HOVEDPROSJEKT:

# BIB-*it*

en verktøykasse for **BIBTEX**

FORFATTERE:

Guro Bu
Geir Engen
Lubomir Mihailov
Stein Strindhaug

Dato:

22.05.2006

# SAMMENDRAG AV HOVEDPROSJEKT

| | | |
|---|---|---|
| Tittel: | Bib-*it* – en verktøykasse for BibTeX | Nr. : 1 |
| | | Dato : 22.05.06 |
| Deltaker(e): | Guro Bu | |
| | Geir Engen | |
| | Lubomir Mihailov | |
| | Stein Strindhaug | |
| Veileder(e): | Arne Magnus Bakke | |
| Oppdragsgiver: | Høgskolen i Gjøvik | |
| Kontaktperson: | Førsteamanuensis Ivar Farup, IMT, Høgskolen i Gjøvik | |
| Stikkord (4 stk) | LaTeX, BibTeX, bibliografi, referanseliste | |

| | | |
|---|---|---|
| Antall sider: 6 + 140 | Antall bilag: 7 (A-G) | Tilgjengelighet (åpen/konfidensiell): åpen |


Kort beskrivelse av hovedprosjektet:

Høgskolen i Gjøvik har mange brukere av LaTeX og BibTeX blant både ansatte og studenter. LaTeX er et verktøy som brukes mye for å skrive spesielt tekniske dokumenter med mye formler, figurer og kryssreferanser. Til LaTeX finnes også hjelpeprogrammet BibTeX som holder orden på bibliografisk innhold, som referanser og sitering av disse. Brukeren skriver informasjon om referansene han eller hun vil sitere (forfatter, tittel, årstall, utgiver osv.) i en tekstfil (vanligvis kalt en bib-database). Denne bib-databasen vokser etter hvert til å bli stor og uoversiktelig og vanskelig å vedlikeholde for hånd.

Vi har i dette hovedprosjektet utviklet Bib-*it* – en GUI-applikasjon som forenkler vedlikehold av bib-databaser. Bib-*it* inneholder også mer avansert funksjonalitet som hjelper brukeren med å holde bib-databasen konsistent (duplikatsøk), samt en stilgenerator som lar brukeren definere og vedlikeholde sine egne bibtexstiler.

Bib-*it* er plattformuavhengig og applikasjonen er utviklet i Java.
Vi har brukt en inkrementell utviklingsmodell. Dette gjorde det mulig å gi ut flere betaversjoner av Bib-*it* i løpet av prosjektperioden. Disse betaversjonene ble testet av oppdragsgiver og andre interesserte, og slik fikk vi nyttige tilbakemeldinger som har vært med på forme det endelige resultatet.

# Forord

BIB-*it* – en verktøykasse for BIBTEX, er utviklet av fire studenter ved Høgskolen i Gjøvik våren 2006. Prosjektoppgaven ble utformet i samarbeid med oppgragsgiver Høgskolen i Gjøvik representert ved Dr. Ivar Farup, og har bestått i å utvikle en applikasjon som håndterer *bib-databaser* på en oversiktlig og lettvint måte. Det har blitt lagt stor vekt på brukervennlighet og funksjonalitet i applikasjonen.

Vi ønsker å takke følgende personer for deres bidrag til prosjektet:

- Dr. Ivar Farup, vår kontaktperson, som under prosjektperioden har vært en god ressurs når det gjaldt å finne løsninger på noen av våre utfordringer under utformingen applikasjonen.
- Arne Magnus Bakke som var vår veileder og som med sin gode tekniske innsikt og kjennskap til både LATEX og BIBTEX, veiledet oss på en utmerket måte gjennom hele prosjektet.
- Alle personer som vi har fått tilbakemelding av gjennom diverse forum på Internett der vi har reklamert for BIB-*it*.

Gjøvik, 22. mai 2006

| | | | |
|---|---|---|---|
| ..... | ..... | ..... | ..... |
| Guro Bu | Geir Engen | Lubomir Mihailov | Stein Strindhaug |

# Innhold

# Figurer

# 1 Innledning

LaTeX er et verktøy som brukes mye for å skrive spesielt tekniske dokumenter med mye formler, figurer og kryssreferanser. LaTeX er et formateringsprogram og ikke et tekstbehandlingsprogram. Med det menes at LaTeX ikke er interaktivt og du ser ikke uten videre hvordan dokumentet du jobber med i LaTeX kommer til å se ut. Du jobber på en mer logisk måte når du jobber i LaTeX ved at du kun bryr deg om selve oppbyggingen og strukturen av dokumentet samt at du konsentrerer deg om selve teksten i dokumentet.LaTeX bruker et tilleggsprogram som heter BibTeX. Brukeren skriver informasjon om referansene han vil sitere (forfatter, tittel, årstall osv.) i en tekstfil (vanligvis kalt en bib-database), og BibTeX formaterer referanselisten til LaTeX-dokumentet ved hjelp av informasjonen i bib-databasen.

## 1.1 Oppgavedefinisjon/problemområde

Mange akademikere og forskere verden over bruker LaTeX og BibTeX for å skrive sine artikler. Når de begynner å bruke BibTeX så vil bib-databasen etterhvert vokse seg uforholdsmessig stor og uoversiktelig og det blir tungvint og tidkrevende å vedlikeholde denne filen.

Bib-*it* er en *GUI*-applikasjon som gjør det lettere å håndtere og vedlikeholde bib-databaser.

Nå finnes det en del lignende applikasjoner som Bib-*it* tilgjengelig allerede, men de er, i følge vår oppdragsgiver og andre med han, ikke tilfredstillende når det gjelder administrering av bib-databaser. Det programmet som er mest brukt i så måte er *JabRef*. Dette programmet er relativt avansert med mange funksjoner, men har noen mangler i forhold til oppdragsgivers behov. I avsnitt 2.4 beskriver vi JabRefs viktigste funksjoner nærmere og i punkt 2.5 begrunner vi hvorfor og hvordan JabRef ikke svarer til oppdragsgivers behov.

### 1.1.1 Resultatmål

Resultatet skal være et praktisk og effektivt verktøy for BibTeX som er brukbart på flere plattformer. Verktøyet skal bestå av et bibliotek for BibTeX[1] som en klientapplikasjon - en GUI-applikasjon - skal jobbe mot. I utgangspunktet skulle vi også utvikle en konsonllbasert applikasjon, men på dette punktet ble den opprinnelige målsetningen endret (se punkt 8.1.1 for en begrunnelse).

[1] som er av en slik kvalitet at andre programmerere som ønsker det, kan lage sine egne brukergrensesnitt mot dette biblioteket

### 1.1.2 Avgrensning

Programmet kommer ikke til å være en komplett LaTeX-parser, det betyr blant annet at spesialtegn og annen LaTeX-kode i navn og titler vil vises slik det står i bib-databasen. En konsekvens av dette vil være at alle felter ikke vil bli sortert helt korrekt alfabetisk, men vil følge ASCII-rekkefølgen for de tegnene som utgjør et spesialtegn. (Dvs. "Händel" blir sortert som `H{\"a}ndel`.)[2] Dette betyr også at søk i databasen må skrives korrekt med LaTeX-skrivemåte eller at man benytter en litt mindre streng søkemetode (f.eks. regulære uttrykk).

Parseren skal kunne lese (om enn ikke tolke) vanlig LaTeX i bib-databasen, men vi kommer ikke til å garantere at de mest kompliserte LaTeX-konstruksjonene alltid vil fungere. Hvis parseren vår ikke takler en streng skal den i hvertfall oppgi linjenummerene for starten av feltet og stedet feilen oppstod, og en melding om typiske feil (ikke støttede kommandoer etc.) og at man bør teste om BibTeX faktisk aksepterer filen. Vi dokumenterer underveis kommandoer vi velger å ikke støtte og hvilke kommandoer som kan brukes i stedet.

Ved lagring av bib-databaser kommer vi til å velge én kodestandard og vi kommer ikke nødvendighvis til å lagre postene i samme rekkefølge som i orginalfilen, men dataene skal være de samme.

## 1.2 Målgruppe

Målgruppen for Bib-*it* er som før nevnt alle akademikere og forskere verden over som bruker LaTeX og BibTeX til sine publiseringer. Dette forutsetter selvsagt at Bib-*it* blir kjent ute i den store verden, noe som er avhengig av hvordan vi profilerer produktet. Målgruppen for denne rapporten er oppdragsgiver, veileder og andre som er interessert i å benytte systemet.

## 1.3 Prosjektorganisering

| | |
|---|---|
| Oppdragsgiver: | Høgskolen i Gjøvik (HiG) |
| Kontaktperson: | Dr. scient. Ivar Farup ved HiG |
| Veileder: | Arne Magnus Bakke |

[2] Forøvrig er det umulig å sortere ord med spesialtegn (ikke-engelske bokstaver) helt "korrekt" siden f.eks. det norske og danske alfabetet slutter med "æøå" mens det svenske slutter med "ääö".

### 1.3.1 Gruppesammensetning

| | |
|---|---|
| Gruppeleder: | Guro Bu |
| Drift av utviklingsmiljø: | Geir Engen og Lubomir Mihailov |
| Kodeansvarlig: | Stein Strindhaug |
| Sekretær: | Geir Engen |
| Dokumentansvarlig: | Guro Bu |

## 1.4 Gruppens faglige bakgrunn

Gruppen består av fire personer som har ganske ulik bakgrunn og erfaring. Tross denne ulikheten har alle medlemmene tatt fordypning i programutvikling[3],har programmering som interesse og har gjennom studiene programmert en del. Et par av medlemmene har tatt faget kompilatorteknikk[4] mens de to siste medlemmene tok faget systemadministrasjon[5]. Kompilatorteknikk viste seg å være et svært nyttig fag i forbindelse med parsing av bib-databaser, som var en nødvendighet for å kunne lage applikasjonen vi skulle lage. Alle medlemmene har tidligere vært med på å utvikle applikasjoner til forskjellige formål, dog i forskjellige programmeringsspråk, og vi hadde derfor relativt gode kunnskaper om programmering ved oppstarten av prosjektet.

## 1.5 Utviklingsmodell

Da vi skulle avgjøre hvilken systemutviklingsmodell vi skulle bruke var det tre retninger som ble diskutert som aktuelle.

Den ene retningen baserte seg på XP (eXtreme Programming), der vi kan programmere parvis, noe som vi så som positivt i og med at vi er fire stykker på gruppa. Det andre som tiltalte oss med XP var at den har en uformell tilnærming og vi kan få ut tidlige versjoner av programmet. Det som talte i mot XP var at vi ikke hadde noen On-site Customer fordi oppdragsgiveren vår, representert av en lærer ved skolen, hadde gått ut i permisjon og kunne følgelig ikke være tilstede under utviklingen. I tillegg til dette var det klare krav i prosjektet og disse kravene hadde lite av potensielle forandringer. XP funger spesielt godt for prosjekter hvor endringer er svært sannsynlige, dette gjør denne utviklingsmodellen mindre aktuell for vårt prosjekt.

Den andre retningen vi vurderte var en *Open Source*-modell som brukes slik at en tidlig versjon legges ut på Internett til vurdering og kommentar fra andre i tillegg til at man også kan få inn forslag til forbedringer og ferdig kode på applikasjonen.

[3] IMT3281 Programutvikling ved HIG
[4] IMT3052 Kompilatorteknikk ved HIG
[5] IMT3292 Systemadministrasjon ved HIG

Dette virket i utgangspunktet som en grei metode, men vi vurderte den til å være litt for ambisiøs med hensyn på tidsrammen vi har på prosjektet. Det var et krav fra oppdragsgiver at resulterende applikasjon skulle være Open Source, og det ville derfor vært naturlig å legge vår kode ut på nettet også underveis i utviklingen, i tråd med Open Source-modellen, men på grunn av den relativt korte tiden vi har til disposisjon og det faktum at det er andre mer etablerte programmer som også er Open Source og samtidig har mange likehetstrekk med vår løsning, vil det kanskje være liten respons fra andre potensielle deltakere på Internett. Vi vurderer imidlertid denne modellen som aktuell.

Den tredje retningen vi så på var en inkrementell utviklingsmodell. Denne modellen passer vårt prosjekt på den måten at det er klare krav til applikasjonen og den er veldig modulær i måten man kan utvikle den på, slik at det er naturlig å dele opp utviklingen i inkrementer.

Disse inkrementene kan vi kjøre test på og legge de ut som ferdig kode i løpet av kort tid. Vi har også muligheten til å kjøre refaktorering på noen av inkrementene hvis de viser seg å ikke holde mål. Vi har også klare tidsfrister å forholde oss til som også gjør denne modellen veldig aktuell.

I tillegg til at vi dermed velger å bruke den inkrementelle modellen kommer vi til å ta i bruk noen av eXtreme Programming sine fremgangsmåter ved å bruke par-programmering på inkrementene.

Vi planlegger også å bruke litt av Open Source modellen ved å legge ut ferdige inkrementer på Internett til testing, men er bevisste på at det muligens ikke blir veldig respons på dette på grunn av tidligere diskuterte argumenter.

Som konklusjon tar vi dermed i bruk en inkrementell metode som hovedmodell med innslag av eXtreme Programming og Open Source modellene.

## 1.6 Valg av utviklingsmiljø

Et av kravene fra oppdragsgiver var at både GUI-applikasjonen og konsollapplikasjonen skal være plattformuavhengige. Vi vurderte flere alternativer; Qt og wxWidgets er open source vindusbiblioteker som kan brukes sammen med mange av de mest brukte programmeringsspråkene (bl.a. C++, Python, Smalltalk, Perl og Java), og som gir plattformuavhengige GUI-applikasjoner i den forstand at programmereren kan kompilere og kjøre den samme kildekoden med minimale endringer på ulike plattformer.

Valget falt tilslutt på Java/Swing. Alle i gruppa er kjent med språket og vi vil på denne måten spare mye tid ved å slippe å sette oss inn i et nytt språk. Java er i stor grad plattformuavhengig, siden det vanligvis kompileres til bytekode som tolkes av en virtuell maskin.

## 1.7 Arbeidsform

Gruppen ble tildelt grupperom A030 sammen med en annen gruppe på tre personer. Stort sett ble alt arbeidet gjort på grupperommet, med noen unntak. Det var ikke noe stort hinder å jobbe hjemmefra på grunn av et versjonhåndteringssystem på serveren (*Subversion*) som sto på skolen og som holdt orden på alle filer og forandringer hvert enkelt gruppemedlem gjorde på den enkelte fil. Til tross for dette systemet valgte vi å gjøre mesteparten av arbeidet på grupperommet, siden dette gjorde det lettere å kommunisere og samarbeide med de andre gruppemedlemmene. Arbeidstiden varierte litt fra dag til dag og fra person til person, men vi prøvde å få minimum ca 5 arbeidstimer hver dag i gjennomsnitt. En fullstendig arbeidslogg er tatt med i vedlegg H.

## 1.8 Organisering av rapporten

Vi har ved organiseringen av rapporten basert oss i hovedsak på disposisjonsmalen som Høgskolen i Gjøvik har satt opp.

### 1.8.1 Kapitteloppsummering

**Kapittel 1 - Innledning** Beskriver på en overordnet måte hva programmet er og hva det skal gjøre samt hvilke avgrensninger vi har gjort. Dette kapittelet beskriver også hvilken utviklingsmodell og hva slags utviklingsmiljø vi valgte.

**Kapittel 2 - Bakgrunn** Beskriver bakgrunnsteori som er nødvendig for forståelsen av resten av rapporten.

**Kapittel 3 - Kravspesifikasjon** Beskriver utviklingsprosessen, krav fra oppdragsgiver ved prosjektstart og en fullstendig beskrivelse av de endelige kravene.

**Kapittel 4 - Design** Inneholder en beskrivelse av hvilket design vi valgte på løsningen samt designvalg vi har måttet gjøre underveis.

**Kapittel 5 - Implementering** Inneholder en beskrivelse av hvordan utvalgte deler av systemet er implementert.

**Kapittel 6 - Testing** Inneholder en beskrivelse av hvordan vi har testet applikasjonen. Vi har tatt med noen eksempler på feil som ble funnet og hvordan vi hør rettet disse.

**Kapittel 7 - Diskusjon av resultater** Inneholder en vurdering av arbeidsprosessen og resulterende programvare. Kapittelet inneholder også en vurdering av avvik fra kravspesifikasjonen og den opprinnelige tidsplanen, samt en liste over mulige forbedringer i programvaren.

**Kapittel 8 - Konklusjon** Inneholder prosjektets konklusjon og en kort oppsummering.

### 1.8.2 Layout

Rapportens hoveddeler er inndelt i kapitler. Disse kapitlene er inndelt i underkapitler for hvert emne innen hvert kapittel. Disse emnene er igjen inndelt i underkapitler. Disse tre første inndelingene er nummerert i henhold til innholdsfortegnelsen. Videre inndeling går da på punktlisteformat, og er følgelig ikke oppført i innholdsfortegnelsen.

Ord som finnes i ordlisten (Vedlegg A), er formatert med *kursiv* første gangen de brukes i rapporten. Alle andre ganger de brukes er de formatert på vanlig vis for å få et ryddig og helhetlig inntrykk.

# 2 Bakgrunn

BibTeX er et program som brukes sammen med LaTeX, og som holder orden på referanser og sitering av disse. I dette kapittelet går vi nærmere inn på syntaksen til BibTeX-formatet, forklarer sammenhengen mellom bibliografistilsystemer og BibTeX-stiler og ser på hvordan referanselister kan genereres automatisk i LaTeX, før vi gir en oversikt og presentasjon av eksisterende verktøy for å vedlikeholde bib-databaser.

## 2.1 BibTeX-databaseformatet

En bibtex-database (.bib) er en ASCII-tekstfil som inneholder bibliografiske poster strukturert som nøkkelord/verdi-par. Syntaksen er enkel; en post består av tre deler: en entrytype (en typespesifikator, kan f.eks. ha verdien *book*), en bibtexkey[6] og 1 eller flere datafelter (nøkkelord/verdi-par). Det er bibtexkey'en som brukes som argument i en LaTeX `\cite`-kommando for å referere til en bestemt post i databasen. Det eneste obligatoriske datafeltet er crossref, som brukes i for eksempel en referanse til et kapittel for å angi hvilken bok kapittelet er hentet fra, se eksemplet under.

```
Eksempel på en \gloss{bibdatabase}:

@ARTICLE{luke:2003:ECJ,
    author = {Luke, Sean},
    journal = {Evolutionary Computation},
    number = {1},
    pages = {67 --106},
    title = {Modification Point Depth and Genome Growth in
                Genetic Programming},
    volume = 11,
    year = 2003,
    keywords = {genetic algorithms, genetic programming,
                Introns, Inviable Code, Code Bloat, Crossover
                Point},
}

@INBOOK{Mittelbach2004:12,
    title = {Managing Citations},
    booktitle = {The LaTeX Companion},
```

[6] Denne fungerer på samme måte som en primærnøkkel i en database - den skal identifisere en post i bib-databasen. Imidlertid er ikke BibTeX-formatet fullt så strengt; når en post med ikke-unik bibtexkey siteres gir BibTeX en advarsel om dette og hvis ingen poster med ikke-unik bibtexkey siteres men er tilstede i databasen som brukes ignoreres det. Det anbefales imidlertid på det sterkeste at man velger unike bibtexkey'er.

```
    chapter = {12},
    crossref = {Mittelbach2004}
}

@BOOK{Mittelbach2004,
    author = {Mittelbach, Frank and Goossens, Michel and Braams, Johannes
              and Carlisle, David and Rowley, Chris},
    title = {The LaTeX Companion},
    year = 2004,
    edition = {Second},
    publisher = {Addison-Wesley}
}
```

## 2.2 Bibtexstiler

### 2.2.1 Bibliografistilsystemer

En referanseliste blir vanligvis generert automatisk ut fra en database som inneholder bibliografisk informasjon (for eksempel en bib-database). Listen (og siteringer til referanser) kan formateres på mange ulike måter, men det finnes visse fellestrekk som har gitt opphavet til ulike bibliografistilsystemer. Mittelbach et al. [2004b] beskriver fire vanlige bibliografistilsystemer: *short-title, author-date, author-number* og *number-only.*

**short-title**
Eks. (fotnote eller i teksten)

```
Brett, M. G. Biblical Criticism in Crisis? The Impact of the
Canonical Approach on Old Testament Studies. Cambridge 1991.
Cambridge University Press.
```

I short-title-systemet gis all bibliografisk informasjon om en publikasjon direkte i teksten eller som en fotnote. Første gang en publikasjon blir sitert er det vanlig å ta med mye informasjon, i senere sitering til samme publikasjon brukes gjerne en forkortet versjon som består av for eksempel forfatterens navn, en kort tittel eller året. I short-title-systemet er det vanlig å ikke ta med en fullstendig referanseliste på slutten av dokumentet. Short-title er mye brukt i humaniora.

**author-date**
Eks. (sitering)

```
Brett 1991: Kap.2
```

I author-date-systemet (også kjent som Harvard-systemet) gis også referansen direkte i teksten, men bare i form av forfatterens navn og publikasjonsår. Fullstendig bibliografisk informasjon gis i en separat referanseliste. Author-date brukes ofte i naturvitenskap og samfunnsvitenskap.

**author-number**
Eks. (sitering)

`Brett[3]`

I author-number-systemet gis referansen i teksten i form av forfatterens eller forfatternes navn fulgt av et nummer som indikerer hvilken publikasjon av denne forfatteren som er sitert. I bibliografilista numreres alle publikasjonene per forfatter.

**number-only**
Eks.: (sitering)

`[3]` eller `(3)` eller lignende
I number-only-systemet nummereres publikasjonene sekvensielt. Siteringene i teksten refererer til disse numrene.
Author-number og number-only er ikke så mye brukt som short-title og author-date.

### 2.2.2 Bibliografistilsystemer i LaTeX / BibTeX

Til tross for at short-title og author-date tradisjonelt er de mest brukte stilsystemene, er det number-only som er standardsystemet i LaTeX, og var lenge det eneste systemet LaTeX støttet.

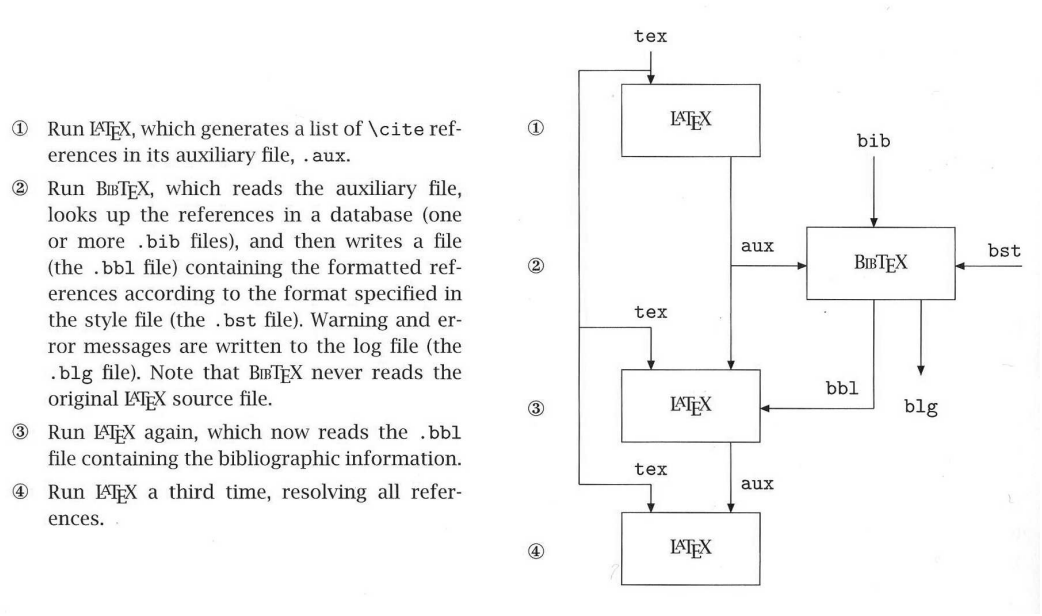
Det har blitt gjort mange forsøk på å få støtte for author-date-siteringer i LaTeX. Natbib (NATural sciences BIBliography), utviklet av Patrick Daly, er den mest vellykkede av disse.

I LaTeX-miljøer er det i dag number-only og author-date som er de mest brukte stilsystemene.

### 2.2.3 Eksempel på hvordan referanselister kan lages i LaTeX / BibTeX

Utgiver definerer gjerne eksakte regler for hvordan en referanseliste og siteringer til referanser (i teksten) skal formateres, og disse reglene må forfatteren forholde seg til. Mittelbach et al. [2004c] belyser flere problemstillinger knyttet til å produsere bibliografilister for hånd:

- det er vanskelig å gjøre siteringene og bibliografiinformasjonen konsistent, for eksempel kan det forekomme ulik bruk av forkortet/ikke forkortet fornavn, kursiv




Figur 1: Hvordan inkludere en referanseliste i et LaTeX-dokument. Figuren er hentet fra Mittelbach et al. [2004b].

eller ikke kursiv tittel, forskjellige skrivemåter: "utg.", "Utg." eller "Utgiver" og ulike varianter av kombinasjonen journal/volum/nummer.

- En bibliografi som er lagt ut i én stil (for eksempel sortert alfabetisk etter forfatter og år) er ekstremt vanskelig å konvertere til en annen stil (for eksempel numerisk sortering)
- Det er vanskelig å vedlikeholde en stor database med bibliografiske referanser[7] for hånd

I BibTeX / LaTeX automatiseres denne prosessen på følgende måte (se også figur 1);

- data om referanser samles i en bib-database (for eksempel ved hjelp av et bibliografihåndteringsverktøy som Bib-*it* )
- referansene siteres i LaTeX-dokumentet ved hjelp av ulike varianter av `\cite`-kommandoer og `\nocite`-kommandoer (for eksempel `\cite{Mittelbach2004}`)
- man leter fram en passende standard *BibTeX-stil*, evt. endrer denne slik at den gir nøyaktig den formateringen man ønsker, eller skriver en egen BibTeX-stil.

[7] for gjenbruk i andre dokumenter

- bib-databasen og BibTeX-stilfilen angis i LaTeX-dokumentet slik:

```
\bibliographystyle{minBibtexstil}
\bibliography{minBibDatabase}
```

Ved hjelp av bib-databasen, BibTeX-stilfilen og informasjonen i LaTeX-dokumentet, skriver LaTeX ut referanselisten for eksempel på slutten av LaTeX-dokumentet.

### 2.2.4 BibTeX-stiler

Som nevnt tidligere, er det BibTeX-stilen som bestemmer hvordan referanselisten skal se ut.
I en LaTeX-implementasjon (som f.eks. TexLive) følger det med et antall standard BibTeX-stiler (abbrv.bst, alpha.bst, plain.bst, plainnat.bst, unsrt.bst og unsrtnat.bst er de vanligste). I tillegg har mange organisasjoner og enkeltpersoner laget egne BibTeX-stilfiler som samsvarer med stilen som brukes i bestemte journaler eller av utgivere.

**BibTeX-stilspråket** BibTeX-stiler bruker et postfiks stakkespråk for å fortelle BibTeX hvordan referansene i referanselisten skal formateres. Språket har 10 kommandoer som brukes for å manipulere konstanter, variabler, funksjoner, stakken og entrylisten. Det finnes to typer funksjoner; innebygde funksjoner (implementert i BibTeX) og brukerdefinerte funksjoner (defineres vha. MACRO-kommandoen eller FUNCTION-kommandoen). Strengkonstanter defineres mellom to doble anførselstegn (" ").
Språket har 10 reserverte tegn. Konstanter og variabler kan ha typen integer eller string, og det finnes 3 typer variabler.

**Globale variabler** , deklareres slik:

```
INTEGERS { my.int.var }
STRING { et.al.string }
```

**Entry-variabler** er heltalls- eller strengvariabler. Hver av disse får en verdi for hver referanse i listen lest fra en bib-database. Eksempel:

```
ENTRY{
  ...
}
{ ... }
{ label short.list dummy }
```

**Felter** er read-only strengvariabler. Disse variablene lagrer informasjonen fra bib-databasen. Det er en verdi for hver referanse, og variablene settes med READ-kommandoen.

```
ENTRY{
  author
  editor
  title
  year
}
{ ... }
{ ... }
```

Funksjonene får sine parametre fra stakken. Noen av funksjonene har returverdi, som legges tilbake på stakken.
Noen av de mest brukte innebygde funksjonene er:

**call.type$** Kaller en funksjon hvor navnet er entrytypen til en referanse (f.eks. `FUNCTION { book }`)

**duplicate$** Dupliserer referansen.

**empty$** Returnerer 1 hvis parameteren er en tom streng eller et felt som ikke eksisterer, 0 ellers.

**format.names$** Formaterer en streng som representerer et eller flere navn i *BibTeX name format* og returnerer den formaterte strengen. Strengen formateres i følge spesifikasjonene gitt som parameter (eks. med formateringsparameter `"{f.}{ ll}"` vil navnestrengen "Nina Beate Larsen" returnere den formaterte strengen "N.B. Larsen").

**if$** Tar tre parametre (to funksjonsliteraler og et heltall). Hvis heltallet er større enn 0 blir den andre funksjonen kjørt, hvis ikke blir den første funksjonen kjørt.

**pop$** Popper av det øverste elementet på stakken.

**write$** Skriver ut parameteren til output-bufferet.

Se Patashnik [1988] for en fullstendig beskrivelse av kommandoer og innebygde funksjoner i stakkespråket.

### 2.2.5 CustomBib

CustomBib er et system som man kan bruke for å generere egne BibTeX-stiler. CustomBib er utviklet av Patrick Daly, som startet på prosjektet i 1993. For å få generert en stil må brukeren besvare en rekke spørsmål (over 70), og en bst-fil bygges opp utfra svarene brukeren gir. Spøsmålene presenteres i form av en konsollbasert veiviser.

Alle spørsmålene, svarene og de valgte svaralternativene skrives til en dbj-fil i et format som kan prosesseres av *DOCSTRIP-programmet* og som dessuten er lesbart for mennesker. En masterfil (merlin.mbs brukes hvis ikke brukeren velger en annen fil) inneholder alternativ koding som avhenger av DOCSTRIP -valg. Når brukeren kjører LaTeX på dbj-filen bygges en bst-fil opp fra masterfilen ved å strippe bort kode for å beholde bare den koden som samsvarer med valgene i dbj-filen.

## 2.3 Eksisterende verktøy for å vedlikeholde bib-databaser

Til tross for at BibTeX har eksistert lenge, finnes det ikke mange virkelig gode verktøy for å vedlikeholde bib-databaser. De vi har klart å oppspore (gjennom Mittelbach et al. [2004a], *googling* og LaTeX/TeX-diskusjonsgrupper) er noen enkle kommandolinjeverktøy og noen GUI-applikasjoner som ikke tilfører spesielt mye funksjonalitet i forhold til å redigere en tekstfil.
Disse verktøyene er beskrevet i Mittelbach et al. [2004a]:

### Kommandolinjeverktøy

**bibtools** er en samling av kommandolinjeverktøy.

- **aux2bib** er et perlskript som vha. av en .aux-fil lager en bib-database som inneholder bare de referansene som trengs for et bestemt dokument.
- **bibkey** er et C-shellskript som bruker verktøyene *sed*, *egrep* og *awk* for å skrive ut en liste over alle referanser som har en gitt streng som (en del av) bibtexkey'en. Søkestrengen må være et gyldig egrep-uttrykk.
- **looktex** er et C-shellskript som ved fritekstsøk i en bib-database skriver ut en liste av alle referanser i en bib-database som oppfyller søkestrengen.
- **makebib** er et C-shellskript som genererer en bib-database fra et gitt sett med bib-databaser og en (valgfri) liste med siteringer.
- **printbib** er et C-shellskript som lager en .dvi-fil fra en bib-database. Referansene sorteres på bibtexkey og feltene som skrives ut er *keyword* og *abstract*.
- **bib2html** er et perlskript som produserer en HTML-versjon ut fra en eller flere bib-databaser.

**bibclean etc.** er en annen samling av kommandolinjeverktøy

- **bibclean** er et C-program som er en *prettyprinter*, syntakssjekker og leksikalsk analysator for bib-databaser.
- **bibextract** ekstraherer (fra en liste av bib-databaser) de referansene som oppfyller et par av spesifiserte regulære uttrykk, sammen med alle preamble og string-deklarasjoner.

**citefind og citetags** er to Bourne shell-skript. *Citefind* ekstraherer BibTeX-*siteringstagg*ene fra LaTeX-dokumenter eller aux-filer og sender dem til standard output. Deretter plukker *citefind* dem opp og prøver å finne de gitte bibtexkey'ene i den spesifiserte bib-filen. Den resulterende nye bibliografifilen skrives til *stdout*.

**bibtool** er et kommandolinjeverktøy som kombinerer mye av funksjonaliten fra de tidligere diskuterte programmene og i tillegg inneholder det noe ekstra funksjonalitet. Med bibtool kan man blant annet prettyprinte en bib-database, slå sammen flere bib-databaser, sortere, normalisere (som i denne sammenheng vil si å gjøre gjøre om syntaksen slik at den blir konsekvent, f.eks. bruke bare "'er ELLER bare {'er rundt feltverdier), søke etter referanser, generere bibtexkey mm.

### GUI-verktøy

**pybliographer** er et utvidbart vektøy for å vedlikeholde bibdatabaser, som støtter BibTeX, ISI, Medline, Ovid og Refer/EndNote. Pybliographer har funksjonalitet for å legge til/endre/slette referanser, sortere, søke og generere bibtexstiler.
At applikasjonen er utvidbar betyr at brukeren selv kan skrive og kjøre egne Python-skript for å få utført operasjoner som ikke er tilgjengelige gjennom Pybliographers grafiske brukergrensesnitt.
Pybliographer er skrevet i Python og dermed tilpasset UN*X-plattformer. Det finnes riktignok Windows-implementasjoner av applikasjonen, men disse fungerer ikke spesielt bra.

**BibTexMng** er et BibTeX-databasehåndteringsverktøy for Windows. BibTeXMng har funksjonalitet for å legge til/endre/slette referanser, søke, sortere og flytte/kopiere referanser fra en fil til en annen. BibTeXMng kan verken importere fra eller eksportere til andre formater (med unntak av eksportering til .bbl-filer og html-filer), og programmet kjenner kun til standard BibTeX entrytyper. Dessuten blir alle andre felter enn standardfeltene ignorert første gangen en BibTeX-database leses inn, noe som gjør det problematisk å jobbe med andre bibtexstiler som f.eks. jurabib. BibTeXMng støtter heller ikke @String-deklarasjoner.

Mittelbach et al. [2004a] nevner også JBibtexManager og BibKeeper, som nå er slått sammen til JabRef. JabRef blir beskrevet grundig i neste avsnitt.

## 2.4 JabRef

Oppdragsgiver kjente ikke til JabRef før vårt prosjekt startet, men prosjektgruppen oppdaget denne applikasjonen i løpet av forprosjektet. JabRef er et mye bedre produkt enn alle de andre GUI-applikasjonene vi fant for samme formål, og vi bestemte oss derfor for

å teste JabRef grundig helt i begynnelsen av prosjektet, for å se hvor godt denne applikasjonen samsvarte med kravene vi hadde fått fra oppdragsgiver. På grunnlag av denne testingen kunne vi avgjøre hva vi skulle gjøre videre. I samråd med oppdragsgiver kom vi fram til at vi hadde tre alternativer:

1. utvikle en applikasjon fra bunnen, som planlagt
2. bli med som utviklere av JabRef. Implementere og endre JabRef for å få inn ønsket funksjonalitet.
3. basere oss på JabRef og videreutvikle denne som en egen applikasjon

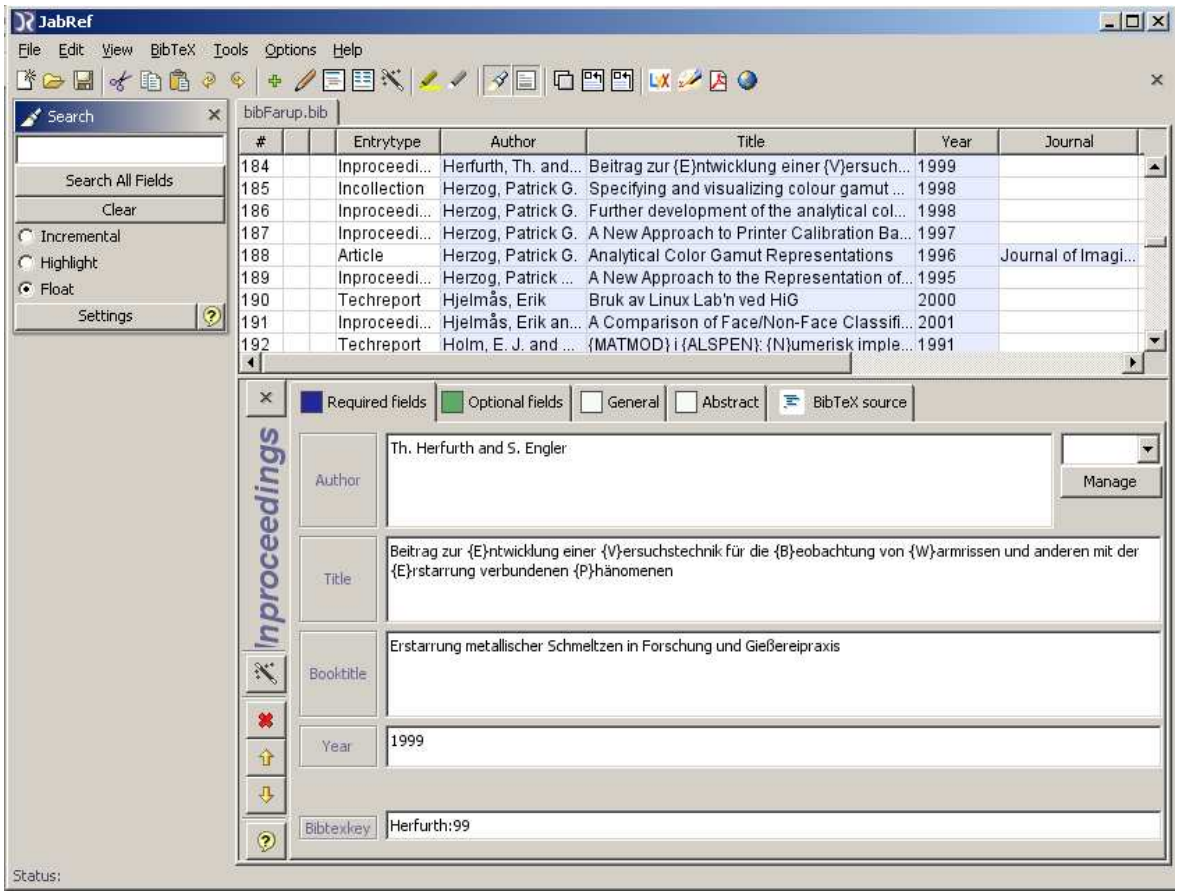
I dette avsnittet vil vi først gi en kort presentasjon av JabRef, deretter gjengi hovedpunktene fra testingen og tilslutt begrunne hvorfor vi valgte alternativ 1.

### 2.4.1 Hva er JabRef?

JabRef er et Open Source-verktøy for å håndtere BibTeX-bibliografidatabaser. JabRef er skrevet i Java, og er dermed plattformuavhengig. De viktigste funksjonalitetene til JabRef inkluderer;

- Legge til/endre og slette referanser
- Søke i alle felter, *regexp-søk*
- Gruppere referanser etter f.eks. keyword (eller andre felter)
- Importere fra diverse formater
- Eksportere til diverse formater
- Legge til egne felter til en BibTeX entrytype
- Tilpasse det grafiske brukergrensesnittet (farger, fonter osv.)
- Kjøre eksterne applikasjoner
- Automatisk generering av bibtexkey
- Medline/citetseer-søk

Første versjon av JabRef kom i november 2003. Prosjektet startet som en sammenslåing av to eksisterende bibliografidatabase-verktøy; *JBibtexManager*, utviklet av Nizar Batada, og *BibKeeper*, utviklet av Morten O. Alver m.fl.




Figur 2: JabRefs hovedvindu

### 2.4.2 Testing av JabRef

Ved prosjektets begynnelse (desember 2005) testet vi versjon 1.8.1 av JabRef.
Førsteinntrykket var at JabRef er en applikasjon med mye og til dels avansert funksjonalitet, men at det er litt vanskelig å skille det vesentlige fra det uvesentlige (menyene og valgene i GUIen kunne kanskje vært litt bedre organisert). I tillegg mangler JabRef noe av funksjonaliteten som var ønsket av oppdragsgiver.

**Grafisk brukergrensesnitt**

Som vist i figur 2 består hovedvinduet hovedsaklig av en tabell som viser alle referansene i den åpne filen. Når en referanse er valgt presenteres informasjonen om denne i "skjemavisning" nederst, hvor feltene er gruppert etter hvorvidt de er obligatoriske eller ikke, og det er også mulig å vise/redigere bibtexkildekoden i en egen tab her. Hvilket panel som vises helt til venstre er avhengig av valgt operasjon. F.eks. når man velger tools -> search dukker det opp et søkepanel hvor man skriver søketekst og velger søkekriterier.

I *referansetabellen* presenteres alle referansene i den gjeldende filen. Det er ikke mulig å redigere informasjon om en referanse direkte i tabellen.

Det er bare denne ene tabellen som brukes for å presentere referanser. Ved søking vil referansene som inngår i søkeresultatet presenteres i denne tabellen, og når man velger en *gruppe* presenteres alle referansene i denne gruppen i den samme tabellen. Ulike uthevingsteknikker av radene i tabellen brukes for å skille "resultatpostene" fra postene som representerer den åpne filen.

For å endre rekkefølge på feltene og hvilke felter som skal være synlige må man inn i options -> perferences -> entry table columns, det er ikke mulig å endre rekkefølgen med "dra og slipp" direkte i GUIen. Vi synes det er litt underlig at man faktisk kan velge å vise felter som ikke eksisterer (dvs. felter som ikke er standardfelter og som man heller ikke har lagt til selv i en av entrytypene).
Det er vanskelig å se hvilke(t) felt(er) det er sortert på siden tabellheaderen ikke har noen form for piler eller annet som viser sorteringsfelt og sorteringsretning.

**Legge til/slette/endre referanse**

En ny referanse kan legges til på to måter;

1. Velge bibtex -> new entry, velge ønsket entrytype og legge til data om referansen i skjemavisning
2. Velge bibtex -> new entry from plain text og velge ønsket entrytype. I dialogboksen som dukker opp kan man lime inn plain tekst (dvs f.eks. bare dataer til en referanse), tilordne teksten til felter og velge *accept*.

For å endre data i en allerede eksisterende referanse kan man;

1. markere en entry og fylle ut tekstfeltene i *required*, *optional*, *general* og *abstract* tabbene.

2. markere en entry lime inn bibtexkildekode i tabben *bibtex source*.

**Entrytyper og felter**

Det er gode muligheter for å spesialtilpasse entrytyper og felter - man kan legge til egne entrytyper og for alle entrytypene kan man bestemme hvilke felter som skal brukes (og angi om et felt er obligatorisk eller valgfritt i en gitt entrytype).

**Søking**

Søking fungerer bra (raskt), men det er ikke mulig å søke i enkeltfelter. Resultatet av søket kan vises på 3 ulike måter (men i alle de tre tilfellene presenteres resultatet i *hovedtabellen*); *incremental* - som vil si at når man klikker søk markeres den raden i tabellen som er først i søkeresultatet, og med tastetrykk kan man iterere gjennom resultatet, *highlighted* - medfører at alle referansene som inngår i søkeresultatet blir markert (men står på samme sted) og *float* - hvor alle referansene som inngår i søkeresultatet flyttes og plasseres på toppen av tabellen.

**Sortering**

Alle felter betraktes som rene tekstfelter ved sortering. Det vil si at man får et litt underlig resultat ved sortering på f.eks. "year"-feltet.

Stigende sortering på årstall blir derfor slik i JabRef:

| |
|---|
| 1996 |
| 1998 |
| 213 |
| 987 |
| about 1996 |
| omkring 1995 |

mens en mer naturlig rekkefølge ville være;

| |
|---|
| 213 |
| 987 |
| omkring 1995 |
| 1996 |
| about 1996 |
| 1998 |

### Generere bibtexkey

En bibtexnøkkel kan genereres for en eller flere markerte referanser, og formatet for hvordan bibtexkey'en skal bygges opp kan defineres for hver entrytype. Man kan også sette et default-format som brukes for alle entrytyper hvor ikke et eget format er spesifisert.

### Gruppering

Gruppering brukes for å ordne referansene i kategorier. JabRefs grupperingsfunksjonalitet er ganske avansert. Når man lager en ny gruppe kan man velge *statisk*, hvor referansene må legges til i gruppen manuelt eller *dynamisk*, hvor innholdet i gruppen bestemmes ut fra et søk.

### Søke etter duplikate referanser

Søk etter eventuelle duplikate referanser startes med å velge *tools -> find duplicates*. Brukeren får presentert en dialogboks som viser innholdet i to referanser som muligens er duplikater. Brukeren kan velge å beholde den øverste, den nederste, eller beholde begge, og først etter at dette valget er gjort får man se på det neste paret av "potensielle duplikater", og slik fortsetter det til man har vært gjennom alle parene.
Etter litt testing kommer vi til følgende vurdering av duplikatsøk-funksjonaliteten;

1. Måten duplikatene er presentert på (GUIen) gjør det vanskelig å avgjøre om to referanser er duplikater eller ikke. Man kan velge mellom to typer forhåndsvisning. Den ene viser dataene om en referanse i plaintekst, og det går ikke fram av GUIen eller av hjelpefunksjonaliteten hvilke felter som vises til brukeren og heller ikke hvilke felter som er sammenlignet. Den andre forhåndsvisningen er bibtexkildekoden til begge referansene og fungerer mye bedre, men denne er gjemt bort i en annen tab.

2. Det er veldig frustrerende at man kun ser to referanser om gangen, og at man må ta en avgjørelse på om disse er duplikater før man kan vise neste par. Man må da huske på hvilke referanser man har slettet og hvilke man har tatt vare på, f.eks. i situasjoner hvor det er 3 eller flere referanser som er veldig like hverandre.

3. Det finnes ingen mulighet i JabRef for å spesifisere hvor streng duplikatsøkealgoritmen skal være.

4. Brukeren får ingen informasjon om hvor mange "potensielle duplikater" som ble funnet før dialog-veiviseren er helt ferdig. Store bibliografidatabaser har gjerne mange potensielle duplikater, særlig dersom databasen er en sammenslåing av bib-databaser skrevet av flere personer, og i slike tilfeller hadde det vært greit å vite om det bare er noen få potensielle duplikater eller om det er flere titalls.

## 2.5 Vurdering av Jabref mot kravene fra oppdragsgiver

Mye av den grunnleggende funksjonaliteten som var ønsket av oppdragsgiver er ivaretatt i JabRef: søking (delvis), legge til/endre/slette referanser, sortere, eksportere til ulike formater, legge til egendefinerte felter (delvis) og generere bibtexkey'er. Duplikatsøkfunksjonaliteten i JabRef syntes verken oppdragsgiver eller vi var tilfredsstillende, og JabRef har ingen funksjonalitet for å generere egne bibtexstiler.

### 2.5.1 Et felt, flere verdier?

Et av de sentrale kravene til vårt prosjekt var at applikasjonen skal fungere som om bibtexformatet var *normalisert*. Den flate filstrukturen er nettopp en av de store ulempene med å redigere bib-databaser i vanlige teksteditorer. *Author* og *editor* feltene er gode eksempler på dette. Disse feltene inneholder ofte mer enn en verdi (dvs. flere personer), og samme enkeltverdi (en person) brukes svært ofte i flere referanser. En sentral karakteristikk ved mange av de allerede eksisterende GUI-verktøyene vi testet var at GUIen besto av en tabell som ikke gjør noe annet enn å liste opp innholdet i feltene som rene tekstverdier, på samme måte som i kildefilen. Dette gjelder også JabRef; alle feltene (deriblant author og editor feltene) betraktes som rene tekstfelter. Dette har flere ulemper:

- brukeren må skrive inn verdien i forfatter/redaktør-feltet i korrekt *BibTeX name format* (JabRef har ingen syntakssjekking på det som skrives inn i forfatter/redaktørfeltene). Dette krever at brukeren er kjent med syntaksen til BibTeX name format.
- brukeren må skrive inn navnet på en forfatter/redaktør HVER gang han eller hun legger til en referanse. Dette gjør at bib-databasen fort blir inkonsistent ved at navnet til en bestemt person skrives inn litt forskjellig hver gang.

Grunnleggende ting som å skrive inn navn på forfattere er noe man gjør *svært ofte*. Å la sluttbrukeren kunne se BibTeX som et normalisert databaseformat gir en større opplevelse av å ikke redigere en tekstfil. Et navn oppfattes av brukeren som et personobjekt som kan kan slettes og legges til i en refersanse (noe som er mer naturlig for GUI-brukere enn å betrakte en liste med navn som en tekststreng). Med en datastruktur som støtter opp om dette kan man implementere funksjonalitet som gjør det enkelt å holde bib-databasen konsistent. Eksempler på slik funksjonalitet er:

**drag/drop-funksjonalitet** sparer brukeren for mye arbeid ved at han for det første slipper å skrive inn samme navn mange ganger, og for det andre slipper å dobbeltsjekke at navnet han skrev inn faktisk er riktig

**funksjonalitet for å vedlikeholde personer globalt** . Hva om brukeren oppdager at han i 5 år har skrevet navnet til en russisk forfatter feil? Da kan han på en enkel måte endre stavemåten til forfatterens navn på *alle* referansene av denne forfatteren ved å redigere navnet på ett sted

og man kan implementere ytterligere funksjonalitet som hjelper brukeren med å få oversikt over innholdet i bib-databasen og finne fram til det han eller hun leter etter:

**vise alle referanser skrevet av en gitt person** ved å velge en eller flere personer fra en liste kan man få presentert alle referanser til publikasjoner som er skrevet av eller redigert av denne personen. Dette kan selvsagt gjøres i f.eks. JabRef med et regexp-søk, men det er avhengig av at brukeren kan regexp-syntaksen.

Å legge til eller endre forfattere og redaktører på en referanse er oppgaver som utføres *svært ofte* når man vedlikeholder en bib-database. Ved å gjøre disse operasjonene raskere og sikrere vil man oppnå en stor gevinst sammenlignet med tradisjonell tekststrengredigering.

For å kunne implementere funksjonaliteten nevnt ovenfor i BIB-*it* (eller eventuelt i JabRef), er det en forutsetning at datastrukturen er bygd opp på en slik måte at det lar seg gjøre.

### 2.5.2 Ny applikasjon eller videreutvikling av eksisterende?

Etter testingen av JabRef sto vi igjen med tre mulige alternativer til hvordan vi skulle fortsette arbeidet, og det var ikke gitt hvilket alternativ vi skulle velge. De tre alternativene var, som tidligere nevnt:

1. utvikle en applikasjon fra bunnen, som planlagt.
2. bli med som utviklere av JabRef. Implementere og endre JabRef for å få inn ønsket funksjonalitet.
3. basere oss på JabRef og videreutvikle denne som en egen applikasjon

**Manglende funksjonalitet og overflødig funksjonalitet**

JabRef inneholder mye funksjonalitet som oppdragsgiver ikke var interessert i; dette gjaldt først og fremst de utallige mulighetene for å importere og eksportere fra/til ulike formater, gruppering av referanser etter egendefinerte grupper og kommunikasjon med andre programmer. Som nevnt tidligere mangler JabRef to viktige funksjonaliteter, nemlig å kunne generere egne BIBTEX-stiler og godt fungerende funksjonalitet for å søke etter duplikate referanser.

I tillegg ønsket oppdragsgiver en normalisert datastruktur, som beskrevet i forrige avsnitt. Dette var argumenter som veide tungt for å fortsette som planlagt (jfr. punkt 1 over).

Men dette alternativet har også sine ulemper. Mye av den planlagte funksjonaliteten i Bib-it var jo allerede på plass og godt uttestet i JabRef, så hvorfor gjøre alt dette

på nytt? Ville det ikke vært bedre å bygge på noe som allerede eksisterer, og på denne måten kunne bruke mesteparten av tiden på den mer avanserte funksjonaliteten som oppdragsgiver ønsker?

**Delta i et veletablert Open Source-miljø**

Alternativ 2 framsto som et spennende alternativ. Ingen av oss hadde erfaring med å bidra som utviklere i veletablerte Open Source-prosjekter, så dette var en gyllen mulighet til å få prøvd seg på det. JabRef er et stort prosjekt som har pågått lenge, og de har klart å trekke til seg mange brukere. Om vi bestemte oss for å bli en del av JabRef-teamet ville vi også kunne dra nytten av andre (og mer erfarne) utviklere sine kunnskaper.

På den andre siden; det er ikke bare bare å hoppe inn i et godt etablert prosjekt, legge fram kravene fra oppdragsgiver og kjøre på. For det første ville vi måttet bruke en god del tid på å sette oss inn i hvordan applikasjonen er bygd opp. Vi var allerede godt inne i prosjektperioden, og å få fullstendig oversikt over en såpass stor applikasjon er ikke gjort på én kveld. Vi ville også blitt nødt til å endre oppgavebeskrivelsen for hovedprosjektet, og ikke minst selge oss inn i JabRefs utviklermiljø.

Open Source-utvikling er en demokratisk prosess. Hva om JabRef'erne ikke delte vårt og oppdragsgivers syn på hvordan JabRef bør endres og utvides? Og hva med JabRefs øvrige brukere? Hva ønsker de? JabRef er som sagt et veletablert prosjekt som har pågått helt siden 2003. Allerede fra den første uttestingen av JabRef ante vi at det måtte til store endringer i datastrukturen for å få implementert den funksjonaliteten vi ønsket (det gjelder spesielt funksjonaliteten knyttet til personregisteret og flerverdifelter). Å endre datastrukturen i JabRef ville i beste fall vært en stor oppgave, og mest sannsynlig ville ikke "eierne" av prosjektet gå med på dette, siden det tross alt er datastrukturen all funksjonalitet i JabRef bygger på.

Et tredje alternativ er å basere seg på JabRef, men å videreutvikle denne som en egen applikasjon under et annet navn. Dette er (i teorien) fullt mulig siden JabRef er lisensiert under GNU Public License. På denne måten vil vi unngå problemstillingen rundt hvorvidt JabRef-utviklerne går med på og ønsker de nødvendige endringene og utvidelsene. Men det ville naturligvis fortsatt være like mye kode å sette seg inn i og vi ville hatt de samme problemene knyttet til nødvendige endringer i JabRefs datastruktur som for alternativ 2. Det største problemet med dette alternativet er derimot at det ville føre til en splitting av JabRef-miljøet. Ville JabRef'erne godtatt dette? Hva med JabRefs brukere? Og hva med framtiden til JabRef og vår egen nye JabRef-avart? Man kan argumentere med at siden de har valgt en så åpen lisens må de regne med at noen gjør nettopp dette. Men Open Source-utvikling handler også om moral. Å gjøre seg til uvenner med JabRef-miljøet ville neppe være noen god idè.

### 2.5.3 Bib-*it* blir løsningen

Etter diskusjoner med oppdragsgiver og veileder og samtaler internt i gruppa kom vi fram til at å prøve å selge seg inn i JabRefs utviklingsgruppe med oppdragsgivers krav til endringer og utvidelser ville være et såpass vanskelig og tidkrevende prosjekt at det ikke ville gi noen gevinster i forhold til å utvikle hele applikasjonen fra grunnen. Et velkjent prinsipp innen systemutvikling er at det tar ofte lengre tid å endre på en eksisterende applikasjon enn å bygge opp alt fra scratch, spesielt dersom det må gjøres endringer i hovedarkitekturen, som vi mener er nødvendig. Vi bestemte oss dermed for å fortsette som planlagt, og tok fatt på kravspesifisering og implementering av Bib-*it*.

# 3 Kravspesifikasjon

I dette avsnittet går vi nærmere inn på kravene til Bib-*it*, med utgangspunkt i ønskene fra oppdragsgiver ved oppstarten av prosjektet.
*Use cases* er verktøyet vi har brukt for å utdype kravene underveis i prosessen (i samarbeid med oppdragsgiver). Usecase-beskrivelsene fungerer også som en dokumentasjon på kravene.

## 3.1 Innledning

Oppdragsgiver hadde i forkant av hovedprosjektet klare formeninger om hvilken funksjonalitet den resulterende applikasjonen skulle ha. Den ønskede funksjonaliteten var som følger:

1. Søke på enkelt felt eller fri tekst
2. Legge til/endre og slette referanser
3. Sortere på hva som helst
4. Generere forskjellige lister i forskjellige formater (eksportere til EndNote og html)
5. Kjenne til alle obligatoriske og valgbare felter
6. Kunne legge til egendefinerte felter
7. Operere som om BibTeX-formatet var normalisert
8. Håndtere kryssreferanser (f.eks. kapittel i bok, sider i bok)
9. Kunne generere nøkler, men også tillate at brukeren definerer dem selv
10. Finne mulige duplikate referanser
11. Muligens også generere og vedlikeholde bib-stiler (omfattende!)

Det skulle utvikles to brukergrensesnitt - et grafisk og et tekstbasert. Et annet krav var at resulterende programvare skal være open source. I tillegg skulle det være mulig å eksportere til EndNote-format.

I løpet av de første møtene med oppdragsgiver og veileder gikk vi dypere inn på kravene (se usecase-beskrivelser, vedlegg A) og satte opp en fornuftig prioritering som ble utgangspunktet for inndeling i inkrementer.

### 3.1.1 Endring av oppgaven

Tidlig i andre inkrement endret kravene seg noe fra oppdragsgivers side. Etterhvert som det grafiske brukergrensesnittet ble bedre og bedre, falt behovet for et tekstbasert brukergrensesnitt bort. Vi ble derfor enige om å avslutte arbeidet med det tekstbaserte grensesnittet og heller bruke tid på å få funksjonaliteten og det grafiske brukergrensesnittet så bra som mulig. Se kapittelet 8.1.1 for en diskusjon av dette.

## 3.2 Brukerbeskrivelse

### 3.2.1 Omgivelser

Vi kan tenke oss at en person som ønsker å redigere bib-databaser ved hjelp av en teksteditor gjerne har følgende argumenter for å bruke en teksteditor;

- Det er kjapt - å åpne en tekstfil for å legge til/endre tekst tar noen knappe millisekunder
- Jeg har "full kontroll" - jeg trenger ikke å være redd for at omstendigheter jeg ikke har kontroll over endrer dataene mine på en måte jeg ikke ønsker
- Jeg slipper å lære meg å bruke en ny GUI-applikasjon. Teksteditoren jeg bruker er favoritt-teksteditoren som jeg også bruker til andre formål og derfor er godt kjent med.

Siden oppdragsgiver (og veileder) selv har brukt teksteditorer til å redigere bib-databaser i mange år, var et av hans viktigste mål for Bib-*it* at den skulle gjøre det *enkelt* å utføre ting på en bib-database som er vanskelig (eller kanskje umulig) å få til ved teksteditorredigering, og at programmet derfor skulle være nyttig ikke bare for de som foretrekker GUI til alt, men også for de som er vant til teksteditorredigering.

Noen ulemper ved å redigere en bib-database i en teksteditor;

- Det er vanskelig å passe på at navnet til en og samme person skrives på samme måte overalt til enhver tid
- All informasjon må skrives inn hver gang (eneste snarvei er å kopiere tekst fra en annen referanse)
- Man får lite hjelp til å sjekke om syntaksen er riktig
- Man må selv passe på hvilke felter som er obligatoriske og valgfrie. Det er fristende å fylle ut så få felter som mulig istedenfor å sjekke hvilke felter som bør fylles ut, og dette oppdager man først når man bruker BibTeX til å lage referanselister ut fra databasen.

- Man kan i de fleste teksteditor utføre både vanlig- og regexp-søk, men det er svært vanskelig å få en god oversikt over resultatet
- Det er vanskelig å holde oversikt over referansene og f.eks. finne fram til en referanse man ikke husker så mye om (ikke mulig å sortere på noen fornuftig måte)
- Det er vanskelig å holde oversikt over forbindelser mellom referansene (i praksis ryssreferanser)

For oppdragsgiver var det svært viktig at GUI-applikasjonen skulle *tilføre* noe i forhold til å redigere bib-databasen i en teksteditor. Dette var også bakgrunnen for oppgaven.

### 3.2.2 Systemets brukere

Systemets brukere vil være personer som bruker LaTeX og BibTeX for å skrive og vedlikeholde tekniske dokumenter eller andre typer dokumenter. Det forutsettes at brukeren er kjent med LaTeX og BibTeX; vi kommer ikke til å legge ved en detaljert manual for LaTeX eller BibTeX.

## 3.3 Inkrementene

**Inkrement 1**

- Enkel parser, datastruktur
- Legge til/endre/slette referanser
- Lagre
- Sortering

Målet for første inkrement var å designe og implementere en robust datastruktur og implementere en enkel parser, og parallelt med dette lage et enkelt grafisk og tekstbasert brukergrensesnitt for å få testet ut parseren og datastrukturen.

**Inkrement 2**

- Forbedring av parseren, tolkning av spesialtegn
- Søking (enkel, feltsøk, regexp)
- Legge til egendefinerte felter
- Autogenerering av bibtex-key

Målet for andre inkrement var å forbedre parseren og implementere grunnleggende funksjonalitet i biblioteket og gjøre denne funksjonaliteten tilgjengelig gjennom GUIen og TUIen.

Det skulle være mulig å søke etter referanser ved å søke i alle felter eller velge en eller flere felter man vi søke i. I tillegg var det ønskelig å la mer avanserte brukere kunne søke med regex-er.

Egendefinerte felter skulle kunne legges til på hver enkelt referanse (dvs. at om man legger til feltet *sponsororganization* på en referanse er det kun denne ene referansen som får dette feltet).

**Inkrement 3**

- Legge til referanser fra annen database
- Finne duplikater

Målet for tredje inkrement var å gjøre det mulig å slå sammen flere bib-databaser til én (ved å tilføye postene fra en annen bib-database til den åpne bib-databasen), samt å implementere funksjonalitet for å lokalisere eventuelle duplikate referanser.

**Inkrement 4**

- Generere og vedlikeholde bib-stiler

Dette kravet var i utgangspunktet et tilleggskrav fra oppdragsgiver, siden han vurderte dette til å være en omfattende oppgave. Etter en nærmere diskusjon kom vi fram til at dette både ville være en veldig nyttig funksjonalitet; vi ikke har funnet noen andre grafiske bibdatabaseverktøy som har funksjonalitet for dette og dessuten hørtes det ut som en spennende utfordring. Derfor ble vi i samråd med oppdragsgiver enige om å sette av et helt inkrement til å implementere funksjonalitet for å generere og vedlikeholde bib-stiler.

**Inkrement 5**

- Eksportere til forskjellige formater

Etter samtale med oppdragsgiver ble det klart at dette kravet (som var nokså uklart formulert i oppgaveteksten) kun dreide seg om eksportering til EndNote-formatet.

## 3.4 Funksjonalitet foreslått underveis

Underveis dukket det opp flere ideer og ønsker til funksjonalitet, både fra oppdragsgiver, gruppemedlemmer og andre engasjerte Bib-it-brukere;

1. vise alle referanser av en eller flere forfattere/redaktører

2. slå sammen flere personer til én og endre navn på personer (med virkning i hele filen)
3. legge til/slette egendefinerte felter
4. lime inn en referanse (i BibTeX-format)
5. vise/skjule kolonner i hovedtabellen og endre rekkefølge på kolonnene
6. gruppere kryssreferanser
7. forhåndsvisning av informasjonen i en eller flere referanser
8. angre/gjøre om operasjoner
9. finne ikke-unike referanser (dvs. referanser med en ikke-unik bibtexkey)

Disse ønskene ble prioritert fortløpende og implementert innimellom i ledige stunder. Punkt 1 og 2 var mulig fordi vi hadde valgt å operere som om BibTeX-formatet var normalisert og de viste seg å være veldig nyttige funksjoner. Vi valgte å ikke ta med usecase-beskrivelsene for disse tilleggskravene.

## 3.5 Usecase-diagram

Figur 3 viser et usecase-diagram for hele systemet. Se vedlegg A for fullstendige usecase-beskrivelser.

## 3.6 Supplementærspesifikasjon

### 3.6.1 Funksjonalitet

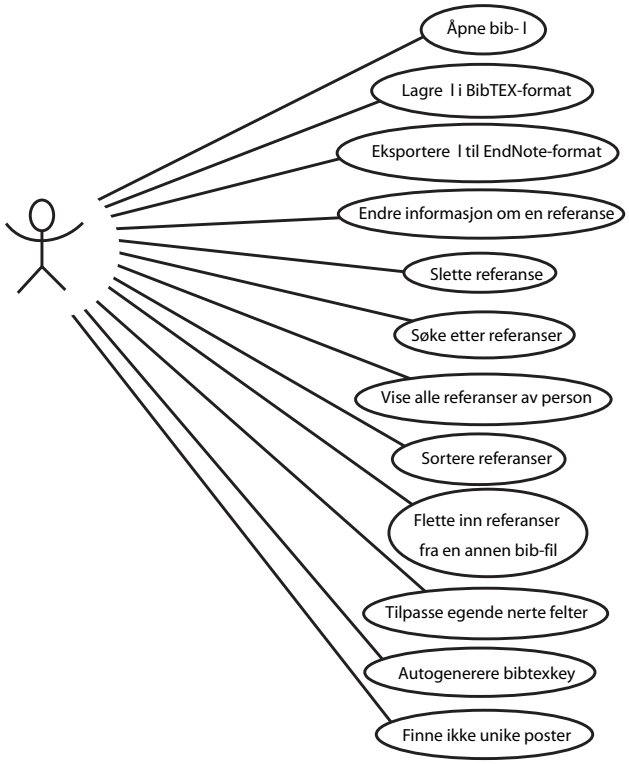
Alle knapper (bortsett fra ok, cancel o.l.) skal ha tooltip-tekst.

### 3.6.2 Brukervennlighet

- Applikasjonen skal være internasjonalisert slik at det er enkelt å tilrettelegge den for andre språk.
- Brukergrensesnittet skal være så enkel som mulig å bruke. Menyene i GUIen skal være organisert slik at de er enkle å finne fram i, og de viktigste menyvalgene skal også være tilgjengelige som knapper på en verktøylinje.

### 3.6.3 Ytelse

Applikasjonen skal kunne håndtere store bib-databaser (opptil ca 5000 referanser eller filstørrelse på opptil 8mB).




Figur 3: Usecase-diagram for Bib-*it*

### 3.6.4 Støtte

- Applikasjonen skal være mest mulig plattformuavhengig, et minstekrav er at den skal fungere på Windows, alle de største Linux-distribusjonene og Macintosh.

# 4 Design

## 4.1 Inndeling i lag og pakker

Hovedfokuset for denne oppgaven har vært å lage et bibliotek mot BibTeX og et grafisk brukergrensesnitt som bruker dette biblioteket. Det var et viktig poeng at biblioteket og brukergrensesnittet skulle være klart adskilt, slik at det er mulig ved en senere anledning å lage nye brukergrensesnitt (grafiske eller tekstbaserte) mot biblioteket. Dette ble utgangspunktet for oppdeling i lag og pakker. Videre forsøkte vi å følge de grunnleggende ideene for strukturering av et system som beskrevet i Buschmann et al. [1996]:

- Organize the large-scale logical structure of a system into discrete layers of distinct, related responsibilities, with a clean, cohesive separation of concerns such that the "lower" layers are low-level and general services, and the higher layers are more application specific.
- Collaboration and coupling is from higher to lower layers; lower-to-higher layer coupling is avoided

Vi delte systemet i to lag; presentasjonslaget og domenelaget. Koblingene går hovedsaklig nedover fra høyere lag til lavere lag, med unntak av koblingen fra domenelaget til event-pakken. Dette er et vanlig prinsipp i software arkitektur og det var nødvendig blant annet for at GUI-komponentene i en del av applikasjonen skal få vite om endringer i objektene i domenelaget når disse endringene skjer i GUI-komponenter i en annen del av applikasjonen.

Biblioteket er implementert i pakken *domain* og Bib-its grafiske brukergrensesnitt er implementert i pakken *ui::gui*. Klassene i domain-pakken har ingen direkte kjennskap til ui-pakken, kun indirekte gjennom pakken *event* (jfr. pakkediagrammet, figur 4).

Resten av dette avsnittet vil beskrive hovedtrekkene i presentasjonslaget og domenelaget.

## 4.2 Presentasjonslaget

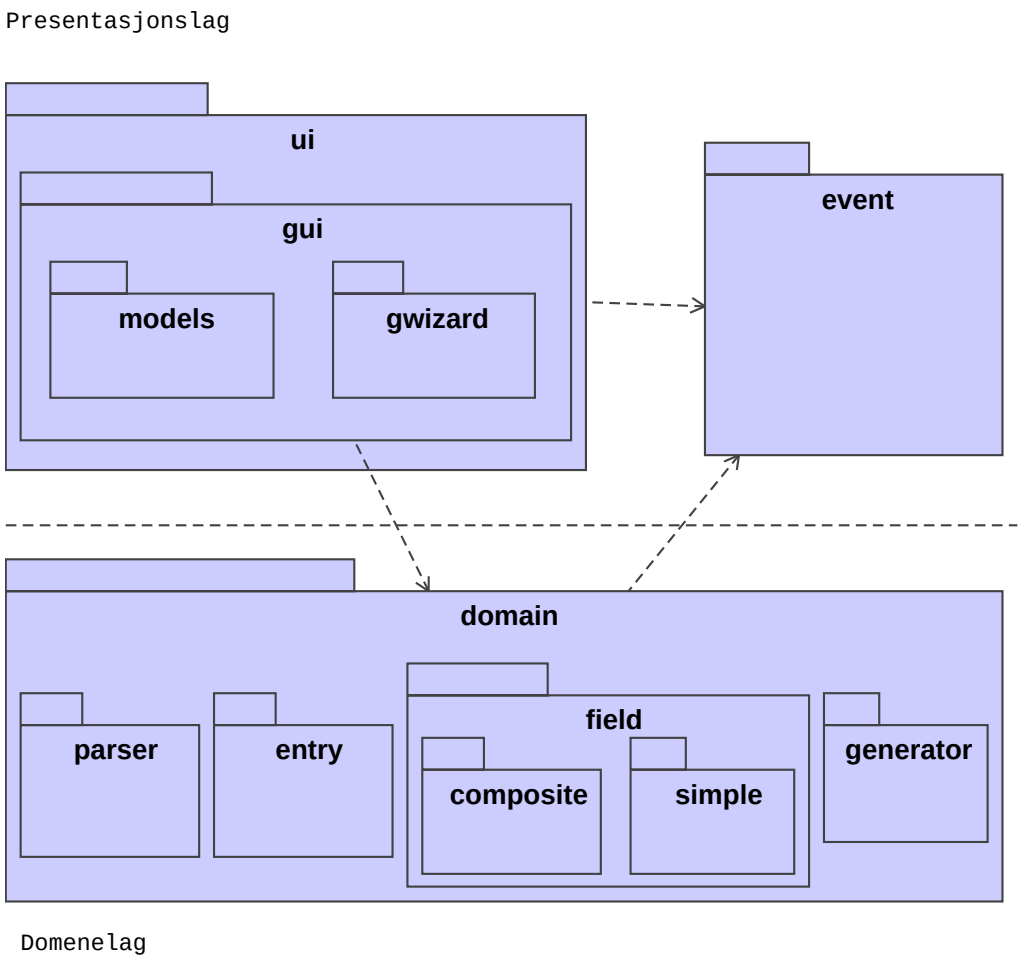
### 4.2.1 Grafisk brukergrensesnitt

**Tabellene**

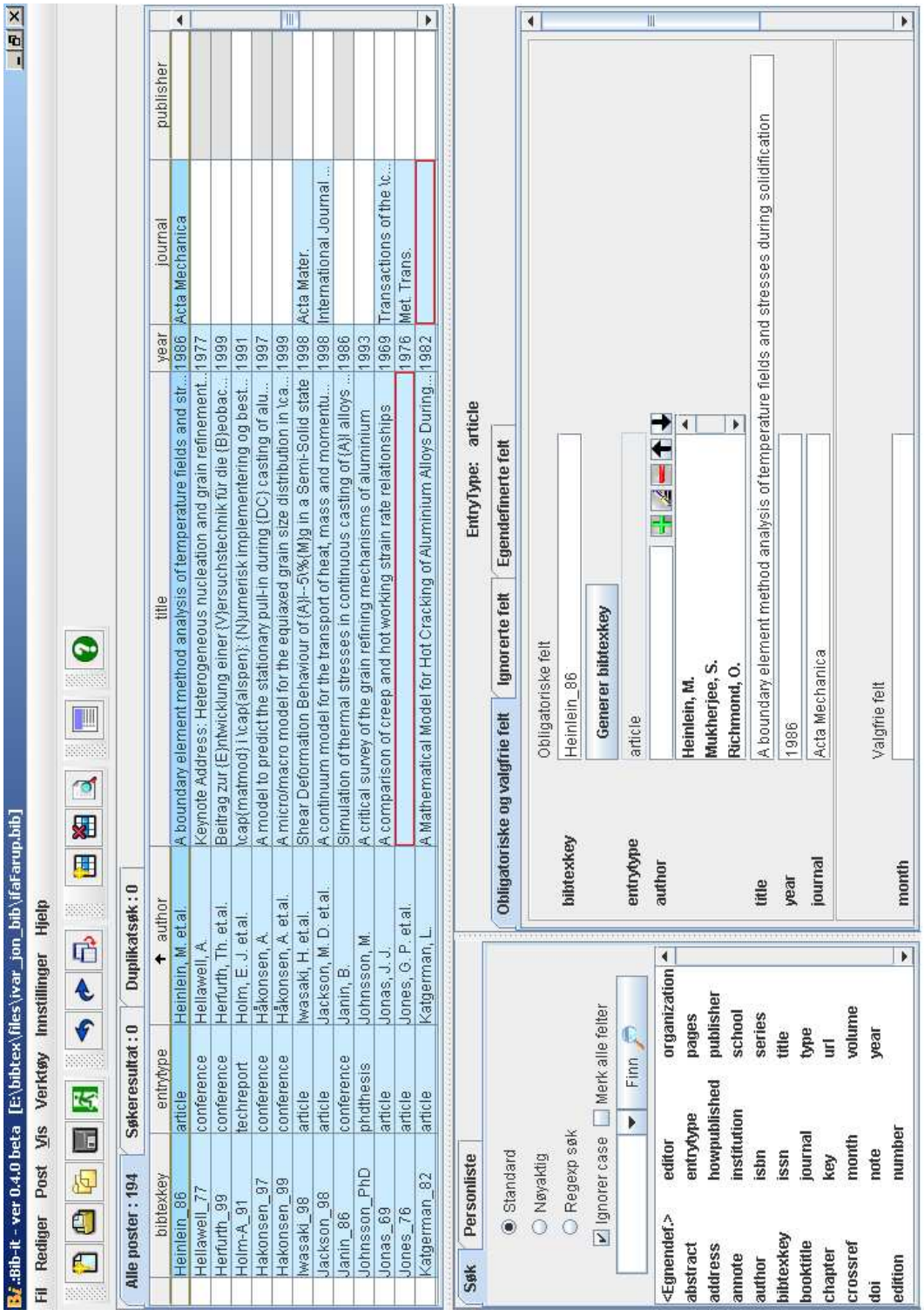
I *tabellpanelet* (se figur 5) kan man vha. tabbene øverst bytte mellom tre forskjellige visninger:

**Hovedtabellen** viser alle referansene i den åpne bib-databasen

**Søkeresultat-tabellen** viser alle referansene i resultatet fra et søk




Figur 4: Pakkediagram for Bib-it




Figur 5: Bib-*its* hovedvindu

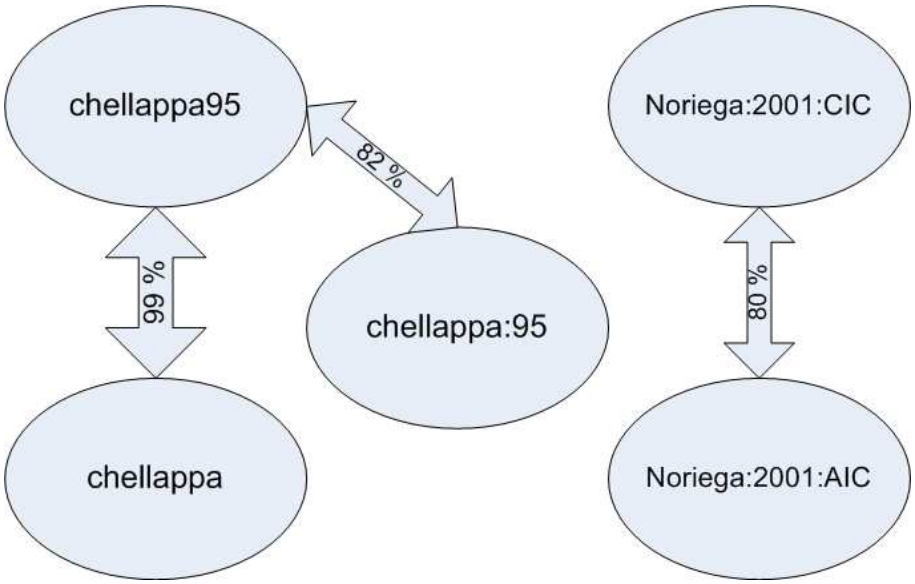
**Duplikatsøkeresultat-tabellen** viser alle referansene i resultatet fra et duplikatsøk

Vi har lagt vekt på å gjøre de tre tabellene så like som mulig (funksjonelt og grafisk), slik at det eneste som endrer seg når man bytter fra en tabell til en annen er dataene i radene. På denne måten unngår vi at brukeren fester seg ved detaljer som ikke har noe med det faktiske innholdet i bib-databasen å gjøre, f.eks. at størrelsen på kolonnene endrer seg, rekkefølgen og hvilke kolonner som vises endrer seg osv.
Tabellene kan **sorteres** på et hvilket som helst felt. Pilene i tabellheaderen viser hvilket felt det er sortert på og sorteringsretning (stigende eller synkende). Se punkt 5.1 for detaljer om hvordan de enkelte feltene sorteres.
**Tilpasse kolonnene**
Hvilke kolonner som vises i tabellene kan endres i dialogboksen *vis/skjul kolonner* (Vis -> Vis/skjul kolonner). Kolonnebredde og rekkefølgen på kolonnene endres ved "dra og slipp" på tabellheaderne. Valgene lagres slik at når man starter Bib-it er visningen den samme som ved forrige kjøring.
**Fargene** i tabellen indikerer obligatoriske, valgfrie og ignorerte felter (dette avhenger av valgt entrytype). Obligatoriske felter som ikke er fylt ut markeres med en rød ramme rundt cellen.

*Markering av rader og viewport view* har betydning for hvor godt brukeren klarer å orientere seg i bib-databasen. Når man bytter mellom to tabeller beholdes markert rad og viewport view i den enkelte tabellen. Dvs. at om man bytter fra hovedtabellen til søkeresultattabellen og tilbake igjen, vil viewport vinduet og den raden som er merket være de samme som forrige gang hovedtabellen ble vist.
Når et søk utføres (duplikatsøk eller vanlig søk) settes den aktuelle resultattabellen til valgt tabell i tabellpanelet (forutsatt at resultatet har 1 eller flere referanser).
*Antall referanser* i de ulike tabellene vises ved siden av tabellnavnet på hver tab.

I søkeresultattabellen og duplikatsøkeresultat-tabellen kan man endre dataene i referanser og slette referanser, men det er ikke mulig å legge til referanser. Når en referanse slettes i en av disse tabellene slettes den fra bib-databasen (og dermed fra hovedtabellen). Tabellene er statiske; det vil si at hvis brukeren endrer data om en referanse på en slik måte at denne referansen ikke lenger hører til i resultatet, forsvinner ikke raden av den grunn. For å få et oppdatert søkeresultat må man mao. utføre søket på nytt.

**Duplikatsøk**
Å presentere resultatet etter et duplikatsøk på en god måte viste seg å være en utfordring. Sammen med oppdragsgiver vurderte vi flere mulige alternativer.
For å få en totaloversikt over likheten mellom referanser i et sett referanser, kunne man presentere referansene i et diagram, som vist i figur 6



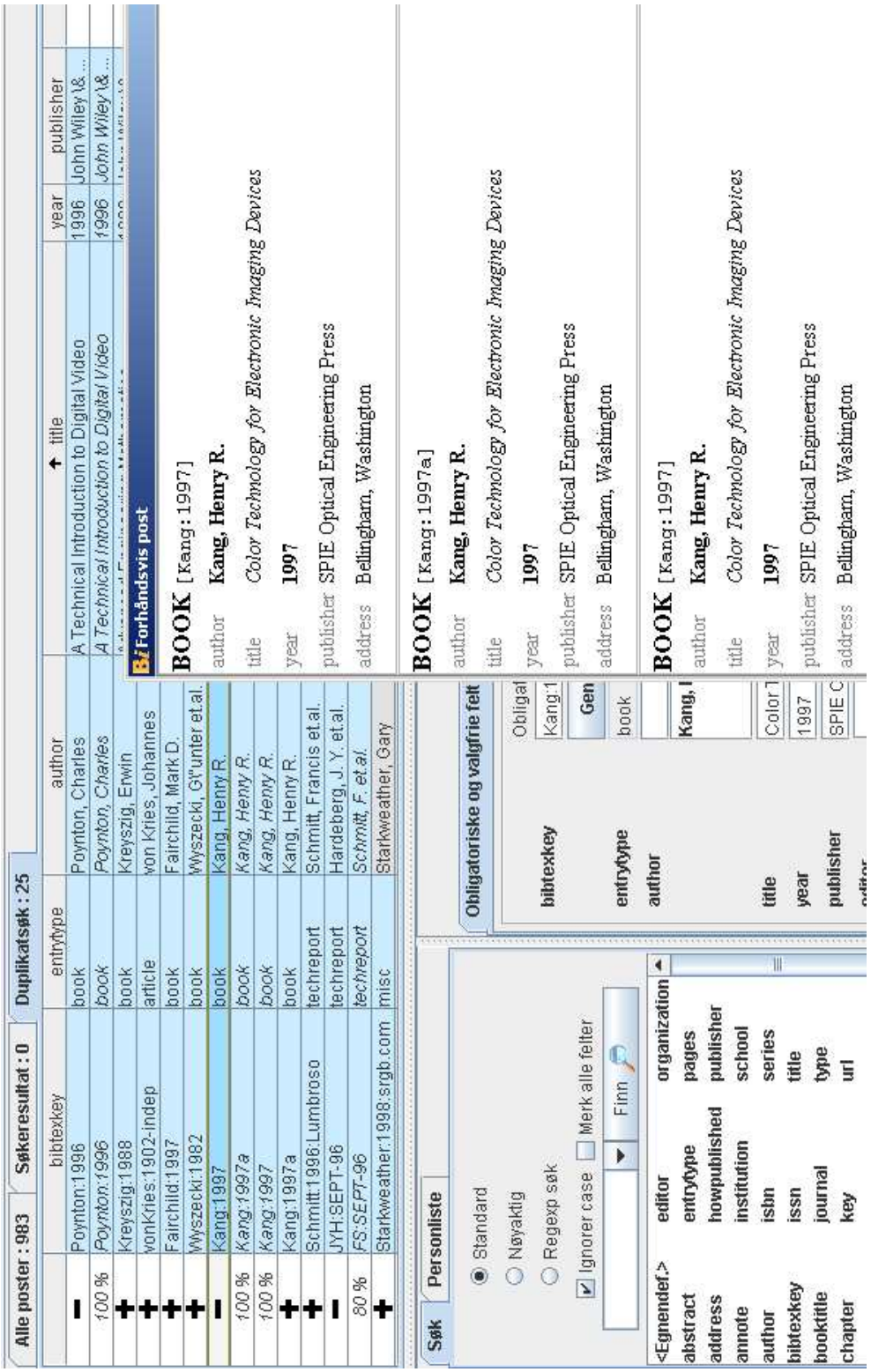

Figur 6: Duplikatsøkresultat illustrert som diagram

Her er det lett å visualisere mange-til-mange forbindelser mellom referansene. For å undersøke likheten nærmere kunne brukeren f.eks. merke en eller flere referanser og få opp en forhåndsvisning av disse i et annet panel eller vindu.
Dette bød imidlertid på relativt store programmeringstekniske utfordringer. Siden duplikatsøket er ikke en del av kjernefunksjonaliteten til Bib-*it*, ble vi enige med oppdragsgiver om å gå for en enklere GUI.

En annen mulighet er å presentere referansene i en eller annen form for liste. Men hvordan kunne vi få vist fram sammenhengen (mht. likhet) mellom referansene? Det sier seg selv at mange-til-mange-sammenhenger er vanskelig å få fram i en liste, men en-til-mange-sammenhenger, derimot, er mulig.
Vi testet ut forskjellige måter dette kunne gjøres på. Valget falt på en enkel "tre-tabell"-GUI-komponent; en tabell som kan vise fram grupper av beslektede referanser og hvor det er mulig å vise eller skjule referansene som ligger innunder en gruppe.

Ved å høyreklikke på en av referansene i gruppen og velge *Forhåndsvis* presenteres alle data om alle referansene i denne gruppen i et vindu slik at det er lett å se hva som er ulikt i referansene.

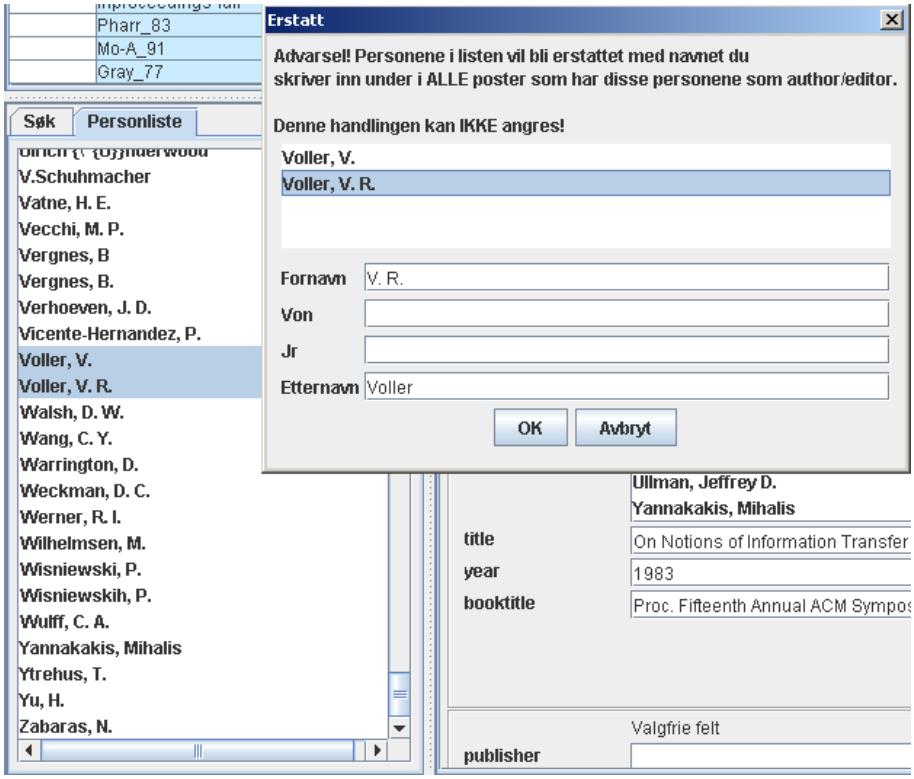
**Personlisten**
Personlisten (se figur 8) viser innholdet i det globale personregisteret, det vil si alle personer som er lagt til på en eller flere referanser i den åpne fila.




Figur 7: Bib-*it*s Duplikatsøk-GUI

Når et nytt navn legges til (som forfatter eller redaktør) på en referanse legges dette navnet til i det globale personregisteret hvis det ikke finnes der fra før. Når et navn slettes fra en referanse fjernes dette navnet fra det globale personregisteret kun dersom det ikke lenger er noen referanser som har denne personen som forfatter eller redaktør.

Det er ikke mulig å slette eller legge til personer direkte fra personregisteret. Skal man legge til en ny person gjør man det ved å legge den til PÅ en referanse og å slette en person fra personregistet vil bety at man ønsker å slette denne personen fra alle referansene den er lagt til på, og det gir ingen mening.




Figur 8: Det globale personlisten (sortert på navn) og dialogboks for å slå sammen flere personer til en.

*Drag and drop:* en eller flere personer kan velges i personlista og legges til på en referanse ved å dra navnene til forfatterlisten eller redaktørlisten til en referanse i skjemavisning.

*Slå sammen flere navn til ett og endre navn på person (se figur 8):*

- Ved å velge én person i personlsita, høyreklikke og velge *erstatt* dukker det opp en *erstattdialogboks*. Navnet på den valgte personen endres i tekstfeltene.

- Ved å velge to eller flere personer, høyreklikke og velge *erstatt* kommer *erstattdialogboksen* opp og brukeren kan velge et navn fra listen øverst og eventuelt endre dette navnet. Alle de valgte personene slås sammen til én person med det navnet brukeren oppga i tekstfeltene.

*Vise referanser skrevet av en eller flere personer:* Ved å velge en eller flere personer i personlista og klikke enter eller dobbelklikke, blir alle referanser som er skrevet av eller redigert av en eller flere av disse personene vist i søkeresultattabellen.

## 4.3 Domenelaget

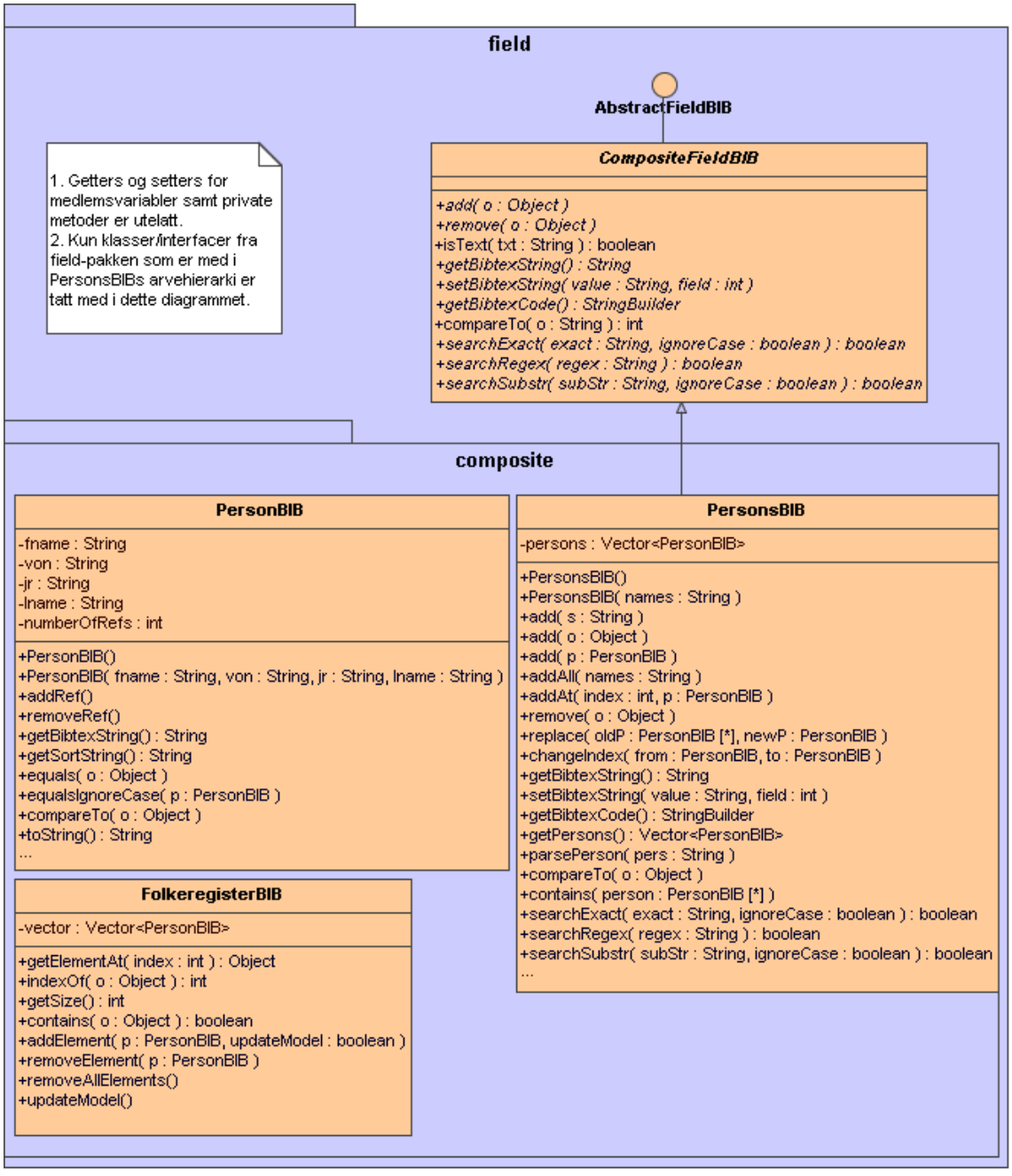
Vi brukte mye tid de første ukene på å finne fram til hvilke klasser domenelaget skulle bestå av og hvordan klassene skulle knyttes til hverandre. Domenelaget utgjør selve grunnmuren i en lagdelt applikasjon, og for å gjøre det mulig å bygge ut applikasjonen med ny funskjonalitet i senere inkrementer, er det viktig å få på plass et stabilt domenelag tidlig i utviklingen. Det var spesielt to faktorer det var viktig å ta hensyn til;

1. domenelaget må ha et grensesnitt som gjør det enkelt å koble nye grafiske eller tekstbaserte brukergrensesnitt til domenelaget.

2. datastrukturen skal være normalisert (se punkt 2.5.1 for en diskusjon rundt normalisert datastruktur)

Basert på kunnskaper om domenet og de to overstående faktorene kom vi fram til følgende datastruktur:

| Pakke | Klasse | Arver fra | Beskrivelse |
|---|---|---|---|
| no.hig.bibit.domain.entry | EntryBIB | | |
| | EntryTypeBIB | | |
| no.hig.bibit.domain.field | AbstractFieldBIB | Comparable | Representerer et felt i en bibtex-entry |
| | SimpleFieldBIB | AbstractFieldBIB | Representerer et felt som kan ha kun én verdi. |
| | CompositeFieldBIB | AbstractFieldBIB | Representerer et felt som kan ha mer enn én verdi |
| | FieldBIB | Comparable | |
| no.hig.bibit.domain.field.composite (se figur 9) | PersonsBIB | CompositeFieldBIB | Representerer et felt av typen "name" |
| | PersonBIB | | Representerer 1 person i et PersonsBIB-felt |
| | FolkeregisterBIB | | En samling av flere PersonBIB-instanser |
| no.hig.bibit.domain.field.simple | BibtexkeyBIB | SimpleFieldBIB | |
| | MonthBIB | SimpleFieldBIB | |
| | TextBIB | SimpleFieldBIB | |
| | TitleBIB | SimpleFieldBIB | |
| | YearBIB | SimpleFieldBIB | |
| no.hig.bibit.domain | EntryModelBIB | | |




Figur 9: Pakken domain::field::composite

### 4.3.1 Generisk datastruktur

Leser man en bok eller artikkel om BibTeX (for eksempel Mittelbach et al. [2004a]), kommer man fort borti begrepene *standard entrytypes*, *required fields* og *optional fields*, og man får gjerne presentert en oversikt over standard entrytyper og hvilke felter som er obligatoriske og valgfrie for hver av disse entrytypene. Men det er viktig å merke seg at det ikke er selve bibtex-programmet som bestemmer disse standard entrytypene og hvilke felter som er obligatoriske og valgfrie. Derimot er det BibTeX-stilfilen som brukes til å formatere en gitt bib-database som bestemmer hvilke entrytyper denne bib-databasen kan inneholde og hvilke felter som må eller bør være tilstede for en gitt entrytype.

Siden et av kravene til vår applikasjon var å lage en stilgenerator som gjør det enkelt for brukeren å lage sine egne BibTeX-stilfiler, var det naturlig å gjøre datastrukturen generisk slik at brukeren selv kan bestemme hvilke entrytyper og felter han eller hun vil bruke i en gitt bib-database.

Dette gjør også applikasjonen så fleksibel at den kan brukes til andre formål. Som et eksempel kan vi nevne at vi i utarbeidelsen av denne rapporten har benyttet oss av LaTeX-pakken Gloss – Gloss er en pakke som gjør det enkelt å lage ordlister ved å bruke BibTeX. Pakken er utviklet av Jose Luis Diaz og Javier Bezos. Ordforklaringene defineres i bib-databaser og bruker samme syntaks som bibliografioppføringene; eks:

```
@GD{gnu,
  word = {gnu},
  definition = {Extrange animal}
}
```

Ved å definere entrytypen *GD* og feltene *word*, *definition* og øvrige Gloss-felter i Bib-*it*s .ini-fil, kunne vi bruke Bib-*it* til å vedlikeholde ordlistefilen vår.

### 4.3.2 Domenelagets grensesnitt

EntryModelBIB er en samling av relaterte EntryBIB-objekter. I en applikasjon som bygger på Bib-*it*s bibliotek vil en åpen bib-database typisk representeres ved ett EntryModelBIB-objekt.
EntryModelBIB er et grensesnitt mot domenelaget - klassen inneholder metoder som kan kalles for å få utført typiske operasjoner mot dataene. De viktigste operasjonene er:

- legge til/slette referanser
- endre verdien i felter
- søke etter referanser som oppfyller gitte søkekriterier

- søke etter mulige duplikate referanser
- lese data fra fil og skrive data tilbake til fil (i bibtex-format)
- sortere etter et hvilket som helst felt
- generere en unik bibtexkey for en gitt referanse

En viktig operasjon som ikke er tigjengelig gjennom EntryModelBIB, er stilgenerering. Stilgeneratoren er helt uavhengig av en eventuell bib-database som måtte være åpen i øyeblikket, og fungerer derfor omtrent som et eget program som kjøres fra BIB-*it*.

## 4.4 Parseren

BIBTEX har flere lovlige varianter av syntaksen. Man kan f.eks. velge om man vil bruke parenteser `( )` eller "krøllparenteser" `{ }` rundt feltene i en referanse. Å lage en parser som håndterer mange forskjellige parenteser er ikke helt enkelt men å lage en oversetter som konverterer forskjellig syntaks til en enkelt stil er mye lettere. Og når man har en konsistent syntaks er det enklere å lage en parser for den syntaksen. Derfor bestemte vi oss for å dele parseren i to deler som vi kalte CleanerBIB og ParserBIB.

**En liten "omvei" via C++**

Vi skrev Cleaner-programmet opprinnelig i C++ og kjørte det fra parseren som en prosess. Vi gjorde dette mest som et eksperiment for å se om det var mye raskere enn en tilsvarende Java implementasjon og fordi det var interessant å prøve å kjøre en prossess i Java, men vi valgte etterhvert å skrive programmet om til Java fordi det ble enklere å gjøre programmet plattformuavhengig da. Det viste seg at C++-implementasjonen sammenlignet med Java-implementasjonen faktisk ble noe raskere, men det var bare snakk om noen få millisekunder forskjell på en stor fil (og en treg maskin). Når selve parsingen av den samme filen tar flere sekunder og Cleaner i seg selv bare bruker sjelden bruker mer enn noen få 100 millisekunder fant vi ut at det ikke var stor å spare på å bruke en komplisert C++ og Java løsning.

Vi valgte etterhvert å alltid kjøre Cleaner, selv om filer som BIB-*it* har skrevet tidligere kunne parses uten at cleaner var kjørt.

## 4.5 Duplikatsøk

### 4.5.1 Ulike alternativer

**Sammenligning av feltverdier**

Når man jobber med samme bib-database over lengre tid, og spesielt om man redigerer

filen i en teksteditor hvor man har begrensede muligheter til søking og sortering, er det fort gjort å legge inn samme referanse to ganger. Det skal godt gjøres at man skriver inn referansen nøyaktig likt hver gang - sannsynligheten er stor for at man ikke har fylt ut akkurat de samme feltene eller at innholdet i to motsvarende felter er litt forskjellig.

Ulik bruk av store og små bokstaver, tegn og mellomrom og skrivefeil kan forekomme, men kanskje mer vanlig er det å skrive inn selve informasjonen på forskjellig måte fra gang til gang. Av og til kan det være vanskelig å vite om man skal bruke både hovedtittel og undertittel i tittelfeltet for en bok eller om man bare skal bruke hovedtittelen.
En artikkel fra tidsskriftet *Computer Graphics, 1997, nr 2* forekommer som to ulike referanser i en av Nelson Beebes bib-databaser

1. "Artist's View: Computer Animation and Theatre of the Absurd: Some Thoughts"
2. "Computer animation and theatre of the absurd: some thoughts"

En inproceeding fra *The ACM digital library* forekommer som to ulike referanser i samme bib-database

1. "Cellular growth: Wriggon"
2. "Cellular growth: Fossy"

I begge eksemplene er det kun tittelen skiller de to referansene fra hverandre. Hvordan skal en datamaskin kunne vite at referansene i det første eksemplet er duplikate, mens referansene i eksempel 2 ikke er det?

Utgivere skrives av og til forkortet og andre ganger fullt ut. Bibtex-formatet er dessuten veldig fritt med hensyn til hva som skal skrives inn i feltene. Alle felter behandles som tekstfelter, og selv om det har blitt forsøkt å lage regler for hvordan enkelte felter skal fylles ut, varierer det veldig hvor nøye hver enkelt bib-database-forfatter følger disse reglene.

Når man tar med i betraktningen at mange bib-database-forfattere ikke skriver inn alle referansene sine selv, men kanskje kopierer fra en kollega sin bib-database eller kopierer referanser fra nettet, vil et duplikatsøk som baserer seg på eksakt-likhet-sammenligning av felter overse mange par av referanser som i praksis er duplikater.

Et alternativ til å bruke eksakt-likhet-sammenligning er en sammenligning hvor likheten mellom de to feltene som sammenlignes angis i prosent. Etter litt testing kom vi fram til at denne typen sammenligning ville fungere best for vårt formål. I avsnittet 5.4 beksriver vi hvordan vi implementerte dette i Bib-*it*.

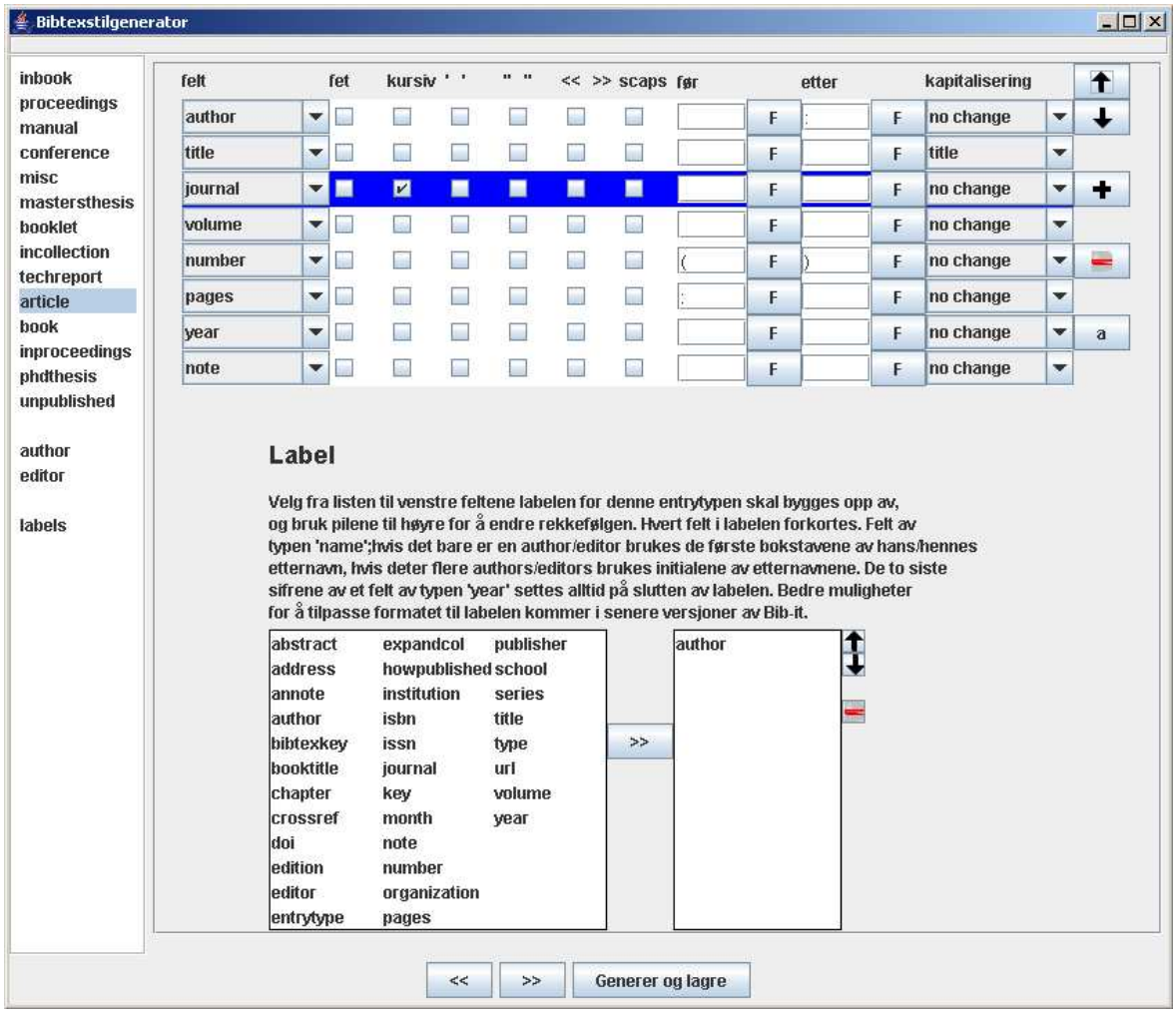
**Hvilke felter skal brukes i sammenligningen?** Hvor godt en duplikatsøkalgoritme fungerer er ikke bare avhengig av hvordan to felter sammenlignes, men selvfølgelig også av hvor mange og hvilke felter som sammenlignes. Her fikk vi en utfordring i forhold til den generiske måten BIB-*it* er bygd opp på (dvs. at felt og felttyper bestemmes dynamisk ut fra innholdet i xxxStyle-filen som brukes).

For å avgjøre ved visuell inspeksjon om en referanse A og en referanse B er duplikate, er det naturlig og først se på entrytypen, deretter finne ut hvilke felter som er sentrale for denne entrytypen, og sjekke hvor like disse feltene er i A og B. Hvis både A og B har entrytype "article" kan man med stor sikkerhet si at de er duplikate hvis de er fra samme tidsskrift, samme nummer og årgang av tidsskriftet og tillegg befinner seg på samme sidetall. For entrytypen "book" vil det være naturlig å se på tittel, forfatter, årstall og utgiver, og hvis alle disse feltene er helt like (eller nesten helt like) kan man anta at A og B er duplikater.

Feltene man ser på for en gitt entrytype samsvarer godt med hvilke felter som er obligatoriske i denne entrytypen. Vi tenkte derfor at det kunne være naturlig å ta utgangspunkt i entrytypen og la denne avgjøre hvilke felter som skal sammenlignes (vha. required-vektoren i EntryTypeBIB eller angi i xxxStyle.ini-filen hvilke felter som skal brukes som sammenligningsfelter for hver entrytype). Da kunne vi først sortere vektoren med alle referansene etter entrytype, og for hver referanse sammenligne denne med hver av de andre referansene av samme entrytype.

Vi ville på denne måten kunne gjøre duplikatsøket generisk i tråd med oppbygningen av BIB-*it* ved å lage feltsammenlignings-algoritmer basert på typeinformasjonen i xxxStyle.ini-filen og plassere disse i subklassene til AbstractFieldBIB.

Etter å ha gått denne ideen nærmere i sømmene, fant vi imidlertid ut at den hadde sine **begrensninger**. Noen av de standard entrytypene i bibtex er ikke helt klart definerte, dvs. at man i noen tilfeller kan være i tvil om hvilken entrytype man skal bruke (for eksempel brukes techreport og inproceeding mye om hverandre). En annen sak er at om bib-databasen er redigert med en vanlig teksteditor er det fort gjort at det sniker seg inn skrivefeil i entrytypen slik at den av BIB-*it* vil bli betraktet som en egen entrytype og ikke vil bli sammenlignet med de referansene som egentlig har samme entrytype. For å komme rundt disse problemene kunne man selvfølgelig sammenligne hver referanse med hver av alle de andre referansene (uavhengig av entrytype). Men dette er for det første veldig kostbart mht. søketid. I en funksjon som ikke brukes så veldig ofte (i praksis kanskje en gang i uka) er ikke kort søketid det viktigste, men det er uansett lite attraktivt å måtte vente i flere minutter på et søk. For det andre; hvis en techreport skal sammenlignes med en inproceeding, hvordan skal man avgjøre hvilke felter som skal sammenlignes?




Figur 10: Stilgeneratoren: formateringsvalg for entrytypen 'article'

### 4.5.2 Løsningen

Det vi endte opp med var en duplikatsøkalgoritme som bruker de samme sammenligningsfeltene på alle entrytyper. Feltene som brukes i sammenligningen er year, author, title og journal. På alle felttyper som er subklasse av SimpleFieldBIB (title og journal) bruker vi Levenshtein distance, og på felt av typen "name" (author) bruker vi vår egen feltsammenligningsalgoritme.

## 4.6 Stilgeneratoren

Da vi i fjerde inkrement begynte å tenke på hvordan vi skulle løse oppgaven med å lage en stilgenerator, var det naturlig å undersøke om andre hadde gjort noe lignende tidligere.

Mittelbach et al. [2004a] beskriver CustomBib (se punkt 2.2.5 for en beskrivelse av CustomBib), og vi bestemte oss for å undersøke dette systemet nærmere. Vårt hovedinntrykk var at CustomBib tilbyr en raskere og enklere måte å lage egne BibTeX-stilfiler på sammenlignet med å måtte lære seg å bruke stakkespråket. Men problemet med CustomBib er at det ikke er så lett tilgjengelig for brukere som kun er vant til grafiske brukergrensesnitt og ellers har liten erfaring med mer avansert bruk av en datamaskin (og det er også gjerne denne gruppen av brukere som har største behov for et alternativ til å skrive bst-filer selv).

**Noe av det vi opplevde som problematisk med CustomBib var:**

1. En konsollbasert veiviser er vanskelig og tungvint å bruke; for å generere en stil med CustomBib er man i praksis nødt til å på forhånd sette seg ned med papir og blyant og ut fra egenskapene til den formateringen man ønsker gå gjennom hvert valg i CustomBib-veiviseren for å finne ut hva man skal svare på de ulike alternativene. Det er svært vanskelig å se for seg underveis hvordan bibliografilisten blir seende ut.

2. Spørsmålene må besvares sekvensielt. Det er dermed ikke mulig å gå tilbake å se hva man svarte på et tidligere spørsmål eller endre svar på et tidligere spørsmål. (Men det er riktignok mulig å endre svaralternativene etter at veiviseren er ferdig, ved å endre dbj-filen manuelt.

3. For å kunne svare på enkelte av spørsmålene må man være kjent med typiske BibTex-begreper (som f.eks. *Reference component tags*)

4. Alle svaralternativene er hardkodet, noe som begrenser fleksibiliteten. Et eksempel:

   ```
   JOURNAL VOLUME: (def) Volume plain
   vol-it,% Volume italic
   vol-bf,% Volume bold
   vol-2bf,% Volume and number bold
   ```

   Hva om brukeren isteden ønsker *Volume and number italic*?

5. Spørsmålene og svaralternativene er kun presentert med beskrivende tekst. Dette krever mye finlesing de første gangene man bruker veiviseren.

6. Entrytyper og felter er hardkodet, noe som gjør det umulig å få generert en stil som bruker et felt man f.eks. har definert selv (den eneste måten å få til dette på er å gå inn i .bst-filen etterpå og redigere den manuelt).

7. Rekkefølgen feltene skal skrives ut i (i referanselisten) kan ikke velges fritt. Eksempler:

```
POSITION OF VOLUME AND SERIES FOR INCOLLECTIONS: (def) Series and
volume after the editors
ser-ed,%: Series and volume after booktitle
```

og

```
POSITION OF NUMBER AND SERIES: (def) After chapter and pages
numser,%: Just before publisher
```

For å finne ut mer om CustomBib og hvor godt dette systemet fungerer til å lage egne stiler, lagde vi et enkelt grafisk brukergrensesnitt mot CustomBib. Vi ville også undersøke i hvilken grad en GUI-veiviser kan gjøre et opprinnelig TUI-system mer brukervennlig og dermed mer anvendelig, for å kunne avgjøre om vi skulle lage en stilgenerator fra bunnen eller om vi rett og slett skulle satse på en GUI-tilpasning av CustomBib.

Fordelene med en slik GUI er at man får mulighet til å svare på spørsmålene i den rekkefølgen man ønsker, evt. se alle spørsmålene på en gang, samt at spørsmålene kan illustreres bedre ved bruk av bilder og/eller tekst som er formatert på en mer avansert måte enn man får til i en TUI.

Til tross for dette fungerte ikke denne GUI-tilpasningen av CustomBib noe særlig godt som en GUI-veiviser. Veiviseren fikk et noe "tekstbasert preg". Den eneste naturlige måten å presentere svaralternativene på var i kombobokser med tekst, slik at et spørsmål med en gruppe svaralternativer i CustomBib presenteres i GUIen som én komboboks. Spørsmålene (komboboksene) ble orgranisert i grupper med en gruppe for hver side i veiviseren. Resultatet ble en veiviser på ca. 12 sider (noe som er i overkant mye), med mellom 2 og 10 kombobokser på hver side.

Etter å ha testet ut denne GUI-tilpasningen av CustomBib, ble vi i samråd med oppdragsgiver enige om å forsøke å lage vår egen stilgenerator.

### 4.6.1 Datastruktur

Et av problemene vi opplevde i forbindelse med BibTeX-stiler generelt og eksisterende hjelpemidler for å lage egne stiler (f.eks. CustomBib) er at de er preget av en *hardkodings*- og *klipp og lim*-tankegang, noe som begrenser fleksibiliteten og gjør det vanskelig å gjøre endringer.

CustomBib sin masterstilfil (se punkt 2.2.5) inneholder koden for *alle* BibTeX-stilfilene som er mulig å få generert med dette systemet, og en ny stilfil genereres ved at man plukker ut de delene av masterfilen som tilsvarer valgene man har gjort i veiviseren.

Et annet eksempel er filen btxbst.doc, som er velkjent av de fleste som lager egne BibTeX-stilfiler enten ved å endre på eksisterende stilfiler eller skriver koden selv. Dette er en template-fil som bygger på plain.bst, unsrt.bst, alpha.bst og abbrv.bst og som i tillegg inneholder forklarende dokumentasjon og stakkekoden oversatt til Pascal-lignende pseudokode. Ved å kjøre filen gjennom en C-preprosessor kan man få generert diverse forskjellige stilfiler (eks.: `cpp -P -DPLAIN btxbst.doc plain.txt` gir standardstilfilen plain.bst som output), men først og fremst er denne template-filen kanskje nyttig for å lage egne stilfiler ved å klippe og lime stakkekode.

**Ideen bak Bib-*it*s stilgenerator**

Under utviklingen av stilgeneratoren forsøkte vi derfor å tenke *felter* og *entrytyper* istedenfor *feltet tittel* og *entrytypen article*. Hovedtanken bak generatoren er at brukeren skal kunne

- oppgi et sett med entrytyper og et sett med felter
- for hver entrytype legge til de feltene han vil skal skrives ut i referanselisten
- endre rekkefølgen på feltene
- bestemme hvordan hvert felt i hver entrytype skal formateres
- angi eventuelle skilletegn før og etter hvert felt (f.eks. mellomrom, punktum, komma, eller tekster som "side" eller "utgitt av") og eventuelt angi hvordan disse strengene skal formateres

# 5 Implementasjon

## 5.1 Felttyper

Siden vi ønsket å

1. gjøre det enklere og kjappere for brukeren å redigere informasjonen i en referanse med Bib-*it* enn det som er mulig med en teksteditor, og;

2. la brukeren få en opplevelse av at bibtex-formatet er normalisert (se punkt 2.5.1 for en diskusjon rundt normalisert datastruktur)

var det nødvendig å innføre noen felttyper. Et felt kan enten ha en enkelt verdi (*SimpleFieldBIB*) eller være sammensatt av flere enkeltverdier (*CompositeFieldBIB*).
Felttypene bestemmer først og fremst hvordan et gitt felt skal parses og hvordan det skal sorteres, men er også sentralt for grafiske brukergrensesnitt hvor man kanskje vil presentere feltene på forskjellig måte avhengig av hvilken type de har.

Det måtte litt eksperimentering med bibtex til før vi fant ut hvilke felttyper som var aktuelle. En sterk føring her lå i hvordan bibtex behandler felter og hvilke regler (formelle og uformelle) som finnes for hva slags verdier et felt kan inneholde. Av bibtex betraktes de fleste feltene som vanlige tekstfelter; unntaket er at man har en implisitt definert type *name* (stakkespråket som BibTeX-stilfilene skrives i har en innebygd funksjon format.name$ som tar som parameter en streng som representerer et navn, og det finnes regler (BibTeX name format) for hvordan denne strengen skal bygges opp.
Utover dette er det den enkelte stilfilen som eventuelt legger begrensninger på hva slags verdier et felt kan inneholde (for eksempel krever de fleste standardstilene at year-feltet inneholder et årstall (fire sifre), som skal stå helt tilslutt).

Løsningen ble et kompromiss mellom veldig sterke typer og kun en teksttype. Typene som er definert i .ini-filen samsvarer direkte med subklassene av *CompositeFieldBIB* og *SimpleFieldBIB*, og man kan selvfølgelig (ved en senere utvidelse av Bib-*it*) definere flere eller andre typer ved å lage klasser som arver fra en av de to abstrakte klassene beskrevet over.

## 5.2 CleanerBIB

CleanerBIB leser en bib-database tegn for tegn, og formaterer fila slik at alle `" "` og `( )` blir endret til `{ }`. Hvert felt ligger på egen linje (untatt konkatenerte verdier der hver del etter `#` tegnet ligger på en egen linje).
Siden Cleaner kjøres hver gang bygde vi etterhvert om parser-pakken slik at cleaner skriver en avart av BibTeX som ikke er helt standard. Noen av disse endringene er blant annet

at kommaene fjernes (de er unødvendige når feltene alltid står på en egen linje) og at konkatenerte felt deles. (Se eksemplet nedenfor.)

**En eksempelfil:**

```
En eksempelfil (dette er en kommentar). @comment{   Dette er en
annen kommentar.} @stRing{ enStringDef="Dette er en forkortelse" }

@artICle(key1, AUthor = "Jaan Quees", TITLE = {The dutch man},
journal= 32 # "Nature" # enStringDef
                    , year = 1992)
```

**Etter at CleanerBIB har kjørt:**

```
En eksempelfil (dette er en kommentar). @comment{ Dette er en annen
kommentar. }

@stRing{ enStringDef={Dette er en forkortelse} }

@artICle{key1
 AUthor = {Jaan Quees}
 TITLE = {The dutch man}
 journal= 32 #
 {Nature} #
 enStringDef
 year = 1992
}
```

Feltet `journal` består i eksempelet av tre verdier(et tall, en vanlig tekststreng og en forkortelse) som er konkatenert med `#`. Cleaner deler feltet slik at parseren lett kan kjenne igjen konkateneringer og parse dem.

## 5.3 Stilfilene

Før vi kan beskrive parseren må vi ta for oss "stilfilene". Stilfilene er er `ini`filer som inneholder data om hvordan Bib-*it* skal lese bib-databasene. En stilfil har tre deler med følgende obligatoriske overskrifter: `[Special]`, `[Fields]` og `[EntryTypes]`.

Under `[Special]` er en liste med typenavn på oppføringer som starter med `@` men ikke skal oppfattes som en referanse. I standard BibTeX[8] er dette oppføringene: `string` som

[8] Med *Standard BibTeX* mener vi opførselen til de fire standard stilene som er utgangspunktet for Bib-*it*stilfilen plainStyle.ini.

er en forkortelsesdefinisjon, `preamble` som inneholder LaTeXmakroer og `comment` som er en kommentar. Etter hver spesielle oppføring står det et likhetstegn og en bokstav som i standard stilfilen `plainStyle.ini` (Se vedlegg C):

```
[Special] string   = a preamble = k comment  = k
```

**a** står for "abbrevation" altså forkortelse. En forkortelse parses som en omtrent som en vanlig oppføring.

**k** står for "keep it" dvs. at feltet skal leses (men ikke parses) og lagres når fila evt blir lagret igjen.

**i** står for "ignore" dvs. at feltet skal ignoreres slik en kompilator ignorerer kommentarer. Tidligere var sto det `comment = i`, men vi endret det fordi vi fant ut at det var litt drastisk å fjerne alle kommentarer sånn helt uten videre.

## 5.4 ParserBIB

Det første parseren gjør når den skal lese en fil er å lage en midlertidig `tmp` fil som CleanerBIB kan skrive til. Parseren leser linje for linje til den finner en linje som begynner med `@` dvs. starten på en referanse. Alt utenom referansene blir lagret i en array med strenger som senere kan leses og redigeres under navnet "filhode". Hvordan parserene oppfatter hva som er "utenfor" eller "inni" en referanse ved syntaks feil kan variere noe.

Vår målsetting med parseren er at mest mulig skal aksepteres uten at parsingen skal avbrytes. Parseren vil stortsett bare gi feilmeldinger og avbryte seg selv når det er stor fare for at å fortsette vil ødelegge data.[9]

Hvis et felt mangler likhetstegn mellom navnet og verdien vil navnet og verdien føyes sammen med navnet på neste felt, dermed får man et "selvdefinert felt" med et høyst uvanlig navn, men det vil ikke komme noen feilmelding (BibTeX godtar også selvdefinerte feltnavn). Hvis det ikke er noe felt etterpå, eller det er noen andre feil, vil all slik tekst som ikke blir oppfattet som et lovlig felt av parseren lagt inn i et selvdefinert felt med navnet "UgyldigeFelt"[10], og parseren vil gi en advarsel om ugyldig felt i en post.

[9] En slik "fatal feil" er hvis en forkortelse blir redefinert. Parseren kan ikke avgjøre om dette er meningen eller om å fortsette parsingen vil kunne føre til at noen poster får feil innhold.

[10] Navnet er forskjellig avhengig av språkinstillingene.

Ved en del syntaksfeil (en sluttparentes for mye kan være nok) kan det hende at parseren tror referansen er ferdig før den egentlig er det, resten blir lagret i filhode teksten som kommentarer. Her blir det heller ikke noen feilmelding, fordi parseren naturligvis godtar kommentarer uten protester. Første gang man leser inn en fil som ikke BIB-*it* har skrevet, bør man ta en titt på filhodet og sjekke at alt som står der skal være der. Man bør også se etter ukjente poster med ukjent "entrytype"[11] som ofte er et tegn på at noe har gått galt ved parsingen.

## 5.5 Duplikatsøk

### 5.5.1 Datastruktur

Valget av GUI for framvisning av resultatet etter et duplikatsøk (se punkt 4.2.1) ga visse føringer for datastrukturen. Vi definerte noen referanser som "far", og de referansene som hadde en viss likhet med en far ble "barn" av denne faren. En far er en instans av ComparedEntryBIB-klassen, og et barn en instans av EntryBIB (se figur 11).

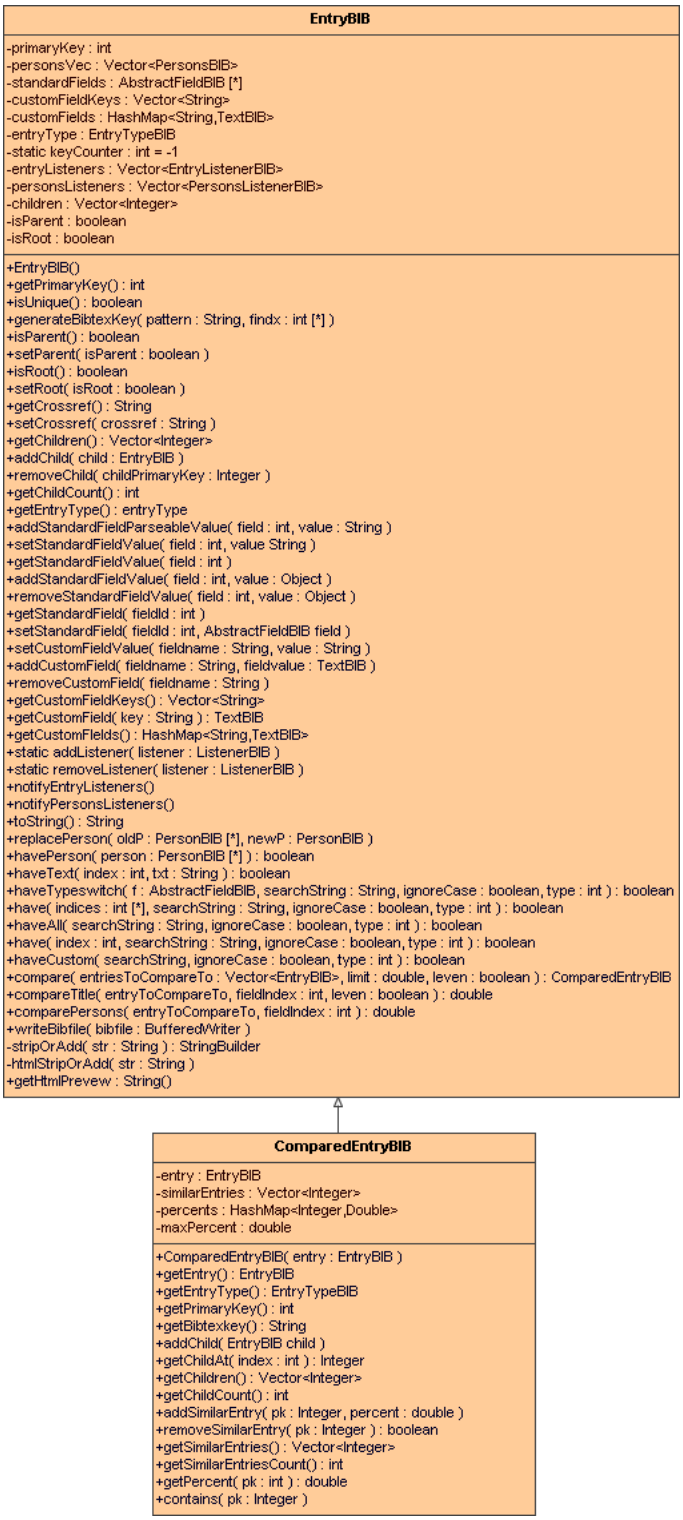
### 5.5.2 Duplikatsøkalgoritmen

Duplikatsøkalgoritmen itererer gjennom en vektor med referanser. Hver referanse pakkes inn i et ComparedEntryBIB-objekt (A) og sammenlignes med alle de andre referansene i vektoren. Hvis likheten mellom A og referansen den sammenlignes med (X) er større enn en forhåndsbestemt grenseverdi, legges X til i As vektor av "referanser som ligner på meg". Hastighetsproblemer (ved store bib-databaser) førte imidlertid til at vi måtte gjøre noen optimaliseringer av denne algoritmen.

#### Sammenligne kun referanser med samme årstall

Feltet "year" er obligatorisk for de standardentrytyper som er mest brukt (article, book, inbook, incollection, inproceedings, masterthesis, phdthesis, proceedings og techreport). I tillegg krever alle standard bibtexstiler at de fire siste tegnene i dette feltet er tall. I BIB-*it* er et felt av typen "year" bygd opp av en streng etterfulgt av et heltall. For å gjøre duplikatsøkalgoritmen raskere sorteres først alle referansene etter 'year'-feltet, og kun de referansene med samme årstall (dvs. hvor 'year'-delen av 'year'-feltet er likt; strengdelen ignoreres) sammenlignes. Dette har selvsagt den bieffekten at duplikatsøket aldri vil foreslå to referanser som er ganske like bortsett fra at de har forskjellig årstall som mulige duplikater. Men siden vi la vekt på å lage en duplikatsøkalgoritme som fungerer så bra som mulig for de aller fleste tilfeller (i praksis er det en svært liten andel av referansene i en bib-database med tomt 'year'-felt), vurderte vi denne bieffekten som nærmest ubetydelig opp mot en betraktelig raskere duplikatsøkalgoritme.

[11] Ukjente sorteres før kjente typer så det er lett å finne dem.




Figur 11: EntryBIB og ComparedEntryBIB

Duplikatsøkalgoritmen er implementert i BIB-*it* som metoden *getDuplicateEntries* i klassen domain.EntryModelBIB:

```
/**
* Retrieves possible duplicate entries from this EntryModel. All entries in
* the model are sorted (descending) on the first field of type year. The
* first entry ('entry') is extracted and all other entries with the same
* year ('cmpEntry') are compared in turn to entry, using EntryBIB's
* compare-method.
* If the equality between 'entry' and 'cmpEntry' is greater than limit,
* cmpEntry is added to 'entry'-s similarEntries vector. Then the second
* entry is extracted from the model, and compared to all entries with the
* same year except the first entry in the model... When all entries i
* EntryModel have been compared to all other entries with the same year, a
* vector of the entries in EntryModel that has more than 0 similarEntries
* is returned.
*
* @since     0.5
* @param     limit
* @see       no.hig.bibit.domain.entry.EntryBIB#compare
* @return    a vector of CompareEntryBIBs.
*/
public Vector<EntryBIB> getDuplicateEntries(double limit,boolean leven){
    Integer intYearField = InitBIB.getIndicesFromClassname("year").get(0);
    int yearField = 0;

    if(intYearField != null) yearField = intYearField.intValue();
    else return null;

    EntryModelBIB tmpModel = new EntryModelBIB(getEntries());
    Vector<EntryBIB> tmpEntries = tmpModel.getEntries();
    Vector<EntryBIB> comparedEntries = new Vector<EntryBIB>();
    int j;
    int year2=0;
    EntryBIB cmpEntry = null;

    tmpModel.sortData(yearField,false,false);

    for(int i=0; i<tmpModel.getRowCount(); i++){
        EntryBIB entry = tmpEntries.get(i);
        AbstractFieldBIB field = entry.getStandardField(yearField);
        int year1 = ((YearBIB)field).getYear();

        Vector<EntryBIB> entriesToCompare = new Vector<EntryBIB>(15);

        if(year1 != 0){
            j=i+1;

            if(j < tmpModel.getRowCount()){
                cmpEntry = tmpEntries.get(j);
                field = cmpEntry.getStandardField(yearField);
                year2 = ((YearBIB)field).getYear();
            }else year2 = -1;

            while(year1 == year2){
                if(cmpEntry != null){
                    entriesToCompare.add(cmpEntry);
                }
                j++;

                if(j < tmpModel.getRowCount()){
                    cmpEntry = tmpEntries.get(j);
```

```
60                      field = cmpEntry.getStandardField(yearField);
61                      year2 = ((YearBIB)field).getYear();
62                  }else year2 = -1;
63              }
64              ComparedEntryBIB comparedEntry =
65                   entry.compare(entriesToCompare,limit,leven);
66
67              if((comparedEntry != null) &&
68                  (comparedEntry.getSimilarEntriesCount() > 0))
69                  comparedEntries.add(comparedEntry);
70          }
71      }
72      return comparedEntries;
73  }
```

På linje 62 kalles EntryBIBs *compare*-metode som finner feltene som skal brukes i sammenligningen, oppretter et ComparedEntryBIB-objekt som wrappes rundt dette EntryBIB-objektet og sammenligner dette EntryBIB-objektet med alle EntryBIB-objektene i entriesToCompareTo ved å kalle EntryBIB-metodene *compareTitle* (for felt av typen tittel) og *comparePersons* for felt av typen navn. Disse metodene returnerer en *double* mellom 0.0 og 1.0 som angir likheten mellom de to referansene som ble sammenlignet. Alle delprosentene ganges sammen for å få en totalprosent for likheten mellom to referanser. Hvis denne totalprosenten er større enn eller lik *limit*, legges referansen fra sammenligningsvektoren til i vektoren *similarEntries* i ComparedEntryBIB-objektet.

### Sammenligning av felter

I vårt første forsøk på duplikatsøkimplementasjon laget vi en enkel feltsammenligningsalgoritme som gikk ut på å finne likheten mellom to feltverdier ved å sammenligne felter på "ordnivå" - dele opp feltverdiene i ord-arrayer og finne ut hvor mange eksakt like ord de to ord-arrayene som sammenlignes har. Denne algoritmen ble brukt på tittelfelter.

For felter av typen "name" ble likheten regnet ut på en tilsvarende måte, men istedenfor å bruke ordarrayer ble vektoren av PersonBIB-objekter i de to feltene brukt, og likheten ble bestemt hvor mange eksakt like personer de to feltene som sammenlignes har (ved å bruke equals-metoden i PersonsBIB).

Denne algoritmen viste seg å ikke være spesielt rask, og den fungerer heller ikke særlig bra på felt med mye skrivefeil ("referanse" vil betraktes som like ulik "referanes" som "høyfjellshotell").

Vi vurderte også muligheten for å lage en algoritme som sammenligner en og en bokstav, og fant en veletablert og mye brukt algoritme som gjør akkurat dette; nemlig Levenshtein distance (se vedlegg B for en beskrivelse av algoritmen). På nettet fant vi en

java-implementasjon av denne algoritmen[12] som vi kunne bruke i BIB-*it* uten endringer. Koden under viser hvordan begge disse to alternativene er implementert i metodene *compareTitle* og *comparePersons*.

```
/**
* Compares one of this entry's title fields (identified by fieldindex) to
* the correspondig title field in entryToCompareTo. If leven is true
* Levenshteins algorithm is used by calling Distance's LD method to
* get the number of steps required to transform title1 to title2, then
* 1 - (numSteps / size of the longest string) gives the equality percent.
* If leven is false a simpler algorithm is used; title1 and title2 are
* splitted into word arrays words1 and words2
* (divided by either spaces or -'s), step is set to 0.5,
* each word in the longest word-array which is also
* contained in the shortest word-array adds (step / length of the longest
* word array) to the equality percent, each word in the longest word-array which
* is NOT contained in the shortest word-array subtracts (step / length of the
* longest word array) from the equality percent.
*
*
* @param        entryToCompareTo  the entry to compare to this entry
* @param        fieldIndex        identifies the field to be compared
* @param        leven             true if the Levenshtein algorithm should
*                                 be used in the comparision, false otherwise
* @see          #compare
* @see          no.hig.bibit.domain.Distance#LD(String,String)
* @return       the equality percent between the to fields which have been
*               compared
*               (1.0 if the field values are equal, 0.0 if the field values are
*               not equal at all).
*
*
*/
public double compareTitle(EntryBIB entryToCompareTo,int fldInd, boolean leven){
        double percentAcc = 0.5;
        double step;

        String title1 = getStandardFieldValue(fldInd).trim();
        String title2 = entryToCompareTo.getStandardFieldValue(fldInd).trim();

        if(leven){

                //using levenshtein
                Distance dist = new Distance();
                int numSteps = dist.LD(title1,title2);

                int size = 0;
                if(title1.length() >= title2.length()) size = title1.length();
                else size = title2.length();

                if(size == 0) return 1.0;

                double percent = 1.0 - ((double)((double)numSteps/(double)size));

```

[12] http://www.merriampark.com/ld.htm#JAVA
Siden Michael Gilleland ikke har oppgitt lisensinformasjon på denne siden, kontaktet vi han på mail og fikk tillatelse til å bruke koden.

```
51                  return percent;
52
53          }else{
54                  String[] words1 = title1.split("␣");
55                  String[] words2 = title2.split("␣");
56
57                  //TODO hvordan skal uutfylte felt telle???
58                  if((words1.length < 1) || (words2.length < 1)){
59                          return 1.0;
60                  }
61
62                  //e.g: j-AEROSPACE-AMERICA, j-AM-PROG seems to be normal
63                  //journal names (java.bib)
64                  if(words1.length == 1){
65                          words1 = title1.split("-");
66                  }
67                  if(words2.length == 1){
68                          words2 = title2.split("-");
69                  }
70
71                  Vector<String> vecWords1 = new Vector<String>();
72                  for(int i=0; i<words1.length; i++){
73                          vecWords1.add(words1[i]);
74                  }
75
76                  Vector<String> vecWords2 = new Vector<String>();
77                  for(int i=0; i<words2.length; i++){
78                          vecWords2.add(words2[i]);
79                  }
80
81                  if(vecWords1.size() > vecWords2.size()){
82                          step = 0.5 / vecWords1.size();
83                          for(String word : vecWords1){
84                                  if(vecWords2.contains(word)) percentAcc+=step;
85                                  else percentAcc-=step;
86                          }
87                  }else{
88                          step = 0.5 / vecWords2.size();
89                          for(String word : vecWords2){
90                                  if(vecWords1.contains(word)) percentAcc+=step;
91                                  else percentAcc-=step;
92                          }
93                  }
94
95                  return percentAcc;
96          }
97  }
```

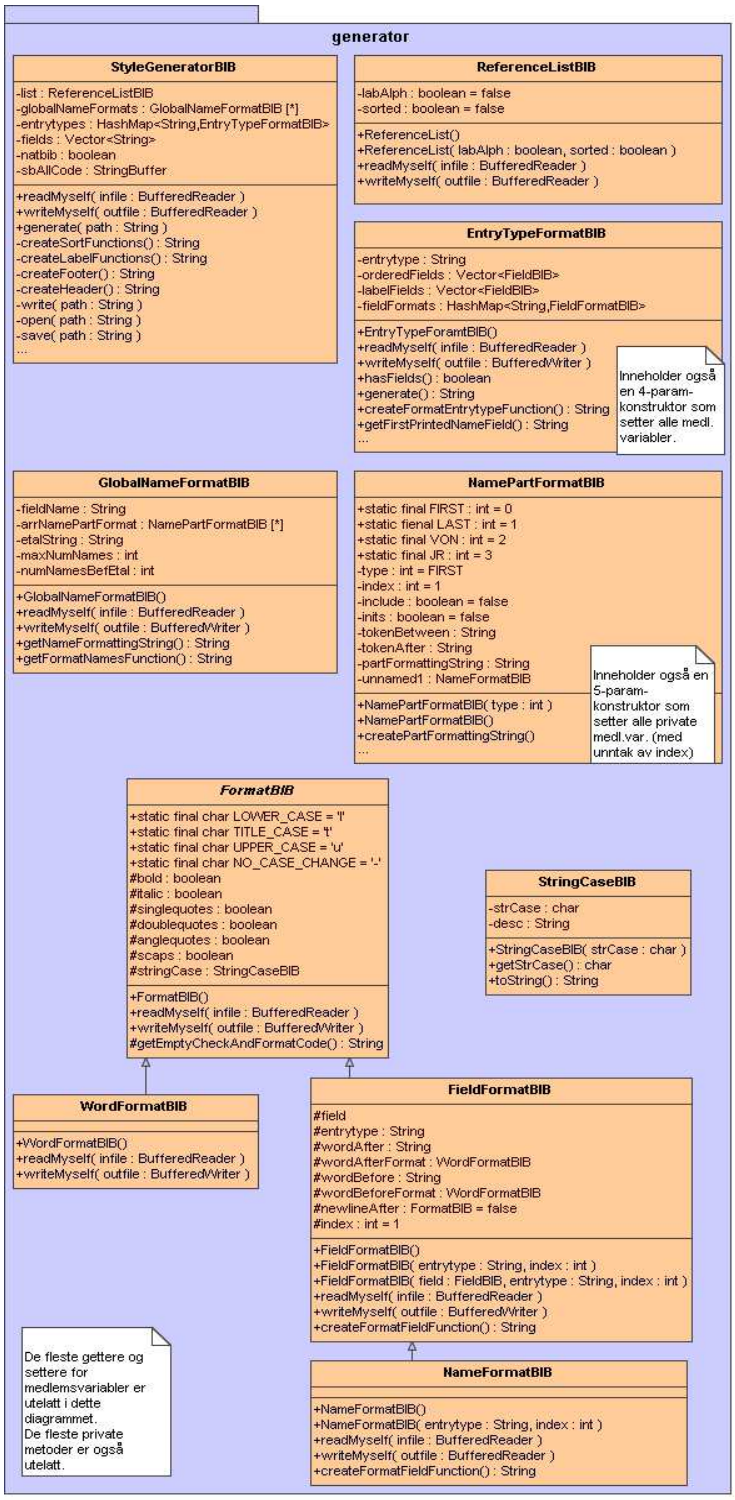
```
1   /**
2   * Compares one of this entry's persons fields (identified by fieldindex) to
3   * the correspondig persons field in entryToCompareTo.
4   * step is set to 0.5.
5   * Each person in the longest persons vector which is also
6   * contained in the shortest persons vector adds (step / length of the longest
7   * persons vector) to the equality percent, each person in the longest person
8   * vector which is NOT contained in the shortest persons vector
9   * subtracts (step / length of the longest persons vector) from the equality
10  * percent.
```

```
11  *
12  *
13  * @param         entryToCompareTo        the entry to compare to this entry
14  * @param         fieldIndex              identifies the field to be compared
15  * @see           #compare
16  * @return        the equality percent between the to fields which have been
17  *                compared
18  *                (1.0 if the field values are equal, 0.0 if the field values
19  *                are not equal at all).
20  *
21  *
22  */
23  public double comparePersons(EntryBIB entryToCompareTo,int fldInd){
24          double step;
25          double percentAcc = 0.5;
26
27          AbstractFieldBIB field1 = getStandardField(fldInd);
28          if(!(field1 instanceof PersonsBIB)) return 0.0;
29          PersonsBIB persons1 = (PersonsBIB)field1;
30
31          AbstractFieldBIB field2 = entryToCompareTo.getStandardField(fldInd);
32          if(!(field1 instanceof PersonsBIB)) return 0.0;
33          PersonsBIB persons2 = (PersonsBIB)field2;
34
35          Vector<PersonBIB> vecPersons1 = persons1.getPersons();
36          Vector<PersonBIB> vecPersons2 = persons2.getPersons();
37          int size1 = vecPersons1.size();
38          int size2 = vecPersons2.size();
39
40          if((size1 < 1) && (size2 < 1)) return 1.0;
41
42          if(size1 > size2){
43                  step = 0.5 / size1;
44                  for(PersonBIB person : vecPersons1){
45                          if(vecPersons2.contains(person)) percentAcc+=step;
46                          else percentAcc-=step;
47                  }
48          }else{
49                  step = 0.5 / size2;
50                  for(PersonBIB person : vecPersons2){
51                          if(vecPersons1.contains(person)) percentAcc+=step;
52                          else percentAcc-=step;
53                  }
54          }
55
56          return percentAcc;
57  }
```

## 5.6 Stilgeneratoren

### 5.6.1 Hovedklassene

Et *EntryTypeFormatBIB*-objekt bestemmer hvordan referanser av typen *entrytype* (medlemsvariabel i EntryTypeFormatBIB) skal formateres. HashMap'en *entrytypes* i *StyleGen-*




Figur 12: Stilgeneratorens datastruktur

*eratorBIB* inneholder et par av entrytypenavn-EntryTypeFormatBIB for hver entrytype. Klassen *FormatBIB* (og subklasser av denne) bestemmer hvordan et element (f.eks. et felt (FieldFormatBIB) eller en streng (WordFormatBIB)) skal formateres. HashMap'en fieldFormats i EntryTypeFormatBIB inneholder et key-value-par for hvert felt som skal skrives ut i referanselisten for denne entrytypen, hvor key er et feltnavn og value er et FieldFormatBIB-objekt. Metoden *createFormatFieldFunction* i FieldFormatBIB returnerer en streng med stakkekode som formaterer verdien i alle FieldFormatBIB (en stakkekode-funksjon med navn
`{FUNCTION {format.<entrytype>.<feltnavn>}`
f.eks. `FUNCTION { format.book.title }` ).
*StyleGeneratorBIB* er hovedklassen i generator-pakken. Når en stil genereres opprettes en instans av denne klassen. Metoden *generate* genererer stakkekode som formaterer referanselisten i følge formateringsvalgene i medlemsvariablene *list* (instans av ReferenceListBIB med formateringsvalg som gjelder *hele* referanselisten), *globalNameFormats* og *entrytypes* for deretter å skrive ut koden til to bst-filer (en vanlig bst-fil og en natbib-kompatibel bst-fil) og lagre formateringsvalgene til en .save-fil.

### 5.6.2 StyleGeneratorBIB

Metoden *generate* i StyleGeneratorBIB kaller hjelpemetodene *createHeader*, *createLabelFunctions*, *createSortFunctions* og *createFooter* som tilsammen genererer stakkekoden til èn stil (all stakkekoden legges i et StringBuffer, *sbAllCode*, og tilslutt skrives innholdet i dette bufferet til fil. Hver av disse metodene genererer kode for et antall stakkespråkfunksjoner. Siden en funksjon i stakkespråket må defineres før den kan kalles, er det viktig at funksjonene skrives ut i riktig rekkefølge til bst-filene. I StyleGeneratorBIBs generate-metode kalles først createHeader. Deretter itererer den gjennom alle EntryTypeFormatBIB-objektene og kaller EntryTypeFormatBIBs generate-metode på hver av dem for å generere funksjoner som formaterer en referanse (f.eks. `FUNCTION { book }`, `FUNCTION { article }`). Deretter skriver StyleGeneratorBIBs generate-metode ut en READ-kommando til *sbAllCode*, og tilslutt kalles metodene createLabelFunctions, createSortFunctions og createFooter.

#### createLabelFunctions

Metoden *createLabelFunctions* genererer kode for funksjonene som bygger opp *labelen*. I referanselister basert på alpha (dvs. alfabetiske labler) er det labelen som står først for hver referanse i en referanseliste - labelen er nøkkelen som man bruker for å finne igjen en referanse i referanselisten. I GUIen kan brukeren bestemme hvilke felter denne labelen skal bygges opp av og rekkefølgen på feltene. Vi valgte å gjøre det mulig å bygge opp

labelen på forskjellig måte avhengig av entrytype (vektoren labelFields i EntryTypeFormatBIB inneholder feltene labelen skal bygges opp av). I praksis bruker man stort sett de samme feltene i labelen (uavhengig av entrytype) - svært vanlig er det at labelen består av author og year. Det kan imidlertid være situasjoner der brukeren ønsker å spesialtilpasse labelen etter entrytype. Hovedfunksjonen som generers av *createLabelFunctions* er `FUNCTION { calc.label }`. Denne kan se f.eks. slik ut i bst-filen:

```
1:  FUNCTION {calc.label}
2:  { type$ "article" =
3       'editor.label
4:      { type$ "book" =
5:          'author.label
6:          'skip$
7:        if$
8:      }
9:    if$
11:   duplicate$
12:   year field.or.null purify$ #-1 #2 substring$
13:   *
15:   'label :=
16:   year field.or.null purify$ #-1 #4 substring$
17:   *
18:   sortify 'sort.label :=
19: }
```

I dette eksemplet lages det bare labler for referanser med entrytypen *article* og entrytypen *book*. Hvis typen til gjeldende referanse er *article*, kalles funksjonen `FUNCTION { editor.label }` som formaterer referansens editor-felt i henhold til formateringsvalgene [13] og popper den resulterende strengen på stakken. Hvilken funksjon som kalles her er selvfølgelig avhengig av innholdet i *labelFields* i EntryTypeFormatBIB, slik:

`FUNCTION { <labelField1>.<labelField2>.<labelField3>.label }`.

Koden fordenne funksjonen genereres også av *createLabelFunctions*.
Siden det er høyst uvanlig at labler i referanselister *ikke* slutter med årstall konkateneres strengen som nå ligger på stakken med de to siste sifrene i year-feltet (linje 12 og 13) og den resulterende strengen legges på stakken. På linje 15 settes labelvariabelen lik strengen

[13] I senere versjoner av Bib-*it* skal det være mulig for brukeren å bestemme hvordan feltene i labelen skal formateres, f.eks. hvis feltet er av typen name skal brukeren kunne oppgi hvilke deler av navnet som skal brukes og hvor mange bokstaver som skal være med. Nå er denne formateringen hardkodet i funksjonene som genereres av StyleGeneratorBIB - for felter av typen *name* brukes de tre første tegnene av etternavnet til forfatteren (hvis en forfatter) eller initialene i etternavnene på forfatterne (hvis flere forfattere) og på vanlige tekstfelter brukes de tre første tegnene.

som nå ligger på toppen av stakken.
På linje 18 settes variabelen sort.label lik label-variabelen. Denne variabelen brukes i sorteringen av referanselisten (se under).
Labler brukes kun når brukeren har valgt "alpha" i panelet "labels" i stilgeneratoren. Om brukeren har valgt "numerisk" (jf. number-only i punkt 2.2.1) brukes ikke labelFields i EntryTypeFormatBIB. I dette tilfelle genererer *createLabelFunction*-metoden kode for numeriske labler.

**createSortFunctions**

Metoden *createLabelFunctions* genererer stakkekode for `FUNCTION { presort }` som bygger opp en streng for hver referanse som brukes til sortering av referanselisten, samt flere hjelpefunksjoner. `FUNCTION { presort }` bygger opp sorteringsstrengen ved å konkatenere sort.label-variabelen med en streng som med alle forfatternavnene. presort-funksjonen kaller `FUNCTION { sort.format.names }` for å formatere forfatternavnene (fullt etternavn og initialene i fornavnet) og sette dem sammen til en streng.
Referanselisten sorteres ved å kalle BibTeXs innebygde SORT-kommando. Denne sammenligner sorteringsstrengene til alle referansene (bygd opp av presort-funksjonen) og ordner referansene i stigende rekkefølge.

### 5.6.3 EntryTypeFormatBIB

Klassen EntryTypeFormatBIB inneholder en metode *generate* som kalles av StyleGeneratorBIB sin generate-metode (se punkt 5.6.2). Denne metoden itererer gjennom alle FieldFormatBIB-objektene i vektoren *fieldFormats*, og kaller FieldFormatBIB sin createFormatFieldFunctionmetode. Metoden createFormatFieldFunction genererer kode for funksjoner, F.eks. `FUNCTION { format.book.title }`, som formaterer ett enkelt felt for en bestemt entrytype, i følge formateringsvalgene gitt av brukeren. Disse funksjonene bruker verdiene i feltene for den gjeldende referansen, formaterer dem, og legger den formaterte strengen tilbake på stakken. Tilslutt kaller EntryTypeFormatBIBs generate-metode metoden *createFormatEntrytypeFunction*. Denne metoden genererer kode for funksjoner (eks. `FUNCTION { book }`, `\verb`FUNCTION article ) som sørger for å skrive ut alle feltene i referansen i riktig rekkefølge og med riktig formatering (funksjonen kaller format.<entrytype>.<felt>- funksjonene (beskrevet over) for å formatere feltene).

## 5.7 Internasjonalisering

Factor et al. [2005] forklarer internasjonalisering slik:

> The process of designing software so that it can be adapted (localized) to various languages and regions easily, cost-effectively, and in particular without engineering changes to the software

For å tilpasse programvaren til et nytt språk/en ny region trenger man bare å legge til komponenter (f.eks. oversatt tekst) spesifikke til språket og regionen det er snakk om. *Locales* i J2SE er identifikatorer som kan brukes for å få oppførsel spesifikk til en region. *Localization* er støttet ved ResourceBundle-klassen, som gir tilgang til objekter (f.eks. strenger) som er spesifikke for en region.

### 5.7.1 Internasjonalisering i Bib-*it*

For Bib-*it*s vedkommende var det tekstene i brukergrensesnittet som var mest aktuelt å internasjonalisere. Vi implementerte en klasse *Babelfish*:

```
1  package no.hig.bibit;
2
3  import java.util.*;
4  import java.io.*;
5
6  /**
7   * The Babel fish.
8   * <blockquote cite="http://en.wikipedia.org/wiki/Babelfish">
9   * <p>
10  * The Babel fish is small, yellow and leechlike,
11  * and probably the oddest thing in the Universe.
12  * It feeds on brainwave energy [...] if you stick a
13  * Babel fish in your ear you can instantly understand
14  * anything said to you in any form of language.
15  * </p>
16  * </blockquote> Douglas Adams, 'The Hitchhiker's Guide to the Galaxy'.<br>
17  *
18  * Used for internationalization.
19  *
20  */
21 public class Babelfish{
22
23         /**  The ResourceBundle with current language (.properties) file. */
24         private static ResourceBundle resources = init();
25
26         /**
27          * @param string the name of the string to get.
28          * @return a localized string.
29          */
30         public static String getString(String string){
31                 return Babelfish.resources.getString(string);
32         }
33
34         private static ResourceBundle init(){
35                 return ResourceBundle.getBundle("Bib-it", ConfigFile.useLocale);
36         }
37 }
```

*ConfigFile.useLocale* er et *Locale*-objekt [14].

Istedenfor å hardkode teksten i labler, menyelementer og lignende i GUIen, kaller disse objektene Babelfish-metoden *getString*, som returnerer den ønskede teksten på språket

[14] variabelen ConfigFile.useLocale settes når brukeren endrer innstillingen om hvilket språk han vil bruke

som brukes for øyeblikket.

Tekststrengene som skal brukes i objektene hentes fra *språkfilene*. Dette er properties-filer med navn etter hvilket *Locale* de representerer. Om ingen språk er valgt, brukes properties-filen *Bib-it.properties* som inneholder alle tekststrengene på engelsk - engelsk er altså defaultspråket i Bib-*it*. For å oversette programmet til et nytt språk trenger man bare å lage en ny språkfil, f.eks. slik:

```
#File menu
fileMenu = File
saveAsBib = Salva come...
saveBib = Salva
openBib = Apri
openReadOnlyBib = Apri sola lettura
mergeBib = Combinazioni di file
new = Nuovo
exit = Uscita
notUniqueSearch = Trova voci non univoche
```

og gi den navn etter hvilket språk den inneholder (f.eks. `Bib-it_it_IT.properties` for italiensk språk).

Objektet *actOpenBIB*[15] instantieres slik:

```
1 actOpenBib = new AbstractAction(Babelfish.getString("openBib"),
2                                 ImageBIB.getInstance().getImage("open.gif")){
3                                    public void actionPerformed(ActionEvent ae){
4                                          ...
5                                    }
6                                 };
```

Her brukes nøkkelen *openBib* for å få tak i teksten på språket som er i bruk i øyeblikket. Babelfish-metoden getString kaller ResourceBundle-objektet sin getString-metode med denne nøkkelen som parameter, og denne ResourceBundle-metoden bruker nøkkelen for å hente riktig tekststreng fra riktig språkfil. Tekststrengen returneres til actOpenBib-objektet, slik at teksten som brukes i menyelementet openBib ved kjøring av Bib-*it* med italiensk språk blir *Apri*.

[15] actOpen er et Action-objekt som kan knyttes til et MenuItem-objekt eller en knapp slik at actionPerformed-metoden til Action-objektet kjøres når brukeren trykker på menyelementet eller knappen

# 6 Testing

Målet med software-testing er å bygge opp tillit til systemet. Både kunde og utviklere må bli trygge på at systemet fungerer slik det skal og at det oppfyller kravene slik de er gitt i kravspesifikasjonen.

Vi valgte å hovedsaklig benytte oss av brukersentrert testing, men under utvikling av datastrukturen benyttet vi også enhetstesting.

## 6.1 Enhetstesting

Enhetstestingen ble fortrinnsvis brukt på klasser i datastrukturen. Formålet med enhetstesting er å finne ut hvorvidt metodene i en klasse virkelig gjør det de skal gjøre og gir det resultatet utviklerne forventer.

For å lage og kjøre enhetstester brukte vi verktøyet JUnit. Verktøyet er utviklet av Erich Gamma og Kent Beck. [16] JUnit er et rammeverk for å skrive enhetstester, og kombinert med Ant, som automatiserer build-and-deploy-prosessen i et systemutviklingsprosjekt, er det enkelt å utføre automatisk testing.
*Test case*, *Test fixture* og *Test suite* er sentrale begreper i forbindelse med enhetstesting i JUnit. Hver klasse man skriver har vanligvis ett test case. Et test fixture sørger for nødvendige ressurser; variabler og objekter som testene trenger for å kjøre. En test suite er en samling av relaterte test cases.

### 6.1.1 Eksempler fra enhetstestingen: parsing av navn

Det første test-caset for klassen PersonBIB så slik ut:

```
1  package no.hig.bibit.domain.field.composite;
2
3  import no.hig.bibit.domain.field.*;
4  import no.hig.bibit.domain.*;
5
6  //JUnit imports
7  import junit.framework.*;
8  import junit.extensions.*;
9
10 /*
11 *
12 * @version 0.1
13 */
14
15 public class TESTPersonBIB extends TestCase{
16
17         private PersonBIB person1;
18         private PersonBIB person2;
```

[16] Kent Beck er mannen bak utviklingsmetodologien XP.

```
19          private PersonBIB person3;
20          private PersonBIB person4;
21          private PersonBIB person5;
22          private PersonBIB person6;
23
24          public TESTPersonBIB(String name){
25                  super(name);
26          }
27
28          public static Test suite(){
29                  return new TestSuite(TESTPersonBIB.class);
30          }
31
32          public static void main(String[] args){
33                  junit.textui.TestRunner.run(suite());
34          }
35
36          protected void setUp(){
37                  //InitBIB.init();
38                  person1 = new PersonBIB();
39                  //person2 = new PersonBIB("Heidi","van den","jr","Bjørke");
40                  PersonsBIB persons = new PersonsBIB("van den Bjørke, jr, Heidi");
41                  person2 = persons.getPersons().get(0);
42                  person3 = new PersonBIB("J.","","","Winden");
43                  person4 = new PersonBIB("Johan","","","");
44
45                  person5 = new PersonBIB("Heidi","van den","jr","Bjørke");
46                  person6 = new PersonBIB("Johan","","","");
47          }
48
49          public void testGetBibtexString(){
50                  String bibtexString = "van den Bjørke, jr, Heidi";
51                  String bibtexString2 = person2.getBibtexString();
52                  assertEquals("the bibtexString back from the person ",
53                          bibtexString,bibtexString2);
54
55                  String emptyBibtexString = "";
56                  String emptyBibtexString2 = person1.getBibtexString();
57                  assertEquals("the bibtexString back from the empty person ",
58                          emptyBibtexString,emptyBibtexString2);
59
60                  String fnameLname = "Winden, J.";
61                  String fnameLname2 = person3.getBibtexString();
62                  assertEquals("the bibtexString back from the fnameLname person ",
63                          fnameLname,fnameLname2);
64
65                  String fname = ", Johan";
66                  String fname2 = person4.getBibtexString();
67                  assertEquals("the bibtexString back from the fname person ",
68                          fname,fname2);
69          }
70
71          public void testGetSortString(){
72                  String sortString = "Bjørke, Heidi";
73                  String sortString2 = person2.getSortString();
74                  assertEquals("the sort string back from the person ",
75                          sortString, sortString2);
76
77                  String emptySortString = "";
78                  String emptySortString2 = person1.getSortString();
79                  assertEquals("the empty sort string back from the person ",
80                          emptySortString, emptySortString2);
```

```
81
82                  String fname = ", Johan";
83                  String fname2 = person4.getSortString();
84                  assertEquals("the sort string back from the fname person ",
85                          fname, fname2);
86          }
87
88          public void testEquals(){
89                  assertEquals(person4,person6);
90                  assertEquals(person2,person5);
91                  assertEquals(new PersonBIB(" ","","","","),
92                          new PersonBIB("","","","",""));
93          }
94  }
```

Denne testen kan kjøres separat ved å kjøre TESTPersonBIB-klassen, eller man kan inkludere den i en større test suite som kjører flere tester på en gang:

```
1   public class TESTAllTests{
2           public static void main(String[] args){
3                   junit.textui.TestRunner.run(suite());
4           }
5
6           public static Test suite(){
7                   InitBIB.init();
8                   TestSuite suite = new TestSuite("All JUnit Tests");
9                   suite.addTest(TESTEntryBIB.suite());
10                  suite.addTest(TESTPersonBIB.suite());
11                  suite.addTest(TESTEntryModelBIB.suite());
12
13
14                  . . .
15
16                  return suite;
17          }
18  }
```

Ved første kjøring ga TESTPersonsBIB følgende feil:

```
Time: 0,07
There were 3 failures:
1) testGetBibtexString(no.hig.bibit.domain.field.composite.TESTPersonBIB)
    junit.framework.ComparisonFailure: the bibtexString back from the
    person
    expected:<[van den Bjørke, jr, Heidi]>
    but was:<[den Bjørke, jr, van}>
    at no.hig.bibit.domain.field.composite.TESTPersonBIB.testGetBibtexString(
    TESTPersonBIB.java:52)
    ...
```

```
    ...
2) testGetSortString(no.hig.bibit.domain.field.composite.TESTPersonBIB)
    junit.framework.ComparisonFailure: the sort string back from the
    person
        expected:<Bjørke, [Heidi]>
        but was:<Bjørke, [van]>
    at no.hig.bibit.domain.field.composite.TESTPersonBIB.testGetSortString(
    TESTPersonBIB.java:75)
    ...
    ...
3) testEquals(no.hig.bibit.domain.field.composite.TESTPersonBIB)
    junit.framework.AssertionFailedError:
    expected:<den Bjørke, jr, van>
    but was:<van den Bjørke, jr, Heidi>
    at no.hig.bibit.domain.field.composite.TESTPersonBIB.testEquals(
    TESTPersonBIB.java:91)
    ...
    ...

FAILURES!!!
Tests run: 3, Failures: 3, Errors: 0
```

I den første testen som ga feil, ble tekststrengen
`String bibtexString = "van den Bjørke, jr, Heidi";` (expected) sammenlignet med PersonBIB-objektet person2 (den virkelige verdien, omtales som *was*):

```
PersonsBIB persons = new PersonsBIB("van den Bjørke, jr, Heidi");
person2 = persons.getPersons().get(0);
```

Her opprettes først et PersonsBIB-objekt for å få parset navnet vha. PersonsBIB sin doParsePerson-metode (som tar som parameter en streng som representerer ett navn i bibtex name format og returnerer et PersonBIB-objekt), deretter hentes PersonBIB-objektet ut fra PersonsBIB sin person-vektor (vi vet at det må være det første objektet i denne vektoren).
getBibtexString-testen ga følgende resultat:

```
    expected:<[van den Bjørke, jr, Heidi]>
    but was:<[den Bjørke, jr, van}>
```

Dette impliserer to mulige feil i koden; enten er getBibtexString-metoden feil, og hvis ikke må en eller flere av navnedelvariablene (fname, lname, von, jr) ha feil verdi.
I den tredje testen som ga feil, ble person2-objektet (opprettet som forklart over) sammenlignet med person5-objektet, som bruker en PersonBIB-konstruktor som tar navnedelene enkeltvis og setter navnedelvariablene direkte:

`person5 = new PersonBIB("Heidi","van den","jr","Bjørke");` person2 og person5 er forventet å være like, men det er de ikke. equals-testen ga følgende resultat:

```
expected:<den Bjørke, jr, van>
but was:<van den Bjørke, jr, Heidi>
```

Dette viser at det må være parsinga av navnestrengen
`van den Bjørke, jr, Heidi`
det er noe galt med, »»»> .r1025 altså metoden doParsePerson i PersonsBIB, nærmere bestemt parses jr-delen feil. Dette fikk vi bevist ved å legge til en testemetode i TESTPersonsBIB som tester variabelen sånn:

```
1 public void testGetVon(){
2         String expected = "van␣den";
3         String actual = person2.getVon();
4         assertEquals("getVon:␣",expected,actual);
5 }
```

Det viste seg at doParsePerson parser navnestrengen riktig ved en von-del bestående av kun ett ord, men fikk problemer hvis von-delen besto av flere ord.
Ved å gjøre noen små endringer i doParsePerson løste vi problemet:

```
1
2 if(lastCh == '␣' && ch != '␣'){
3         if(Character.isLowerCase(ch)){
4                 if(end != 0 && vonSet == 0){
5                         if( first == null){
```

ble endret til

```
1 if(lastCh == '␣' && ch != '␣'){// if first letter in word
2         if(Character.isLowerCase(ch)){ // if lowercase (e.g. von part)
3                 if(vonSet == 0){ //if NOT first token and first von token
4                         if(end != 0 && first == null){
```

Hele doParsePerson-metoden er tatt med i vedlegg D.
Etter at disse endringene ble utført kjørte vi testen TESTPersonBIB på nytt, og nå ga den følgende resultat:

```
Time: 0,051
OK (4 tests)
```

## 6.2 Brukersentrert testing

Som vist i eksemplet over fungerer enhetstesting veldig godt for å avdekke visse typer feil i programvaren (f.eks. problemer i datastrukturer, algoritmer o.l.), mens andre typer feil er nesten umulige å avdekke vha. enhetstesting. For å teste et grafisk brukergrensesnitt, er det svært lite hensiktsmessig å bruke enhetstesting. Her er det mer naturlig å bruke en form for akseptansetesting, dvs. at sluttbrukeren tester applikasjonen ved å kjøre en ferdig versjon og sjekke hvor godt funksjonaliteten i den ferdige versjonen tilfredsstiller kravene dokumentert i kravspesifikasjonen.

Siden vi brukte en inkrementell utviklingsmodell hvor vi la ut beta-versjoner av produktet minst en gang for hvert inkrement, utførte oppdragsgiver uformelle akseptansetester flere ganger i løpet av prosjektperioden. Han ga oss tilbakemelding om feil og problemer han oppdaget og kom med forslag til endringer i kravspesifikasjon (ny funksjonalitet og endring av eksisterende funksjonalitet) pr mail, og på grunnlag av disse tilbakemeldingene justerte vi de detaljerte arbeidsplanene for hvert inkrement.
På en god del av funksjonaliteten utførte også vi (utviklerne) brukersentrert testing.

### 6.2.1 Testing av API (Program-grensesnitt)

For å teste at vi ikke hadde laget for sterke koblinger mellom data og presentasjon, lagde vi et lite tekstbasert bruker grensesnitt (TUI). Det var opprinnelig meningen at vi skulle lage en fullstendig tekstbasert versjon i tillegg til en grafisk versjon (GUI), men vi fant ut at mange av de funksjonene det var ineressant å ha i BIB-*it* ville være enten nesten umulig[17] eller veldig upraktisk[18] å ha i en konsollversjon. Vi lagde en minimal konsollversjon (no.hig.bibit.ui.tui.CmdBibit) som kan gjøre følgende operasjoner:

| Consoll kommando: | Funkjon: |
|---|---|
| Bibit *fil* find *uttrykk evt. feltnavn* | Finner alle poster i filen *fil* som har *uttrykk* i *evt. feltnavn* eller alle felt. |
| Bibit *fil* dupkeys | Finner alle poster som ikke har unike referanse nøkler. |
| Bibit *fil1* insert *fil2* | Fletter inn *fil2* i *fil1* (så lenge det ikke er noen like nøkler eller noen annen feil oppstår). |

Disse funkjonene fungerer helt greit så lenge man bare skal gjøre en operasjon og man

[17] F.eks.: navnelista med navnesøk: denne funksjonen egentlig bare mulig i en GUI, fordi den baserer seg på å jobbe med en stor liste med navn og en annen stor liste med poster som har valgte navn, noe som blir fullstendig umulig å gjøre i en TUI (Konsoll bufferet er ofte ikke større enn ca. 30–40 linjer før de eldste linjene forsvinner).

[18] F.eks.: Redigering (fletting av filer, sletting av poster, etc.) blir i beste fall en slags teksteditor (som det finnes flere av: VIm, emacs, nano, notepad) eller i værstefall veldig upraktisk.

vet akkurat hva man skal finne og vet at søket gir bare to–tre treff, eller vet at det går bra; med andre ord i de tilfellene der man kunne like gjerne gjort det selv med en teksteditor. I de tilfellene tekstredigering blir tungvint fordi det ikke går helt knirkefritt å flette en fil manuelt eller det blir vanskelig å få oversikt over hva man leter etter fordi fila er stor, er CmdBibit til liten hjelp. Det tar dessuten uhensiktsmessig mye tid å parse filen slik BIB-*it* gjør[19] for slike oppgaver, derfor sluttet vi å videreutvikle TUIen.

Men som en test på BibTeX-APIen var CmdBibit likevel nyttig. Vi oppdaget bla. at datastrukturen og presentasjonsstrukturen til hver oppføring ikke var godt nok adskilt.

### 6.2.2 Parseren

Parseren ble testet kontinuerlig i hvert inkrement.
I begynnelsen skrev vi enkle bib-databaser for å teste at parseren klarte å lese de nye tingene. De første testfilene inneholdt noen få felt og alle feltene ble parset som tekst. Når parseren kunne lese vanlige felt lagde vi test filer med forfatter felt og prøvde å parse disse slik fortsatte vi med mer og mer komplisert forfatter syntaks.

I begynnelsen fokuserte vi mest på at parseren skulle kunne lese helt korrekte og pent skrevne filer, men den neste utviklingsfasen begynte vi å fokusere på hastighet og robusthet[20]. For å slippe å bygge en veldig kompleks parser som kunne takle det meste, lagde vi forprosessoren Cleaner som renskriver filene før parseren leser dem. Cleaner leser en fil og skriver til en ny fil, dermed kunne vi teste Cleaner uavhengig av parseren. Nå prøvde vi å finne store filer på internett for å teste hastigheten. For å teste Cleaner "saboterte" vi forskjellige filer så de bare så vidt var korrekt syntaksmessig for teste at cleaner gjorde det den skulle.

Etter dette var parseren, dvs. ParserBIB, stort sett ferdig. Detaljparsingen som er spesiell for hver felttype blir utført av felttypens egen klasse. Disse ble testet hver for seg etterhvert som de ble implementert. Parseren ble videre i utviklingen jevnlig testet siden enhver test av BIB-*it* stort sett startet med å lese en testfil. Etter en stund oppdaget vi at noen store bib-databaser hadde mange rare feil; mye av postene ble oppfattet som kommentarer og flere poster og felter ble konkatenert til underlige typer. Noen hadde riktignok feil som BIBTEX oppdaget mens andre bare ble feil i BIB-*it*. Det viste seg at felt med anførselstegn som start- og slutt-tegn og anførselstegn inni (med unntak av \" som

[19] CmdBibit implementasjonen er *spesielt* treg fordi den parser filen for *hver* kommando, men det kunne vi selvsagt endret hvis vi hadde følt at TUIen hadde noen funksjon.

[20] Faktisk ble Cleaner og parseren så robust at det kan være vanskelig få feilmeldinger. Vi prøvde bla. å lese en Word-fil for å teste hva som skjedde; verken Cleaner eller parseren protesterte. For å få feilmeldinger må faktisk fila være ganske korrekt, derfor bør man alltid sjekke at resultatet er ok før man lagrer.

var spesialbehandlet) var grunnen. Ved å inspisere ustskriften fra Cleaner kunne vi slå fast at feilen var i Cleaner[21]. Etter å ha studert koden i Cleaner, fant vi ut at vi hadde glemt å implementere en funksjon i `" "`-delen som var der i `{ }`-delen.

### 6.2.3 Duplikatsøk

I forbindelse med testing av duplikatsøk-funskjonaliteten var det spesielt to ting som var interessante;

1. hvor godt fungerer algoritmen (hvor mange av referanseparene som får en likhetsprosent på ca. 98-100 virkelig duplikater? blir alle referanseparene som er virkelige duplikater fanget opp av algoritmen?)
2. hvor godt fungerer det grafiske brukergrensesnittet (presenteres resultatet av duplikatsøket på slik måte at brukeren lett ser sammenhengen og lett kan orientere seg? er det lett å endre verdier i referanser og slette den/de referansene man ikke ønsker å ta vare på?)

Å måle hvor godt algoritmen fungerer viste seg å være svært vanskelig. Det var flere årsaker til dette. For det første trenger man en stor mengde reelle data (men selv om de skal være reelle bør de kanskje inneholde litt uvanlig mange duplikater, slik at man har noe å teste). Testingen kan ikke automatiseres - bare et menneske kan avgjøre om et gitt potensielt duplikatpar virkelig er duplikater, dvs. at det kan fort ta svært lang tid å gå gjennom hele resultatet grundig. Et annet kompliserende moment er - hvordan skal man finne ut om alle virkelige duplikater fanges opp av algoritmen? For å finne ut av dette må man manuelt gå igjennom hele bib-databasen man har utført duplikatsøk på, og finne alle duplikater for å sammenligne det man fant med resultatet fra duplikatsøket. Dette er en meget tidkrevende oppgave.

Vi mente allikevel at det var nødvendig å *Black Box-teste* duplikatsøkfunksjonaliteten for i det minste å få en formening om hvor godt algoritmen fungerer. Vi utførte brukergrensesnittesting parallellt med testing av algoritmen.

**Duplikatsøktesten**

For å en bib-database med reelle data som samtidig er et godt grunnlag for å teste duplikatsøkfunksjonaliteten, slo vi sammen 3 bib-databaser; ifa.bib[22], jonsbib.bib[23] og en av Nelson Beebes bib-databaser med hele 3856 referanser. Alle disse bib-databasene omhandler samme fagområde og vi visste at de inneholdt mange referanser til samme publikasjoner.

[21] Referansene var formatert som om de virkelig sluttet der, i den midlertidige filen.
[22] oppdragsgiver Ivar Farup sin bib-database med 1015 referanser
[23] Jon Yngve Hardeberg sin bib-database med 821 referanser

Vi utførte duplikatsøk med en grense på 70%. Søkeresultatet inneholdt 798 grupper av duplikatkandidater. Av disse var det ca. 570 referanser som hadde en likhet på 100% til en annen referanse i søkeresultatet, ca. 135 referanser med 99% likhet, ca 10 referanser med 90-96% likhet, 35 referanser med 80-89% likhet og 48 referanser med 70-79% likhet til en annen referanse i søkeresultatet.
Sammenligningsfeltene som blir brukt i algoritmen er year, author, title og journal.
Av referanseparene som hadde en likhet på 100% i duplikatsøkresultatet, var det bare 8% som i virkeligheten *ikke* var duplikater (ved manuell inspeksjon av referansene).
Av referanseparene som hadde en likhet på mellom 75% og 79% var det 33% (4 av 12) som var duplikate, mens av referanseparene som hadde en likhet på mellom 70% og 74% var det 14% (5 av 36) som var duplikate.
Disse tallene tyder på at duplikatsøkalgoritmen gir et rimelig fornuftig resultat.

**Feil som ble funnet og rettet** BIB-*it*s parser godtar ikke ugyldige kryssreferanser. Black Box-testen avslørte at når referanser slettes i duplikatsøket, slettes ikke automatisk verdien i crossref-feltet i de referansene som kryssrefererer referansen som ble slettet. Dette kan resultere i mange ugyldige kryssreferanser, noe som gir uventede resultater når filen leses inn neste gang. Denne feilen ble rettet ved å gå gjennom alle referansene i bib-databasen og nullstille verdien i crossref-feltet for de referansene som har crossref=bibtexkey'en til den slettede referansen.

### 6.2.4 Stilgeneratoren

Stilgeneratoren ble testet på følgende måter:

**1**
**Mål**: Undersøke om resulterende referanselisten virkelig blir formatert slik valgene i GUIen tilsier)
**Metode**: Vi testet den genererte stakkekoden ved å kjøre BIBTEX og LATEX på en .tex-fil (med siteringer) som bruker en .bst-fil generert av vår generator til å formatere referanselisten

**2**
**Mål**: Undersøke om .save-filformatet fungerer og om GUIen oppdateres riktig når verdiene fra .save-filene leses inn
**Metode**: Bruke GUIen til å lese inn en .save-fil og observere verdiene til objektene som inneholder formateringsvalgene

**3**
**Mål**: Undersøke hvor godt stilgeneratoren fungerer til å lage praktisk nyttig formatering av referanselister

**Metode**: Legge inn formateringsvalg i GUIen som ligner mest mulig en av standardstilene, og sammenlignet resulterende referanseliste med en referanseliste formatert av denne standardstilen

Stakkespråket krever at en funksjon er definert før den kan kalles (i motsetning til f.eks. Java hvor en metode kan kalles på et sted lenger opp i filen enn den er definert). En vanlig feil i en tidlig utgave av stilgeneratoren var at stakkekoden som kaller en funksjon kom lenger opp i .bst-filen enn koden som definerte funksjonen. Vi fikk følgende feilmelding fra BibTeX når vi testet den genererte .bst-filen:

```
This is BibTeX, Version 0.99c (MiKTeX 2.4)
The top-level auxiliary file: pages.aux
The style file: test.bst
format.names.author is an unknown function---line 294 of file
test.bst
format.names.author is an unknown function---line 444 of file
test.bst
format.names.author is an unknown function---line 586 of file
test.bst
Database file #1: ifa.bib
(There were 3 error messages)
```

Vi løste problemet ved å organisere koden til funksjonene (i *StyleGeneratorBIB*) på en litt annen måte. Vi samlet funksjoner som blir mye brukt av andre funksjoner i en metode createHeader som blir skrevet ut først i .bst-filen, og funksjoner som kun kalte andre funksjoner i en metode createFooter som blir skrevet ut helt tilslutt i .bst-filen.

# 7 Verktøy, distribuering og installering

## 7.1 Verktøy

Som kodestandard og utviklingsmiljø ble Java programmeringsspråket det mest naturlige valget siden Bib-*it* skulle være en plattformuavhengig applikasjon (se 1.6 for flere detaljer knyttet til valg av utviklingsmiljø).

Verktøyet vi brukte i forbindelse med kodingen var TextPad (en enkel tekstediteringsapplikasjon). Denne har integrert plug-in for kompilering av Java-kode. Noen av gruppemedlemmene testet også ut mer avanserte verktøy som for eks. Eclipse og UltraEdit, men TextPad ble foretrukket fordi den hadde akkurat det vi trengte og ikke mye "undøvendig" funksjonalitet.
TextPad ble også brukt til som selvstendig teksteditor for å skrive språkegenskapesfilene for flerspråkstøtte i Bib-it, bortsett fra de østeuropeiske språk filene og delvis noen av de vesteuropeiske hvor vi ble nødt til å skape egen Unicode - escape generator for å støtte Cyrillic og noen spesielle karakterer fra vestlige alfabet.

**Av andre hjelpeverktøy som ble nødvendig for oss å ta i bruk i dette prosjektet, kan nevnes:**

- Velkjente MS-Office-pakken, hvor MS Word ble brukt til midlertidig dokumentering av aktiviteter, og MS Visio ble brukt til å lage noen av figurene til rapporten. MS Project ble verktøyet vi brukte i forbindelse med utarbeiding av Gantt-skjema i både forprosjektrapporten og den endelige prosjektrapporten.
- En demo versjon av MagicDraw UML ble brukt til å skape UML-diagrammer.
- En av de mer sentrale applikasjonene som ble tatt i bruk spesielt i en senere fase av prosjektet var Latex. Dette verktøyet ble brukt i til å skrive både forprosjekt - og den endelige prosjektrapporten.
- For å skape de grafikkbittene som vi har brukt i Bib-it, som diverse ikoner og animasjoner, brukte vi en rekke gratis tilgjengelige verktøy som for eks. den allsidige GIMP. Mens til å lage plakat til vårt prosjekt trengte lit mer avansert program som Adobe Photoshop.

Hele dokumentasjonskriving og kodingsarbeid ble utført på datamaskiner med Windows eller Linux.

## 7.2 Distribusjon og installasjon

Et viktig poeng med distribueringsprosessen er å sørge for at brukeren har den riktige versjonen av Java-runtime. Videre er det naturlig å utnytte til en vis grad finessene til den aktuelle install-makeren for å gi brukeren mulighet til å tilpasse installeringen etter eget behov. Det er et utall distribueringsverktøy ute i verden. En del gratis og upålitelige mens resten krever lisens fra produsenten for å kunne brukes. I et ikke-profitterende studentprosjekt som dette, er det ikke naturlig at medlemmene investerer i diverse verktøy. Høgskolen i Gjøvik hadde lite å stille med i form av lisensierte kopier av diverse install-makere. De noen få gratis install-makerne tilgjengelige på weben var enten for kompliserte i bruk eller krevde en eller annen form for runtime for å kjøre. Eks:

- IzPack Java Software Installer
  `http://www.izforge.com/izpack/index.php?page=download`
- Webstart teknologien fra Java.

Disse var lite hensiktsmessige for vår applikasjon. Noen av gruppemedlemmene hadde erfaring med Setup Factory install-meker fra Indigo Rose corporation, men denne er ikke tilgjengelig i gratisversjon, og omstendighetene rundt bruken av demoversjonen passer ikke helt med lisensen som Bib-*it* er distribuert under. Vi tok kontakt med Indigo Rose[24] og forklarte situasjonen og Indigo Rose ga oss rettighetter til å bruke Setup Factory 7.0 i fullversjon i forbindelse med prosjektet vårt.
Den faktiske distribuering vil skje vha. en komprimert Setup.exe fil for Windows-brukere og i form av en komprimert fil av typen ZIP, TAR.GZ eller HQX for Linux- og MAC-brukere.

### 7.2.1 Installasjon

**Windows-brukere:**

1. Etter nedlasting av installasjonspakken for Bib-*it*, dobbeltklikk på Bib-it_1_0_Setup.exe for å starte installasjonsprosessen.
2. Før Bib-it_1_0_Setup.exe aktiverer Windows-installeren, vil Bib-it_1_0_Setup.exe sjekke hvilken versjon av Java Runtime Environment som er installert på den aktuelle datamaskinen.
3. Avhengig av resultatet, vil Bib-it_1_0_Setup.exe fortsette med installasjonen eller foreslå for brukeren å installere den nødvendige java runtime pakken.
4. Før installasjonen kan fortsette, må brukere velge ett av tre alternativer: Installere Java-runtime; Ikke installere Java runtime eller kansellere installeringen av Bib-*it*.

[24] http://www.indigorose.com/suf/index.php

5. For alternativet Installer Java-runtime, vil JRE ver 1.5 bli installert på brukeren sin PC før installasjonen av BIB-*it* fortsetter.

6. For alternativet Ikke installere Java runtime, vil installasjonen av BIB-*it* fortsette uten å installere medfølgende Java-runtime-versjon.

7. Deretter vil brukeren måtte svare på en rekke enkelte og mer eller mindre obligatoriske spørsmål om godkjenning av lisensavtale samt diverse sti- og stasjonsbokstavbekreftelser.

I installeringsrutinene er det implementert automatisk reparasjon av skadde filer, som betyr at i tilfelle installasjonen mislykkes pga feilaktig bruker- og / eller programoperasjon, vil brukeren kunne starte prosessen på ny for å reparere installasjonen og få applikasjonen til å virke.

**Linux-brukere:**
Etter nedlasting av BIB-*it*-installasjonspakken for det aktuelle OS, pakke ut innholdet i installasjonspakken og kjør JAR-filen vha. kommandoen`java -jar BibitX_X_X.jar`.

**MAC-brukere:**
Pakke ut innholdet i installasjonspakken og dobbeltklikke på jar-filen for å kjøre programmet.

**Installere og kjøre BIB-*it* fra kildekode (For alle platformer)**

1. Last ned kildekodearkiv fra BIB-*it* sin webside og pakk ut innholdet i en lokal mappe. Inni denne mappen finner man kildekoden til BIB-*it*. Man trenger også JDK versjon 1.5.0 eller senere for å kompilere denne koden.

2. Start terminalvinduet og naviger til mappen som inneholder kildekoden. Kompileringen utføres ved å kjøre følgende kommando `javac no/hig/bibit/Bibit.java` i en Unix/Linux-terminal, eller `javac no\hig\bibit\Bibit.java` i Windows/Dos kommandolinjen.

3. Etter at kompilasjonsprosessen har gått uten feilmeldinger, kan en kjøre BIB-*it* ved hjelp av kommandoen `java no.hig.bibit.Bibit`

# 8 Diskusjon av resultater

Alt i alt er vi veldig godt fornøyd med prosjektperioden - både gjennomføringen og resultatet. Prosjektet har resultert i en fungerende applikasjon som tilfredsstiller alle hovedelementene i kravspesifikasjonen. Underveis i utviklingsprosessen har vi lagt ut betaversjoner av Bib-*it* som har blitt testet av oppdragsgiver, veileder og andre interesserte (bl.a. studenter og ansatte ved Høgskolen i Gjøvik og Universitetet i Oslo.
Det har blitt avdekket en del feil som har blitt rettet underveis, men muligheten for at det fortsatt finnes feil er selvfølgelig til stede. Vi har også mottatt ønsker om tilleggsfunksjonalitet som vi har gjort vårt beste for å få implementert i løpet av prosjektperioden.
Bib-*it* har blitt godt mottatt hos dem som har testet applikasjonen.

Vi har vært i kontakt med Dag Langmyhr, overingeinør ved Universitetet i Oslo, som har sagt at han ønsker å prøve å få lagt opp Bib-*it* som et standardprogram ved Universitetet i Oslo.

Vi ønsker å fortsette utviklingen etter at hovedprosjektet er slutt, og vi har derfor tatt med et eget avsnitt på slutten i denne rapporten hvor vi nevner mulige utvidelser og forbedringer av eksisterende funksjonalitet (se punkt 8.9).

## 8.1 Avvik fra kravspesifikasjon og tidsplan

I de tre første inkrementene fikk vi ingen nevneverdige avvik fra tidsplanen - funksjonaliteten som skulle implementeres i disse inkrementene var stort sett på plass innen slutten på hvert inkrement. Det oppsto derimot en endring av målsetting for prosjektet i løpet av første og andre inkrement - dette blir diskutert grundigere i punkt 8.1.1.

I fjerde inkrement kom vi ikke så langt med stilgeneratoren at vi følte at den var bra nok til å inngå i en offisiell versjon, derfor ble det kun lagt ut mer uoffisielle versjoner etter fjerde inkrement, som oppdragsgiver og veileder brukte til å teste stilgeneratoren.
På grunn av gjenstående arbeid fra fjerde inkrement (stilgeneratoren) og at rapportskrivingen tok litt mer tid enn vi hadde regnet med, ble vi også litt forsinket i femte inkrement, som medførte at vi ikke ble ferdig med funksjonaliteten som skulle implementeres i løpet av femte inkrement.

Vi synes vi klarte å få til en fornuftig inndeling i inkrementer og fordeling av funksjonalitet som skulle implementeres i de enkelte inkrementene; det har alltid vært nok å gjøre men samtidig har vi stort sett klart å bli ferdig innen tidsfristene (med unntak av femte inkrement, som vi utdyper nedfor). Detaljinndelingen i hvert inkrement derimot, fikk vi ikke til så godt - her ble det en del tidsestimeringer som ikke henger på greip, f.eks. kan det

nevnes at vi satte av hele 5 dager til funksjonaliteten "flette inn fil", noe som i virkeligheten tok bare noen timer å implementere. Forklaringen til at vi allikevel alltid har hatt nok å gjøre, ligger nok i at vi underveis har implementert en god del funksjonalitet som ikke var planlagt før første inkrement og som derfor ikke var medregnet i den første versjonen av Gantt-diagrammet, samt at det var en del ting i forbindelse med brukergrensesnittet som ikke var direkte funksjonaliteter, men som allikevel måtte gjøres for å få til en fungerende applikasjon. I tillegg til dette brukte vi også en del tid underveis i inkrementene til testing og andre nødvendige aktiviteter.

### 8.1.1 Endring av målsetting

I løpet av de to første inkrementene diskuterte vi det tekstbaserte grensesnittet mhp. hvilken funksjonalitet som skulle være tilgjengelig her og hvordan grensesnittet skulle fungere. Etterhvert som GUIen vokste fram ble oppdragsgiver imidlertid i tvil om det var behov for et tekstbasert grensesnitt. Det han før prosjektets start mente han ville foretrekke å gjøre gjennom et tekstbasert grensesnitt (enkle operasjoner som å flette inn referanser inn i en eksisterende bib-database og søke etter referanser), fant han etterhvert ut at han like gjerne kunne gjøre i GUIen. GUIen etterhvert kjapp å starte og måten referansene presenteres på i aller høyeste grad var å foretrekke i et grafisk brukergrensesnitt framfor i en TUI (nesten all funksjonaliteten innebærer presentasjon av store mengder tekst, noe som er svært vanskelig å få til på en god måte i et tekstbasert brukergrensesnitt).
Vi ble derfor i samråd med oppdragsgiver og veileder enige om å endre oppgaven til å kun omfatte biblioteket og et grafisk brukergrensesnitt.
Det bør imidlertid også nevnes at et av målene for prosjektet var å lage *to ulike* brukergrensesnitt, for å på denne måten kunne vise at biblioteket er mest mulig frittstående og kan kobles til nye brukergrensesnitt på en enkel måte. Vi har derfor implementert små, enkle tekstbaserte brukergrensesnitt (men ikke fullstendige nok til at de kan inngå i en release) for å teste hvor godt biblioteket fungerer.

### 8.1.2 Første inkrement

I første inkrement gikk det meste som planlagt. Vi la ut versjon 0.1.1 ved slutten av inkrementet. Denne versjonen består av et forholdsvis enkelt grafisk brukergrensesnitt og et bibliotek med en enkel parser. GUIen består av en tabell (hovedtabellen) som viser alle referansene i den åpne bib-databasen. Her er det mulig å åpne en eksisterende bib-database, lagre bib-database, legge til/slette/endre referanser og sortere. I tillegg var diverse funksjonalitet for å vedlikeholde forfattere og redaktører påbegynt (se punktet 3.4). I versjon 0.1.1 var det mulig å legge til/slette/endre en person knyttet til en referanse, og endre rekkefølgen på personene, men den globale personlisten var ikke implementert.

På møtene med oppdragsgiver og veileder underveis i første inkrement diskuterte vi blant annet datastrukturen (spesielt koblingen mellom domenelaget og GUI-tabellen) og vurderte og testet ut ulike alternativer for organisering av GUIen (blant annet om vi skulle ha skjemavisning for å redigere en enkelt referanse og hvordan denne skulle organiseres). Vi hadde en utfordring knyttet til parseren - det tok svært lang tid å parse store bib-databaser. Datastrukturen med klasser som representerer felttyper (AbstractFieldBIB, CompositeFieldBIB, SimpleFieldBIB og alle subklasser av CompositeFieldBIB og SimpleFieldBIB) kom på plass i løpet av første inkrement.

EntryModelBIB gjennomgikk store endringer i løpet av første og andre inkrement. I de tidligste versjonene av denne klassen ble vi litt for mye opphengt i et ønske fra oppdragsgiver om å kunne gruppere referanser under en referanse den kryssrefererer og lot dette hierarkiet gjenspeile seg helt ned i datastrukturen ved at EntryBIB-vektoren i EntryModelBIB innehold kun inneholdt *parent*-EntryBIB-objektene (dvs. referanser som ikke har noen kryssreferanse), og *child*-EntryBIB-objektene (referanser som har kryssreferanse til en annen referanse) kun var tilgjengelige gjennom den referansen den kryssrefererer (EntryBIB hadde en vektor *children* som inneholdt EntryBIB-objekter). Denne datastrukturen fungerte veldig bra mot *vårt* grafiske brukergrensesnitt, men førte til at enkle, ikke-hieraktiske, presentasjoner av alle referansene i en bib-database ble unødvendig komplisert å implementere - dette oppdaget vi under utviklingen av en TUI som skulle bruke det samme biblioteket. Siden oppdragsgivers krav om et klart adskilt brukergrensesnitt veide tungt, kjørte vi en refaktorering på deler av datastrukturen (*modell*-klassene *EntryModelBIB* og *TableModelBIB*), noe som resulterte i mindre avhengighet mellom data og presentasjon.

### 8.1.3 Andre inkrement

I andre inkrement gikk også alt etter planen.
Under implementasjonen av "legge til egendefinerte felter" fant vi fort ut at det også var behov for å kunne slette egendefinerte felter fra en referanse. Vi valgte å gi alle egendefinerte felter typen 'text' (TextBIB). Det kunne også selvsagt vært nyttig å kunne legge til felter med andre typer. Dette kan enkelt implementeres senere ved å gjøre noen små endringer i et par metoder i EntryBIB (addCustomField og removeCustomField) og la vektoren customFields i EntryBIB ta AbstractFieldBIBobjekter istedenfor TextBIBobjekter, og tilpasse brukergrensesnittet tilsvarende.

Som en konsekvens av kravet om å kunne generere bibtexkey automatisk, var det også nødvendig å lage enkelt formateringsspråk slik at brukeren kan bestemme hvordan bibtexkeyen skal bygges opp (hvilke felter bibtexkeyen skal bygges opp av og rekkefølgen på disse).

I tillegg til funksjonaliteten som var planlagt for andre inkrement rakk vi å implementere en del av ønskene som kom opp i løpet av andre inkrement; søke etter ikke-unike poster (dvs. poster med eksakt lik bibtexkey) og GUI-spesifikk funksjonalitet for å skjule/vise kolonner i tabellen og slå av og på gruppering av kryssreferanser.

### 8.1.4 Tredje inkrement

I tredje inkrement ble all funksjonalitet planlagt for dette inkrementet (finne duplikate referanser, legge til referanser fra annen bib-fil) implementert og en ny betaversjon ble lagt ut tirsdag 21. mars. Vi fikk også tid til å gjøre noen generelle forbedringer i GUIen (knyttet til menyer, dialogbokser og åpne/lagre-funksjonalitet). Det ble også implementert funksjonalitet for å redigere preamble (og andre @-kommandoer i bib-databasens filhodes.

### 8.1.5 Fjerde inkrement

Ved slutten av fjerde inkrement hadde vi to ulike (uferdige) versjoner av stilgeneratoren. Den ene generatoren bygger på CustomBib og er en GUI-tilpasning av CustomBib-TUI-veiviseren (se punkt 2.2.5), den andre generatoren er helt uavhengig og genererer stakkekoden selv. Vi lot oppdragsgiver og veileder teste de to alternative stilgeneratorene, og det ble bestemt at vi skulle sats på vår helt egenutviklede stilgenerator og videreutvikle denne i femte inkrement.

### 8.1.6 Femte inkrement

I begynnelsen av femte inkrement var vi ikke helt ferdig med all funksjonalitet fra fjerde inkrement (stilgeneratoren). Det gikk også med en god del mer tid til rapportskriving enn vi hadde regnet med. Dette kombinert med at tre av gruppemedlemmene hadde flere eksamner midt i femte inkrement, førte til at vi dessverre ikke kom i mål med funksjonaliteten for å eksportere til EndNote-formatet, og ble nødt til å utsette dette arbeidet til etter at hovedprosjektet er levert.
Femte inkrement ble hovedsaklig brukt til rapportskriving, samt å lage en Windows-installer for Bib-*it*. Vi gikk også igang med å undersøke hvordan vi skulle eksportere til EndNote, noe som viste seg å være vanskeligere enn vi trodde. Etter en nærmere undersøkelse av EndNote-formatet (som er veldig ustabilt, det kommer med jevne mellomrom nye versjoner med så store endringer fra forrige versjon at en eksporteringsfunksjon fort ville gå ut på dato) og testing av andre programmer som eksporterer fra BibTex-format til EndNote-format.

## 8.2 Vurdering av utvalgte deler av BIB-*it*

### 8.2.1 Duplikatsøk

**Grafisk brukergrensesnitt (dupsøkresultat-tabellen m. forhåndsvisning)**
I GUIen er prosentene ganske godt skjult (ingen av radene er expanded i utgangspunktet). Det fungerer bra siden det ikke er prosentene brukeren skal henge seg opp i, men innholdet i referansene som kanskje er duplikater (som man får presentert i forhåndsvisningen). Det burde kanskje gått bedre fram av GUIen at ved å sortere på expandkolonnen sorteres gruppene etter maxprosent (dvs. den referansen i gruppen som har høyest prosentverdi er den som brukes i sorteringen).

Når en referanse er valgt og man velger forhåndsvisning, vises detaljer om alle referansene i gruppen, i den rekkefølgen de opptrer i tabellen. Sletting av en av referansene (eller hele gruppen) gjøres i tabellen. Det er gjerne i forhåndsvisningen man finner ut hvilken av referansene man vil slette, og da må brukeren passe på at han sletter riktig referanse i tabellen. Bedre hadde det kanskje vært om det kunne vært mulig å slette referanser i forhåndsvisningen (men da hadde det kanskje vært naturlig å kalle det for noe annet enn forhåndsvisning), særlig hvis flere av referansene i gruppen har ganske like prosentverdier. Dette er imidlertid ikke noe stort problem siden i de fleste tilfeller inneholder gruppen få (ofte bare to) referanser.

Det er mulig å endre verdier i referanser i skjemavisning også fra duplikatsøk-tabellen, og man kan naturligvis velge hvilken av referansene i en duplikatgruppering man vil slette. Dette viste seg å være veldig nyttig. Gitt to referanser fra duplikatsøk-tabellen som brukeren ved inspeksjon finner ut er virkelige duplikater. Ofte inneholder disse referansene litt forskjellig informasjon, i den ene er kanskje ikke number-feltet utfylt (men det er fylt ut i den andre), og i tillegg har denne referansen kanskje en kommentar i note-feltet som brukeren er interessert i å ta vare på. Da kan brukeren enkelt redigere referansene slik at en av dem (den han vil ta vare på) har all informasjonen som trengs, og han kan slette den andre. Dette sparer brukeren for å måtte gå inn i tabellen med alle referansene, finne de to referansene det er snakk om og endre opplysningene, for så å huske på hvilken referanse han skal ta vare på og gå tilbake til duplikatsøktabellen for å slette denne.

**Hvor godt fungerer algoritmen?** Algortimen fungerer veldig bra på de fleste typer referanser (jf. test av duplikatsøk i punkt 6.2.3).
Men algoritmen har problemer med referanser av typen artikler hvor feltene author, title, year og journal er eksakt lik de tilsvarende feltene i en annen referanse. Noen av disse viser seg å faktisk ikke være duplikater. Et eksempel (fra en av Nelson Beebes bib-databaser):

```
@ARTICLE{Anonymous:1996:SIGa,
```

```
    author = {Anonymous},
    title = {{1997 Symposium on Interactive $3$D Graphics}},
    year = {1996},
    month = may,
    pages = {62--??},
    journal = {Computer Graphics},
    volume = {30},
    number = {2},
    issn = {0097-8930},
    coden = {CGRADI, CPGPBZ},
    bibdate = {Wed Aug 14 17:53:29 MDT 1996},
}

@ARTICLE{Anonymous:1996:SIGb,
    author = {Anonymous},
    title = {{1997 Symposium on Interactive $3$D Graphics}},
    year = {1996},
    month = nov,
    pages = {82--??},
    journal = {Computer Graphics},
    volume = {30},
    number = {4},
    issn = {0097-8930},
    coden = {CGRADI, CPGPBZ},
    bibdate = {Tue Dec 3 18:59:19 MST 1996},
}
```

Duplikatsøkealgoritmen gir 100% likhet mellom disse referansene (feltene author, title, year og journal er eksakt like). Studerer man dem nærmere ser man at feltene number, month og pages ikke er like. Hadde kun ett av disse feltene hatt ulike verdier kunne man anta at det var en "trykkfeil" i en av referansene, men når det er tre felter med ulike verdier er dette lite trolig.
En bruker som har mange referanser av denne typen, kan ha nytte av å legge til et ekstra felt i duplikatsøksammenligningen, f.eks. et av number, volume eller pages. På bakgrunn av dette tenkte vi at feltene author, title, year og journal bør være faste sammenligningsfelter, men at det skal være mulig å legge til et eller flere felter i tillegg. Slik duplikatsøket er i dag er feltene author, title, year og number hardkodet i duplikatsøkalgoritmen. En mulig fremtidig endring av duplikatsøket er å legge noen linjer i plainStyle.ini som angir hvilke felter som skal brukes, f.eks. slik:

```
##Fields to use in duplicate search
#required fields
authorfield = author
```

```
titlefield = title
yearfield = year
journalfield = journal

#field name of additional fields to use
number
month
```

Det må imidlertid vurderes nærmere om det er nødvendig å gjøre det så komplisert, og om en gjennomsnittlig i det hele tatt kommer til å sette seg såpass godt inn i duplikatsøket at han vil benytte seg av muligheten til å legge til tilleggsfelter. At det forekommer en og annen gruppe hvor duplikatsøkalgoritmen gir 100% likhet i tilfeller som vist over, er sannsynligvis ikke noe stort problem siden søkeresultatet er organisert slik som det er i en oversiktlig tabell.

## 8.3 Fordeler og ulemper med BIB-*it* vs. JabRef

Som antydet tidligere er JabRef etter vår oppfatning den eneste GUI-applikasjonen for å håndtere BibTeX-databaser som er såpass bra at en gjennomsnittlig bruker vil foretrekke å bruke denne framfor å redigere tekstfilen direkte. Vi har tidligere begrunnet hvorfor vi valgte å starte utviklingen av en ny applikasjon istedenfor å bidra i videreutviklingen av JabRef, med vekt på utviklingsprosessen og ønsket funksjonalitet for BIB-*it* (se punkt 2.5).

Ved prosjektets slutt er det derfor naturlig å sammenligne vårt resultat med JabRef, og vurdere med den ene i forhold til den andre.(se 2.4.2) Sammenligningen er hovedsaklig basert på testing av JabRef (versjon 1.8.1 [25]) og BIB-*it* som ble utført 5. mai. Det må imidlertid understrekes at målinger av ytelse og hastighet ikke er utført vitenskapelig, men vi har prøvd å gjøre testbetingelsene så identiske som mulig. Testene ble utført på en stasjoner Intel Pentium III 731 mHz med 384 mB RAM.

JabRef har mye funksjonalitet, deriblant en del funksjonalitet som ikke er tilgjengelig i BIB-*it*. Et poeng her er at JabRef har vært i kontinuerlig utvikling helt siden 2003, mens utviklingen av BIB-*it* så dagens lyst for bare 5 måneder siden. Noe av denne funksjonaliteten kunne det kanskje vært aktuelt å implementere i BIB-*it* etterhvert, men enkelte ting, som f.eks. å kunne kjøre eksterne applikasjoner, ser verken vi eller oppdragsgiver den store nytten av. Gruppering er en annen funksjonalitet som JabRef har, men som vi foreløbig ikke har planer om å implementere i BIB-*it*.

[25] Dette var nyeste versjon av JabRef da vi startet med hovedprosjektet

### 8.3.1 Testing av ytelse og hastighet

Bib-*it* og JabRef tar omtrent like lang tid å starte opp (Bib-*it* er litt raskere, men ikke så mye at det har noen praktisk betydning). Minnebruken derimot (rett etter oppstart) er i JabRef på ca 24 mB, mot ca 19 mB i Bib-*it*. Bib-*it*s parser viser seg også å være en god del raskere enn JabRefs parser. Det tar 7,5 sekunder å lese inn bib-databasen ieeecg.bib (893 kB, 1038 referanser), mens det bare tar 4,6 sekunder å lese inn samme fil i Bib-*it*. Bib-*it* har også en progressbar slik at brukeren kan se at det faktisk skjer noe. Minnebruken er omtrent den samme i Bib-*it* og JabRef rett etter at denne filen er åpnet (ca. 32 mB).

### 8.3.2 Sammenligning av brukergrensesnitt

Hovedtankene bak organiseringen av brukergrensesnittet er den samme i JabRef og Bib-*it*. Hovedvinduet består i begge applikasjonene av et tabellpanel (øverst), en panel for skjemavisning (nederst) og et ekstra panel til venstre. I begge applikasjonene presenteres referansene i den åpne filen i en tabell, mens man kan endre informasjon i en referanse i skjemavisningen nederst. Panelene i JabRefs hovedvindu er imidlertid noe mer fleksible enn de er i Bib-*it*; panelet til venstre og panelet nederst kan vises og skjules etter behov. Dette er det imidlertid enkelt og få til også i Bib-*it*, hvis det skulle bli aktuelt.

Da utviklingen av Bib-*it* startet hadde vi ikke noe mål eller ønske om å "kopiere" JabRefs inndeling av paneler i hovedvinduet eller bruk av tabell og skjemavisning. Vi startet med et åpent sinn og prøvde ut forskjellige organiseringer av hovedvinduet og vi valgte det oppdragsgiver likte best etter å ha lekt litt med brukergrensesnittet.

Hvordan tabellen(e) brukes for å vise fram referanser er ganske forskjellig i Bib-*it* og JabRef. Dette er beskrevet nærmere i punkt 4.2.1 (for Bib-*it*) og i punkt 2.4.2 (for JabRef), men hovedforskjellene består i at Bib-*it* bruker en tabell for alle referanser og en tabell for hvert søkeresultat, mens JabRef viser søkeresultatet direkte i hovedtabellen.
Oppdragsgiver var ikke særlig begeistret over måten søkeresultatet ble presentert på i JabRef. I samarbeid med oppdragsgiver kom vi fram til tabellorganiseringen som er beskrevet i 4.2.1 som har vist seg og fungere meget godt.

### 8.3.3 Sammenligning av kjernefunksjonaliteten

Referansene kan i begge applikasjonene sorteres på et hvilket som helst felt, stigende eller synkende. Til forskjell fra JabRef er det i Bib-*it* *typen* til et felt som bestemmer hvordan dette feltet skal sorteres. Det vil si at f.eks. et årstallsfelt som sorteres på følgende måte i JabRef:

| 1996 |
|---|
| 1998 |
| 213 |
| 987 |
| about 1996 |
| omkring 1995 |

vil bli sortert slik i BIB-*it*

| 213 |
|---|
| 987 |
| omkring 1995 |
| 1996 |
| about 1996 |
| 1998 |

Det sammme gjelder et felt av typen 'month' - dette vil i BIB-*it* bli sortert slik at *jan* kommer før *feb* ved stigende sortering. Til tross for en litt mer komplisert sortering i BIB-*it*, går ikke dette utover hastigheten - på en vanlig pc er tiden det tar å sortere på et felt i en bib-database med opptil et par tusen referanser så kort at GUIen ikke oppleves som "hakkete" for brukeren.
BIB-*it* har god støtte for å enkelt legge til forfattere/redaktører på en referanse samt å vedlikeholde personene både globalt og på en enkelt referanse, som beskrevet i punkt 4.2.1. Dette var noe av det de personene som har testet ut betaversjonene av BIB-*it* likte veldig godt.
I tillegg til at det er mulig å søke både eksakt, vanlig og vha. regexp'er (som i JabRef), kan man i BIB-*it* begrense søket til å gjelde bare ett felt eller en kombinasjon av flere felter.

Vi valgte en litt annen løsning å presentere søkeresultatet på enn JabRef har gjort, men dette er smak og behag. Oppdragsgiver likte best BIB-*it*s løsning hvor søkeresultatet (og duplikatsøkeresultatet) vises i egne tabeller, så lenge det er en fornuftig sammenheng mellom tabellene slik at det er lett å orientere seg, noe vi mener vi har klart å få til i BIB-*it* (se beskrivelsen av tabellene i punkt 4.2.1).

### 8.3.4 Duplikatsøk

Duplikatsøk er tilgjengelig i begge applikasjonene. Det var spesielt denne funksjonaliteten oppdragsgiver syntes fungerte dårlig i JabRef, så målet for BIB-*it* var å få til et godt fungerende duplikatsøk.

Det mener vi at vi har klart. Duplikatsøket i BIB-*it* har blitt testet ut flere ganger på reelle bib-databaser. Duplikatsøket er noe av det oppdragsgiver er meget fornøyd med i BIB-*it*; både med hensyn på hvor godt det klarer å fange opp reelle duplikater og hvordan duplikatene presenteres i det grafiske brukergrensesnittet. Se 6.2.3 for et eksempel på hvordan vi har testet duplikatsøkfunksjonaliteten.

Vi mener dermed at vi med rette kan påstå at vi har klart å få til en duplikatsøkfunksjonalitet som fungerer langt bedre enn duplikatsøket i JabRef.

### 8.3.5 Stilgeneratoren

JabRef har ikke funksjonalitet for å generere BIBTEX-stiler. Siden dette var noe oppdragsgiver ønsket men anså som en relativt omfattende oppgave å utvikle, må vi si oss fornøyd med hva vi har fått til her. Stilgeneratoren kan imidlertid forbedres på en del områder (se 8.9.2).

## 8.4 Arbeidsprosessen

Det var nytt for alle i gruppa å jobbe med et prosjekt av et så stort omfang. Vi har jobbet med prosjektet 4-5 dager i uken i nesten fem måneder. Vi har gjort vårt beste for å planlegge arbeidet og evaluere resultatene underveis, for at alle gruppemedlemmene skulle ha noe å gjøre hele tiden og så vi skulle komme i mål med prosjektet. Dette har vært en stor utfordring, og vi kunne nok gjort det bedre på dette området - i perioder har noen av gruppens medlemmer hatt lite å gjøre.

## 8.5 Parprogrammering

I begynnelsen av prosjektperioden bestemte vi oss for å teste ut parprogrammering. Ingen av oss hadde prøvd dette før, og vi syntes i utgangspunktet det så ut som en spennende måte å jobbe på. Imidlertid fant vi fort ut at dette var en arbeidsmåte som ikke passet så godt for oss. Vi har isteden fordelt arbeidet slik at vi har jobbet på hver vår del av koden, men vi har selvfølgelig måttet snakke mye sammen, diskutert mye og hatt god kjennskap til hverandres deler av koden for å få det til å fungere sammen.

## 8.6 Faglig utbytte

Alle gruppemedlemmene hadde en god del erfaring med programmering fra tidligere i studiet, men ingen av oss hadde vært med på å utvikle systemer av denne størrelsen før. Av programmeringsdelen har vi særlig lært mye om kodestrukturering og kodefordeling. LaTeX, BibTeX og TeX var nytt for alle gruppemedlemmene, og dette har vi også lært

en god del om. Fra tidligere emner har vi vært borti systemutviklingsmodeller, men fortrinnsvis i teorien. Vi brukte en inkrementell utviklingsmodell og har lært mye om hvordan denne fungerer i praksis.

## 8.7 Valg av utviklingsmodell

Vi er veldig godt fornøyd med valg av utviklingsmodell. Det at vi har hatt relativt korte inkrementer og at vi satte oss som mål å legge ut en betaversjon av BIB-*it* har vært medvirkende til at vi har klart å jobbe jevnt og effektivt gjennom hele prosjektperioden. En bivirkning av å legge ut betaversjoner ofte var at vi etterhvert også ble gode på å distribuere programvare (som omfatter blant annet å lage jar-filer, skrive readme-filer, reklamere for programvaren) - dette sparte oss for mye tid i de hektiske dagene helt på slutten av prosjektperioden da vi skulle distribuere BIB-*it* 1.0.

Men først og fremst har en inkrementell utviklingsmodell vært nesten uunnværlig for at vi har kunnet ende opp med et *ferdig* produkt og ikke minst et produkt som tilfredsstiller oppdragsgivers *reelle* krav. Dette vil vi gå nærmere inn på i avsnittene som følger. Disse tilbakemeldingene har i aller høyeste grad vært med å forme BIB-*it* og gi det resultatet vi endte opp med.

### 8.7.1 Dokumenterte krav vs. reelle krav

I systemutviklingsprosjekter er det svært ofte et gap mellom det som blir dokumentert i en kravspesifikasjon og det brukeren forventer at han får. Hadde vi valgt en ren fossefallsbasert utviklingsmodell er det mye større sannsynlighet for at vi ved prosjektets slutt ville ha levert et produkt som ikke var det kunden *egentlig* ville ha, dette til tross for at produktet stemmer til punkt og prikke med kravspesifikasjonen.
Dette kan ha flere årsaker. Det er vanskelig for brukeren ved oppstarten av et prosjekt å se for seg hva han vil ha og det er spesielt vanskelig å klare å beskrive det han vil ha på en slik måte at han og utviklerne har *samme* oppfatning av hva det er som skal utvikles.

I et fossefallsprosjekt blir gjerne kravene utdypet, nedtegnet og *frosset* helt i starten av prosjektet; det samme gjelder alle de andre fasene (design, implementasjon...) - hver fase utføres bare en gang og resultatet (i form av artefakter) fryses før man går i gang med neste fase.

### 8.7.2 Veien blir til mens du går...

Den store fordelen med inkrementell utvikling er at prosjektet deles opp i inkrementer slik at alle fasene kjøres en gang i hvert inkrement, og da vil man ha minst like mange muligheter for å rette opp misforståelser (selv etter at de er implementert) som man har

inkrementer. Brukeren får se og prøve programvaren underveis, og han har dermed store muligheter til å påvirke utviklingen og foreslå endringer og rette opp i misforståelser under hele prosessen.

Det var dette vi strebet etter under utviklingen av BIB-*it* - vi var i kontinuerlig samtale med oppdragsgiver og veileder og hadde meget stor nytte av tilbakemeldingene de ga oss etter å ha testet betaversjonene.

## 8.8 Profilering

Som tidligere nevnt så har BIB-*it* blitt godt mottatt av de som har testet applikasjonen. På grunn av at applikasjonen er et Open Source produkt, så har vi lagt ut en link til vår hjemmeside, der både applikasjonen og kildekode kan lastes ned, på diverse Internett forum slik at de som er interessert kan utvikle applikasjonen videre eller forandre koden til sin egen fordel, samt gi oss en tilbakemelding på produktet. Vi har ikke fått den helt store responsen, men av de som faktisk har lastet ned og testet programmet, så har det vært en god mottakelse.
Vi har også fått god tilbakemelding fra overingeniør Dag Langmyhr ved Universitetet i Oslo som var svært begeistret for applikasjonen og ytret også ønske om å kunne bruke applikasjonen ved Universitetet i Oslo. Nå finnes det også mange andre liknende programmer som også har mye av samme funksjonalitet som BIB-*it*, men det er vel ingen reelle programmer som er i særlig utstrakt bruk, bortsett fra JabRef. Dette er et program med mye funksjonalitet, men som har mangler i forhold til enkelte ting som oppdragsgiver setter pris på (se 8.3).

## 8.9 Mulige forbedringer

Vi er godt fornøyd med hva vi har klart å få til i løpet av prosjektperioden, men som med alt annet er det alltid rom for forbedring og videreutvikling. Vi har hatt mye moro gjennom arbeidet med hovedprosjektet, og har derfor lyst til å fortsette å videreutvikle BIB-*it* etter at prosjektet er avsluttet. Som nevnt tidligere er det allerede en del folk som har tatt BIB-*it* i bruk, noe som tyder på at applikasjonen har en fremtid. I avsnittene under nevner vi noe av det vi på dette tidspunktet mener at det kan være mest aktuelt å jobbe videre med.

### 8.9.1 Generelt

- Sortere (på f.eks. bibtexkey) før lagring
- Hardkode utskrift av plainStyle.ini slik at den kan regenereres hvis den skulle bli ødelagt

- Lage visning og redigering av forkortelser (`@String`).
- Legge inn dropdown-bokser i editentrypanelet, også.
- Utføre duplikatsøk når en bruker er i ferd med å legge til en ny referanse og eventuelt gi brukeren tilbakemelding om han vil beholde denne til tross for at den er veldig lik en referanse som finnes fra før
- Duplikatsøk i personregisteret

### 8.9.2 Stilgeneratoren

- bedre bibtex-feilmeldinger i den genererte stilen
- gjøre label-oppbygningen mer fleksibel
- legge inn "snarveier" som gjør det raskere å lage en stil (fra scratch)
  1. gjøre det mulig for brukeren å (for et felt A) kunne velge et annet felt B i en bestemt entrytype for å formatere felt A likt som feltet B, som medfører at formateringsvalgene for felt B presenteres i GUIen i formateringsvalgene for felt A
  2. gjøre det mulig for brukeren å (for en entrytype C) kunne velge en annen entrytype D for å formatere entrytype C likt som entrytype D

### 8.9.3 Duplikatsøk

- vise statistikk over duplikatsøket når søket er utført (fordeling, hvor stor andel som har 100%, 80% osv) (f.eks. grafisk framstilling). Dette vil gi brukeren et raskt overblikk over "tilstanden" til bib-databasen, hvor inkonsistent den er.
- legge til et valg "fjern automatisk alle eksakte duplikater" (dette forutsetter at algoritmen tar en ekstrasjekk på de som fikk 100% for å finne ut som ALLE feltene i disse er like)
- gjøre duplikatsøket uavhengig av standard BibTeX ved å la brukeren kunne definere hvilke felt duplikatsøket skal bruke i sammenligningen i plainStyle.ini
- Midlertidig ekspandere strenger ved dupsøk hvis de ikke allerede er ekspandert.

### 8.9.4 Mulige forbedringer av utviklingsprosessen

- Organisert testingen bedre (laget testplaner)
- Brukt mer tid på å sette opp utviklingsmiljø (med automatisk kompilering, testing og *deploying*)
- Brukt mer tid på å skissere og diskutere løsningsmetoder og fordele oppgavene bedre

# 9 Konklusjon

Et hovedprosjekt er en del av opplæringsprosessen, hvor mye av det vi har lært i de tidligere semestrene skal utprøves i den virkelige verden. Det har vært veldig lærerikt for oss å få erfaring med hvordan et programutviklingsprosjekt fungerer i praksis, og vi har nok erfart at det på mange områder er langt mellom teori og praksis.

Vi har endt opp med et produkt, Bib-*it*, som vi i det store og det hele er veldig godt fornøyd med. Applikasjonen ble raskt mottatt med en rekke positive tilbakemeldinger, og vi har hatt mye moro under hele prosjektperioden. Vi kommer derfor til å fortsette utviklingen av Bib-*it* også etter at hovedprosjektet er avsluttet.

Vi kan med glede konstatere en utmerket samarbeidsvilje og god støtte fra oppdragsgiver og veileder. Vi hadde en god og kreativ dialog i samtalene med oppdragsgiver og veileder på statusmøter og andre mindre formelle møter.

Hver gruppemedlem føler seg betydelig rikere på kunnskap og ideer om hvordan samarbeide på en mer effektiv måte. Et av lyspunktene var selve oppstarten av gruppearbeidet hvor vi kom raskt og effektivt igang med konstruktive forslag om datastruktur, tidsmessig planlegging og allokering av resurser. Vi har også hatt et stort faglig utbytte av hovedprosjektperioden; vi har blitt flinkere til å programmere og lært endel om LaTeX og BibTeX (på godt og vondt), og sist men ikke minst har vi lært masse om programvareutvikling og prosjektorganisering.

# Ordforklaringer

## B

**bib-database** er en tekstfil i BibTeX format. Hver "post" representerer en referanse og feltene (f.eks. author, title, year, ...) gir informasjon om denne referansen., s. I.

**BibTeX name format** Regler for hvordan en streng med ett eller flere navn skal bygges opp for at det skal kunne parses riktig av BibTeX (eks.: "van den Berg, Claus, Jr and Beate Holm" er en streng bestående av to navn i gyldig BibTeX name format.), s. 12.

**BibTeX-stil** en kildekodefil (skrevet i BibTeXs navneløse postfiksstakkespråk) som ut fra et LaTeX-dokument og en bib-fil produserer en bbl-fil med LaTeX-kode. Bbl-filen inneholder en `\bibitem-kommando` for hver referanse som skal skrives ut i referanselisten, og hver `\bibitem-kommando` inneholder den formaterte teksten for den aktuelle referansen slik den skal se ut i referanselisten. Man kan benytte seg av en av standardstilene (plain.bst, unsrt.bst, alpha.bst) eller lage sin egen .bst-fil, s. II.

**Black Box-teste** En måte å teste programvare på. Testingen utføres av analytikere og brukere. De som tester har kun kjennskap til den ytre funksjonaliteten; den interne kodestrukturen er irrelevant., s. 70.

## D

**DOCSTRIP-programmet** Et program som ekstraherer dokumentasjon fra C-lignende kildekodefiler eller skript, og skriver det ut i et gitt format. En DocStrip-blokk er en sekvens av tekstkarakterer og makroer som starter med en backslah., s. 13.

## E

**eXtreme Programming** er en systemutviklingsmodell, s. 3.

## F

**felttype** i Bib-*it*: typen til et BibTeX-felt. Typene som finnes er name, title, year, month, crossref, key og org., s. 47.

## G

**googling** Å google betyr å utføre et Web-søk, vanligvis vha. søkemotoren Google.com, for å finne ut mer om et bestemt tema, s. 13.

**GUI** Graphical User Interface, s. 1.

## J

**JabRef** JabRef er et "open source" bibliografihåndteringsverktøy, s. 1.

## N

**normalisert** I databasesammenheng (relasjonsdatabaser) er normalisering en prosess som eliminerer redundans, organiserer dataene effektivt og gjør dataene mer konsistende. Den formen for normalisering vi først og fremst tenker på her, er *første normalform (1NF)*. En database i 1NF er kjennetegnet ved at et felt bare kan inneholde én (atomisk) verdi., s. 20.

## O

**Open Source** Åpen kildekode, s. 3.

## P

**prettyprinter** Et lite program som tar som input en tekst (f.eks. kode) og gir som output en tekst formatert på en bestemt måte. En prettyprinter for et programmeringsspråk formaterer koden pent med innrykk og konsekvent plassering av krøllparenteser o.l., s. 13.

## R

**regexp-søk** er så søke med et regulaert uttrykk. Et regulaert uttrykk er en tekststreng som inneholder en del spesial tegn i tillegg til vanlige tegn for å beskrive et mønster., s. 15.

## S

**siteringstagg** er i standard BibTeX `\cite{nøkkel}` for lage en refereranse i teksten en merkelapp i teksten og `\nocite{nøkkel}` legger inn en referanse i lista uten merke i teksten., s. 14.

**Subversion** A "version control system" som er en utfordrende konkurrent til CVS (Concurrent Versions System) i "open source" fellesskapet, s. 5.

## T

**TUI** Text-like user interface, s. 68.

## U

**Universitetet i Oslo** Universitetet i Oslo, s. 76.

## V

**viewport view** Det for øyeblikket synlige området i f.eks. en tabell som er lagt inn i et scrollpane - dvs. det utsnittet av tabellen som er synlig., s. 33.

# Referanser

Paul E. Black, editor. *Levenshtein distance*. CRC Press LLC, 1999. URL `http://www.nist.gov/dads/HTML/Levenshtein.html`. In Dictionary of Algorithms and Data Structures [online].

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, West Sussex, England, 1996.

Alexander Factor, Cay S. Horstmann, and Gary Cornell. *Internationalization*, chapter 10. The Java Series. Prentice Hall PTR, seventh edition, 2005.

Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The LaTeX Companion*. Addison-Wesley, second edition, 2004a.

Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *Managing Citations*, chapter 12. In Mittelbach, Mittelbach et al. [2004a], second edition, 2004b.

Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *Bibliography Generation*, chapter 13. In Mittelbach, Mittelbach et al. [2004a], second edition, 2004c.

Oren Patashnik. Designing bibtex styles. webside, 1988. URL `ftp://tug.ctan.org/pub/tex-archive/biblio/bibtex/contrib/doc/btxhak.pdf`.