WTK) [99] er en verktøykasse med hjelpemidler for utvikling av Java ME-programvare. Et av de viktigste verktøyene i denne pakken er en emulator, slik at man kan kjøre Java ME-kode på en simulert mobiltelefon. Emulatoren har også støtte for å simulere Bluetoothforbindelser, slik at man kan starte to eller flere instanser av emulatoren, og la disse kommunisere via simulerte Bluetooth-forbindelser. Verktøykassen inneholder

[1] Se avsnittet om JUnit.

også analyseverktøy, slik at man blant annet kan overvåke ressursbruken til programmet under kjøring.

Figur 15.4: Skjermbilde fra mobiltelefonemulator

16 Utviklingsmetode

Smidig systemutvikling er en utviklingsmodell som har en del fundamentale forskjeller i forhold til UP. I 2001 samlet 17 ledende skikkelser innenfor det som på den tiden ble kalt lettvektsmetodikk og lagde et manifest som ble grunnlaget for det som i dag er kjent som "Agile software development". Manifestet tar et standpunkt som beveger seg vekk fra det UP har som sine hjørnesteiner:

> *Vi oppdager bedre måter å utvikle programvare ved å utføre det og hjelpe andre gjøre det. Gjennom dette arbeidet har vi omfavnet følgende verdier:*
> *Individer og interaksjon fremfor prosesser og verktøy.*
> *Fungerende programvare fremfor omfattende dokumentasjon.*
> *Samarbeid med kunde fremfor kontraktsforhandlinger.*
> *Handle etter forandringer fremfor å følge en plan.*
> *Det er verdi i elementene til høyre, men vi verdsetter de til venstre mer.*[100]

Til dette manifestet er det en del prinsipper som konkretiserer essensen i manifestet [100]:

- Vår høyeste prioritet er å tilfredsstille kunden ved å tidlig og kontinuerlig levere verdifull programvare.
- Møte forandringer i kravene med åpne armer, selv sent i utviklingen. Smidige prosesser omfavner forandringer for å styrke kundens konkurranseevne.
- Levere fungerende programvare regelmessig, fra et par ukers til et par måneders intervall, med en preferanse for korte intervall.
- Forretningsfolk og utviklere må jobbe sammen daglig gjennom prosjektet.
- Bygg prosjekter rundt motiverte individer. Gi dem det miljøet og støtten som de trenger, og stol på at de får jobben utført.
- Den mest effektive måten å formidle informasjon til og internt i et utviklingsteam er verbal kommunikasjon.
- Fungerende programvare er den primære målenheten for fremgang.
- Smidige prosesser foregår i et jevnt tempo. Sponsorene, utviklerne og brukerne må kunne holde et jevnt tempo til evig tid.
- Kontinuerlig fokus på teknisk perfeksjonisme og god design øker fleksibiliteten.
- Enkelhet - kunsten å maksimere mengden arbeid ikke utført - er viktig.
- De beste arkitekturer, krav og design er utspring fra selvorganiserende team.
- Regelmessig bør teamet reflektere over måter som bidrar til bedre effektivitet og gjør endringer deretter.

Selv om det er grunnleggende forskjeller i hvilke verdier som vektlegges i UP og smidig utvikling, finnes det noen punkter hvor de har fellestrekk. Begge metodikkene benytter iterativ utvikling, men forskjellen ligger i hva som gjøres i de forskjellige iterasjonene. UP er mer formell og prøver å opprette et design før implementeringen starter. I en smidig utvikling vil forandringer komme fortløpende og fokuset vil derfor være i nær framtid.

16.1 eXtreme Programming

Utviklingsmetodikken eXtreme Programming (XP) har røttene sine i smidig systemutvikling. XP har alle verdiene i smidig utvikling, men konkretiserer hvordan dette faktisk bør gjennomføres i et prosjekt. Disse "øvelsene" er knyttet sammen slik at komponentene styrker hverandre og gir mest mulig utbytte. Øvelsene er enkle å forstå og utføre. Sammen gir de en smidig utviklingsprosess.

XP er utviklingsmetodikken som ble valgt for denne delen av oppgaven. De forskjellige øvelsene i XP er kort oppsummert i følgende avsnitt [100].

16.1.1 Øvelser i XP

Kunden deltar i teamet

XP-team vil at kunden skal kunne arbeide sammen med utviklingsteamene. På denne måten har teamene rask tilgang til en ressurs som kan svare på spørsmål. Dette vil også gi kunden et større innblikk i hvordan programvaren tar form og partene kan hjelpe hverandre med å komme i mål.

I beste tilfelle er kunden tilgjengelig i samme rom som utviklerne. Hvis dette ikke er mulig, bør det være kort avstand. Desto lenger vekk kunden er, jo vanskeligere er det å ha en dialog.

Brukstilfeller

Brukstilfeller (eng: User Stories) er korte “historier” om hvordan forskjellige situasjoner oppstår i kundens hverdag. Brukstilfellene behøver ikke å være detaljerte - alt som er nødvendig er essensen i hva tilfellet inneholder. Programmererne behøver ikke alle detaljer for å estimere hvor lang tid et brukstilfelle tar å implementere. Noen grove detaljer kan være nyttig, men de små detaljene vil også kunne være utsatt for endringer senere i utviklingen.

Korte iterasjoner

Iterasjonene bør være korte; normalt rundt to uker. På dette tidspunktet skal utviklerne vise en fungerende versjon til oppdragsgiver for å vise hvor langt prosjektet er kommet og for å få tilbakemeldinger på hva som bør endres.

En iterasjonsplan inneholder en liste over hvilken funksjonalitet (brukstilfeller) som skal implementeres til neste demonstrasjon for kunde. Det er kunden som bestemmer hvilken funksjonalitet som skal implementeres først. Etter en iterasjon har begynt har kunden ikke lov til å endre prioriteringene eller forandre hvilke brukstilfeller som skal implementeres. Det er opp til utviklingsteamene å dele opp de aktuelle brukstilfellene i oppgaver. Utviklerne blir enige om i hvilke rekkefølge det er teknisk lurt å implementere de forskjellige oppgavene.

En milepælsplan skisserer hvilke brukstilfeller som skal bli tatt med i en fullverdig versjon av systemet. Normalt gjelder en milepælsplan for rundt seks iterasjoner (ca tre måneders arbeid). På dette tidspunktet skal programmet kunne settes i produksjon hvis kunden ønsker dette. I hver milepælsperiode kan kunden velge hvilke brukstilfeller som skal implementeres. Kunden står også fritt til å endre, legge til og fjerne brukstilfeller underveis. Brukstilfellene som skal inn i en milepælsperiode, blir til slutt fordelt av kunden inn i de forskjellige iterasjonene.

Akseptansetester

Detaljene til et brukstilfelle defineres som forksjellige krav tilsystemet. Disse kravene bør være mulig å skripte slik at det er mulig å opprette automatiske akseptansetester. Når først et krav er innfridd, så er det ikke lov for utviklerne å gjøre endringer slik at kravet ikke lenger er oppfylt. Dette betyr at når først en funksjonalitet er testet ok, så skal alle endringer i koden medføre at akseptansetestene fremdels gir grønt lys. De automatiske testene bør kjøres flere ganger daglig. Hvis en av testene feiler skal hele programmet få status som "ikke godkjent".

Parprogrammering

Med parprogrammering danner to utviklere et par som arbeider sammen. Paret skal fungere som en enkelt programmerer med fire øyne og to hjerner. Det skal kun være ett tastatur og èn mus. Paret bytter på hvem som skal "styre" (dvs. bruke mus og tastatur). Ofte blir rollene navngitt sjåfør og navigatør. Sjåføren styrer mus og tastatur, mens navigatører følger nøye med og kommer med innspill etter hvert som han/hun ser forbedringspotensial. Det skal være aktiv diskusjon innad i paret om hvilken vei som er den riktige å gå. På denne måten kjenner to personer koden godt, og siden disse nesten alltid vil ha litt varierende kompetanse, så økes sannsynligheten for god kode.

Parene bør bytte medlemmer minst en gang per dag. På denne måten vil hver programmerer jobbe med flest mulig av de andre i utviklingsteamet. På denne måten overføres det kompetanse og alle er kjent med store deler av systemet.

Undersøkelser har vist at ved parprogrammering, så minsker ikke programmerernes samlede effektivitet. En nedgang i antall feil er derimot funnet. [101]

Testdrevet utvikling

Testdrevet utvikling er tanken om at all kode/funksjonalitet skal testes. Testene skal verifisere at implementeringen fungerer slik den skal. Dette gjøres ved at det først lages tester til et kodesegment/metode som skal implementeres senere. I testen skal det sjekkes at metoder oppfører seg som planlagt. Dette gjøres ved å sende bestemte data inn, for så sammenligne resultatet med en gitt fasit. En stor fordel med dette er at mange feil blir oppdaget på et meget tidlig tidspunkt, i stedet for å bli oppdaget senere, hvor det kan bli både vanskelig og dyrt å rette disse.

Etter en test er laget, skal den kjøres for å bekrefte at den feiler. Testen vil feile på dette punktet da implementeringen ikke er gjort enda. Ved implementering, så skal utviklerne kun utføre implementeringen slik at testen passeres. Hvis funksjonaliteten til koden skal utvides skal det først skrives tester som sjekker funksjonaliteten, for så utvide implementeringen slik at testen igjen passeres. På denne måten kan man gjøre til dels radikale endringer i koden og fremdeles sørge for at all funksjonalitet er tilstede ved å kontrollere at testene viser at alt er ok.

Det finnes mange studier angående testdrevet utvikling, og resultatene kan variere noe med hensyn på faktisk oppnådd effekt. Noen studier viser forbedring i kvaliteten, og nedgang i produksjon, mens andre studier viser det motsatte. En studie fra National Research Council Canada [102], skrevet av Hakan Erdogmus viser blant annet at kodekvaliteten i en testgruppe gikk litt ned ved bruk av testdrevet utvikling, men at kodekvaliteten var mye jevnere i forhold til en testgruppe som ikke brukte testdrevet utvikling, men som skrev tester i ettertid for å teste koden. Produksjonen var også høyere i testgruppen som brukte testdrevet utvikling. Den samme rapporten summerer også opp resultatene fra en del andre undersøkelser, som grovt

summert viser at testdrevet utvikling ikke alltid er bedre på alle kriterier, men gir best resultat totalt.

Kollektivt eierskap av kildekoden

Et hvert par har rett og mulighet til å sjekke ut kode, gjøre endringer for så sende koden inn igjen. Det er et kollektivt eierskap og ansvar for koden. Ingen enkelt utvikler eller par "eier" noen deler at kildekoden. Alle på utviklingsteamet har mulighet til å forbedre andres kode eller endre den.

Kollektivt eierskap hindrer ikke noen å være spesialister innenfor et felt (for eksempel grafiske brukergrensesnitt eller databaser). En som er spesialisert innenfor et felt, er bare mer aktuell til å få tildelt oppgaver innenfor det området. En viktig del av XP er å overføre kunnskap mellom utviklerne. Dette gjøres ofte ved at en spesialist “parer” seg med en som ønsker mer kunnskap innenfor feltet.

Kontinuerlig integrasjon

Etter hvert som parene blir ferdige med oppgavene blir kildekoden sjekket inn i det sentrale kodelageret. Hvis to eller flere par har jobbet med den samme koden, får det første paret den enkleste jobben. De andre parene må selv flette inn sine egne endringer. For at dette arbeidet skal være enklest mulig, er det anbefalt at alle par sjekker inn koden sin med jevne mellomrom. Her gjelder også at koden skal passere testene. All kode som er sjekket inn skal være kjørbar. På testserverne blir det regelmessig kjørt tester som sjekker at systemet lar seg bygge og er i fungerende stand.

Stabilt tempo

Et programvareprosjekt er en maraton, ikke en sprint. Jevnt tempo er nøkkelordet. Overtid er ikke tillatt; eneste unntak er uken før en milepæl.

Åpent arbeidsområde

Et åpent kontorlandskap hvor det er mye veggplass der teamene kan henge opp lapper og tegninger virker bra inn på miljøet. Det skal være lav snakking i hele lokalet. Parene skal sitte såpass nær hverandre at de kan høre når noen får problem, men det bør være på et nivå slik at det ikke er forstyrrende.

Planleggingsspillet

Planleggingsspillet er et møte med oppdragsgiver hvor funksjonalitet som skal implementeres i neste iterasjon blir bestemt. Dette foregår ved at utviklerteamet på forhånd har gitt hver av brukstilfellene en poengsum som indikerer hvor mye arbeid hver av disse representerer. Teamet forteller også arbeidsgiver hvor mange poeng denne kan bruke. Antall tilgjengelige poeng avhenger av teamets totale kapasitet. Oppdragsgiver velger så ut hvilke brukstilfeller som skal implementeres, innenfor poenggrensen.

Enkelt design

Kode skrevet etter XP-prinsippet skal være så enkel som mulig for å løse den oppgaven man skal løse der og da. Det skal ikke opprettes store komplekse strukturer bare fordi det er mulig at det blir et behov for disse i fremtiden. Designet skal representere behovet i nåværende iterasjon. Trenger man mer avanserte strukturer senere, legger man til disse da og refaktorerer etter behov.

Refaktorering

Refaktorering er et viktig prinsipp i XP. For å unngå "dårlig" kode utføres refaktorering kontinuerlig slik at koden blir mer optimal. Dersom kode ikke trengs lenger slettes denne, og klasser og metoder får nye navn slik at disse stemmer overens med hva klassene og metodene faktisk gjør, slik at koden blir oversiktlig.

Metaforer

XP bruker metaforer for å uttrykke funksjonalitet. For eksempel kan en database bli beskrevet som et varehus, dataene i databasen som kasser, nettverket mellom klienten og databasen som en motorvei, og IP-pakkene som lastebiler. Disse blir brukt for å skape en bedre felles forståelse innad i teamet om hvordan applikasjonen skal utføre sin funksjonalitet.

16.1.2 Bruk av XP i prosjektet

Et av fokusområdene med dette delprosjektet var å benytte XP så "korrekt" som mulig, uten å ta snarveier eller hoppe over elementer som ikke passet utviklerne. De fleste elementene i XP har derfor blitt brukt i sin helhet, med minimalt med modifikasjoner.

Kunden deltar i teamet

Som XP tilsier, bør kunden være så tilgjengelig som mulig. Kunden trenger ikke være en av programmererne i teamet, men bør være lett tilgjengelig for teamet til enhver tid. Dette viste seg å være rimelig grei, da kunden hadde kontor rett under rommet hvor utviklingen fant sted, noe som medførte at utviklerne hadde veldig lett tilgang på kunden når det var behov for kontakt, og det var også lett for kunden å kontakte utviklerne.

Brukstilfeller

Her hadde utviklerne en mye større kjennskap til prinsippet, som medførte at alle de initiale brukstilfellene ble skrevet av utviklerne og gitt til kunden for godkjenning/korreksjon. Dette var nok uheldig, fordi det lot utviklerne beskrive hvordan de ville ha applikasjonen, i stedet for å få beskrevet fra kunden hva det var han ville ha. Her burde nok utviklerteamet ha overlatt dette helt og holdent til kunden, og latt vedkommende ta jobben med å skrive brukstilfellene. Det bemerkes her at kunden hadde absolutt en rolle i denne prosessen, og skrev en god del av de senere brukstilfellene, men det ble allikevel litt feil at utviklerne hadde en så stor rolle her. Utviklerne tar selvkritikk for dette.

Korte iterasjoner

Her ble det bestemt at iterasjonene skulle gå over fem arbeidsdager. Ved å sette fem arbeidsdager i stedet for en uke, oppnådde man den effekten at alle iterasjonene ble nøyaktig like lange, uavhengig av helligdager og lignende. Dette medførte også at det ble lett å holde et jevnt og stabilt tempo. Dette medførte også at ukedagen for iterasjonsslutt skiftet jevnlig, men dette var ikke noe problem da det var avklart på forhånd.

Akseptansetester

Det ble ikke opprettet noe formelt rammeverk for akseptansetester i dette prosjektet. Disse ble i hovedsak utført ved at kunden testet applikasjonen manuelt for å sjekke at alt var i orden i forholdt til levert funksjonalitet på daværende stadium.

Parprogrammering

Siden prosjektet hadde to utviklere ble parprogrammering konsekvent benyttet. All kode, med unntak av enkelte mindre feilrettinger ble skrevet ved hjelp av parprogrammering. Utviklerteamet opplevde dette som en meget positiv erfaring, og det medførte at de fleste feil ble oppdaget da koden ble skrevet. Utviklerteamet opplevde at de enkelte utviklerne hadde en tendens til å bare fortelle den som satt ved tastaturet hva vedkommende skulle skrive, i stedet for bare å ta tastaturet og skrive dette selv. Teamet vil derfor ta litt selvkritikk på dette området, og innser at her kunne man vært flinkere til akkurat denne biten. Det kunne nok ha økt effektiviteten ytterligere dersom dette hadde blitt bedre praktisert.

Testdrevet utvikling

Testdrevet utvikling ble benyttet i veldig stor grad ved utviklingen av kjernen i programmet, men utviklerne må innrømme at de noen ganger ble litt overivrige, og skrev kode før testen var utviklet. Testen ble da i disse tilfellene skrevet etterpå. Dette strider mot selve prinsippet ved testtrevet utvikling, og kan føre til at testen ikke tar høyde for alle eventualiteter på samme måte som hvis testen hadde blitt skrevet på forhånd.

Da det grafiske grensesnittet ble utviklet, ble dessverre testene litt ignorert, mye fordi det var vanskelig, og også tidkrevende, å skrive gode tester for grafiske grensesnitt. Det er lett å teste om et enkelt punkt har riktig farge, men å teste om den resulterende illustrasjonen på skjermen virkelig viser en kløver ni når den skal vise en kløver ni er mye vanskeligere, siden det her er snakk om en grafisk illustrasjon og ikke en enkel tekststeng. Størstedelen av programlogikken ligger likevel i kjernen, som hovedsakelig ble utviklet ved hjelp av testdrevet utvikling. Alle de viktigste elementene er dermed korrekt utviklet etter dette prinsippet. Utviklerne brukte også en utvidelse til NetBeans, Code Coverage Plugin, som markerte all kode som hadde blitt testet med egen farge, som bidro til å sørge for at det ble skrevet tester som dekket alle kode skikkelig, også alle mulige veier i valgstrukturer.

Utviklerne erfarte at testdrevet utvikling er et godt verktøy for å forsikre seg om at koden gjør det den skal, og at den fremdeles virker korrekt etter at man har gjort endringer. På den negative siden opplevde utviklerne at det kan være litt frustrerende å ikke kunne skrive kode direkte når man vet hva man skal skrive, og at det kan være vanskelig å lage gode tester som tester det man faktisk ønsker å teste. Det er her svært viktig at testene faktisk tester de forskjellige eventualitetene som kan oppstå, og ikke bare tester det ene resultatet man forventer skal inntreffe.

Kollektivt eierskap av kildekoden

Her besto utviklerteamet kun av to personer, noe som medførte at det kun ble ett fast par, og all kode tilhørte da dette paret. Paret fikk dermed et sterkt eierforhold til denne koden, noe som er vanskelig å unngå i denne situasjonen. Paret opplevde derimot koden som parets kode, og ikke tilhørende noen av enkeltindividene.

Kontinuerlig integrasjon

Dette punktet er svært vanskelig å praktisere når man bare er ett par, siden prinsippet baserer seg på at man skal sjekke inn kildekoden ofte for å få minimalt med konflikter, og at de som får konflikter tar hånd om disse. Med bare ett par blir det naturligvis aldri noen konflikter, og insentivet til å sjekke inn ofte forsvinner og prinsippet blir mer eller mindre meningsløst i denne sammenhengen. Utviklerne sjekket inn kildekoden minst en gang for dagen, men uten noen å "konkurrere" mot, fikk ikke disse noen reell følelse med dette prinsippet.

Stabilt tempo

Dette prinsippet ble praktisert i stor grad av utviklerne. Antall arbeidstimer brukt var omtrent det samme i hver iterasjon. Utviklerne føler her at dette har vært en veldig god løsning, da man ikke "brenner ut", og mister lysten til å jobbe med prosjektet.

Åpent arbeidsområde

Med bare to personer på prosjektet skal det ikke så mye til for å oppnå denne effekten. Arbeidsrommet som ble benyttet sørget for at hver av utviklerne hadde god plass til seg selv, men også at man satt så nært hverandre at kommunikasjon kunne foregå problemfritt uten å måtte bevege på seg. Utviklerne hadde en fast arbeidsplass, slik at man kunne bruke vegger til å henge opp notater og informasjon.

Planleggingsspillet

Denne delen av metoden fungerte veldig bra. Dersom man ser bort i fra at deler av bruks-tilfellene ble skrevet av utviklerteamet, som beskrevet over, var dette en suksess. Poengene kunden fikk tilgjengelig til å benytte ble regnet ut i forhold til hvor mye teamet hadde produsert i forrige iterasjon, og poengsummene på brukstilfellene ble også sjekket/justert før hver iterasjon. Kunden fikk da velge selv hvilke elementer som skulle implementeres i neste iterasjon. Det var ofte en god dialog mellom utviklere og kunde for å komme frem til hva som var mest hensiktsmessig å ta med.

Enkelt design

All kode ble hovedsakelig utviklet etter dette prinsippet. Målet var å unngå unødvendig kompleksitet. Dette kunne i noen tilfeller være litt frustrerende, siden begge utviklerne hadde en tendens til å tenke litt frem i tid for å forutse hva som kom til å trenges, og dermed tenke unødvendig komplekst. Utviklerne lykkes stort sett med å undertrykke disse impulsene, og det meste av koden ble utviklet etter dette prinsippet.

Refaktorering

Refaktorering ble flittig brukt. Prosjektet gjennomgikk noen få større refaktoreringer for å fjerne klasser som bare medførte unødvendig kompleksitet, men mindre refaktoreringer ble kontinuerlig brukt for å gjøre koden enklere og mer lesbar. En av farene kan være at utviklerne ikke har lyst til å fjerne kode de har fått ett forholdt til selv om denne er unødvendig, men i dette prosjektet oppstod ikke noen slike problemer. Utviklerne var positiv til å både rydde opp og fjerne så mye unødvendig kode som mulig, for å få en ren og enkel kodebase.

Metaforer

Utviklerne prøvde seg her på å lage noen metaforer, men kom raskt til den konklusjonen at metaforene var mye vanskeligere å forstå enn den direkte forklaringen, så dette ble gitt opp. Utvilerne ser her at dette kan være nyttig når ikke alle deltagerne i prosjektet er på samme tekniske nivå, gjerne spesielt hvis kunden er på et lavt teknisk nivå, men også dersom det er store forskjeller mellom utviklerne. I denne sammenhengen ble konklusjonen at metaforer bidro til å forvanske bildet, og ble derfor ikke brukt.

Konklusjon

Prosjektdeltagerne er meget fornøyd med XP som utviklingsmetode. De aller fleste prinsippene i XP ble fulgt slik metoden beskriver, og resultatet ble meget bra. Utviklerne føler at utviklingen gikk meget jevnt og smertefritt, og at tid ikke ble kastet bort på å dokumentere bare for å dokumentere. Deltagerne ser helt klart behovet for dokumentasjon, og XP er ikke en unnskyldning for ikke å skrive dokumentasjon, men det hele faller mer naturlig sammen med utviklingsprosessen, og blir ikke en separat prosess.

Siden det i dette tilfellet var mer enn en deltager, betydde det også at XP kunne praktiseres i sin helhet, i motsetning til tilsvarende prosjekter med bare en deltager, hvor en stor del av elementene i XP må elimineres eller endres i stor grad. Man kan derfor ikke unngå å stille seg kritisk til valget av XP som utviklingsmetodikk for mange enkeltmannsprosjekter som hevder å bruke dette, selv om XP er et vanlig valg i oppgaver av denne typen. Det skal også være sagt at XP har en del elementer som er nyttige også når man er alene, men forfatterne av denne oppgaven er av den oppfatning at når man bare plukker ut et lite knippe egenskaper fra XP, så bruker man ikke XP som utviklingsmetode. Det presiseres allikevel her at det finnes metodikk tilgjengelig for å tilpasse XP til en enkelt utvikler. En studie av dette ble gjort av Gareth Cronin ved Universitet i Auckland [103].

17 Design og oppbygning av systemet

17.1 Java

Programmeringsspråket Java har fått mye oppmerksomhet siden det først ble lansert i 1995, først og fremst på grunn av sin plattformuavhengige struktur. Java er etter hvert blitt et begrep som de fleste brukere av datasystemer har i alle fall et overfladisk forhold til. Spesielt i den senere tid, etter hvert som det er blitt vanlig at mobiltelefoner også støtter Java, selv om nok ikke alle engang vet at de bruker dette. Hva en stor del av disse ikke vet, er at Java ikke er et entydig begrep. Java finnes i flere utgaver. De to utgavene som er relevant for dette prosjekter er J2SE (Java 2 Standard Edition) [104] og J2ME (Java 2 Micro Edition) [105]. En annen mye brukt utgave av Java er J2EE (Java 2 Enterprise Edition) [35], men denne er ikke relevant for dette prosjektet, og vil ikke bli diskutert.

17.1.1 Grunnprinsipper i Java

Java er som nevnt over et plattformuavhengig språk. Dette vil si at Javakode som er skrevet på en datamaskin i prinsippet kan kjøres på en hvilken som helst annen maskin uavhengig av prosessorarkitektur og operativsystem. For å muliggjøre dette må det installeres en såkalt JVM (Java Virtual Machine) [106] på alle maskiner som skal kjøre programmet. En JVM er en virtuell maskin, et programvarelag som ligger mellom operativsystemet og koden som skal kjøres. Når man kjører en Java-applikasjon er det da JVMen som oversetter Javas plattform-uavhengige kode til plattformspesifikk kode under kjøring. Dette medfører at selve JVMen er sterkt plattformavhengig, mens Javakoden som den kjører ikke er det.

På de fleste moderne operativsystemer er en JVM allerede inkludert, og Java finnes som nedlasting til de aller fleste arkitekturer og operativsystemer i dag. Begrensningen i plattformuavhengigheten blir da at det er tilgjengelig en JVM for plattformen programmet skal kjøres på. JVMen er også forskjellig for de forskjellige utgavene av Java, så man kan ikke kjøre J2SE-kode på en J2ME-JVM (Her brukes også en lettvekts JVM som refereres til som KVM [107]). Det er dette som gjør at man ikke uten videre kan kjøre en Java-applikasjon beregnet for en datamaskin på en mobiltelefon, selv om begge støtter Java.

17.1.2 J2SE versus J2ME

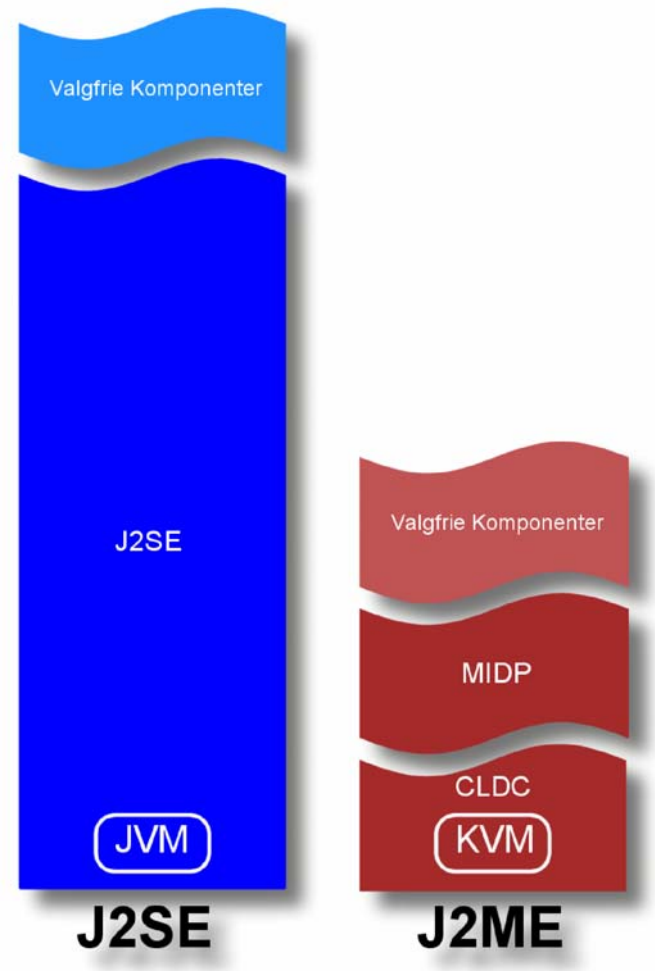
J2SE er den utgaven av Java som de fleste referer til når de snakker om Java. Det er denne utgaven av Java man installerer på vanlige PC-er for å kjøre Javaprogrammer.

J2SE er et rikt språk [108], med bred støtte for nettverk, multimedia, og en rekke andre teknologier. Når man installerer J2SE på en datamaskin, får man en komplett pakke som kan benytte seg av de fleste funksjonene på datamaskinen uansett merke og modell.

Gitt Javas rykte som plattformuavhengig, er det lett å tro at man uten videre kan flytte en Java-applikasjon som er skrevet for PC-en over på en mobiltelefon, forutsatt at mobilen er utstyrt med nødvendig hardware, slik som trådløst nettverk for en applikasjon som krever tilgang til nettverket. Det viser seg at dette ikke er så lett som man skulle ønske. Problemet her er at mobiltelefoner, og de fleste andre håndholdte enheter, kjører J2ME. J2ME er en lettvektsutgave av Java [109], spesialtilpasset for bruk på små enheter med lite minne og liten prosessorkraft. Og selv om syntaksen i de to Javautgavene er det samme, støtter J2ME kun et lite utvalg av alle de funksjonene man finner i storebroren, J2SE. En annen forskjell mellom

de to Javautgavene er at mens J2SE støtter de samme funksjonene overalt[1], så er J2ME forskjellig fra enhet til enhet. Alle enheter har en felles grunnpakke, men denne inneholder bare et minimum av funksjonalitet. På toppen av denne tilbyr hver mobile enhet et sett med profiler, som representerer de funksjonene som enheten har. Hvilke profiler som er tilgjengelig på hvilke enheter varierer sterkt, og gjør derfor at totalpakken som utgjør J2ME varierer sterkt mellom forskjellige enheter.

Forskjellene mellom J2SE og J2ME er illustrert på figuren under. Her ser man forholdet mellom innebygd funksjonalitet i de to Java-utgavene. Som man ser av figuren så kjører også J2ME en KVM i stedet for en JVM. Dette er en lettvektsversjon av JVMen, som krever veldig lite ressurser.



Figur 17.1: Sammenligning av Java SE og Java ME

17.1.3 Profiler

Det finnes et stort antall forskjellige profiler for J2ME. Noen av de viktigste profilene er som følger:

[1] Med forbehold om at det er samme versjonen av J2SE som er installert. Det er stor forskjell mellom for eksempel J2SE 1.4 og J2SE 1.5.

- Connected Limited Device Configuration (CLDC) [110]: Dette er en av to grunnpakker i J2ME. Denne profilen inneholder kjernefunksjonaliteten i J2ME, og brukes på enheter med meget begrenset minne og prosessorkraft, slik som mobiltelefoner og andre små, håndholdte enheter. Denne profilen kommer i to versjoner, versjon 1.0 og 1.1. Alle nyere enheter støtter versjon 1.1.
- Connected Device Configuration (CDC) [111]: Dette er den andre grunnpakken som blir brukt i J2ME. Den er beregnet for enheter med litt kraftigere tekniske spesifikasjoner, slik som avanserte håndholdte datamaskiner, set-top bokser og lignende. En enhet er basert på enten CLDC eller CDC, ikke begge.
- Mobile Information Device Profile (MIDP) [112]: Dette er en standardprofil til bruk på mobile enheter som benytter CLDC, og tilbyr grunnfunksjoner for mobile enheter, som blant annet mulighet for grafisk grensesnitt og nettverk. MIDP finnes også i to versjoner, versjon 1.0 og 2.0.
- Java APIs for Bluetooth (JSR-82) [94]: Denne profilen inneholder nødvendig funksjonalitet for Bluetooth-støtte.

I tillegg finnes det profiler for multimedia, andre typer nettverksbærere, og annen funksjonalitet som man i dag kan finne i en mobil enhet.

17.2 Bluetooth

Applikasjonen benytter seg av Bluetooth for å kommunisere mellom de selvstendige mobile enhetene. Det er derfor viktig å forstå de grunnleggende tekniske elementene ved Bluetooth for å ha et klart bilde av hvordan kommunikasjonen fungerer på et underliggende nivå. Dette kapittelet tar for seg virkemåten til Bluetooth i grove trekk, men for å få en dypere forståelse av Bluetooth anbefales en av de mange lærebøkene på markedet, samt Bluetooth-spesifikasjonene utgitt av Bluetooth SIG (Special Interest Group), som finnes offentlig tilgjengelig. [113] [114]

17.2.1 Hva er Bluetooth?

Bluetooth er en standard for trådløs kommunikasjon over radiobølger. Bluetooth opererer hovedsakelig på distanser i området fra en til ti meter, avhengig av sendestyrke på radioen i enheten. Bluetooth blir benyttet til små nettverk. Ofte består dette nettverket av kun to enheter, som for eksempel en mobiltelefon med håndsfri, som er et svært populært bruksområde for Bluetooth i dag.

17.2.2 Bluetooth som nettverk

Master/slave

Når to eller flere enheter er koblet sammen i ett nettverk, er den ene av disse enhetene master, og alle andre er slaver. Det er masterenheten som har kontrollen over kommunikasjonen, og det er denne som må ta initiativet til all kommunikasjon. Figur 17.2 illustrerer oppkoblingen mellom master og slave. En slave kan kun sende data til en master når denne eksplisitt ber om dette. Når to enheter kobler seg sammen er det enheten som initierer oppkoblingen som blir master, og den andre blir slave. En enhet kan godt være både master og slave samtidig dersom den er koblet opp til mer enn én enhet. Den er da master i den ene oppkoblingen, og

slave i den andre. En enhet kan kun ha en oppkobling som slave, men kan være master for inntil syv andre enheter.

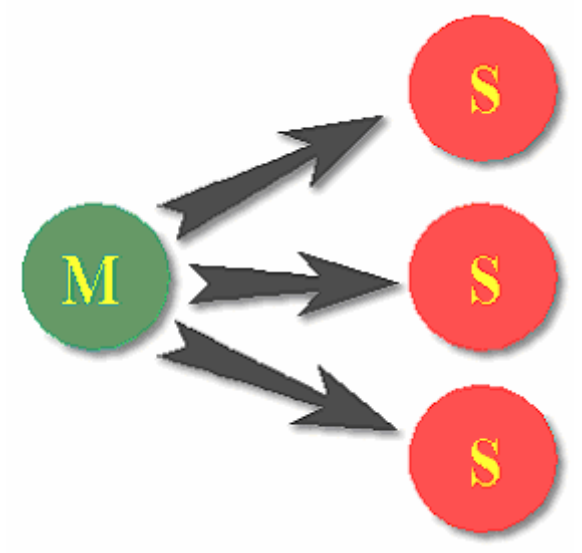
Dersom to enheter ønsker å bytte roller er det også mulig å utføre et såkalt master/slave-bytte, hvor den tidligere slaven nå blir master og motsatt.

Figur 17.2: Bluetooth Master/Slave tilkobling

Piconett

Når to eller flere enheter er koblet sammen i et Bluetooth-nettverk får vi det vi kaller et piconett. En av enhetene er da master, mens de andre er slaver. En master kan være koblet til inntil syv slaver i et piconett. Figur 17.3 viser et piconett med én master og tre slaver.

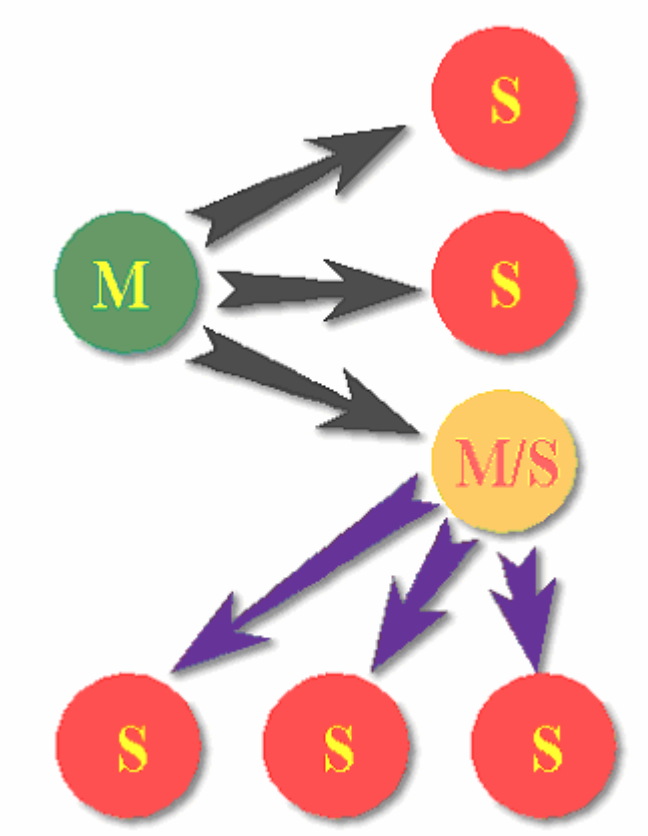
En viktig egenskap med et piconett er at all kommunikasjon alltid går mellom en master og en slave. To slaver kan aldri snakke direkte sammen, så skal en slave kontakte en annen må dette håndteres av masterenheten, ved at denne videreformidler kommunikasjonen. Det er ikke noe underliggende støtte i Bluetooth for dette, så det må den enkelte applikasjon selv implementere.



Figur 17.3: Bluetooth piconett

Scatternett

Siden et piconett er svært begrenset med tanke på antall enheter har også Bluetooth muligheten for såkalte scatternett. Dette er flere piconett slått sammen, hvor en enhet som er slave i ett piconett er master for et annet. I praksis viser det seg at det er meget vanskelig å sende data mellom klienter i to forskjellige piconett, og scatternett blir derfor normalt ikke benyttet i dag. Figur 17.4 viser et Bluetooth scatternett, hvor slaven i ett piconett er master i det andre nettet, og har selv tre slaver koblet til seg.



Figur 17.4: Bluetooth scatternett

17.3 Sentrale Problemområder

17.3.1 Bluetooth

Implementering av Bluetooth-støtte i en applikasjon har en del potensielle problemområder som må håndteres. Alle disse kan greit håndteres, så fremt implementeringen er fornuftig planlagt. For utviklere som ikke er så godt kjent med implementering av Bluetooth-støtte er det ikke alltid like lett å vite om disse områdene. De problemstillingene dette prosjektet møtte på i forbindelse med Bluetooth er beskrevet her.

Scatternett versus piconett

På grunn av problemene med scatternett ønsker man å forsikre seg om at man bruker et piconett så sant som dette er mulig. Problemstillingen her er at når en enhet prøver å koble seg til en annen enhet, er det enheten som kobler seg til som blir master. Prøver man å la tre enheter (klienter) koble seg til en fjerde (tjener), ender man opp med tre klienter som alle er

master enheter og som vil ha tjenerenheten som slave. Siden hver master enhet da danner sitt eget piconett, får vi tre piconett i et scatternett, og to feilede oppkoblingsforsøk siden tjener-enheten kun kan være slave i ett av nettene.

Master/slave-bytte

I utgangspunktet er master/slave-bytte en løsning på problemet over. Når en enhet mottar sin andre oppkobling kan den ikke være slave i dette nettet også, og skal da automatisk kunne ta rollen som master. Man kan også programmere dette slik at umiddelbart etter at den første oppkoblingen er gjort, så gjør tjeneren et master/slave-bytte, og blir da master for sitt eget piconett. Problemet her er at master/slave-bytte ikke er påkrevd implementert i henhold til Bluetooth-spesifikasjonen, og i praksis viser det seg at de fleste mobile enheter ikke støtter dette.

Løsningen

En av løsningene på dette problemet er å snu om på den klassiske klient/tjener-tankegangen, og få tjeneren til å kontakte klientene i stedet. Ved å gjøre klientene søkbare kan tjeneren finne disse enhetene, og initiere oppkoblingene fra tjenersiden. Da vil tjeneren være master siden det er denne som initierer oppkoblingen, og klientene vil automatisk bli slaver. Man ender da opp med bare en master, og ett enkelt piconett. Dette bidrar i stor grad til å forenkle håndteringen av Bluetooth-nettverk.

17.3.2 Java ME og Java SE

Da applikasjonen skulle utvikles, ble det bestemt av utviklerteamet at det ville være hensikts-messig å separere de forskjellige delene av applikasjonen i egne prosjekter i utviklings-miljøet.

Dette ble gjort for å kunne utvikle en applikasjonskjerne som ville være lik uansett miljø, og så utvikle forskjellige grafiske grensesnitt som brukte denne kjernen, herunder et grensesnitt skrevet i J2SE, for å kjøre applikasjonen på en vanlig PC, og et annet grensesnitt i J2ME for å kjøre applikasjonen på en mobil enhet. Når programmet så ble kompilert var det bare å kompilere det riktige grensesnittet, så inkluderte dette kjernen automatisk.

Siden mobiltelefonene som skulle brukes til utvikling ikke var ankommet da prosjektet startet, begynte utviklingsteamet først med å utvikle kjernen, og det grafiske grensenittet til å kjøre på PC-en. Denne utviklingen gikk relativt smertefritt, siden det her var snakk om tradisjonell J2SE-programmering. De største utfordringene her var å få kjernen til å oppføre seg korrekt slik at reglene for bridge ble fulgt.

Først da teamet begynt på utviklingen av grensenittet på mobile enheter ble ett av de større problemene avdekket. Kjernen var skrevet i Java versjon 1.6, mens J2ME bruker en sterkt redusert versjon av J2SE 1.3. Mellom Java 1.3 og Java 1.6 er det kommet en god del forbedringer, slik som annotations, generics, foreach-løkker[1] og en del andre nyttige funksjoner som var blitt benyttet i kjernemodulen. Her måtte det gjøres en jobb for å konvertere kjernen til rett Java-versjon. Utviklingsmiljøet som ble benyttet, NetBeans, hadde muligheten til å kunne definere hvilken versjon av Java koden skulle bruke, og NetBeans var til stor hjelp med å identifisere hvilke deler av koden som ikke var gyldig under Java 1.3, og

[1] Disse begrepene, samt andre Java-spesifikke begreper i dette avsnittet er forklart i begrepslisten som finnes i vedleggsseksjonen til rapporten.

dermed måtte skrives om. Dessverre var det et annet lite problem her. Siden Java ME er en redusert versjon av Java 1.3 klarte ikke NetBeans å avdekke alle problemområdene. Grunnen til dette var at kjernen var blitt definert som et vanlig J2SE-prosjekt, og ikke et J2ME-prosjekt. NetBeans trodde da at koden var gyldig, fordi den forventet at den skulle være J2SE 1.3-kode, og ikke J2ME-kode. Dette medførte at utviklerne måtte finne og rette disse feilene manuelt. Dette foregikk mye via prøv-og-feil metoden, og flittig bruk av kompilatoren. Autofullfør og lignende hjelpesystemer i utviklingsmiljøet virket heller ikke helt perfekt etterpå, siden den foreslo alt som var lovlig i J2SE 1.3, mens prosjektet krevde at koden var lovlig J2ME-kode.

Det hadde vært mulig å gjøre kjernemodulen om til et J2ME-prosjekt, med det ville ha brutt med designet om å gjøre denne så plattformuavhengig som mulig.

En av de største forskjellene som ble oppdaget mellom J2SE 1.3 og J2ME lå i Collections-klassen, som var svært mangelfull. I prosjektet var blant annet ArrayList benyttet flittig, men i J2ME fantes ikke denne, og bruk av denne måtte skiftes ut med Vector. Siden J2ME heller ikke støtter generics, måtte også typecasting taes i utstrakt bruk, noe som ikke hadde vært nødvendig dersom generics var støttet. Dette gjorde at koden muligens blir litt tyngre å lese, og det er vanskeligere for utenforstående å gjøre funksjonskall. Dette siden utviklingsmiljøet ikke vet hva en returnert Vector skal inneholde, og derfor heller ikke kan fortelle programmereren like godt hva som er feil.

17.3.3 Unntakshåndtering på mobile enheter

Mobiltelefonene hadde variabel støtte for å vise feilmeldinger (unntak) på mobilskjermen. Ved enkelte tilfeller viste Nokia-telefonen unntak i varselbokser og Sony Ericsson-telefonene ignorerte unntaket. Denne rare oppførselen ble også observert viseversa.

Det ble også opplevd en del uforklarlige feil under test på Sony Ericsson-telefonene. Under opprettelse av spillere i kjernemodulen oppstod det uforklarlige nullpekerunntak. Disse unntakene oppstod verken på mobilemulatoren eller på Nokia-telefonen. Dette problemet var ikke av typen som stoppet programmet, men hvis man fortsatte å trykke på "start spill"-knappen seks ganger fortsatte applikasjonen som om ingen feil hadde oppstått. Under testing vil dette ikke være et praktisk problem, men symptomene er av den art at de ikke kan være tilstede i produksjonsversjonen. I andre tilfeller oppstod lignende problemer på Nokia-telefonen. Denne kom med en dialogboks som fortalte om unntaket som hadde oppstått, men den lot også brukeren trykke på "fortsett" og fortsette spillet som om ingenting hadde skjedd.

I Java finnes det to forskjellige typer unntak; "unchecked"[1] og "checked"[2]. Unchecked har den egenskapen at de ikke må håndteres i koden. Er typisk eksempel på unchecked unntak er NullPointerException som indikerer at en variabel som blir brukt ikke blir initiert skikkelig. Disse unntakene vil "boble" opp til den virtuelle maskinen (JVMen) med mindre disse blir eksplisitt behandlet. Oppførselen til telefonene her viser at de ser ut til å praktisk talt ignorere unntak av typen "unchecked". Dette er en meget uheldig oppførsel, da en nullpeker som regel skjer fordi et objekt som ikke er blitt initiert blir brukt. For eksempel kan det være at spiller øst ikke er blitt opprettet. Å ignorere en slik feil betyr i praksis at applikasjonen er i en udefinert tilstand, og at man ikke kan stole på at det applikasjonen gjør er korrekt.

[1] java.lang.RuntimeException

[2] java.lang.Exception

Løsningen ble å sømfare applikasjonen for å forsikre seg om at slike unntak aldri kan skje. Applikasjonen ble i stedet satt til å sjekke om objektet er opprettet, og opprette et nytt korrekt objekt dersom dette ikke er tilfelle.

17.3.4 Skjermstørrelser

En av utfordringene med å lage en applikasjon for mobile enheter var å lage den på en slik måte at den ikke var bundet til noen spesifikk skjermstørrelse. Som diskutert i seksjonen om rammeverk tidligere klarte ikke utviklerne å finne et rammeverk som kunne bidra til å gjøre dette automatisk når man benyttet seg av lavnivå GUI i J2ME. Løsningen måtte derfor bli å finne en god måte å gjøre dette på selv.

Første punktet som ble behandlet var behovet for et kvadratisk bridgebord, samt en plass å vise diverse informasjon. Siden skjermer på mobile enheter så godt som aldri er kvadratiske, ble det bestemt at kvadratet som ble dannet ved å bruke skjermens minste dimensjon skulle bli bordet, mens resten av skjermen ble til et informasjonsfelt. Siden noen mobiler har liggende skjerm, mens andre har stående, ble det brukt en enkel formel for å tegne opp informasjonsfeltet alt etter om det kom på toppen eller venstresiden av skjermen. Selve bordet var kvadratisk, så det ville bli tegnet likt uansett orienteringen på skjermen.

Som tidligere diskutert, så ble minste akseptable skjermstørrelse på skjermen definert som 128x160 punkter (siden applikasjonen håndterer stående og liggende skjermer korrekt, er 128x160 og 160x128 begge akseptable skjermstørrelser). Denne størrelsen ga da et bord som var minimum 128x128 punkter stort. I den andre enden er vanligvis den største tilgjengelige skjermen i dag 240x320, som gir et bord på 240x240 punkter. Disse to ble derfor definert som praktiske yttergrenser for applikasjonen. Det ble så brukt en enkel matematisk formel for å plassere de visuelle elementene slik at disse ble korrekt plassert uansett skjermstørrelse. Enkelte mindre viktige elementer, slik som baksidene på motstandernes kort ble tildelt en liten prosentandel av tilgjengelig skjermplass, slik at på større skjermer var disse mer synlige, mens på mindre skjermer forsvant disse nesten helt. På denne måten fikk kjerneelementene plass på skjermen og så visuelt korrekt ut uansett størrelse innenfor det definerte området. Større skjermer enn dette vil fremdeles ha et korrekt visuelt bilde, med korrekt sentrering, men vil ha mindre reell nytte av den større skjermen.

Et annet problem med visningen var at forskjellige mobiltelefoner har forskjellige skriftstørrelser. For eksempel bruker Nokia N95-telefonen en mye større skriftstørrelse enn Sony Ericsson W960i-telefonen. Dette medførte at selv med like mange punkter tilgjengelig, var det ikke plass til å skrive det samme på skjermene. Dette var heldigvis relativt enkelt å løse, siden Java har en innebygd metode for å regne ut høyden og bredden på en gitt bokstav. Dermed var det en smal sak å bruke disse verdiene til å plassere riktig mengde tekst på skjermen. Et veldig synlig eksempel på dette er tabellen i meldedelen hvor spilleren velger hvilken melding vedkommende ønsker å gi. Antall elementer i denne er veldig forskjellig mellom de to testtelefonene, til tross for at skjermstørrelsen er lik.

17.4 Oversikt over løsningens oppbygning

17.4.1 AI-spiller

En av utfordringene med utviklingen av denne applikasjonen var at bridge krever fire spillere, verken mer eller mindre. For å teste ut applikasjonen underveis ville det være hensiktsmessig at utviklerne ikke måtte styre alle spillerne selv. Man ville da ikke fått testet applikasjonen

slik den vil oppleves av en bruker, som naturlig nok bare vil ha kontroll over én av spillerne (eller to hvis makkeren er blindemann). Også for enhetstesting av metoder som krevde input fra spillerne ville det være en fordel at utviklerne ikke måtte manuelt legge inn disse, siden slike tester helst skal være helautomatiske. Siden testene skulle teste reell fremdrift i et bridgespill, var det heller ikke helt hensiktsmessig å bare teste med forhåndsdefinerte input.

Utviklerne bestemte seg da for at den beste løsningen her ville være å lage en såkalt AI-spiller (Artifical Intelligence – kunstig intelligens), som kunne ta del i spillet akkurat som en vanlig spiller. Denne spilleren skulle kunne både melde og spille kort, og den måtte også holde seg til reglene for bridge, slik at den ikke laget problemer i spillet.

Det ble bestemt at for å ikke bruke for mye tid på å lage avanserte algoritmer, så skulle den lages så enkel som mulig. Løsningen ble derfor å gjøre handlingene til spilleren helt tilfeldig. For meldedelen hentet AI-spilleren inn listen over lovlige meldinger akkurat da, og valgte tilfeldig blant de fem første. Dette ble gjort for at meldedelen skulle bli noenlunde normal, til tross for AI-spillerens tilfeldige valg. Denne listen med lovlige meldinger er den samme som blir hentet inn og vist i det grafiske grensesnittet når en menneskelig spiller skal melde. Listen inneholder spesielle meldinger som pass, dobling/redobling dersom noen av disse er lovlig på gjeldende tidspunkt, og alle meldinger som er høyere en den nåværende gjeldende meldingen.

I selve spilldelen plukker AI-spilleren ut et helt tilfeldig kort fra hånden, og sjekker så med selve spillmotoren om dette kortet er gyldig akkurat nå. Dersom kortet ikke er gyldig fortsetter den med å velge et nytt tilfeldig kort til den finner ett som er gyldig. Med en gang den har funnet et kort som er gyldig, spiller den dette. Denne måten å plukke kort på kan være tidkrevende dersom man har svært mange kort på hånd, men i bridge har man maks 13 kort, og i snitt vil ca 40 % av kortene man har til enhver tid være gyldige[1]. Dette medfører at det skal relativt få forsøk til for å finne et gyldig kort, og funksjonen som sjekker om et kort er gyldig er heller ikke spesielt krevende. Denne løsningen ble dermed akseptert som god nok for den bruken den har, spesielt siden AI-spillerne ikke er tiltenkte å benyttes i den ferdige applikasjonen uansett. AI-spillere kan nok være et interessant tillegg, men det ligger utenfor omfanget til dette prosjektet.

Alt i alt var inklusjonen av AI-spillere meget nyttig for utviklingen av prosjektet. Enhetstestene kunne kjøres med input fra et "reelt" bridgespill, uten å måtte ty til manuell input, og man kunne også teste applikasjonen ved å sette fire AI-spillere til å spille og bare observere spillets gang. Siden det kun var to utviklere på prosjektet, ble det også gjort en del testing av applikasjonen med to menneskelige spillere og to AI-spillere, noe som ga en veldig god følelse av hvordan det applikasjonen oppførte seg under faktisk bruk.

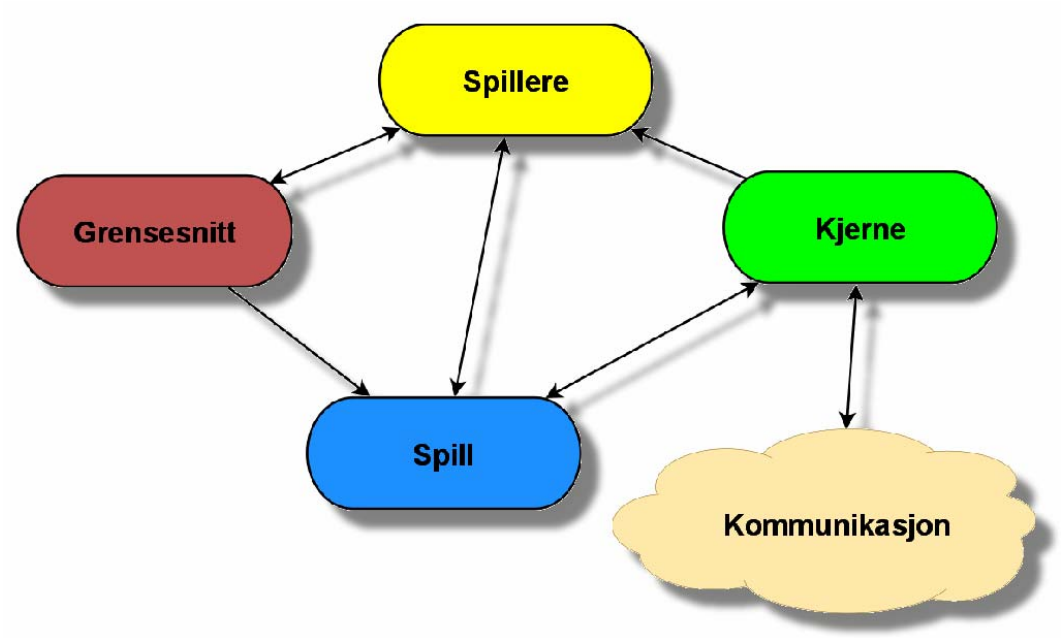
17.4.2 Arkitektur

Da applikasjonen skulle designes ble den delt inn i en gruppe med hovedkomponenter. Siden denne applikasjonen ble utviklet med XP som utviklingsmetode var ikke alle disse hoved-komponentene påtenkt ved oppstart av prosjektet, så denne beskrivelsen representer inn-delingen ved prosjektets slutt. Disse hovedkomponentene er som følger:

[1] Basert på at man i snitt har like mange kort i hver farge siden kortstokken er uniformt fordelt, og at man i snitt spiller ut det første kortet i et stikk ca 25 % av tiden, hvor 100 % av kortene er gyldige. Andelen 40 % er bare et grovt estimat, og ikke en nøyaktig statistisk utregning siden dette ikke har noen hensikt i denne sammenhengen.

- Kjerne
- Spill
- Spillere
- Kommunikasjon
- Grensesnitt

Figuren under viser hvordan disse hovedkomponentene kommuniserer med hverandre.



Figur 17.5: Arkitekturen til bridge-applikasjonen

Kjernen er grunnkomponenten i hele systemet. Denne er ansvarlig for å drive spillet fremover. Kjernen er implementert som en tilstandsmaskin. I hver av de forskjellige tilstandene driver den spillet fremover ved å kontinuerlig iterere gjennom et gitt sett med kommandoer. For eksempel når den er i tilstanden for meldedelen, ber den hele tiden spillet om å hente melding fra neste spiller, og når den er i selve spilldelen kaller den hele tiden spillet for å starte et nytt stikk når det forrige er fullført. Kjernen vet ikke hvilken spiller som skal melde, eller hvor mange stikk som er spilt. Dette er informasjon som spillet holder rede på.

Spillet er et objekt som drives av kjernen. Spillet har metoder som kalles av kjernen for å utføre handlinger i spillet. Spillet selv har ikke noe forhold til hvilken fase spillet er i, og kjører kun metoder etter hvert som kjernen kaller disse. Spillet inneholder metoder for å hente informasjon om spillet, slik som hvor mange stikk hvert makkerpar har tatt, hva kontrakten er, hvilke meldinger som er gitt så langt i runden, hvilket spill man er kommet til, og metoder for å validere om et kort eller en melding er gyldig på nåværende tidspunkt. Spillet inneholder også metoder for å regne ut poeng etter et spill er ferdig.

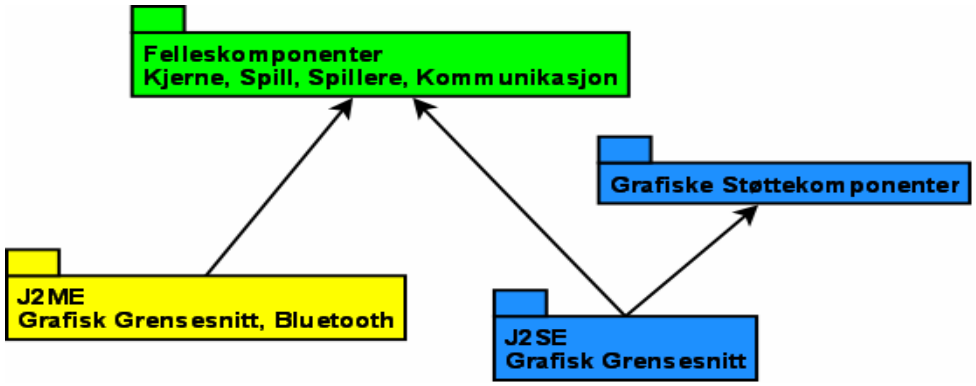
Spillerne er objekter som representerer de faktiske spillerne i spillet (nord, sør, øst og vest). Det finnes flere typer spillere, som for eksempel AI-spillere som handler automatisk, nettverksspillere som får instruksjonene sine fra en annen enhet i nettverket, og vanlige

spillere som er styrt av personen som spiller spillet. En nettverksspiller representerer vanligvis en menneskelig spiller på en annen enhet, men denne kan representere en hvilken som helst type spiller som styres fra en annen fysisk enhet. Det er den andre enheten som er ansvarlig for å holde rede på dette.

Kommunikasjonsdelen tar hånd om datakommunikasjon mellom enheter, og sørger for at hendelser blir sendt videre til tilkoblede enheter. Den enheten som opererer som tjener sørger for at all kommunikasjon denne får inn fra en enhet blir sendt ut til alle de andre enhetene i nettverket.

Grensesnittet er brukerens vindu mot applikasjonen. Grensesnittet presenterer brukeren med den nødvendige informasjonen, og gir brukeren mulighet til å gi de nødvendige inputer til programmet. For eksempel så vises en meldingstabell under meldedelen, hvor brukeren kan velge ønsket melding, mens under selve spilldelen lar grensesnittet brukeren velge et kort som skal spilles.

I tillegg til inndelingen over, var det også påtenkt at programmet skulle kunne kjøre på forskjellige typer enheter, både enheter som kjørte J2SE og enheter med J2ME. De grafiske komponentene og tilgjengelige metodene er svært forskjellig mellom disse to Java-utgavene, så det ble utviklet ett grensesnitt for hver av disse. For å organisere prosjektet på en oversiktlig måte, ble felleskomponenter laget som et eget prosjekt i utviklingsmiljøet, mens det ble opprettet egne prosjekter for grensesnittet. Et standard J2SE-prosjekt ble opprettet for et grensesnitt som kunne kjøres på en vanlig datamaskin, mens et J2ME-prosjekt ble opprettet for bruk på mobile enheter. I disse prosjektene ble all kode som var spesifikk til én av de to utgavene av Java plassert. Dette gjelder i hovedsak det grafiske grensesnittet, som har vesentlige forskjeller, men også Bluetooth er rimelig forskjelling mellom J2ME og J2SE, så denne støtten ble også lagt direkte i J2ME-prosjektet. Disse to prosjektene ble så gjort avhengig av prosjektet som inneholdt felleskomponentene, slik at når ett av disse ble kompilert, ble også felleskomponentene automatisk inkludert. Figuren under viser en grafisk illustrasjon av hvordan prosjektene henger sammen.



Figur 17.6: Inndeling i logiske prosjekter

På denne måten unngikk man å vedlikeholde to kopier av samme kode, eller eventuelt manuelt kopiere koden mellom prosjektene når det ble gjort endringer. Som beskrevet tidligere gjorde denne oppdelingen at enkelte problemer oppstod på grunn av kodeversjonen på fellesprosjektet, men her skal det sies at dette skyldtes hovedsakelig kortsiktighet fra utviklerne, og er en erfaring til neste gang. Problemene her ble ryddet opp i, og etter dette fungerte denne løsningen meget bra.

18 Implementering

18.1 Presentasjon av applikasjonen

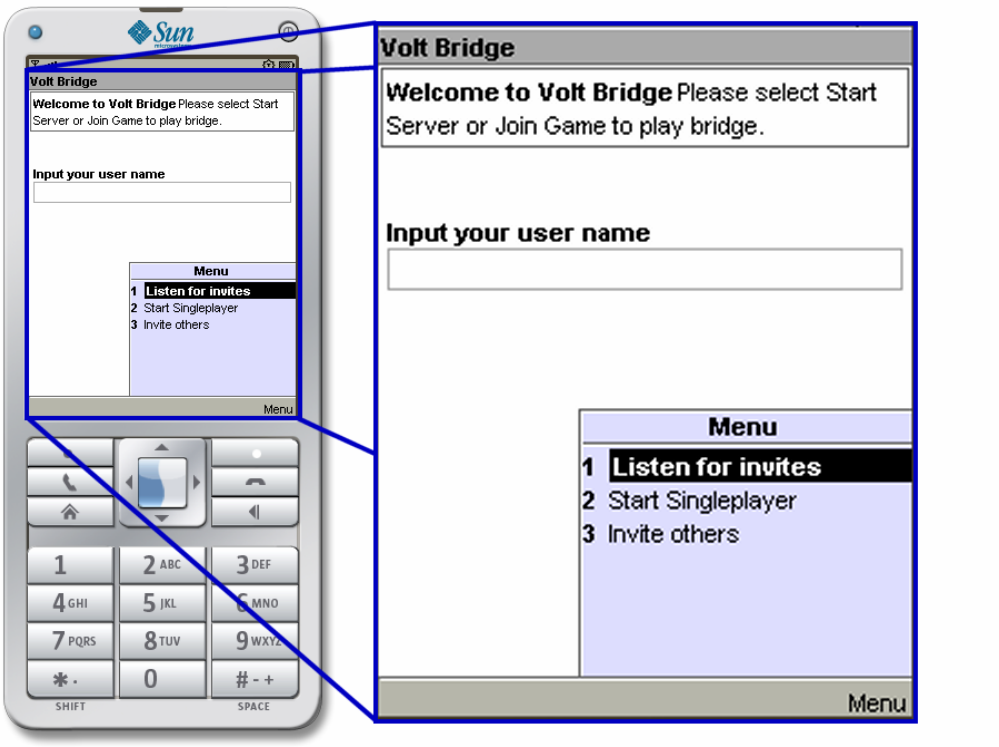
Under utviklingen fikk prosjektet kodenavnet "Volt". Siden selve applikasjonen ikke er helt klar for det åpne marked enda, har den ikke fått noe endelig navn. Den blir derfor omtalt "Volt Bridge".

Volt Bridge er en applikasjon for å kunne spille bridge via en portabel enhet, vanligvis en mobiltelefon. Foreløpig støtter den bare oppkobling til andre enheter via Bluetooth, men applikasjonen er utviklet på en slik måte at det skal være enkelt å implementere andre typer databærere, slik som TCP/IP.

Volt Bridge følger per i dag reglene for bridge korrekt, men den mangler enkelte funksjoner som bridgespillere kan finne nyttig, slik som for eksempel å kreve resten av stikkene. Ingen av de manglende funksjonene er strengt tatt nødvendig for å kunne spille bridge, men de har en praktisk nytte som ville gjøre applikasjonen litt mer brukervennlig dersom de var til stede.

18.1.1 Oppstart av spill, oppkobling

Før man kan begynne spillet må man koble seg opp til de andre spillerne. Foreløpig er den eneste måten å gjøre dette på via Bluetooth. Som tidligere beskrevet i avsnittet om Bluetooth, benytter denne seg av et master/slave-prinsipp. Av praktiske grunner når man skal spille et spill over nettverket, må den ene telefonen være satt opp som tjener, og de andre klienter. På grunn av arkitekturen til Bluetooth, bør også tjeneren være master i et piconett, med klientene som slaver. Under utviklingen ble det oppdaget at veldig få mobile enheter støtter et såkalt master/slave-bytte, og dette betyr i Bluetooth-verdenen at det er tjeneren som må initiere kommunikasjonen mot klientene, og ikke omvendt. Det ble derfor valgt en løsning hvor hver klient satte seg i en lyttemodus, som tillot tjeneren å oppdage dem, og at tjeneren gjorde et søk etter tilgjengelige klienter. Dette kan umiddelbart virke litt bakvendt dersom man er kjent med tradisjonell nettverksarkitektur, men er det mest fornuftige alternativet når man skal sette opp et Bluetooth-piconett med mer enn to enheter.



Figur 18.1: Skjermbilde av emulator og oppstartsskjerm

Første skjermbilde viser åpningsbildet til applikasjonen. Etter å ha skrevet inn ønsket navn, kan man velge hvordan man vil starte spillet. Klienter vil her velge å lytte etter invitasjoner, mens tjeneren vil velge å invitere andre. Tjeneren vil da finne alle klientene som lytter etter invitasjoner, og invitere disse til spillet.

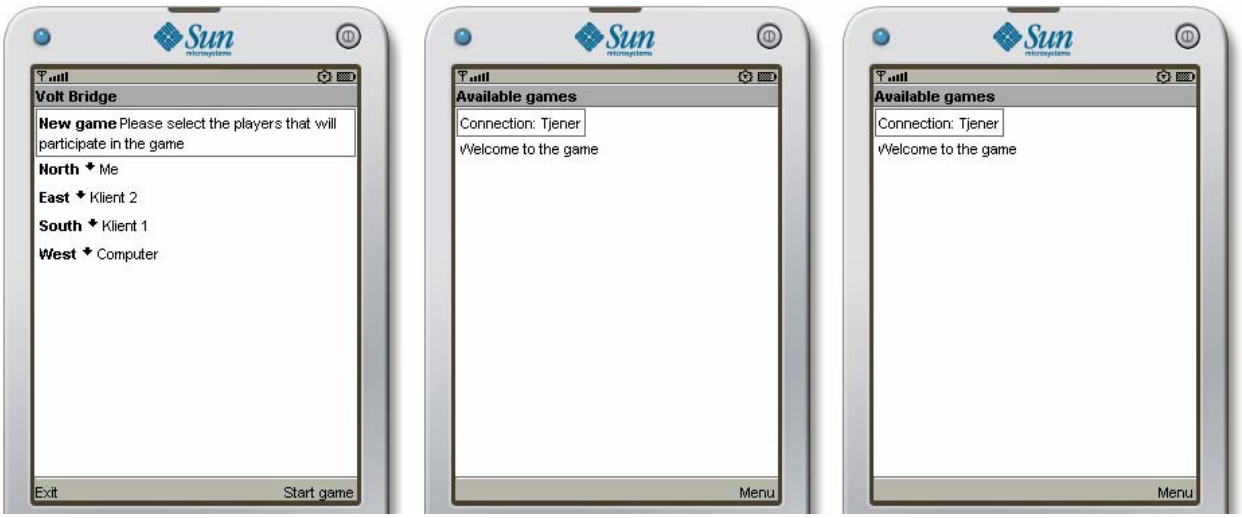
Figur 18.2: Skjermbilder fra klient og tjener under oppkobling

Etter at brukeren har gjort sitt valg i hovedmenyen, vil nå klientene lytte etter tilkoblinger fra tjeneren, mens tjeneren søker etter tilgjengelige klienter. Skjermbildet over viser en tjener til venstre, har funnet to tilgjengelige klienter. Skjermbildene over er hentet fra Suns mobil-telefonemulator, og her har alle enhetene samme Bluetooth-navn, "WirelessToolkit". På en

virkelig mobiltelefon ville tjeneren her ha vist det navnet brukeren har gitt til mobiltelefonen sin. Som standard er dette ofte satt til produsent og/eller modellnummer når telefonen er ny fra fabrikk. Applikasjonen lister her Bluetooth-navnet og ikke navnet brukeren skrev inn da applikasjonen ble startet, fordi dette navnet ikke er tilgjengelig før etter at en forbindelse er satt opp med den andre enheten. Tjeneren lister så opp etterpå de klientene den har klart å koble seg opp til. Etter at oppkoblingen er gjort kan man også hente inn navnet brukeren skrev inn i hovedmenyen da applikasjonen ble startet, og det er dette som vises her.

På klientene ser vi at de har blitt kontaktet av en tjener og navnet på denne. Igjen så er dette navnet brukeren skrev inn når vedkommende startet tjeneren.

Brukerne på klientene må nå vente til tjeneren starter spillet. Tjeneren må først tilordne spillerne hver sin plass rundt bordet før spillet kan begynne.



Figur 18.3: Skjermbilde fra klient og tjener under oppsett av spill

I skjermbildet over har tjeneren gått videre til oppsettet. Dersom det ikke er nok klienter tilkoblet vil de manglene posisjonene bli fylt inn med AI-spillere. Tjeneren kan nå velge hvilken plassering rundt bordet hver av spillerne skal ha. Når tjeneren er fornøyd med posisjonene kan den starte spillet, og alle klientene går da automatisk videre til selve spillet.

18.1.2 Spillet

Meldedelen

Figur 18.4: Skjermbilder fra meldedelen

Første del av hvert spill er meldedelen. Her vises skjermbildet til tre av spillerne rundt bordet. Som på tidligere skjermbilder er tjeneren helt til venstre, mens klientene er de andre to. Dette spiller en mindre rolle på dette stadium, da alle er likeverdige sett fra spillets perspektiv. Kun de underliggende kommunikasjonsrutinene trenger å skille mellom tjener og klient.

Den lokale spilleren på hver enhet vises alltid nede på skjermen, som vist på skjermbildene over. Dette er naturlig både i forholdt til grensesnittene på de fleste andre elektroniske kortspill, og også slik en spiller ser kortene når vedkommende spiller vanlig bridge.

I meldedelen kan man se at aktiv spiller har fått en markør på skjermen som brukes til å velge ønsket melding. De andre spillerne har ingen markør, som betyr at det ikke er deres tur enda. Når første spiller har avgitt sin melding forsvinner markøren fra hans skjerm, og vil i stedet bli synlig på skjermen til neste spiller. Informasjonsruten viser hvilket spillnummer spillerne holder på med, hvilken posisjon man har, og informasjon om hvem som er i faresonen.

Etter hvert som meldingene avgis kan man se at disse blir listet i en tabell på skjermen. Meldingslisten helt til høyre viser også kun de til enhver tid lovlige meldinger basert på hva som allerede er meldt.

Spilldelen

Figur 18.5: Skjermbilder fra spilldelen

Skjermbildet over viser nå et spill som er gått inn i selve spillfasen. Som i meldedelen har aktiv spiller en markør som viser at det er denne spillerens tur, og som spilleren bruker til å velge kort fra hånden med. Her ser man også at sør er blitt blindemann, og at alle spillerne kan se sørs kort. Siden blindemann ikke har en aktiv rolle, og ikke selv kan styre sine kort, får blindemann se kortene til alle spillerne. Dette er gjort for å gjøre det litt mer interessant for blindemann å følge med på spillet. Vi ser også at nord, som er blindemannens makker, har kontroll over blindemannens kort, og kan spille disse. Markøren på nords skjerm vil hoppe mellom egne og blindemannens kort alt etter hvem sin tur det er til å legge på.

Skjermbildet viser også at informasjonslinjen på toppen nå også viser kontrakten, og hvor mange stikk som er tatt av hvert makkerpar.

Poeng

Figur 18.6: Skjermbilder fra poengskjerm

Etter alle stikkene er spilt går spillet videre til poengskjermen. Dette er siste stopp før spillet går tilbake til meldedelen og starter et nytt spill. Skjermen her viser poengsummene for hvert spill, samt samlet poengsum for hele sesjonen. Denne skjermen vises en kort stund, før neste spill startes.

18.2 Tilbakemelding fra brukertest

Ved slutten av applikasjonens utvikling ble den testet av tre medlemmer fra Bergen Akademiske Bridgeklubb. Dette var mennesker som var godt kjent med å spille bridge i elektroniske former, og disse var derfor en veldig god testgruppe. Tilbakemeldingene fra disse medførte at noe av funksjonaliteten i applikasjonen ble endret som følge av deres kommentarer. Disse tilbakemeldingene og endringene er beskrevet nedenfor. Testpersonene hadde også en del tilbakemeldinger som det ikke var mulig å implementere i denne omgang på grunn av tilgjengelig tid. Disse er nærmere beskrevet i avsnitt *19.2 Videre arbeid.*

18.2.1 Tilbakemeldinger og endringer

Kortene forsvant for fort

Testgruppen mente at kortene forsvant for fort fra skjermen når siste kort i hvert stikk var lagt på, slik at det var vanskelig å få med seg hva som faktisk ble spilt.

Dette ble fikset ved å legge inn en liten tidsforsinkelse, slik at spillet ikke gikk videre umiddelbart.

Poengskjerm forsvinner for fort

Poengskjermen i spillet var lagt opp slik at den forsvinner fra alle skjermene samtidig når en av spillerne klikket den bort. Dette var implementert slik for å holde spillet synkront på alle enhetene. Dette mente testgruppen var en dårlig løsning, siden noen gjerne klikket denne bort før de andre fikk lest den.

Her ble dette endret ved å fjerne muligheten til å klikke vekk skjermen, og heller legge inn en liten tidsforsinkelse på fem sekunder. Denne løsningen ble valgt på grunn av liten tilgjengelig tid til utvikling på dette tidspunktet, og bør nok på et senere tidspunkt erstattes med en løsning der spillerne kan klikke vekk skjermen enkeltvis, men at spillet ikke fortsetter før alle spillerne har klikket vekk denne skjermen.

Markøren hopper tilbake til første kortet når et kort blir spilt

Det grafiske grensesnittet tegner i dag opp en liten markør rundt det kortet en spiller har markert så lenge det er denne spillerens tur. Etter at en spiller har lagt på et kort blir denne satt tilbake til det første kortet på hånden, dette grunnet at kortet spilleren la ut blir fjernet fra hånden, og ikke lengre er et gyldig kort å markere. Testgruppen mente dette var litt forvirrende, og gjorde det litt uklart om man faktisk hadde valgt riktig kort.

Dette ble endret ved å fjerne denne markøren helt når spilleren hadde valgt hvilket kort som skulle spilles. Markøren hopper fremdeles tilbake til første kort teknisk sett, men den blir ikke lengre tegnet opp på skjermen.

19 Konklusjon og videre arbeid

19.1 Oppsummering

Utvikling av programvare i Java for mobile enheter har en del aspekter som gjør dette forskjellig fra tradisjonell Java-utvikling for vanlige datamaskiner. Noen av de viktigste aspektene her er at J2ME er en svært strippet versjon hvis den sammenlignes med J2SE. En stor del av systemklassene man kjenner fra J2SE mangler, og blant de som finnes er mange av dem også redusert i funksjonalitet. Et annet viktig aspekt er måten J2ME benytter profiler. Ikke alle enheter har de samme profilene tilgjengelige, og programmer som benytter funksjonalitet fra en profil vil kun fungere på enheter som implementerer denne profilen. I dette prosjektet ble kun standardprofilene (CLDC og MIDP), samt JSR-82 (Bluetooth) benyttet. Siden programmet ble tilnærmet meningsløst uten Bluetooth-støtte var det akseptabelt at denne var påkrevd for at applikasjonen skulle kjøre. For andre applikasjoner, som for eksempel kan spille av lyd, kan dette være mer kritisk, da det kan være ønskelig at applikasjonen skal kjøre selv om lyd ikke kan spilles av, men at lyden selvsagt skal spilles av hvis mulig. Da må man som regel kompilere flere versjoner av programmet, for de forskjellige enhetene. Det tredje aspektet som dette prosjektet opplevde var at en del av grunnfunksjonaliteten i J2ME også var leverandøravhengig. Dette var blant annet tydelig på hvordan de forskjellige telefonene håndterte unntak.

Selve implementeringen av Bluetooth hadde også utfordringer. Enkelte deler av Bluetooth-spesifikasjonen er definert som frivillig for den enkelte leverandør å implementere. Det var tydelig at telefonene som ble brukt under utviklingen ikke støttet alle disse, slik som master/slave-bytte. Dette medførte at utviklerne brukte en god del tid på å få Bluetooth til å fungere som ønsket. Utviklerne opplevde også veldig tydelig forskjellene i implementeringen, ettersom svært enkle endringer i kildekoden medførte at plutselig endret det seg hvilke enheter som ville kommunisere sammen. Som regel var situasjonen at når en endring fikk to enheter til å snakke sammen, så var de to andre som ikke ville snakke i sammen. Utviklerne klarte aldri å få alle tre telefonene brukt i utviklingen til å snakke korrekt og pålitelig sammen. Kommunikasjonen mellom de to hovedtelefonene fungerte derimot utmerket, og av praktiske hensyn holdt utviklerne seg da til disse. Den simulerte Bluetooth-kommunikasjonen mellom emulatorene fungerte også utmerket, og siden disse er utgitt av Sun, som også er ansvarlig for spesifikasjonen (JSR-82), ble det bestemt at dette var et tegn på at applikasjonen fungerte på korrekt måte. Dersom det hadde vært mer tid tilgjengelig kunne det nok ha vært fordelaktig å bruke mer tid på å få Bluetooth til å virke riktig mot alle enheter, men dette ble for tidkrevende for dette prosjektet.

Alt i alt gikk utviklingen relativt bra. Utviklerne brukte en del tid på å løse J2ME-spesifikke problemer som man ikke hadde opplevd under J2SE-utvikling, noe som medførte at applikasjonen totalt sett fikk mindre funksjonalitet enn det som kunne ha vært ønskelig. Dette kom ikke som noen overraskelse på utviklerne, da disse var klar over at J2ME hadde en del særegenheter som kunne ta en del tid å løse. Ved prosjektets start var derimot ikke utviklerne klar over nøyaktig hvor disse særegenhetene lå. For utviklerne har dette prosjektet blitt en innføring i J2ME og forskjellene mellom J2ME og J2SE. Utviklerne føler ikke at J2ME har vært noe uoverkommelig hinder, men kun forårsaket at man måtte bruke en del ekstra tid på problemløsning. Mye av denne ekstra tiden skyldes nok at ingen på teamet hadde noe mer enn overfladisk kjennskap til J2ME, og hadde ikke programmert annet enn enkle applikasjoner i dette miljøet tidligere. Erfaringene som teamet har opptjent under dette prosjektet vil

nok medføre at det blir betraktelig mye lettere å utvikle en J2ME-applikasjon på et senere tidspunkt.

19.2 Videre arbeid

Selv om programmet som ble utviklet i forbindelse med oppgaven fungerer etter hensikten, er det helt klart at det her ligger mye potensial for videreutvikling. Spesielt hvis programmet skal benyttes av allmennheten er det en del som bør gjøres.

For å få kartlagt hvor godt programmet var ved utviklingsslutt, og hva som burde gjøres videre med applikasjonen, fikk en liten gruppe teste programmet. Testgruppen bestod av ivrige bridgespillere, som alle var godt vant til å spille bridge via elektroniske medier, slik som BBO.

19.2.1 Forbedring av brukergrensesnittet

Selv om testpersonene meldte tilbake at programmet var lett å forstå og bruke, hadde de også en del forslag til ting som kunne bli gjort annerledes for å gjøre spillet mer oversiktlig. Ettersom disse tilbakemeldingene kom fra erfarne bridgespillere, må disse betegnes som høyst relevante for å gjøre spillet så intuitivt og attraktivt som mulig. De tilbakemeldingene som ble umiddelbart tatt hensyn til og implementert er beskrevet i avsnitt *18.2 Tilbakemelding fra brukertest*. Resten av tilbakemeldingene er presentert her.

- Grafisk fremstilling av faresone: I dag vises kun faresonen som en tekststreng i informasjonsfeltet til applikasjonen. Testgruppen mente at programmet ville være mer oversiktlig hvis denne ble vist grafisk under meldedelen. Det var også ønske om at denne informasjonen ble vist under spillets gang. Her mente de at det var godt nok å vise at man var i faresonen ved for eksempel å vise kontrakten i rød skrift dersom man var i faresonen, ellers hvit skrift.
- Bedre representasjon av kort på hånd: I dag tegnes håndkortene opp i to rader, fordi dette er en metode som gjør det lett å sikre en jevn fordeling av kort på skjermen, slik at kortene blir synlige selv på små skjermer. Dette er en løsning som testgruppen mente kunne være litt uoversiktlig. Ideelt burde det her være fire rader, en for hver farge. Dette kan muligens løses ved å la applikasjonen sjekke hvilken skrift- og skjermstørrelse som er tilgjengelig på enheten, og tegne dette opp på fire rader dersom det er plass til dette.
- Visning av gjeldende spiller: Spillet viser i dag kun når det er din tur. Midt i et stikk eller i meldedelen er det lett å se hvem sin tur det er ved å se på meldingene eller kortene som allerede er spilt, men i begynnelsen av et spill, før det er gitt en melding, eller før første kortet i et stikk er spilt, kan dette være vanskelig å finne ut av. Det bør derfor implementeres en markør som viser alle spillerne hvem som er den aktive spilleren akkurat nå.

19.2.2 Meldetrening

En viktig del av bridge er å melde riktig. Dette er derfor noe som det er interessant for bridgespillere å trene på. Det er også veldig ofte man ikke er fire stykker samlet, og derfor ikke har muligheten til å spille et fullt spill, for eksempel når man deler et offentlig transportmiddel som buss eller tog med en annen bridgespiller. Det ville derfor være meget interessant å tilby meldetrening for to personer via denne applikasjonen. Siden applikasjonen

allerede støtter meldedelen i bridge, vil dette kunne implementeres med et minimum av arbeid. Her kan man se for seg at meldedelen blir startet på vanlig måte, men man bruker to dummyspillere som motstandere. Disse kan melde pass hele tiden, slik at de ikke påvirker meldetreningen til spillerne. De trenger heller ikke nødvendigvis å vises på skjermen. Dummyspillerne kan også komme med faktiske meldinger, men meldingene bør i så fall være noenlunde fornuftige hvis meldetreningen skal ha noe hensikt. Videre må selve spilldelen hoppes over, siden det her kun er snakk om meldetrening.

19.2.3 Fasitspill

I bridge kan man spille såkalte fasitspill. Det vil si at man spiller et spill med identiske hender til et tidligere spilt spill, og sammenligner så resultatet for å se hvordan man gjorde det i forhold til de som har spilt dette tidligere. Applikasjonen støtter allerede å bruke en forhåndsdefinert kortstokk i stedet for å lage en ny tilfeldig, men det trengs et grensesnitt for å hente denne. Informasjonen om kontrakten og poengene i originalspillet må også legges inn. Dette bør sannsynligvis løses ved at man gjør det mulig å hente en spillfil fra et nettsted, som inneholder både kortstokken, og data om originalspillet. Applikasjonen vil da lese og benytte seg av denne. Til slutt må poengskjermen endres, slik at man kan se både egne resultater, og resultatene fra originalspillet.

19.2.4 Hotseat

Hotseat refererer til en måte å spille på, hvor man bare har en fysisk enhet, og man bytter på å benytte denne etter tur. Dette er en veldig interessant spilleform dersom man ikke har nok mobile enheter tilgjengelig. Med denne spilleformen vil da hver spiller ta sin tur, og så sende enheten videre til neste spiller.

Også dette vil kunne implementeres uten å gjøre større strukturelle endringer i måten spillet fungerer på. De viktigste tingene som må gjøres her er at skjermen skifter synsvinkel mellom hver spiller, slik at hver spiller ser riktig skjermbilde for sin posisjon. Det bør også legges inn en skjerm mellom hvert trekk som indikerer at det er neste spillers tur, slik at spilleren ikke ser neste spillers kort før vedkommende sender enheten videre.

19.2.5 TCP/IP kommunikasjon

I dag kommuniserer telefonene over Bluetooth. Dette er godt egnet når telefonene er i nærheten av hverandre, men ubrukelig på lengre avstander (over ca 10 m). Det ble valgt å bruke Bluetooth da denne applikasjonen ble utviklet, fordi de fleste mobiltelefoner har begrensninger som gjør at de ikke kan være en tjener i et IP-nettverk. Implementeringen av TCP/IP krever derfor at man setter opp en sentral tjener som all kommunikasjon går igjennom. Denne tjeneren vil da også fungere som en adressebok/møterom hvor man kan finne andre spillere. Siden alle trekk da vil måtte sendes igjennom tjeneren, vil tjeneren også kunne tilby andre tjenester, som for eksempel lagring av spillet og/eller resultatene. Siden klientene selv kontrollerer at spillet går korrekt, trenger ikke tjeneren vite noe om bridge eller reglene for bridge, den vil hovedsakelig bare være et kontaktpunkt som videreformidler pakkene klientene sender.

Applikasjonen inneholder allerede en nettverksprotokoll som er uavhengig av transmisjons-medium, så det vil ikke kreve noen endringer i de overliggende systemene i applikasjonen. Alt som behøves her er å implementere underliggende TCP/IP-kommunikasjon på samme måte som Bluetooth er implementert i dag, samt å implementere en initial oppkoblings-prosedyre for å koble seg til tjeneren og møte spillere før selve spillet startes. Denne vil være

vesentlig forskjellig fra den oppkoblingsprosedyren Bluetooth bruker for å oppdage tilgjengelige enheter og koble disse opp.

19.2.6 Oppkobling av Bluetooth-forbindelser

I dagens versjon av applikasjonen kobles automatisk opp de tre første klientene tjeneren finner, og spillet startes. Her bør det nok legges til mulighet for at brukeren selv kan velge hvilke klienter som skal delta, dersom tjeneren finner mer enn tre tilgjengelige klienter. En måte å håndtere dette på er å koble opp alle klientene den finner, og så koble disse ned igjen de klientene som ikke blir valgt til en av spillets fire posisjoner. Bluetooth har en begrensning på maks syv slaver i et piconett, så denne løsningen vil ikke være ideell dersom det er mange telefoner som venter på tilkobling i området. Gitt rekkevidden til Bluetooth, er dette et lite sannsynlig scenario. Det må presiseres at det uansett er bare enheter som har startet applikasjonen, og venter på invitasjon som vil dukke opp i dette søket. Enheter som allerede er i gang med et spill, og enheter som ikke har startet applikasjonen vil ikke bli funnet.

19.2.7 Blandede kommunikasjonsprotokoller

I noen tilfeller kan det også være interessant at enhetene snakker sammen via ulike typer nettverksbærere. For eksempel kan én telefon koble seg til tjeneren via TCP/IP mens den andre kobler seg opp via Bluetooth. Har man ikke fire telefoner tilgjengelig så kan man også se for seg at man kombinerer hotseat med Bluetooth, slik at man kan spille med to spillere på hver enhet. Den grunnleggende strukturen som applikasjonen benytter i dag for å kommunisere over nettverk støtter i prinsippet dette allerede i dag, men noen endringer må nok gjøres for å få dette til å fungere korrekt i praksis. Løsningen som ligger til grunn i dag baserer seg også på at alle enhetene kobler seg direkte opp til tjeneren, slik at denne må i så fall støtte alle protokollene som benyttes.

19.2.8 Avspilling av interessante spill

Et annen interessant mulighet er avspilling av interessante spill. I stedet for å selv spille, kan man se på en tidligere spilt kamp. Som med fasitspill krever dette at man legger til rette for å kunne laste inn et slikt spill fra for eksempel en Internett-side. Her kan man gi brukeren flere muligheter, som for eksempel om vedkommende ønsker å se spillet fra synsvinkelen til en av spillerne, eller for eksempel se kortene til alle spillene.

Implementeringen av en slik funksjon bør gjerne kombineres med muligheten for å lagre et spill man nettopp har spilt, slik at man da kan spille det av senere, eller gi det til andre.

Daniel Lundekvam [115] har skrevet en masteroppgave om dette temaet. Dersom dette vurderes implementert i denne applikasjonen er hans arbeid et svært godt utgangspunkt.

19.2.9 Tilkobling til Bridge Base Online (BBO)

BBO er en svært populær tjeneste for å kunne spille bridge over Internett. I følge hjemmesiden [4] har tjenesten over 100 000 medlemmer, med tusenvis av påloggede spillere fra hele verden til enhver tid. Ved å la den mobile klienten koble seg til denne tjenesten, vil man alltid kunne finne noen å spille mot. En slik mulighet vil gjøre applikasjonen mye mer interessant for brukerne.

Før en slik funksjonalitet kan implementeres er man avhengig av å innhente nødvendige tillatelser fra BBO. Selv om basistjenesten er gratis å benytte, er BBO et kommersielt selskap, som blant annet lever av reklame. Man kan derfor ikke bare koble seg opp til denne

tjenesten uten videre. BBO har heller ikke publisert noe materiale som beskriver nettverksfunksjonalitet og APIer. Uten et samarbeid med BBO vil ikke dette være praktisk gjennomførbart på en akseptabel måte.

19.2.10 Lagring av spill

Det vil også kunne være ønskelig at man kan lagre et spill man har spilt. Et slikt spill vil da kunne brukes enten som et fasitspill, eller spilles av igjen senere i henhold til avspillingsfunksjonen som beskrevet over. Spillet må da lagres med alle nødvendige detaljer, som alle avgitte meldinger, kontrakten, kortfordelingen, og hvilke kort hver spiller spilte i hvert stikk, samt den endelige poengsummen. Siden det finnes andre avspillere tilgjengelig vil det være en fordel å benytte et felles filformat, eller i alle fall ha en mulighet til å eksportere til andre filformater. Spesielt med tanke på at en mobil enhet ofte har begrenset lagringsplass, bør man kanskje gå for sistnevnte løsning, slik at man kan lagre dette i en mest mulig kompakt fil lokalt på den mobile enheten. Formater med mye strukturdata i selve filen, slik som XML vil muligens være et dårlig format til å lagre data lokalt på enheten, men kan være et glimrende format å tilby eksportering til, siden det er lett for andre programmer å tilrettelegge for å importere fra XML.

Videre bør den mobile enheten ha en mulighet til å gjøre filen tilgjengelig for andre, for eksempel ved å kunne laste den opp på en Internett-side. Dette kan nok med fordel kombineres med eksportering til et annet filformat for å gjøre prosessen enklere for brukeren.

19.2.11 Intelligente AI-spillere

Som beskrevet i rapporten ble det i løpet av utviklingen utviklet en meget enkel AI-spiller. Denne kan lett utvikles til å bli en mye mer avansert spiller, som kan melde og spille basert på hvilke kort denne har, i stedet for å velge dette helt tilfeldig. Det er selvsagt et langt steg fra å lage en mer avansert AI-spiller til å lage en AI-spiller som er god til å spille bridge, men dersom dette kunne gjøres ville det kunne tilføre en ekstra dimensjon til applikasjonen. Man kan da for eksempel spille bridge alene ved å la AI-spillere ta opp de andre tre posisjonene, eller man kan trene sammen med makkeren sin og la motstanderne være AI-styrte. En veldig interessant mulighet her ville for eksempel være å implementere en selvlærende AI-spiller basert på kunstige nevrale nett. En slik spiller vil bli bedre og bedre jo mer man spiller mot den, dersom dette blir implementert korrekt.

Oppsummering og konklusjon

20 UP versus XP

I dette prosjektet ble det brukt to forskjellige utviklingsmetoder; Unified Process (UP) og eXtreme Programming (XP). Hver av disse metodene er beskrevet nærmere i kapittelet som omhandler det aktuelle delprosjektet.

Ved å bevisst bruke to forskjellige utviklingsmetoder fikk utviklerne et bedre perspektiv på forskjellene og likhetene mellom disse to metodene.

På overflaten er de veldig forskjellige. Mens XP bygger på god kodeskikk og åpen kommunikasjon innad i utviklerteamet, fokuserer UP på konkret dokumentasjon etter gitte maler. Med andre ord, to veldig forskjellige måter å oppnå samme målet på, nemlig at utviklerne har klart for seg hva som skal implementeres, og at det er lett i ettertid å gå inn og endre/vedlikeholde applikasjonen. Etter et fullført UP-prosjekt sitter man igjen med et stort antall dokumenter som beskriver funksjonaliteten og samspillet mellom både de overordnede systemer, så vel som enkeltkomponenter. Med XP sitter man igjen med en stor mengde enhetstester, så vel som klar og konsis kode gitt at prinsippene i XP er blitt korrekt fulgt.

Ser man litt dypere ned i begge metodene kan man også finne flere likhetstrekk. Både XP og UP har brukstilfeller for å beskrive hva applikasjonen skal gjøre, selv om måten disse formuleres på, og de formelle kravene til oppsett er ulike. Begge metodene utvikler også applikasjonen i iterasjoner, og forut for hver iterasjon blir det valgt ut ett sett med funksjonalitet som skal implementeres i den kommende iterasjonen.

20.1 Kan XP og UP sammenlignes?

Ser man grundig på XP og UP, legger man merke til en ting; XP og UP legger seg på to forskjellige nivåer. XP er veldig fokusert på detaljstyringen av selve utviklingen, og legger stor vekt på prinsipper som parprogrammering, testdrevet utvikling og refaktorering. Ser man derimot på UP, så ser man at denne ikke går ned på et slikt detaljnivå, men er mye mer opptatt av de overordnede delene av prosessen. UP legger ikke noen føringer for hvordan selve utviklingen skal gjennomføres når det gjelder å skrive kode, og prosesser fra XP som parprogrammering og testdrevet utvikling kan derfor benyttes under UP uten å bryte med UPs metodikk. På den annen side har man XP som forteller at UML-diagrammer og skrevne spesifikasjoner ikke er en påkrevd del av XP, men at det kan brukes dersom det er behov for dette, mens UP har definert et stort antall (~100) artefakter som kan benyttes i prosessen. Det viktige ordet her er *kan*. Et stort antall av artefaktene i UP er valgfrie, og kan benyttes avhengig av hva som er fornuftig for prosjektet. Man ser her at ved korrekt bruk og utelatelse av valgfrie komponenter kommer man ganske nært til å kunne benytte både XP og UP samtidig på et prosjekt, uten at prinsippene kommer i konflikt med hverandre.

Hvilken konklusjon kan man så trekke av dette? Det er rimelig klart at bruksområdet for disse er forskjellig, når man ser hvordan disse kan nærmest sameksistere. Den viktigste faktoren er nok størrelsen på prosjektet. Studier, blant annet en rapport fra IBM [116], har vist at XP er helt klart best egnet på mindre prosjekter. IBM sier her ca 10 prosjektdeltagere, men andre har vist at XP kan fungere med helt opp til 50 prosjektdeltagere. Men uansett hvor man setter grensen ser man tydelig at større prosjekter krever en grundigere dokumentert og ordnet prosess for å sikre at den overordnede arkitekturen og samspillet mellom komponentene fungerer som tiltenkt. Som diskutert over, ser man at selv om et stort prosjekt trenger en paraply som UP for å håndtere prosjektet, hindrer ikke dette prosjektet å bruke mange gode

XP-prinsipper som parprogrammering og testdrevet utvikling, dersom dette er ønskelig. En objektiv gjennomgang av UP viser at dette er et rammeverk og ikke en tvangstrøye.

20.2 Utviklernes erfaringer

Etter å ha benyttet både UP og XP, og forsøkt å følge praksisene i størst mulig grad sitter utviklerne igjen med flere erfaringer. Det er viktig å forstå at begge delprosjektene teknisk sett er svært små prosjekter, og at dette nok har farget opplevelsen noe. Konklusjonene må derfor leses med dette i tankene, og ikke aksepteres som generelle sannheter for større prosjekter.

Første delprosjekt ble utviklet under UP. Den første delen av dette prosjektet gikk meget greit, tempoet var fornuftig, og artefakter ble produsert som spesifisert. Med bare to utviklere på prosjektet føltes dette litt kunstig, siden artefaktene i prinsippet ikke skulle brukes av andre enn de samme to personene som skrev dem. Det må allikevel sies at disse hadde stor praktisk nytte. Selv om artefaktene ikke ble referert til i ettertid, bidro prosessen med å produsere disse til at utviklerne sikret en felles forståelse av arkitektur og oppførsel til sluttproduktet. Disse artefaktene har også en stor verdi for utviklere som skal vedlikeholde applikasjonen i ettertid. Problemene oppstod først da prosjektet ble forsinket. Dette er noe som ikke er uvanlig i programvareverdenen, men i dette tilfellet måtte produktet i drift til en spesifikk dato uansett status, eller vente et ekstra år før produktet kunne taes i bruk. Sistnevnte var en meget dårlig løsning, og etter samtale med kunden ble det bestemt at systemet skulle gå live. Systemet var da klart til å håndtere initial bruk, men det var også klart at systemet måtte få implementert mer funksjonalitet i løpet av svært kort tid, og datoene hver av funksjonalitetene måtte være implementert til var også låst.

På dette stadiet klarte ikke utviklerne lenger å følge UP sine prosedyrer. Grunnet manglende tid ble det prioritert å utvikle funksjonalitet fremfor å produsere artefaktene for iterasjonen. Det medførte at det ble skrevet en del kode i denne perioden uten at elementer som for eksempel brukstilfeller var dokumentert tilstrekkelig på forhånd. Følelsen utviklerne satt igjen med, er at det var vanskelig å følge UP til punkt og prikke når tiden ble knapp. Det er selvsagt viktig å presisere at siden det var uviklerne som også produserte artefaktene ble dette rett og slett et prioriteringsspørsmål. I et større prosjekt hvor egne teammedlemmer tar seg av artefaktene, vil ikke tidsnød påvirke prosjektet i samme grad.

For andre delprosjekt ble XP fulgt. Utviklerne fokuserte også her på å følge XP så nøye som mulig for å få en realistisk erfaring. En veldig interessant oppdagelse er at noen av prinsippene i XP, slik som testdrevet utvikling og refaktorering kan føles unaturlig og oppleves som en plikt og ikke en naturlig del av kodingen. Dette er en problemstilling som også rapporten fra IBM refererer til [116]. Fristelsen for å hoppe over skriving av tester er stor når man føler tidspress og stress. Utviklingen under XP føltes veldig naturlig og jevn, uten kunstige avbrudd for å skrive dokumentasjon. Kun en liten del av tiden gikk med til å skrive brukstilfeller, og tilordne disse poeng. Resten av prosessene gikk inn som en naturlig del av selve kodeskrivingen.

Konklusjonen er nok ikke spesielt overraskende. Etter å ha fullført begge prosjektene var utviklingsteamet enig om at for et prosjekt av denne størrelsen, så har man mer igjen for å bruke XP enn UP. XP er mer fleksibelt, og mer fokusert på selve utviklingen, mens UP oppfattes som en tanke mer rigid, og stiller en del krav som krever en del tid å få orden på. Men utviklingsteamet ser også at XP ikke er spesielt velegnet til større prosjekter, da XP ikke har gode nok mekanismer for å få en god helhetlig applikasjonsstruktur og et konsistent samspill

mellom komponenter. Man kan derfor konkludere med at ingen av de to er bedre enn den andre, men at valget av utviklingsmetode bør falle på den metoden som er best egnet for det aktuelle prosjektet.

Man bør heller ikke velge en utviklingsmetode spesifikt for hva denne tilbyr, men også se hvilke tilpasningsmuligheter som er mulig uten å bryte med metoden. Er man for eksempel tilhenger av parprogrammering binder ikke dette en til XP, da dette fint kan utføres under UP også. Til slutt må det også nevnes at det finnes tilpasninger både til XP og UP som gjør disse bedre egnet til andre typer prosjekter. Man trenger ikke følge en spikret mal, men kan heller tilpasse utviklingsmetoden til behovene for prosjektet. For eksempel finnes det tilpasninger til UP som gjør denne mer fleksibel og bedre tilpasset små prosjekter. [117]

20.3 Evaluering av valg av metodikk

Til dette prosjektet ble det valgt to forskjellige utviklingsmetoder, en for hvert delprosjekt. Dette ble gjort for å få erfaring med begge disse utviklingsmetodene på et litt større prosjekt, slik som en masteroppgave er. XP ble valgt som utviklingsmetode til mobilapplikasjonen, fordi at ved dette delprosjektet var kunden tilgjengelig i umiddelbar nærhet, noe som er svært viktig for XP. For administrasjonssystemet var kunden i Oslo, noe som ville ha vært en stor hindring dersom dette systemet skulle ha blitt utviklet under XP. Det ble derfor bestemt at første del av prosjektet, turneringssystemet, skulle utvikles under UP. Det kan diskuteres i ettertid om dette var det beste valget, spesielt med tanke på de tidsmessige problemene utviklerne opplevde, som beskrevet over. Men tatt i betraktning de originale forutsetningene som stod bak disse valgene, vil konklusjonen her være at de valgene som ble gjort var de riktige for dette prosjektet.

21 Arbeid, planlegging og fremdrift

Da denne oppgaven ble påbegynt kunne veilederen tilby flere konkrete oppgaver. Tanken var i utgangspunktet at hver av studentene skulle velge hver sin oppgave. Etter å ha diskutert disse oppgavene bestemte studentene seg for å gå for en annen løsning. I stedet for hver sin oppgave foreslo studentene for veileder å kombinere de enkelte oppgavene til en større fellesoppgave tilpasset to studenter. Litt av tanken bak dette var blant annet at utviklings-metodikker som for eksempel XP kunne brukes mer gjennomført, og også for å få en litt mer reell arbeidssituasjon der man hadde andre å forholde seg til.

Gjennom hele prosjektet ser studentene at dette har vært svært vellykket og brakt inn en god del erfaringer.

For det første er all programvaren utviklet som et samarbeidsprosjekt. Dette betydde ikke at hver av studentene bare kunne programmere halvparten hver, men det var veldig viktig med god planlegging og kommunikasjon. Studentene måtte være enige om standardene og teknologien som skulle benyttes, og utviklingsverktøyene måtte være kompatible. I mange tilfeller hadde studentene forskjellige oppfatninger av hvordan ting skulle gjøres, noe som førte til at begge måtte ta en grundigere analyse av alternativene for å så komme frem til en felles løsning som de kunne være enige om. Dette medførte også at løsningen som ble utviklet ble mer gjennomtenkt, og dro nytte av erfaringene til to personer.

En annen fordel av å være mer enn én person som studentene opplevde i løpet av denne oppgaven, var at det ble mye lettere å arbeide i et fast tempo. Ved å alltid arbeide sammen, som i et reelt arbeidsmiljø, følte studentene et visst press til å møtes for å jobbe når det var planlagt. Dette førte til at det ble satt opp en plan for når det skulle arbeides med oppgaven, og denne ble da også hovedsakelig overholdt med unntak av uforutsette hendelser.

Når det gjaldt selve fremdriften på utviklingen, så gikk ikke denne helt etter opprinnelig plan. Mange av oppgavene tok en god del lenger tid enn opprinnelig estimert. Hovedgrunnen til dette ligger nok i vanskeligheten med å estimere tidsforbruk riktig. Korrekt estimering krever ofte en god del erfaring. Siden ingen av studentene hadde noe tidligere forholdt til bridge, manglet de ofte også en del domenekunnskap, noe som medførte at det også kunne være vanskelig å forstå nøyaktig hva som kreves av en deloppgave på estimeringstidspunktet. Til tross for forsinkelsene kom begge delprosjektene i mål, men med en noe redusert funksjonalitet på enkelte områder i forhold til hva som var opprinnelig tiltenkt.

22 Valg av applikasjoner

I løpet av denne rapporten er en del verktøy og applikasjoner som ble brukt under prosjektet beskrevet. Disse applikasjonene ble valgt ut i fra de forhold som lå til grunn ved starten til hvert av de to delprosjektene. Stort sett var utviklerne godt fornøyd med de valgene som ble gjort, men i noen tilfeller var nok utviklerne litt optimistiske i forhold til det reelle behovet for programvare. Spesielt er nok dette synlig i den overordnede prosjektstyringsprogramvaren, og i en mye mindre grad i utviklingsverktøyene.

Selv om all programvaren som er beskrevet i rapporten faktisk ble brukt til prosjektet, viser det seg at noe av denne kunne nok prosjektet ha klart seg uten. Det beste eksempelet her er Mantis, som ble valgt som feilhåndteringssystemet i prosjektet. Dette er et meget bra verktøy, men på et så lite prosjekt som dette tross alt var, og med bare to utviklere, så ble de fleste feil håndtert på direkten når de ble oppdaget. Dette opplevdes som mye mer effektivt enn å først rapportere inn en feil i et system, og så rette feilen selv etterpå.

WordPress ble nok også mindre brukt enn tiltenkt. En del av grunnen til dette var nok at i løpet av prosjektet ble MediaWiki innført, og overtok en del av den rollen som originalt var tiltenkt WordPress. I dette tilfellet hadde det nok vært det beste å bare brukt en av disse løsningene. Det må selvsagt sies at begge disse løsningene er raske og enkle å sette opp og vedlikeholde så det har ikke vært noen negativ påvirkning på prosjektet å bruke begge disse applikasjonene, men det var strengt tatt unødvendig.

Utviklerne vil her konkludere med at de var meget fornøyd med alle applikasjonene som ble valgt, og grunnen til at enkelte applikasjoner ble brukt mindre enn andre skyldes hovedsakelig at behovet ved starten av prosjektet ble noe optimistisk anslått.

23 Konklusjon

Arbeidet med denne oppgaven har nå foregått i ca 18 måneder. I løpet av den tiden har vi planlagt design, lett etter løsninger, implementert og testet gjentatte ganger. Vi har gjort oss mange erfaringer under denne prosessen, både positive og negative, og det er ikke til å unngå at vi også har samlet oss en del ny lærdom. To av de viktigste erfaringene våre står listet under. Disse kan kanskje virke selvsagte, og til en viss grad er de også det. Problemet er at til tider har man en tendens til å overse også det selvsagte. Dette er ting vi egentlig visste fra før, og som ikke burde ha kommet som noen overraskelse, men områder vi allikevel feilet på.

Ikke overtenk problemstillingen eller klargjør for absolutt alle eventualiteter, selv de usannsynlige. Da vi skulle utvikle BTS gjorde vi nettopp denne feilen. En av de første tingene som ble utviklet var domenemodell og databaseskjema. Etter å ha lagt hjernene i bløt, kom vi opp med et databaseskjema som var korrekt normalisert, og designet etter god databaseskikk, men det var også alt for komplisert i forhold til behovet. Dette førte igjen til at de nødvendige spørringene for å hente og oppdatere data ble unødvendig kompliserte.

Ikke anta at alt kjører nyeste versjon. Denne feilen klarte vi å gjøre flere ganger. Under utviklingen av BTS antok vi at siden NBF kjørte MySQL og PHP, så kjørte de også produksjonsversjonene av disse (versjon 5.x for begge systemene). Antagelsen vår var her feil, tjeneren deres kjørte fremdeles 4.0-versjonene. Da vi kom til mobilprogrammeringen antok vi at nyere versjoner av J2ME var basert på nyere versjoner av J2SE. Også her tok vi rimelig feil, siden nyeste J2SE versjonen er 1.6, mens J2ME er basert på versjon 1.3 fra 2000 [118].

Det ovenstående hadde ikke kritisk innvirkning på prosjektet annet enn å koste litt tid som burde ha vært unødvendig, men det er de elementene som står sterkest i hukommelsen vår angående ting som er helt unødvendige å gjøre igjen.

Alt i alt gikk arbeidet med prosjektet veldig greit, men i noen tilfeller har det nok vært litt vanskelig å estimere forventet tidsforbruk korrekt. Vi står nå på slutten av tiden med to utviklede applikasjoner, begge gode applikasjoner som gjør det de skal, men vi ser at skulle vi ha implementert alle de funksjonene vi selv hadde lyst til, og som andre har foreslått, kunne vi ha brukt hele tiden, og vel så det, på å utvikle kun en av applikasjonene.

Som beskrevet i kapitlene om videre arbeid i hver av de to delrapportene er det nok å ta tak i. Vi synes allikevel at denne todelingen var svært hensiktsmessig for oss, siden vi da fikk muligheten til å prøve to forskjellige måter å tilnærme oss problemstillingene på, nemlig ved bruk av henholdsvis UP og XP som utviklingsmetoder.

Vi synes også det er veldig artig å se arbeidet vårt bli videreført. Videreutvikling av turneringssystemet BTS er nå tildelt som bachelor-oppgave til en gruppe studenter. Disse er nå i full gang med å utvikle ny funksjonalitet til systemet. Vi håper og tror at vi har utviklet et system som er lett for disse å bygge videre på.

Appendiks

Appendiks A: Begrepsliste

Begrep	Forklaring
Annotation	Metadata som benyttes i forbindelse med programmering. I Java er disse en slags kommentarer som letter behandlingen for verktøy og bibliotek, i motsetning til kommentarer som er beregnet for menneskelige øyne.
Blindemann	Under spillet vil en av spillerne i makkerparet som har kontrakten være blindemann. Det er makkeren til denne spilleren, spilleføreren, som bestemmer hvilke kort blindemann skal spille.
BTS	Bridge Turneringssystem. Systemet som ble utviklet i forbindelse med første del av oppgaven.
Typecasting	Prosessen med å konvertere en type objekt til et annet. Dette må blant annet brukes når man bruker lister i Java som ikke er opprettet med generics, siden slike lister ”konverterer” alt man putter inn til grunntypen Object, og man må da konvertere det tilbake til riktig type når man henter det ut igjen for å kunne bruke objektet slik det var tiltenkt.
CLDC	Connected Limited Device Configuration. Grunnkonfigurasjonen i J2ME for mobile enheter. Inneholder grunnfunksjonalitet for Java ME.
Foreach	Løkkestruktur i Java som itererer gjennom hvert element i en liste.
Generics	Generics er en ekstra konstruksjon som kompilatoren bruker for å sjekke at man bruker riktig type elementer i en liste. Sparer programmereren for å bruke typecasting når elementer hentes ut fra en slik liste. For eksempel kan en standard Vector i Java inneholde alle typer objekter, mens en Vector som er deklarert med generics kan kun inneholde den typen man deklarerer den for, slik at det aldri er usikkerhet om hva type objekter den inneholder.
Giver	Spilleren som deler ut kortene denne runden. Denne spilleren åpner også budrunden.
GPRS	General Packet Radio Service. System for å koble seg til Internett via en mobiltelefon. Leveres via mobiltelefonileverandøren, og blir vanligvis belastet for mengde data overført.
J2EE	Java 2 Enterprise Edition. Den tyngste utgaven av Java. Brukes på servere som tilbyr tjenester på Java-plattformen.
J2ME	Java 2 Micro Edition. Lettvektsutgaven av Java som er beregnet å kjøre på mobile enheter, og andre enheter med lite systemressurser.
J2SE	Java 2 Standard Edition. Den "vanlige" utgaven av Java. Brukes på vanlige PC-er.
Java	Programmeringsspråk. Ble brukt i andre del av oppgaven.

Begrep	Forklaring
JSR	Java Spesification Request. Spesifikasjonsbeskrivelse for Java. For eksempel er JSR-82 spesifikasjonen for Bluetooth under Java.
JVM	Java Virtual Machine. Virtuell maskin for å kjøre Javakode.
Kontrakt	Den høyeste meldingen i meldedelen blir kontrakten i spillet. Makkerparet som fikk kontrakten vil prøve å klare denne, mens motstanderparet vil prøve å hindre dem. Man klarer en kontrakt ved å ta minst like mange stikk som kontrakten tilsier.
KVM	Kilobyte Virtual Machine. Virtuell maskin for å kjøre J2SE-kode, optimalisert for å kreve lite ressurser, være portabel og modulær.
Makkerpar	De to spillerne som spiller sammen. Makkerne sitter rett ovenfor hverandre på bridgebordet. Nord/sør er alltid ett makkerpar, mens øst/vest utgjør det andre.
MIDP	Mobile Information Device Profile. Dette er en J2ME-profil som inneholder vanlig grunnfunksjonalitet for mobiltelefoner.
PHP	Programmeringsspråk. Ble brukt til første del av oppgaven.
Spillefører	Under spillet vil en av spillerne i makkerparet som har kontrakten være spillefører. Denne spilleren har ansvaret for å spille både sine egne og blinde-mannens kort.
UP	Unified Process.
Vector	Vector er en klasse i Java som kan inneholde en liste med objekter. En Vector kan for eksempel inneholde en liste med kort som en spiller har på hånden.
Volt	Prosjektnavnet for andre del av oppgaven; Bridgespill for mobil.
WLAN	Wireless Local Area Network. Trådløst nettverk.
XP	eXtreme Programming.

Appendiks B: The Open Source Definition

Hentet fra [14].

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

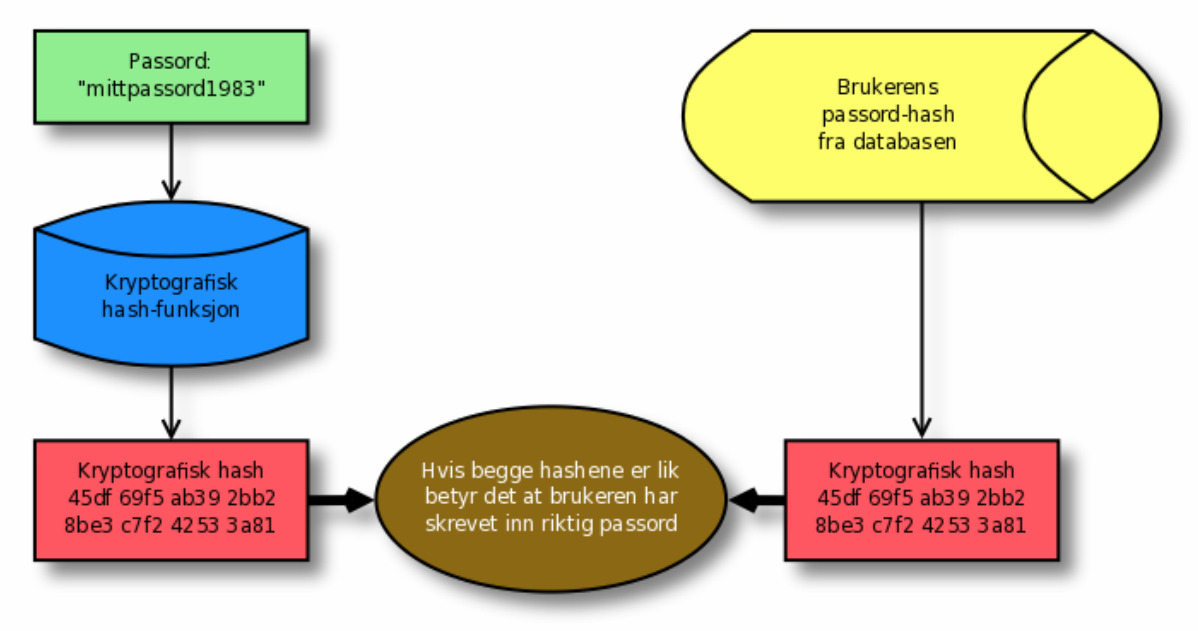
Appendiks C: Kryptografisk hash

En kryptografisk hash-funksjon tar en input som blir transformert til en hash-verdi av en fast lengde. Denne hash-verdien fungerer som et digitalt fingeravtrykk. En liten forandring i inputen vil medføre en stor forandring i hash-verdien. Samme input gir alltid samme output med den samme hash-funksjonen. Da antall tilgjengelige hash-verdier er lavere enn antall unike input vil det inntreffe til tider ”kollisjoner”, hvor to ulike inputer medfører samme hash-verdi. De kryptografiske hash-funksjonene er utviklet for å skille mellom ulike input og sørger for at det skal være veldig store forandringer i en input for at de skal gi identisk hash-verdi.

De mest brukte hash-funksjonene inkluderer MD5 og SHA-familien. Det er funnet en rekke feil i både MD5 og SHA1 som gjør at de i enkelte tilfeller har svakheter. Til bruk i BTS holder begge hash-funksjonene mål siden det er snakk om relativt små inndata. Passordene som brukerne velger må være minst fire tegn og siden de blir memorert av mennesker er det tvilsomt av passordene blir lengre enn 20 tegn.

Input	MD5 hash verdi (hex)	SHA1 hash verdi (hex)
Det var en gang...	d790 3640 5985 f1e2 4aa7 4d0d b4b7 fa55	149c e6bc 5368 d8f6 3a2b 7300 1618 eab5 d44b 5147
det var en gang...	4d28 1242 1bfd 6f4a d0d4 e81f 5d00 98a0	a0ab b38b b593 441e dc18 4d39 3332 9c02 267e 731b
mitthemmeligepassord123	1a10 e4ab bd9c 7826 d57c 56f3 9bb6 8a03	3d1f ae3e 7629 4f97 b9b5 b0f4 9a94 88a3 543e 60fc
mitthemmeligepassord1234	437b a5ad 88dc 622b b48b 3945 fbfb e173	5eff 37ef 56b2 6d70 a9ca 8f76 15e5 534d bce4 78a5
mitthemmeligepassordABC	cc59 feb7 87ce 6b97 e604 b8b8 9fa6 7c32	babd 06f9 9250 cd25 7415 30e6 f6b1 ae93 3f38 e308

Hash-funksjon	Lengde på hash-verdi (i bit)	Antall unike hash-verdier
MD5	128	$3{,}40 * 10^{38}$
SHA1	160	$1{,}46 * 10^{48}$
SHA-256	256	$1{,}16 * 10^{77}$
SHA-512	512	$1{,}34 * 10^{154}$



Figur A - 1: Bruk av kryptografisk hash ved autentisering

Appendiks D: Innholdsfortegnelse filarkiv

Vedlagt denne rapporten finnes et filarkiv som inneholder kildekode, lisenser, utviklingsdokumenter med mer. Dette arkivet er lagt med som et vedlegg i selve PDF-filen i den elektroniske utgaven av denne rapporten. I Adobe Acrobat kan disse hentes ut fra filen ved å vise vedleggspanelet. (View → Navigation Panels → Attachments).

På grunn av at Adobe har lagt inn enkelte sikkerhetsbegrensninger i produktene, var det ikke mulig å legge med en zip-fil direkte uten at brukeren måtte utføre en meget tungvint prosedyre for å hente denne ut igjen[1]. Filen har derfor fått navnet *vedlegg.zip.slett_meg*. For å bruke denne filen er det bare å hente denne ut fra PDF-filen som beskrevet over, og så endre filnavnet til *vedlegg.zip*, så kan filen brukes som en vanlig zip-fil. Utviklerne beklager den litt tungvinte prosedyren, men dette var eneste måten å legge en zip-fil som et vedlegg til et PDF-dokument uten å måtte gi brukeren en enda mer tungvindt forklaring.

Dersom man kun sitter med en papirkopi av denne rapporten vil vedleggene ha blitt levert med som en CD eller tilsvarende. Hvis ikke kan dette vedlegget hentes ut av enhver som har tilgang på den elektroniske utgaven av rapporten, i henhold til prosedyren over.

Innholdet i dette arkivet er som følger:

- **Kildekode**: Kildekoden til begge delprosjekter. Volt Bridge er delt opp i flere underprosjekt om har avhengigheter til hverandre under kjøring.
- **Lisenser**: Inneholder fullstendig versjon av kildekodelisenene omtalt i rapporten. Både i PDF- og tekstformat.
- **Utviklingsmetodikk**
  - **UP-dokumentasjon**: Dokumentasjon brukt i forbindelse med UP-prosjektet BTS.
  - **XP-dokumentasjon**: Dokumentasjon brukt i forbindelse med XP-prosjektet for utvikling av mobilapplikasjonen.

[1] Problemet oppstår med Adobe Acrobat/Adobe Reader under Windows.

Appendiks E: Referanseliste

Alle Internettsider ble sjekket kort tid før innlevering for å verifisere at disse fremdeles var gyldige og hadde riktig innhold. De aller fleste sidene har derfor fått datoen 2008-05-29 som den datoen sidene ble hentet.

[1] Norsk Bridgeforbund (hjemmeside): http://www.bridge.no

[2] Olsen, Arne (2004), *Bridge - Et spill for livet*, 3. utgave. Norsk Bridgeforbund, Oslo

[3] Lag-VM i Shanghai - norsk GULL [Internett]. Norsk Bridgeforbund; 2007-06-10 [hentet 2008-05-29] Tilgjengelig fra: http://www.bridgefederation.no/t2.asp?p=5703&x=1&a=206393

[4] BridgeBase online (hjemmeside): http://www.bridgebase.com

[5] OKbridge (hjemmeside): http://www.okbridge.com

[6] Topbridge (hjemmeside): http://www.topbridge.com

[7] Mantis (hjemmeside): http://www.mantisbugtracker.com

[8] Wordpress (hjemmeside): http://wordpress.org

[9] dotProject (hjemmeside): http://www.dotproject.net

[10] Mediawiki (hjemmeside): http://www.mediawiki.org

[11] Wikipedia (hjemmeside): http://www.wikipedia.org

[12] Subversion (hjemmeside): http://subversion.tigris.org

[13] Åndsverkloven. 1961. Lov om opphavsrett til åndsverk m.v. av 1961-05-12 nr 02

[14] The Open Source Definition [Internett]. Open Source Initiative, 2006-07-07 [hentet 2008-05-29] Tilgjengelig fra: http://opensource.org/docs/osd

[15] Mozilla Firefox (produktside): http://www.mozilla.com/en-US/firefox/

[16] OpenOffice.org (hjemmeside): http://www.openoffice.org

[17] GNU (hjemmeside): http://www.gnu.org

[18] Microsoft Windows (produktside): http://www.microsoft.com/windows/

[19] Adobe Photoshop (produktside): http://www.adobe.com/products/photoshop/index.html

[20] Apple iTunes (produktside): http://www.apple.com/itunes/

[21] Sun Microsystems (hjemmeside): http://www.sun.com

[22] Sun Java (hjemmeside): http://java.com

[23] The BSD Licence [Internett]. Open Source Initiative; 2006-10-31 [hentet 2008-05-29] Tilgjengelig fra: http://www.opensource.org/licenses/bsd-license.php

[24] The MIT Licence [Internett]. Open Source Initiative; 2006-10-31 [hentet 2008-05-29] Tilgjengelig fra: http://www.opensource.org/licenses/mit-license.php

[25] GNU General Public Licence [Internett].. GNU Project; 2008-02-12 [hentet 2008-05-29] Tilgjengelig fra: http://www.gnu.org/licenses/gpl.html

[26] Trolltech Qt (produktside): http://trolltech.com/products/qt/

[27] GNU Lesser General Public Licence [Internett]. Open Source Initiative; 2006-10-31 [hentet 2008-05-29] Tilgjengelig fra: http://www.opensource.org/licenses/lgpl-2.1.php

[28] OpenOffice.org Licence [Internett]. OpenOffice.org; 2008-03 [hentet 2008-05-29] http://www.openoffice.org/license.html

[29] IBM Rational Unified Process [Internett]. Wikipedia; 2008-05-15 [hentet 2008-05-29] http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process

[30] Lunn, Ken (2003), *Software Development with UML*. Palgrave Macmillan, Hampshire, England

[31] Høgskolen i Bergen (hjemmeside): http://www.hib.no

[32] Iterative Development [figur] [Internett]. Wikipedia; 2007-10-16 [hentet 2008-05-30]
Tilgjengelig fra: http://en.wikipedia.org/wiki/Image:Development-iterative.gif

[33] MVC pattern [Internett]. TechRepublic; 2002-10-30 [hentet 2008-05-29]
Tilgjengelig fra: http://articles.techrepublic.com.com/5100-10878_11-1049862.html

[34] HTML (produktside): http://www.w3.org/html/

[35] Sun Java EE (produktside): http://java.sun.com/javaee/

[36] PHP (hjemmeside): http://www.php.net

[37] MySQL (hjemmeside): http://www.mysql.com

[38] XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition) [Internett]. World Wide Web Consortium; 2002-08-01 [hentet 2008-05-29]
Tilgjengelig fra: http://www.w3.org/TR/xhtml1/

[39] Extensible Markup Language (XML) 1.1 (Second Edition) [Internet]. World Wide Web Consortium; 2006-10-29 [hentet 2008-07-01]
Tilgjengelig fra: http://www.w3.org/TR/2006/REC-xml11-20060816/

[40] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification [Internet] World Wid Web Consortium; 2007-07-19 [hentet 2008-05-29]
Tilgjengelig fra: http://www.w3.org/TR/CSS21/

[41] Apache HTTP Server (produktside): http://httpd.apache.org/

[42] PHP Usage Stats [Internett]. PHP.net; 2007-04 [hentet 2008-05-29]
Tilgjengelig fra: http://www.php.net/usage.php

[43] April 2008 Web Server Survey [Internett]. Netcraft; 2008-04-14 [hentet 2008-05-29]
Tilgjengelig fra: http://news.netcraft.com/archives/2008/04/14/april_2008_web_server_survey.html

[44] Oversikt over BSD versjoner: http://bsd.org

[45] Novell (hjemmeside): http://www.novell.com

[46] Mac OS (hjemmeside): http://www.apple.com/macosx/

[47] Perl (hjemmeside): http://www.perl.org

[48] Python (hjemmeside): http://python.org

[49] CakePHP (hjemmeside): http://www.cakephp.org

[50] Eclipse (hjemmeside): http://www.eclipse.org

[51] Subclipse (hjemmeside): http://subclipse.tigris.org

[52] PHP Development Tools (produktside): http://www.eclipse.org/pdt/

[53] Aqua Datastudio (hjemmeside): http://www.aquafold.com

[54] Microsoft Access (produktside):
http://office.microsoft.com/en-us/access/FX100487571033.aspx

[55] AWStats (hjemmeside): http://awstats.sourceforge.net

[56] Prototype (hjemmeside): http://prototypejs.org

[57] script.aculo.us (hjemmeside): http://script.aculo.us

[58] Microsoft Excel (produktside):
http://office.microsoft.com/en-us/excel/FX100487621033.aspx

[59] Generate Excel spreadsheets from your database [Internett]. the Bakery; 2007-05-05 [hentet 2008-05-29]
Tilgjengelig fra: http://bakery.cakephp.org/articles/view/generate-excel-spreadsheets-from-your-database

[60] Ajax [Internett] Wikipedia; 2008-05-30 [hentet 2008-06-01]
Tilgjengelig fra: http://en.wikipedia.org/wiki/AJAX

[61] ODBC overview [Internett]. Mirosoft; 2007-03-29 [hentet 2008-05-29]
Tilgjengelig fra: http://support.microsoft.com/kb/110093

[62] MD5 [Internett]. Wikipedia; 29.05.2008 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/MD5

[63] Captcha [Internett]. Wikipedia; 29.05.2008 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/Captcha

[64] bzip2 [Internett]. Wikipedia; 17.05.2008 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/Bzip2

[65] gzip [Internett]. Wikipedia; 11.05.2008 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/Gzip

[66] zip [Internett]. Wikipedia; 26.05.2008 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/ZIP_%28file_format%29

[67] HTTP [Internett]. Wikipedia; 28.05.2008 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/HTTP

[68] SQL injection [Internett]. Wikipedia; 27.05.2008 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/SQL_injection

[69] Cross Site Scripting [Internett]. Wikipedia; 29.05.2008 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/Cross-site_scripting

[70] Character Entity References in HTML 4 [Internett]. World Wide Web Consortium; 1999-12-24 [hentet 2008-05-29]
Tilgjengelig fra: http://www.w3.org/TR/REC-html40/sgml/entities.html

[71] CAPTCHA (hjemmeside): http://www.captcha.net

[72] Gmail (hjemmeside): http://www.gmail.com

[73] Hotmail (hjemmeside): http://www.hotmail.com

[74] S. Bosworth, M.E. Kabay (1992). Risk Assessment and Risk Management. *Computer Security Handbook*. John Wiley & Sons, Canada

[75] NBF Data (hjemmeside): http://nbfdata.bridge.no

[76] Response Times: The Three Important Limits. [Internett]. Jakob Nielsen; 1993-2007 [hentet 2008-05-28]
Tilgjengelig fra: http://www.useit.com/papers/responsetime.html

[77] The Need for Speed [Internett]. Jacob Nielsen; 1997-03-01 [hentet 2008-05-29]
Tilgjengelig fra: http://www.useit.com/alertbox/9703a.html

[78] Microsoft Windows Internet Explorer (produktside):
http://www.microsoft.com/windows/products/winfamily/ie/default.mspx

[79] Box Modell Hack: Getting Internet Explorer to Play Well with CSS [Internett]. About.com; 2008 [hentet 2008-05-29]
Tilgjengelig fra: http://webdesign.about.com/od/css/a/aaboxmodelhack.htm

[80] Internet Explorer box modell bug [Internett]. Wikipedia; 2008-04-01 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/Internet_Explorer_box_model_bug

[81] Internet Explorer vs. the Standards [Internett]. Bergevin, H. and John; 2005-10-26 [hentet 2008-05-29]
Tilgjengelig fra: http://www.positioniseverything.net/ie-primer.html

[82] Explorer Exposed! [Internett]. Bergevin, H. and John; 2006-12-09 [hentet 2008-05-29]
Tilgjengelig fra: http://www.positioniseverything.net/explorer.html

[83] Quirks mode [Internett]. Wikipedia; 2008-05-15 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/Quirks_mode

[84] Google (hjemmeside): http://www.google.com

[85] Microsoft (hjemmeside): http://www.microsoft.com

[86] Yahoo (hjemmeside): http://www.yahoo.com

[87] Opera (hjemmeside): http://www.opera.com

[88] Browsershots (hjemmeside): http://browsershots.org

[89] Pettersen, B. Nettsted for bridgetjenester. [Masteroppgave]. Bergen: Universitetet i Bergen; 2006. 107 sider

[90] Wi-Fi [Internett]. Wikipedia; 2008-05-28 [hentet 2008-05-29]
Tilgjengelig fra: http://en.wikipedia.org/wiki/Wi-Fi

[91] Bluetooth (produktside): http://www.bluetooth.com/Bluetooth/

[92] J2ME Polish (hjemmeside): http://www.j2mepolish.org/cms

[93] Bluetooth Multiplayer Games Framework (hjemmeside): http://sourceforge.net/projects/bluemgf

[94] JSR 82 Bluetooth API and OBEX API [Internett]. USA: Sun Microsystems; 2006 [hentet 2008-05-29]
Tilgjengelig fra: http://java.sun.com/javame/reference/apis/jsr082/

[95] NetBeans IDE (hjemmeside): http://www.netbeans.org

[96] Nokia (hjemmeside): http://www.nokia.com

[97] Sony Ericsson (hjemmeside): http://www.sonyericsson.com

[98] JUnit (hjemmeside): http://www.junit.org

[99] Sun Java Wireless Toolkit (produktside): http://java.sun.com/products/sjwtoolkit

[100] Martin, Robert C. (2003), *Agile Software Development - Principles, Patterns, and Practices*. Prentice Hall, NJ, USA

[101] An Economic Analysis of Pair Programming. Canada: Erdogmus, H., and Williams, L. , National Research Council Canada; 2002 [hentet 2008-05-29]
Tilgjengelig fra: http://iit-iti.nrc-cnrc.gc.ca/iit-publications-iti/docs/NRC-44961.pdf

[102] On the Effectiveness of Test-first Approach to Programming. Canada: Erdogmus, Hakan, National Research Institute Canada; 2005 [hentet 2008-05-29]
Tilgjengelig fra: http://iit-iti.nrc-cnrc.gc.ca/publications/nrc-47445_e.html

[103] eXtreme Solo: A Case Study in Single Developer eXtreme Programming. Auckland: Cronin, Gareth, University of Auckland; 2002 [hentet 2008-05-29]
Tilgjengelig fra: http://www.croninsolutions.com/writing/eXtremeSolo.pdf

[104] Sun Java SE (produktside): http://java.sun.com/javase/

[105] Sun Java ME (produktside) http://java.sun.com/javame/index.jsp

[106] Java SE HotSpot at a Glance [Internett]. USA: Sun Microsystems; 1994-2008 [hentet 2008-05-29]
Tilgjengelig fra: http://java.sun.com/javase/technologies/hotspot/index.jsp

[107] The K virtual machine (KVM) [Internett]. USA: Sun Microsystems; 1994-2008 [hentet 2008-05-29]
Tilgjengelig fra: http://java.sun.com/products/cldc/wp/

[108] Java™ 2 Platform Standard Edition 6.0 API Specification [Internett]. USA: Sun Microsystems; 1994-2008 [hentet 2008-05-29]
Tilgjengelig fra: http://java.sun.com/javase/6/docs/api/

[109] Java ME Technology APIs & Docs [Internett]. USA: Sun Microsystems; 1994-2008 [hentet 2008-05-29]
Tilgjengelig fra: http://java.sun.com/javame/reference/apis.jsp

[110] Java™ 2 Platform, Micro Edition, Connected Limited Device Configuration 1.1 [Internett]. USA: Sun Microsystems; 1994-2008 [hentet 2008-05-29]
Tilgjengelig fra: http://java.sun.com/javame/reference/apis/jsr139

[111] Connected Device Configuration, version 1.1.2 [Internett]. USA: Sun Microsystems; 1994-2008 [hentet 2008-05-29]
Tilgjengelig fra: http://java.sun.com/javame/reference/apis/jsr218

[112] Java™ 2 Platform, Micro Edition, Mobile Information Device Profile 2.0 [Internett]. USA: Sun Microsystems; 1994-2008 [hentet 2008-05-29]
Tilgjengelig fra: http://java.sun.com/javame/reference/apis/jsr118

[113] Bluetooth Spesification Documents [Internett]. Bluetooth SIG; 2008 [hentet 2008-05-29]
Tilgjengelig fra: http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications

[114] Bluetooth Core Specification v2.1 + EDR [Internett]. Bluetooth SIG; 2008 [hentet 2008-05-29]
Tilgjengelig fra: http://www.bluetooth.com/NR/rdonlyres/F8E8276A-3898-4EC6-B7DA-E5535258B056/6545/Core_V21__EDR.zip

[115] Lundekvam, D. Programvare for å bearbeide og presentere interessante bridgespill. [Masteroppgave]. Bergen: Universitetet i Bergen; 2006. 100 sider

[116] A Comparison of the IBM Rational Unified Process and eXtreme Programming [Internett] USA: John Smith , IBM; 2003 [hentet 2008-05-29]
Tilgjengelig fra:
http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/TP167.pdf

[117] Using the IBM Rational Unified Process for Small Projects: Expanding Upon eXtreme Programming [Internett]. USA: Gary Pollice, IBM; 2003 [hentet 2008-05-29]
Tilgjengelig fra:
http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/tp183.pdf

[118] The Java Platform: Five Years in Review 2 [Internett]. USA: Sun Microsystems; 1994-2008 [hentet 2008-05-29]
Tilgjengelig fra: http://java.sun.com/features/2000/06/time-line.html