HOVEDPROSJEKT:

ESTATICA

Educational Statistics Application

FORFATTERE:

Hermund Andre Torkildsen

Kine Miriam Munkelien

Dato: 23.mai - 2002

SAMMENDRAG AV HOVEDPROSJEKT

Tittel:	Estatica	Nr. : 1
	Educational Statistics Application	Dato : 23/5-02
Deltakere:	Hermund Andre Torkildsen	
	Kine Miriam Munkelien	
Veileder:	Harald Liodden	
Oppdragsgiver:	Høgskolen i Gjøvik	
Kontaktpersoner:	Hans Petter Hornæs	
	Tor Slind	
Stikkord	Statistikkapplikasjon, brukervennlighet, Java, Rational Unified Prosess	
Antall sider: 96 + 62	Antall bilag: 12	Tilgjengelighet: Åpen

Høgskolen i Gjøvik har vært oppdragsgiver for dette prosjektet. Vi har laget en statistikkapplikasjon som foreleserne i statistikk ved skolen kan bruke i undervisning. Applikasjonen kan brukes til å demonstrere ulike emner innenfor statistikk, og studenter skal kunne laste ned applikasjonen for eksperimentering på egenhånd.

Det legges vekt på å lage en applikasjon som er lett å bruke. Brukervennligheten settes i høysetet, formålet med applikasjonen er å illustrere statistikk. Enkel navigering og velkjent miljø er viktige stikkord i denne sammenheng.

Utviklingsprosessen er sterkt preget av RUP (Rational Unified Process). Dette er den systemutviklingsmodellen vi har benyttet oss av. Vi har benyttet UML for å standardisere de dokumenter som er produsert underveis i prosjektet.

Besvarelsen består av denne rapporten og tilhørende vedlegg. I tillegg til dette skal CD med applikasjonen, kildekode, kommentarer og dokumentasjon av kode og javadokumentasjonen regnes som en del av besvarelsen.

FORORD

Endelig kan vi sette siste hånd på verket!

Det ble et produkt til slutt, etter mange timer anstrengelse - ikke forgjeves får en vone. Vi gleder oss nå til endelig å levere et ferdig prosjekt, der man tilsynelatende også leverer hele ingeniørutdannelsen attpå. Innleveringen av dette skrivet setter på en måte et slags punktum, der man nå forsiktig skal ta neste skritt.

Arbeidet med dette prosjektet kunne ikke ha forløpet uten visse personer. Først vil vi rette en stor takk til Hans Petter Hornæs og Tor Slind for en fabelaktig vilje til å hjelpe to studenter. Tiden disse to har ofret, har vært uunnværlig. Disse to har, i tillegg til å representere våre oppdragsgivere, vært med på å skape applikasjonen. Dette tør vi påstå, for uten veiledningen innenfor statistikkfaget, hadde vi ikke kunnet gjennomføre prosjektet. Håper også dere har fått noe ut av samarbeidet.

Vi må også få lov til å takke vår veileder, Harald Liodden, som har geleidet oss gjennom prosjektet. Vi er svært glad for at Liodden hadde tiltro til oss og lot oss gjennomføre prosjektet. Faglige råd og veiledning har vært fornødne og godt anvendt.

Takk også til U2 som har holdt oss med selskap utover kveldene.

Et problem klart stilt er et problem halvt løst.
Charles Kettering

Kine Munkelien

Hermund Torkildsen

INNHOLDSFORTEGNELSE

Kapittel 1

1 INNLEDNING

Comment is free, but facts are sacred.
C.P. Scott

Though this be madness, yet there is method in't.
William Shakespeare

… the wisest prophets make sure of the event first.
Horace Walpole

I claim not to have controlled events, but confess plainly that events have controlled me.
Abraham Lincoln

It is quite a three-pipe problem.
Sir Arthur Conan Doyle

1.1 Oppgaven

1.1.1 Bakgrunn

Endelig har vi kommet dithen i ingeniørstudiet at hovedprosjektet er gjennomført. Dette betyr slutten på et 3-årig studie ved Høgskolen i Gjøvik, og man kan endelig puste lettet ut og ta et skritt videre.

Vi har blitt presentert mange ulike oppgaver for mulige hovedprosjekter, og flere har virket svært interessante. Allikevel har vi valgt å definere vår egen oppgave. Ideen har vi hatt lenge - det manglet bare å ta initiativet. Og det gjorde vi da vi banket på døren til en av statistikklærerne, Hans Petter Hornæs, ved Høgskolen i Gjøvik.

Vi presenterte vår ide for Hornæs, og til all vår glede ble vi møtt med en godvillig holdning, som vi i dag er uendelig takknemlige for. Også Tor Slind, en annen statistikkforeleser ved skolen, var mer en villig til å stille som oppdragsgiver. Dermed hadde vi to oppdragsgivere som har stilt opp annenhver uke, og ikke nølt med å ofre tid utenom planen.

Nå manglet det bare å få prosjektet gjennom nåløyet, og dette skjedde brått etter at vi fikk tildelt vår veileder, Harald Liodden, som hadde troen på oss og klippet den røde snoren. Liodden stilte også opp annenhver uke for veiledning og rådgiving, men har også gitt oss nok spillerom til å nyte arbeidet.

Så hva gikk ideen ut på – ideen vi for første gang luftet til Hornæs? Vi hadde svært lyst til å kombinere to fagområder som har vært sentrale i ingeniørstudiet; data og realfag. Derfor gikk ideen ut på å lage en statistikkapplikasjon som kunne brukes i undervisning for statistikk. Her lå det ubegrensede muligheter, så vi måtte sette oss ned å sette rammer og begrensninger for å få definert oppgaven.

1.1.2 Definering av oppgaven

Det skal utvikles en applikasjon som skal brukes i undervisning for statistikkfagene ved Høgskolen i Gjøvik. Systemet skal kunne brukes i undervisningen for å illustrere ulike emner innenfor statistikkfaget, og man skal raskt kunne generere eksempler for bruk i forelesningstimene. I tillegg skal systemet kunne brukes av studenter for eksperimentering på egenhånd. Målet for vårt prosjekt er å lage et *brukervennlig* program, der man ikke drukner i kompliserte formler og datatekniske detaljer. Innholdet i applikasjonen skal basere seg på pensum i faget statistikk, F166A.

Prosjektet skal gjennomføres i forbindelse med hovedoppgave for siste studieår på dataingeniørutdanningen ved Høgskolen i Gjøvik. Mål for faget Hovedprosjekt, H102G, står beskrevet i ”Studiehåndbok 2001-2002”. Prosjektet skal utgjøre et arbeidsomfang på 6 vekttall (ca. 360 arbeidstimer per gruppemedlem).

Prosjektet gjennomføres med tanke på oppdragsgivers ønsker og behov. Som systemutviklere er vår oppgave å lage et system som tilfredsstiller oppdragsgiver og brukerne sine krav til et slikt system.

Det legges stor vekt på brukervennlighet, og dette må gjennomsyre hele prosjektet. Vi legger opp til en GUI som er lett å forstå og bruke. Kompleksiteten i systemet og de ulike metodene som skal benyttes, bør i størst mulig grad ligge skjult for sluttbrukerne. Brukerne er interessert i bare statistikkdelen.

1.1.3 Eksisterende løsninger

Det finnes selvfølgelig mange programmer ute på markedet, der matematikk- og statistikkverktøy kan brukes. Eksempler på slike applikasjoner er Maple, Matlab og Excel. Disse kommersielle verktøyene, derimot, forutsetter kunnskaper mange studenter ikke har. Programmene inneholder elementer som kan virke forstyrrende på en læringsprosess. Disse verktøyene krever også at man lærer seg å bruke programmet. Å generere eksempler i Maple ”on the fly” er slett ikke så enkelt.

Vi skal *ikke* lage en forenklet versjon av disse verktøyene. Vi skal utvikle en applikasjon som er *enkel* å bruke, og der studenten raskt kan få respons fra systemet. Man skal ikke behøve å gjennomgå en opplæringsprosess for å bruke applikasjonen – basiskunnskaper i statistikk og data skal være nok. Det er et stort poeng å skjule kompleksiteten bak systemet for brukeren.

1.2 Målgruppe

1.2.1 Målgruppe for rapporten

Rapporten skal kunne leses av studenter på vårt nivå. Vi håper at vårt arbeid vil være gjenstand for senere prosjekter, og andre systemutviklere (som da er studenter) vil utgjøre en målgruppe. Sensor og veileder vil selvfølgelig også være en målgruppe for denne rapporten.

Vi har forsøkt å forklare begreper underveis i rapporten, men vi har måttet sette begrensninger på detaljnivået. Derfor må vi forutsette at leser av denne rapporten har noen basiskunnskaper i statistikk og matematikk.

Kapittel 1 og 2 skal kunne leses av alle, mens kapitlene utover krever kunnskaper innenfor programmering og systemutvikling. Vi forutsetter også kunnskaper innen Java, men studenter på vårt nivå skal ha nødvendige kunnskaper for å lese denne rapporten.

For å sørge for at prosjektrapporten er rettet mot riktig målgruppe, har vi utført en analyse av mottakergruppen. Denne mottakeranalysen er å finne i vedlegg **E**. Vi mener å ha lagt oss på et riktig nivå i forhold til mottaker, og vi har forsøkt å belyse de interessante og viktige momentene.

1.2.2 Målgruppe for applikasjonen

Vi har to hovedgrupper av brukere:

- Statistikkforeleserne ved Høgskolen i Gjøvik
- Statistikkstudenter ved Høgskolen i Gjøvik

Statistikkforeleserne: Dette er våre oppdragsgivere, som også vil komme til å benytte seg av systemet. Disse brukerne er høyt kvalifiserte innenfor statistikk og matematikkfagene og skal selvsagt ikke ha noen læringseffekt ut av systemet. Denne brukergruppen vil bruke systemet i demonstrasjoner og generering av eksempler ”on the fly” i undervisning. Statistikkforeleserne vil også ha god kjennskap til hvordan systemet fungerer. Har god kjennskap til bruk av datamaskiner.

Studenter: Denne brukergruppen utgjør den største andelen. En student vil bruke systemet for eksperimentering i statistikkfaget, og skal på denne måten ha en viss læringseffekt. Disse brukerne har noe erfaring med statistikkfaget, og er kjent med grunnleggende begreper. Har god erfaring med datamaskiner.

I tillegg legges det opp til at systemet skal kunne videreutvikles, og på dette nivået vil andre systemutviklere og programmerere være en målgruppe.

Vi har også utført en analyse av målgruppen for applikasjonen, og denne kan finnes i vedlegg **D**.

1.3 Studentenes faglige bakgrunn

Studentene på dette hovedprosjektet har tre års utdannelse ved høgskolen i Gjøvik, og 1-årig realfag studium ved samme høgskole. Noen av de viktigste fagene som kan relateres til prosjektet er Programutvikling (Java), Algoritmiske Metoder 1 og 2, Systemutvikling 1 og 2, Objektorientert Programmering, Grafisk Brukergrensesnitt og Økonomi og Prosjektstyring. Som valgfag har vi også et 5 vekttalls studium i Statistikk.

Vi har ikke vært med på liknende prosjekter tidligere. Allikevel har vi følt at vi har gode forutsetninger for å gjennomføre et slikt prosjekt. Vi har ved tidligere fag gjennomført to mindre prosjekter, og det har ga oss noen pekepinner på hva vi hadde i vente.

1.4 Arbeidsformer

Vårt valg av systemmodell har vært ryggraden gjennom hele prosjektet. Etter nøye analyse og diskusjoner falt det ned på RUP, Rational Unified Prosess. Hovedinndeling av prosjektets prosess har vært i henhold til denne modellen.

RUP definerer 4 hovedinndelinger med påfølgende milepæler slik:

- Inception, påfølgende milepæl LCO (Lifecykle Objective Milestone)
- Elaboration, påfølgende milepæl LCA (Lifecykle Architecture Milestone)
- Construction, påfølgende milepæl IOC (Initial Operational Capability Milestone)
- Transition, påfølgende milepæl PR (Product Release Milestone)

Hver fase har igjen vært oppdelt i iterasjoner, som av og til igjen har inneholdt flere aktiviteter. I hver iterasjon la vi vekt på å utvikle Use Case. Vi fulgte ingen "Waterfall"-modell, men utviklet inkrementer, der vi hele tiden hadde muligheter for å gå tilbake, rette opp feil og gjøre forbedringer.

1.4.1 Hvordan vi fulgte RUP

Inception: Målet i denne fasen er å få en oversikt over prosjektet, her er det ikke vanlig å dele opp i iterasjoner. Da vi nådde denne milepælen hadde vi et klart definert mål. De viktigste Use Casene var identifisert, og vi hadde et førsteutkast av Use Case-modellen. Prosjektplanen som vi skulle følge utover i prosessen var nøye planlagt, estimert og definert. Vi hadde også rukket å utviklet noen GUI-prototyper da denne milepælen var nådd.

Elaboration: Denne fasen delte vi opp i fire iterasjoner. I løpet av denne fasen hadde vi funnet en fordelaktig systemarkitektur, hvor vi landet på en 3-lags arkitektur der det laveste laget var filsystemet (ikke database). Vi hadde fått satt opp en grundig kravspesifikasjon i nært samarbeid med oppdragsgiverne. Systemmodellene for grovdesign/analyse var ferdig definerte, og vi hadde fått satt opp et ønskelig brukergrensesnitt ved hjelp av reelle Use Case og prototyping. I andre iterasjon begynte vi å implementere, og da tok vi for oss modulen som gikk på generering av datasett for envariable data. I tredje iterasjon implementerte vi modulen for grafiske representasjoner. Dette løste vi først etter at vi måtte omstrukturere store deler av datastrukturen. I fjerde og siste iterasjon i denne fasen implementerte vi funksjonalitet for filhåndtering.

Construction: Denne fasen delte vi opp i to iterasjoner. Da fasen var avsluttet hadde vi en betaversjon av applikasjonen og utførte grundig testing av denne. I construction-fasen skal man implementere såkalte frivillige Use Case – de som ikke er så høyt prioritert. Vi valgte derfor å implementere modulen simulering av konfidensintervall i første iterasjon og implementering av tovariable data i andre.

Transition: Vi er nå midt inni denne fasen. Her er målet å få levert produktet og dokumentasjon til avtalt tid. Det betyr at denne fasen består mest av testing og integrering av hele systemet, i tillegg til at det blir en del rapportskriving. Vi har fått nok tilbakemelding fra oppdragsgiverne om mulige utvidelser, slik at vi har kunnet kartlagt disse. Utvidelsene har vi skrevet om i kapittel 8, avsnitt 8.4.

Sammenlikner vi denne fremdriften med vår fremdriftsplan, kan vi si oss godt fornøyd med gjennomføringen av prosjektet. Vi har holdt tidsfrister og oppnådd kravene. Fremdriften har vi dokumentert godt underveis, og dokumentasjonen er å finne i vedleggene:

- Vedlegg F: Gantt – diagram
- Vedlegg G: Statusrapporter
- Vedlegg J: Timelogg
- Vedlegg K: Loggbok

1.4.2 Andre arbeidsformer vi har benyttet oss av

I tillegg til å følge RUP, har vi benyttet oss av andre arbeidsformer underveis slik at det har vært mulig å oppfylle kravene denne systemmodellen stiller. Vi har hatt en god og strukturert arbeidsform, hvor begge gruppemedlemmene har vært oppdatert til enhver tid og derfor kunnet fulgt hele prosessen.

Vi har hatt et nært samarbeid med både veileder og oppdragsgivere underveis. I tillegg til å stille opp på møte annenhver uke, har det heller ikke vært noe problem å få veiledning og hjelp på tidspunkt i mellom disse møtene. Det har også vært aktuelt oppsøke hjelp hos andre forelesere (systemutvikling og Java) underveis i prosjektet, og dette har som regel aldri vært noe problem.

Vi har benyttet oss flittig av ulike typer litteratur. Dette gjelder både under research (mye statistikk ble lest), systemutvikling, design og implementering. Når man leser mye forskjellig litteratur om et emne, er det naturlig at man oppdager motstridende teorier og påstander. Vi har derfor vært nødt til å stille oss kritiske til det vi har lest, og prøvd å sile ut det som har passet best til vårt prosjekt. Mange av algoritmene og metodene måtte vi kode selv for at de skulle passe best mulig til vårt formål.

Vi er godt fornøyd med valget av våre arbeidsformer. Det har vært en krevende og utfordrende periode, som vi føler å ha mestret bra. Når det gjelder bruken av arbeidsmetodikk er det nok noen momenter vi ville ha gjort annerledes, spesielt kommentering av kode underveis. Det hendte flere ganger når vi var på gli og iveren tok overhånd, at mange linjer med kode ble produsert, men dessverre ikke så godt dokumentert. Dette har vi fått igjen nå i sluttfasen.

1.5 Organisering av rapporten

Vi har forsøkt å ha en logisk oppbygging av rapporten. Det skal være mulig å følge utviklingen av systemet trinn for trinn. Slik kan man følge en rød trå gjennom rapporten. I tillegg har vi også lagt opp til at rapporten kan brukes som et oppslagsverk. Man skal ikke behøve å lese hele kapitler for å få med seg noe. En leser av rapporten skal kunne slå opp på for eksempel design av en eller annen pakke, uten å måtte lese alt om alle andre pakker i tillegg.

Kapitlene om krav, design og implementering følger noenlunde samme struktur. Dette gjør at leseren kan identifisere seg med måten vi har representert prosjektet på. Kravspesifikasjonen tar for eksempel seg av krav for Estatica (selve applikasjonen) og deretter krav for statistics (en uavhengig statistikkpakke). Slik er også kapitlene om design og implementering bygd opp. Vi deler applikasjonen opp i mindre og mindre deler, og leser skal kunne følge denne prosessen. Deretter bygges delene sammen, og vi står igjen med en konklusjon til slutt.

Krav: Kapittel 2 tar for seg kravene til applikasjonen som skal utvikles. Kravene er formulert på en slik måte at det skal være mulig å finne ut om mål og krav er oppfylt. Vi har derfor søkt å gjøre kravspesifikasjonen problemorientert og testbar. Dette kapitlet tar for seg *hva* vi skal lage. Vi har puttet de funksjonelle kravene i tabeller, slik at vi enklere kan henvise til kravene utover i rapporten. Til slutt har vi skrevet litt om arbeidet med kravspesifikasjonen.

Design: Designfasen søker å løse problemene skissert i kravspesifikasjonsdokumentet. Vi har delt designkapitlene opp i tre deler: Kapittel 3(*Grovdesign og analyse*, som tar for seg hovedarkitekturen av systemet), kapittel 4 (*GUI-design og brukerinteraksjoner*, som tar for seg brukergrensesnittet og hvordan bruker kan kommunisere med systemet) og kapittel 5 (*Systemdesign*, som tar for seg systemets struktur i en viss detalj). Disse kapitlene tar for seg *hvordan* vi har tenkt å løse oppgaven.

Implementasjon: Implementeringen setter løsningen av oppgaven ut i praksis. Vi har her belyst noen eksempler på hvordan vi implementerte ulike moduler og komponenter. Kapittel 6 inneholder derfor noen eksempler på kode, og vi har vært bevisste på bruk av diagrammer for å illustrere og gjøre lesningen lettere.

Testing: Testing og kvalitetssikring har vi snakket om gjennom hele rapporten. Vi har forsøkt å begrunne de valg vi har gjort underveis. Allikevel har vi valgt å samle viktige momenter og arbeidsmetoder rundt kvalitetssikring og testing i et eget kapittel, 7.

Avslutning: I kapittel 8 har vi prøvd å oppsummere arbeidet og satt resultatet inn i et større perspektiv. Vi har vært kritiske til vårt eget arbeid, men helt uten å underslå at vi faktisk er godt fornøyd med resultatet. Vi har også antydet mulige utvidelser av systemet. Det er også her viktig å nevne at vi ikke har skrevet om alternative løsninger og problemer som oppstod underveis

i rapporten. Slik informasjon synes vi ikke hører hjemme i de kapitlene som beskriver systemet – det kan virke forstyrrende på andre utviklere som skal sette seg inn i hva som har blitt gjort. Denne diskusjonen har vi lagt til avslutningen. Vi har også i dette kapitlet besvart problemstillingen og diskutert om kravene er oppfylt.

Vi har selvfølgelig blitt tvunget til å begrense oss kraftig når vi skrev denne rapporten. Vi har ikke hatt muligheter (eller ønsker) om å belyse alle aspekter ved det ferdige systemet og prosessen. Det kan dog ikke legges skjul på at vi gjerne skulle skrevet mye mer. Mange detaljer, da spesielt under implementasjonen, skulle vi gjerne ha belyst ennå nærmere. Men vi har måttet holde oss på et overordnet og prinsipielt nivå, slik at hovedpunktene blir belyst og ikke drukner i detaljer.

God fornøyelse!

Kapittel 2

2 KRAV

My object all sublime
I shall achieve in time.
W.S. Gilbert

When faced with a decision, I always ask, ”What would be the most fun?”.
Peggy Walker

It is a capital mistake to theorize before one has data.
Arthur Conan Doyle

Get your facts first, and then you can disort them as much as you please.
Mark Twain

2.1 Hva skal vi lage

Vi har hatt mange samtaler med våre oppdragsgivere, og det har vært svært nødvendig for å kunne avdekke ønsker og krav for en statistikkapplikasjon. I avsnitt 2.5 har vi beskrevet kravspesifiseringsprosessen. Oppdragsgiverne har hatt mange konkrete og godt definerte krav. Allikevel har vi erfart at arbeid med å avdekke krav er et delikat stykke problem.

Vi har definert følgende moduler som må utvikles:

- **Tekst editor:** Det må lages en tekst editor der brukeren kan skrive inn sitt datasett i form av tall. En slik tekst editor skal være en tabell der bruker raskt og enkelt kan taste inn tall i cellene.
- **Filhåndtering:** Tekst editoren må ha funksjonalitet for å lagre og hente datasett til og fra fil. Dette bør gjøres på en måte som bruker er godt kjent med fra før.
- **Beregninger på et datasett:** På bakgrunn av et datasett som brukeren har tastet inn, skal det raskt bregnes ulike måldata som for eksempel gjennomsnitt, median, varians og standardavvik.
- **Tegning av diagrammer og grafer:** Bruker skal kunne få grafiske representasjoner av et datasett i form av histogrammer, stolpediagrammer og eventuelt spredningsplott.
- **Tegning av teoretiske fordelinger:** Bruker skal kunne få grafiske representasjoner av teoretiske fordelinger i form av stolpediagrammer eller kurver.
- **Utvikling av datastrukturen:** Et datasett må representeres på en fornuftig måte i datamaskinen. Det stilles krav til en god datastruktur som er effektiv og robust.
- **GUI, brukergrensesnitt:** Brukergrensesnittet bør være så enkelt som overhode mulig. Kompleksiteten bak systemet bør skjules for brukeren, selv om det forutsettes grunnleggende kunnskaper innenfor statistikk. Vi ønsker at bruker skal navigere minst mulig for å få frem et resultat.
- **Utvikling av randomgeneratorer:** For de ulike fordelingene som skal implementeres, skal det utvikles randomgeneratorer for å automatisk generere et datasett. Dette datasettet skal da legges inn i tekst editoren/tabellen.
- **Utviklingen av en statistikkpakke:** Det må utvikles en egen statistikkpakke som har ansvaret for beregninger av data og representasjoner av teoretiske fordelinger. Denne pakken skal inneha all generell funksjonalitet for det statistiske innholdet.
- **Simulering av konfidensintervaller:** Bruker skal ha mulighet for å simulere konfidensintervaller og få disse representert grafisk på skjerm.
- **Enkel hjelpefunksjonalitet:** For hvert vindu skal det finnes en enkel hjelpefunksjonalitet der bruker raskt skal kunne lese seg til hva som kan gjøres i gjeldende vindu.

- **Utvikling av to variabel modeller:** Dette punktet medfører utvidelser på de fleste punktene ovenfor. Om vi gjør gode valg med hensyn på datastruktur og GUI, kan applikasjonen ”enkelt” utvides til å også håndtere to variabel modeller. Dette punktet har foreløpig ikke høy prioritet, da vi ennå ikke vet om vi rekker det. Det kan uansett være et mål for senere utvidelser.

Som man kan se av punktene over, har vi valgt å utvikle en egen statistikkpakke som innehar all funksjonalitet for det statistiske innholdet i applikasjonen. På denne måten blir applikasjonen todelt:

- **Estatica** – er selve applikasjonen og brukergrensesnittet som skal utvikles.
- **statistics** – er en fullstendig uavhengig statistikkpakke utviklet i Java.

Applikasjonen, Estatica, vil måtte benytte seg av statistikkpakken. Statistikkpakken vil være helt uavhengig av applikasjonen som skal utvikles, og her ligger funksjonalitet for det statistiske innholdet. Det er et stort poeng i seg selv å understreke uavhengigheten mellom applikasjonen og statistikkpakken.

Det er flere grunner til at vi har valgt å dele systemet opp i to selvstendige deler:

- Støtter i aller høyeste grad gjenbruk og videreutvikling. Statistics vil være en egen pakke som andre applikasjoner kan benytte seg av. En slik pakke vil også kunne være et objekt for videre utvidelser i form av for eksempel flere fordelinger.
- Vi oppnår god og oversiktlig struktur i vårt design av applikasjonen, hvilket også gir andre systemutviklere god anledning til senere utvidelser av systemet.
- Utvikling i Java legger opp til oppsplitting i pakker. En slik pakke kan enkelt dokumenteres ved hjelp av javadoc. Dokumentasjon ved hjelp av javadoc er godt kjent for utviklere i Java, og dokumentasjonen for pakken får en allment kjent form og oppbygning.
- Vi baserer oss på et objektorientert fundament, og denne utviklingsmåten oppmuntrer til oppsplitting av funksjonalitet og ansvar i form av pakker og klasser.
- Prinsippet om oppdeling støttes av flere av de kjente GRASP patterns, blant annet: Expert-pattern, High Cohesion (høy styrke), Low Coupling (lav kobling), Controller, Don't Talk To Strangers (*henviser til kilde: Applying UML and patterns, Craig Larman*).

Vi har skrevet to kravspesifikasjoner; en for applikasjonen Estatica (avsnitt 2.3) og en for pakken statistics (avsnitt 2.4). Kravspesifikasjonen for statistikkpakken beskriver det matematiske/statistiske innholdet i applikasjonen, og trenger ikke å leses for å få innblikk i hvordan systemet fungerer. Vi har allikevel funnet det nødvendig å skrive en egen kravspesifikasjon for statistikkpakken, da denne pakken kan være gjenstand for senere utvidelser.

2.2 Systemets omgivelser

Systemet skal utvikles i Java, og vil innenfor disse begrensninger være plattform - uavhengig. Vi benytter oss av Java 2 Standard Development Kit versjon 1.4.0, og for å kunne kjøre applikasjonen korrekt, må derfor maskinen ha installert software for å kjøre denne versjonen.

Applikasjonen vil i all hovedsak kjøre på en Windows- eller Linux maskin, og den skal kunne kjøre på en 166 Mhz x86 PC med 32 MB RAM. Systemet vil for det meste kjøre på maskiner med skjermstørrelse på 1024x768 piksler, men minstekravet vil være en skjermstørrelse på 800x600 piksler.

2.3 Kravspesifikasjon for applikasjonen Estatica

For å beskrive kravene for systemet Estatica, har vi valgt å først definere overordnede funksjonelle krav. Dette er de kravene som oppdragsgiverne og vi har kommet frem til. Oppdragsgiverne har kommet med ønsker og krav for hvordan systemet bør fungere, og vi har prøvd å avdekke misforståelser og tvetydige krav. Det er en god måte å skrive de funksjonelle kravene i en nummerert tabell, slik at vi kan henvise til disse senere. De kritiske operasjonelle kravene er også beskrevet.

Vi har identifisert alle Use Casene i denne fasen, og høynivå beskrivelse av disse er lagt med i kravspesifikasjonen.

2.3.1 Overordnede funksjonelle krav

Tekst editor/tabell

K1	Bruker skal ha en tekst editor/tabell tilgjengelig.
K1.1	Bruker skal kunne opprette et nytt ark. Bruker skal ha mulighet til å velge mellom: • Envariabelt diskre ark • Envariablet kontinuerlig ark • Tovariabelt ark
K1.2	Bruker skal kunne legge inn et datasett i et av de tilhørende arkene (K1.1).
K1.2.1	Envariabelt diskret ark skal bare godta heltall.
K1.2.2	Envariabelt kontinuerlig ark skal bare godta reelle tall.
K1.2.3	Tovariabelt ark skal bare godta reelle tallpar.
K1.3	Bruker skal kunne legge inn et datasett manuelt. Dette skal skje ved at bruker taster inn tall i en tabell.
K1.4	Bruker skal ha muligheten for å be systemet generere et ønsket antall tall ut i fra en gitt fordeling med gitte parametre. Disse fordelingene skal være: Uniform diskret fordeling, Normalfordeling, Eksponenitalfordeling, Poissonfordeling, Binomisk fordeling.
K1.5	Bruker skal ha mulighet til å rense (fjerne all data) et ark.
K1.6	Bruker skal ha mulighet til å lukke et ark.
K1.7	Tekst editor/tabellen skal minimum ha 2000 celler.

Filhåndtering

K2	Det skal finnes funksjonalitet for filhåndtering.
K2.1	Brukeren skal kunne be systemet lagre et datasett på fil. Det skal gis mulighet for å velge filnavn og plassering på disk.
K2.2	Brukeren skal kunne hente et datasett fra fil. Det skal gis mulighet for å navigere i filsystemet og åpne ønsket fil.

Beregninger på et datasett

K3	Det skal beregnes og displayes måldata for et datasett.
K3.1	Ut i fra et envariabelt datasett skal systemet regne ut følgende måldata: • Virkelige data: gjennomsnitt, median, varians, standardavvik, antall, maksimumsverdi, minimumsverdi. Dette er uavhengig av om datasettet er skrevet inn manuelt eller om det er generert av systemet • Teoretiske data: type fordeling med tilhørende parametre.
K3.2	Ut i fra et tovariabelt datasett skal systemet regne ut følgende måldata: GjennomsnittX, gjennomsnittY, alpha, beta, empirisk korrelasjon, antall par.

Tegning av diagrammer og grafer

K4	Bruker skal ha mulighet til å få en grafisk representasjon av et datasett til skjerm.
K4.1	For et envariabelt diskret datasett, skal systemet tegne et stolpediagram.
K4.2	For et envariabelt kontinuerlig datasett, skal systemet tegne et histogram.
K4.3	For et tovariabelt datasett, skal systemet tegne et spredningsplott og en regresjonslinje utregnet fra disse punktene.

Tegning av teoretiske fordelinger

K5	Brukeren skal kunne be systemet tegne en grafisk representasjon av en teoretisk modell med gitte parametre til skjerm. Den teoretiske modellen skal displayes over den virkelige modellen.
K5.1	For et envariabelt diskret datasett skal bruker ha mulighet til å få displayet både diskre og kontinuerlige fordelinger ved å angi ønsket fordeling med tilhørende parametre.
K5.2	For et envariabelt kontinuerlig datasett, skal bruker ha mulighet til å få displayet kontinuerlige fordelinger ved å angi ønsket fordeling med tilhørende parametre.
K5.3	For et tovariabelt datasett, skal bruker ha mulighet til å få displayet teoretiske regresjonslinjer ved å angi ønsket alpha og beta.

Simulering av konfidensintervaller

K6	Bruker skal ha mulighet til å simulere konfidensintervaller. Bruker skal selv kunne velge fordeling, type konfidensintervall og konfidensnivå.
K6.1	Det skal være mulig å generere 100 konfidensintervaller.
K6.2	Bruker skal ha mulighet til å legge forsinkelse (0-1 sekund) mellom hvert simulerte konfidensintervall..

Enkel hjelpefunksjonalitet

K7	Brukeren skal kunne åpne et hjelpevindu som kort forklarer hva brukeren kan gjøre. Det skal foreligge denne type funksjonalitet i alle ulike vinduer. Et alternativ kan være å lage tooltip-tekster.

2.3.2 Krav til grafisk brukergrensesnitt

Det stilles høye krav til det grafiske brukergrensesnittet, spesielt med tanke på brukervennligheten. Applikasjonen skal være enkel og lett forståelig. Det skal være intuitivt og velkjent hvordan man skal navigere seg rundt i applikasjonen. For å oppnå dette ønsker oppdragsgiver at applikasjonens brukergrensesnitt minner om Windows. Dette medfører at bruker vil være fortrolig med denne type utseende. Vi må derfor benytte oss av "Windows-LookAndFeel" definert i Java biblioteket, og vi vil følge denne standarden.

En student er interessert i å bruke systemet i den ene hensikt å eksperimentere med statistikk på egenhånd. Det er da ønskelig at dette skal være en lettfattelig og bekvem opplevelse. Vi må derfor sørge for at applikasjonen er lett å manøvrere i og at bruker kan få informasjon underveis. Informasjonen kan da være i form av advarsler og feilmeldinger. Dette skal være med på å gjøre programmet mer brukervennlig. I tillegg er det ytret ønsker fra oppdragsgiverne at systemet skal ha en enkel brukermanual. Vi vil derfor gi en bruker mulighet for å få hjelp til hvordan bruke systemet. Det skal også settes ToolTip-Tekst på alle knapper og felter.

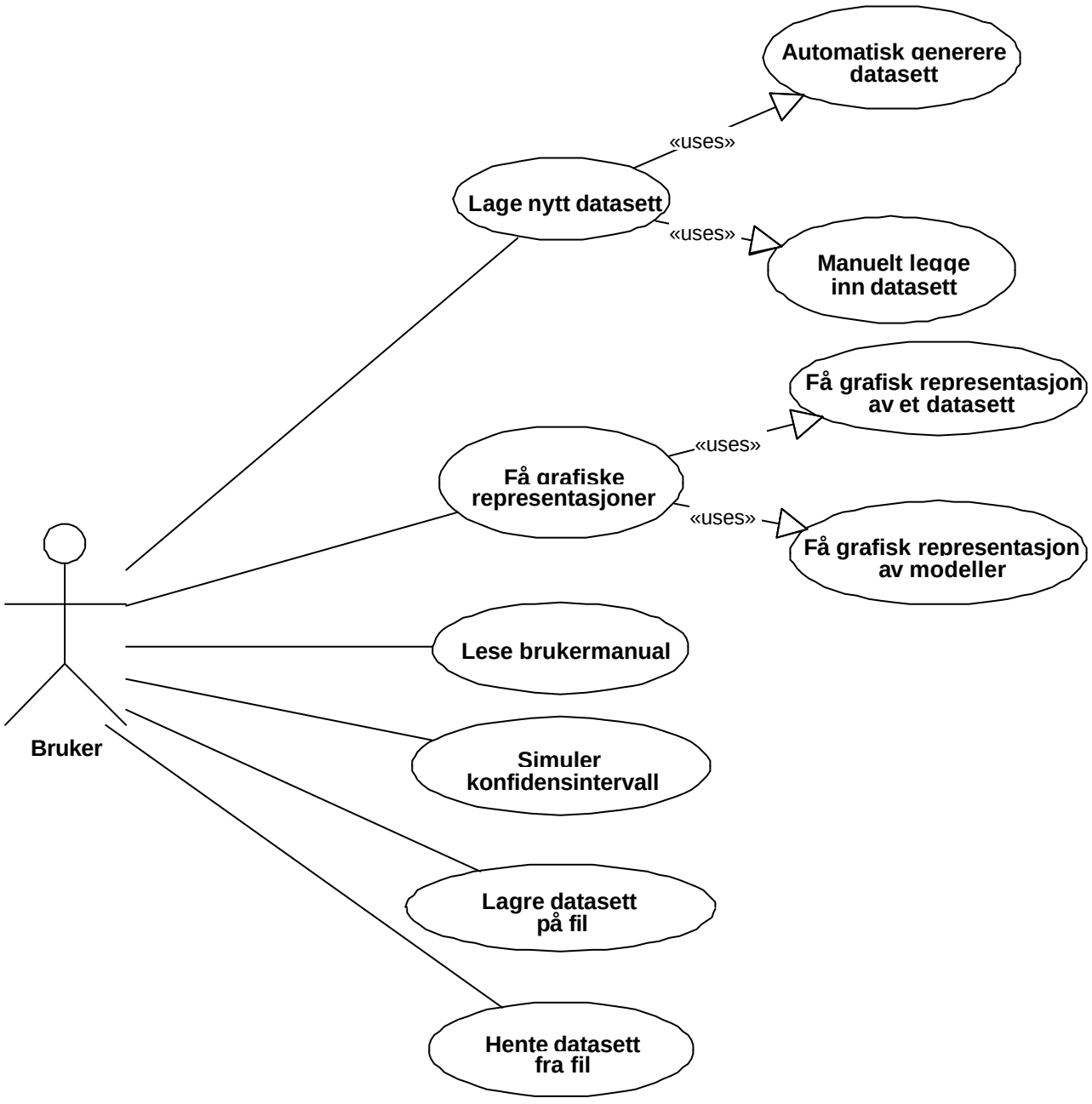
For å øke brukervennligheten ytterligere er det hensiktsmessig med en verktøylinje i tillegg til en standard hovedmeny. En slik verktøylinje skal inneholde de aller mest brukte funksjonaliteter som gjelder for *hele* vinduet. Knappene på verktøylinjene skal ha forklarende ikoner som intuitivt forteller brukeren hvilken funksjon de har. Det kan også være aktuelt med Pop-up-menyer for spesielle objekter som ligger i vinduet. Disse menyene vil da kun gjelde for det aktuelle objektet.

Som foreleser er det ønskelig å raskt kunne demonstrere og generere eksempler i forelesningene. Det er viktig at studentene kan følge med på det essensielle og ikke bli distrahert av andre innfløkte programteknisk spissfindigheter. Det er derfor av stor betydning at tunge regneoperasjoner og annen kompleksitet bak systemet holdes skjult for brukeren.

På dette tidspunktet er fortsatt ikke alle krav til brukergrensesnittet avklart/oppdaget. Av den grunn er det vesentlig at vi kan føre en kontinuerlig prosess med prototyping hvor vi har et nært samarbeid med oppdragsgivere underveis. Dette vil føre til at mange, ennå ikke avdekte krav kan oppdages.

2.3.3 Use Case diagram

Ut i fra de overordnede krav i avsnitt 2.3.1 har vi identifisert Use Casene. Denne modellen beskriver på en god måte hvordan brukeren skal kunne kommunisere med systemet. Vi følger standard UML-notasjon.



Figur 2.1 Use Case diagram for Estatica

2.3.4 High-level UseCase-beskrivelser

For å få bedre oversikt over hvordan brukeren kommuniserer og gir input til systemet, har vi beskrevet Use Casene ved hjelp av UML-standard. De lavnivå Use Casene kommer ikke før i designfasen, men de høynivå beskrivelsene illustrerer på en god måte hva bruker forventer av applikasjonen.

High-level UseCase-beskrivelse for ” Lage nytt datasett ”:

Use Case	**Lage nytt datasett**
Aktør	Bruker
Type	Primært og essensielt use case
Beskrivelse	Bruker ber systemet opprette et nytt ark(tabell). Det er tre typer ark å velge mellom(envariabelt diskret, envariabelt kontinuerlig, tovariabelt). Buker kan velge mellom å legge inn datasettet manuelt eller å få det automatisk generert.

High-level UseCase-beskrivelse for ” Manuelt legge inn datasett ”:

Use Case	**Manuelt legge inn datasett**
Aktør	Bruker
Type	Primært og essensielt use case
Beskrivelse	Det foreligger et ark (tabell), og bruker ønsker å manuelt legge inn et datasett. Bruker legger et og et tall inn i hver sin celle i tabellen, og melder ifra til systemet når han/hun er ferdig. Systemet lagrer datasettet.

High-level UseCase-beskrivelse for ” Automatisk generere datasett ”:

Use Case	**Automatisk generere datasett**
Aktør	Bruker
Type	Primært og essensielt use case
Beskrivelse	Det foreligger et tomt ark (tabell). Bruker ønsker å få generert et datasett. Bruker ber systemet generere et datasett med gitt fordeling og tilhørende parametre. Datasettet lagres i systemet for videre bruk.

High-level UseCase-beskrivelse for ” Få grafisk representasjon av datasett ”:

Use Case	**Få grafisk representasjon av datasett**
Aktør	Bruker
Type	Primært og essensielt use case
Beskrivelse	Det foreligger et datasett. Bruker ønsker å få en grafisk representasjon av dette datasettet. Systemet displayer en grafisk representasjon.

High-level UseCase-beskrivelse for ” Få grafisk representasjon av modeller ”:

Use Case	**Få grafisk representasjon av modeller**
Aktør	Bruker
Type	Primært og essensielt use case
Beskrivelse	Bruker ønsker å få grafisk representasjon av en teoretisk modell. Bruker setter parametre for ønsket modell. Systemet displayer modellen.

High-level UseCase-beskrivelse for ” Legge datasett på fil ”:

Use Case	**Legge datasett på fil**
Aktør	Bruker
Type	Primært og essensielt use case
Beskrivelse	Det foreligger et datasett, og bruker ønsker å lagre dette datasettet på fil. Bruker får mulighet til å navngi filen og plassering, og systemet lagrer filen på disk.

High-level UseCase-beskrivelse for ” Hente datasett fra fil ”:

Use Case	**Hente datasett fra fil**
Aktør	Bruker
Type	Primært og essensielt use case
Beskrivelse	Bruker ønsker å hente et lagret datasett fra fil. Bruker velger hvilken fil han/hun vil hente, og systemet lagrer datasettet klart til bruk.

High-level UseCase-beskrivelse for ” Lese brukermanual”:

Use Case	**Lese brukermanual**
Aktør	Bruker
Type	Primært og essensielt use case
Beskrivelse	For å få hjelp til å bruke programmet, kan bruker be om å få lese brukermanual. Systemet displayer brukermanual. Bruker stenger brukermanualen når han/hun er ferdig.

High-level UseCase-beskrivelse for ” Simuler konfidensintevall”:

Use Case	**Simuler konfidensintevall**
Aktør	Bruker
Type	Primært og essensielt use case
Beskrivelse	Bruker ønsker å kunne simulere konfidensintervall. Bruker ber systemet opprette konfidensintervall vindu og setter parametre for simulering. Systemet simulerer og displayer intervallene.

2.3.5 Operasjonelle krav

Hastighet:

- Det stilles krav til hurtighet ved beregninger av måldata for et datasett. Etter at bruker har tastet inn sitt datasett, skal måldata displayes etter minimum 3 sekunder.
- Ved automatisk generering av et datasett, skal man ikke behøve å vente altfor lenge på respons fra systemet. Det er betydelige beregninger som må gjennomføres, men ved generering av 2000 tall, skal det ikke gå mer enn 5 sekunder til bruker får respons.
- Opptegning av diagrammer bør skje omgående etter at bruker har bedt om å få en grafisk representasjon. Opptegningen skal ikke ha merkbar forsinkelse.
- Hvis bruker ikke har valgt noen forsinkelse, skal systemet ikke bruke mer enn 6 sekunder for å simulere og tegne opp 100 konfidensintervaller.

Utover dette er det ingen krav for hurtighet. Punktene over vil etter all sannsynlighet være de kritiske momenter for hurtighet, og ettersom det er betydelige mengder data som skal beregnes må det designes en god datastruktur som støtter rask behandling av mye data.

Det må taes i betraktning at Java ikke er spesielt raskt i forhold til andre programmeringsspråk. Det er flere grunner til at dette er tilfelle:

- Java har funksjonalitet for behandling av exceptions, noe som er svært tidkrevende.
- Noen av GUI komponentene er trege.
- Ved kompilering genereres objektkode som interpreteres ved hjelp av JVM (Java Virtual Machine). Som kjent er jo interpretering tregere enn å kjøre fullkompilert binærkode.

Robusthet/pålitelighet:

- Et diskret envariabelt ark skal bare godta heltall. Det må lages funksjonalitet som hindrer bruker i å taste inn flyttall, bokstaver eller andre tegn.
- Et kontinuerlig envariabelt eller tovariabelt ark skal bare godta reelle tall. Det må lages funksjonalitet som hindrer bruker i å taste inn bokstaver eller andre tegn.
- Det må finnes funksjonalitet som sier ifra til bruker når han/hun har tastet ulovlige verdier. Dette må skje i form av modale vinduer som forteller hva som er lovlig.
- Det er et krav at systemet responderer korrekt matematisk. De matematiske funksjonene er definert i vedlegg **A**.

Minnebruk/kapasitet:

- Et envariabelt ark skal ha plass til 2000 celler (2000 tall).
- Et tovariabelt ark skal ha plass til 2000 celler (1000 tallpar).
- Ved simulering av konfidensintervaller skal det være mulig å generere 100 intervaller.
- Utover punktene over skal tilgjengelig RAM være eneste begrensning (da spesielt med tanke på antall ark som kan være åpne samtidig).

2.4 Kravspesifikasjon for statistikkpakken statistics

Selv om det opplagt stilles matematiske krav til en statistikkpakke, er det også andre aspekter som må belyses i en kravspesifikasjon. I java finnes det allerede en klasse, java.lang.Math, som inneholder matematiske funksjoner. Vi ønsker, for å fremme muligheten av gjenbruk, å utvikle en statistikkpakke som har noenlunde lik struktur som java.lang.Math. Dette kommer vi nærmere inn på under kap. 5, Systemdesign.

Statistikkpakken skal inneholde følgende funksjonalitet:

- **Randomgeneratorer for de ulike fordelingene**
- **Beregning av måldata**
- **Beregning av sannsynlighet i et gitt punkt for en gitt fordeling**
- **Beregninger av konfidensintervaller**
- **Funksjonalitet for opptegning av teoretiske fordelinger**
- **Funksjonalitet for opptegning av histogrammer og stolpediagrammer**

Det ligger ikke innunder tidsrammene for prosjektet og søke å utvikle funksjonalitet for alle fordelingene. Vi har etter samtale med oppdragsgivere funnet de fordelingene som har høyest prioritering. Det må foreligge randomgeneratorer, muligheter for å beregne sannsynlighet og funksjonalitet for grafiske representasjoner av disse fordelingene. Vi har samlet de funksjonelle krav i en tabell.

Vi har definert de matematiske kravene til systemet. Disse har vi ikke puttet inn i rapporten, men lagt dem med i vedlegg **A**. Det er allikevel verdt å nevne at disse matematiske kravene må sees som en del av kravspesifikasjonen.

2.4.1 Overordnede funksjonelle krav

Randomgeneratorer:

SK 1	Det skal foreligge randomgeneratorer for følgende fordelinger: • Uniform diskret fordeling • Binomisk fordeling • Poissonfordeling • Normalfordeling • Eksponentialfordeling
SK 1.1	Randomgeneratorene for de diskret fordelingene (Uniform, Binomisk og Poisson) skal generere heltall.
SK 1.2	Randomgeneratorene for de kontinuerlige fordelingene (Normal og Eksponential) skal generere flyttall (reelle tall).

Beregning av måldata:

SK 2	Det skal finnes funksjonalitet for beregninger av måldata. Envariabel: Gjennomsnitt, median, varians, standardavvik. Tovariabel: GjennomsnittX, gjennomsnittY, empirisk korrelasjon, regresjonslinje (alpha og beta).

Beregning av sannsynlighet i et gitt punkt for de ulike fordelingene:

SK 3	Det skal være mulig å få sannsynligheten i et gitt punkt for følgende fordelinger: • Uniform diskret fordeling • Binomisk fordeling • Poissonfordeling • Normalfordeling • Eksponentialfordeling

Beregning av konfidensintervaller

SK 4	Det skal være mulig å få beregnet følgende konfidensintervaller: • Konfidensintervall for μ når σ er kjent. • Konfidensintervall for μ når σ er ukjent. • Konfidensintervall for σ.
SK 4.1	Det skal være mulig å beregne intervaller for følgende konfidensnivåer: 80%, 90%, 95%, 98%, 99%, 99.8% og 99.9%

Funksjonalitet for opptegning av teoretiske fordelinger

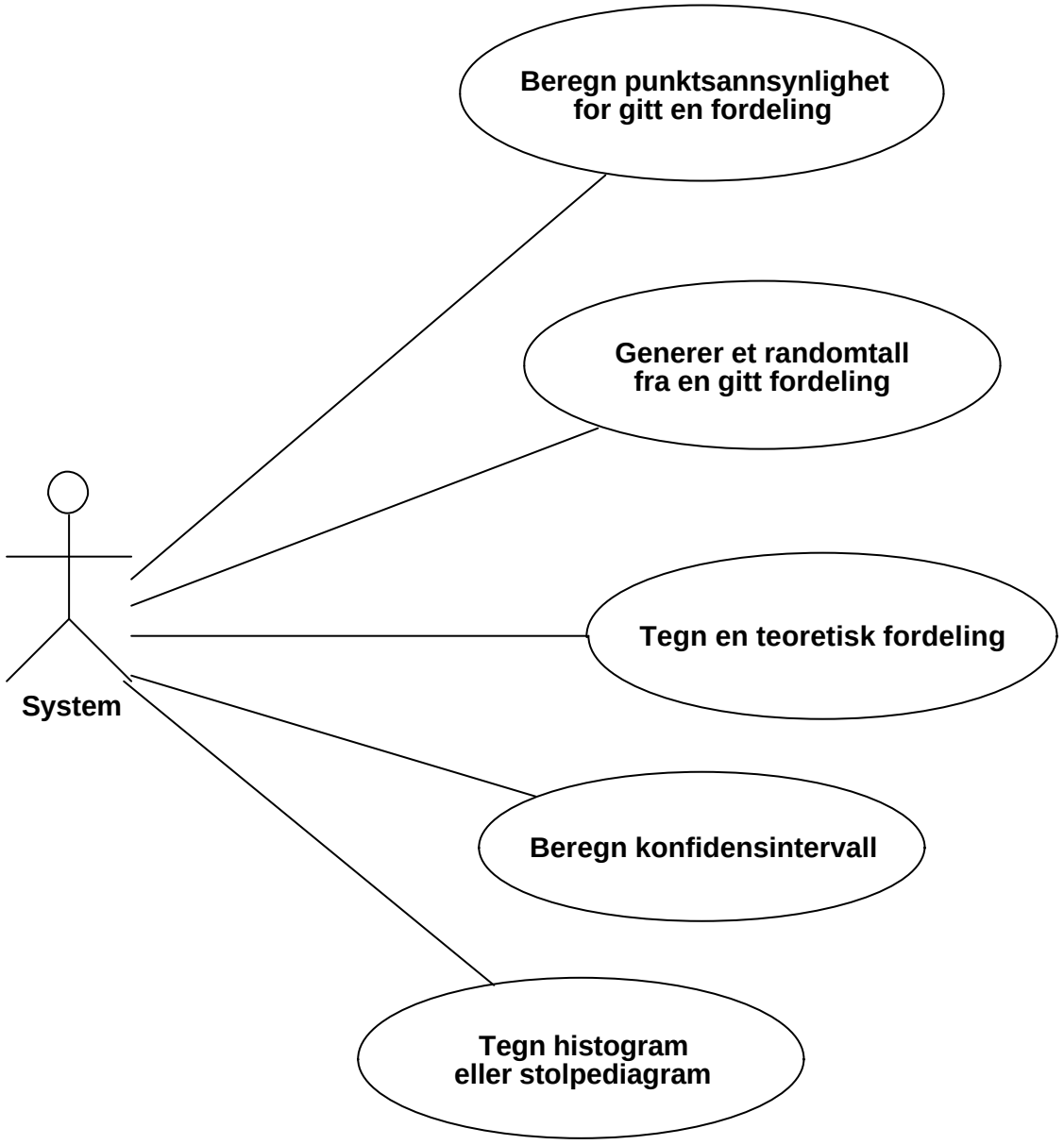
SK 5	Det skal være mulig å få grafiske representasjoner av en gitt fordeling.
SK 5.1	For de diskrete fordelingene (Uniform diskret, Binomisk og Poisson) skal det tegnes et stolpediagram.
SK 5.2	For de kontinuerlige fordelingene (Normal og Eksponential) skal det tegnes grafer/kurver.

Funksjonalitet for opptegning av histogrammer og stolpediagrammer

SK 6	Det skal foreligge funksjonalitet for å tegne histogrammer og stolpediagrammer.

2.4.2 Use Case diagram

Vi har definert Use Caser for statistikkpakken, men har valgt å ikke beskrive disse så detaljert som for Estatica – pakken.



Figur 2.2 Use Case diagram for statistics

2.5 Kravspesifiseringsprosessen

Å avdekke krav og ønsker er ingen enkel prosess. Det kan fort oppstå misforståelser og motstridende krav som oppdages sent. Ofte vet ikke oppdragsgiver selv hva som ønskes, og vi må derfor bidra med å oppdage ønsker og behov. Vi er så heldige at oppdragsgiverne er villig til å ha møte annenhver uke, og på denne måten har vi gode muligheter for tett samarbeid og forhåpentligvis etter hvert ende opp med et produkt som svarer til oppdragsgivernes forventninger.

Oppdragsgiverne er godt bevandret i skogen av dataverktøy for statistikk og matematikk. Dette hjelper oppdragsgiverne og oss å oppdage krav og hva vi ønsker å unngå.

I all hovedsak ser vi oss ferdig med å avdekke de funksjonelle kravene. Vi har fått god følelse av hva systemet skal gjøre, men vi må hele tiden ha øyne og ører åpne for forandringer. Vi skal fortsatt ha et tett samarbeid med oppdragsgiverne under designfasen.

Kravspesifiseringsprosessen har vært preget av kontinuerlig prototyping for å avdekke krav. Dette gjelder da spesielt med tanke på brukergrensesnittet som vi ser på som en viktig del av systemet. Vi har benyttet oss av Delphi for å lage GUI-prototyper. Disse prototypene har vært presentert for oppdragsgiverne og veilederen, slik at vi har fått verdifulle tilbakemeldinger i prosessen. På denne måten har vi fått konkrete tilbakemeldinger. Dette kommer vi til å fortsette med under designfasen.

Vi bestemte oss tidlig for hvordan vi skulle skrive kravspesifikasjonsdokumentet. Vi har skrevet ned alle krav underveis i prosessen, og dokumentet har vært gjenstand for mange forandringer og tilføyelser. Samtaler med oppdragsgiverne har hjulpet oss med å holde kravspesifikasjonsdokumentet godt oppdatert. Organiseringen av dokumentet er basert på en utgitt mal, men vi har sløyfet avsnitt som ikke passer inn for systemet vi skal utvikle. Det er grunn til å påpeke at utviklingen av kravspesifiseringsdokumentet for applikasjonen Estatica og statistikkpakken har foregått uavhengig av hverandre.

Vi har definert alle krav for en statistikkapplikasjon, men det er også en del av kravspesifiseringen å definere krav til prosessen. Et forprosjekt, der vi har gjennomgått en planleggingsfase, har vært med på å sørge for dette. Forprosjektet finnes i vedlegg **L**. I tillegg følger vi Rational Unified Prosess (RUP) som er en kjent og utprøvd systemutviklingsmodell, og dette vil være med på å kvalitetssikre prosessen. Vi har også laget egne standarder som vi plikter å følge (maler, dokumentasjon, standarder for koding osv.). Dette kommer frem utover i rapporten.

Kapittel 3

3 GROVDESIGN OG ANALYSE

Our life is frittered away by detail… Simplify, simplify.
Henry Thoreau

High thoughts must have high language.
Aristophanes

Let's all move one place on.
Lewis Carroll

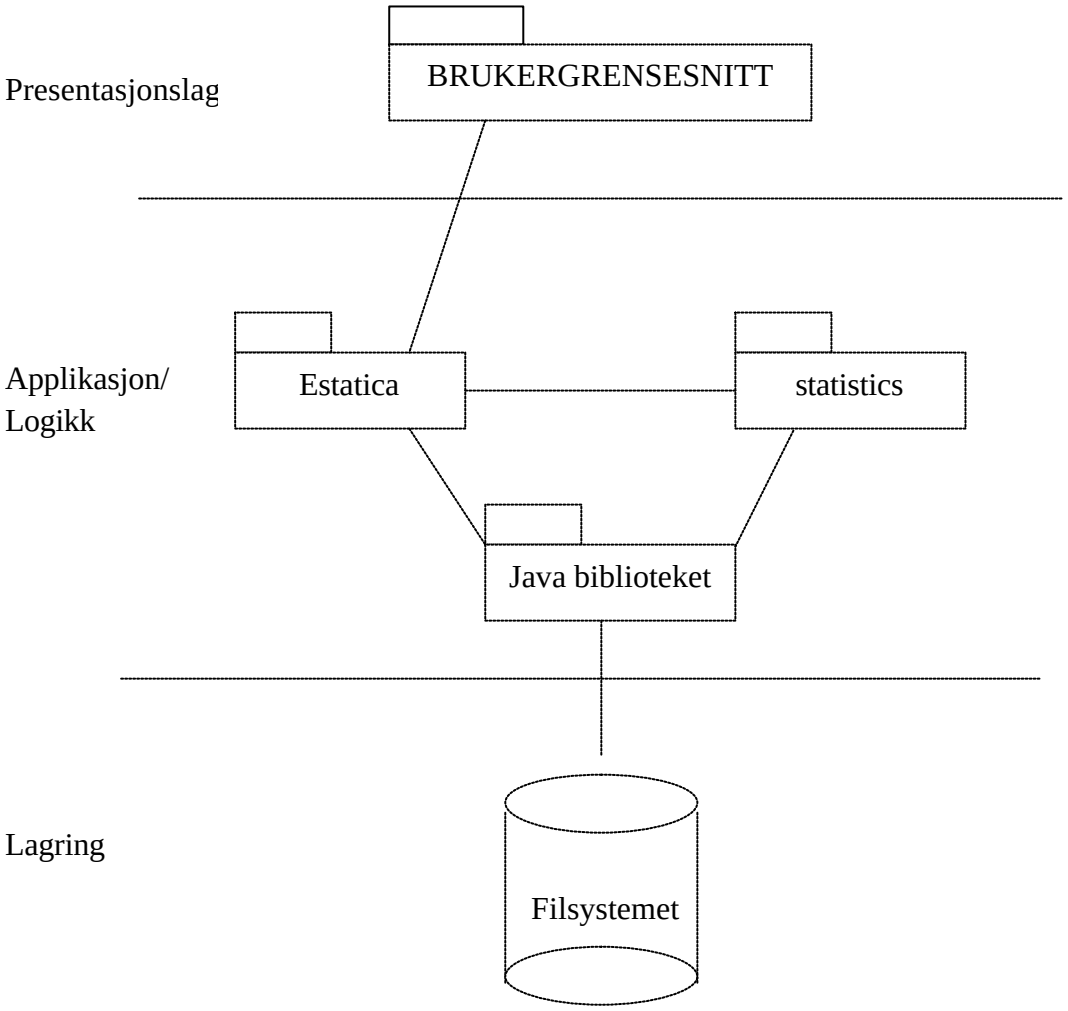
Man is a tool-making animal.
Benjamin Franklin

3.1 Softwarearkitektur/analyse

Systemet som skal utvikles er ikke avhengig av andre systemer bortsett fra operativsystemet. Vi ser derfor ingen verdi i å beskrive applikasjonens avhengighet av andre systemer, da dette taes hånd om av Java og operativsystemet. Estatica vil være en uavhengig applikasjon. Kildekoden kompileres ned til et ”intermediate language” - JVM-kode (Java Virtual Machine). Objektkoden må derfor interpreteres under kjøring, og JVM er en kraftig virtuell maskin som gjør nettopp denne jobben relativt raskt. Avhengigheten av andre systemer er i hvert fall nevnt, men vi utmaler ikke dette nærmere. Vi velger derfor å bevege oss inn i systemet for å se hvordan dette skal bygges opp i lag og pakker.

Applikasjonen skal ha en 3-lags arkitektur. Det er ingen åpenbar grunn til å sette opp en database, men applikasjonen må kunne håndtere filer (jfr. Krav K2). Vi illustrerer overordnet arkitektur med et diagram og kommentarer.

3.1.1 Overordnet arkitektur



Figur 3.1 Systemets overordnede arkitektur

Presentasjonslaget vil være det laget som har ansvar for all kommunikasjon med bruker. Her inngår all GUI, som vinduer, dialoger, tabeller osv. Dette laget vil utgjøre en egen pakke i systemet, og har et avhengighetsforhold til selve applikasjonen (Estatica) som ligger i det midterste laget.

Applikasjon/logikk-laget skal utgjøre den delen av systemet som ligger skjult for brukeren. Dette er selve logikken bak systemet, og danner flere pakker i systemet (disse interne pakkene beskrives senere). Denne pakken skal tilfredsstille kravene beskrevet i avsnitt 2.3.

Statistics er en egen pakke som er uavhengig av systemet vi skal lage. Denne pakken skal tilfredsstille kravene beskrevet i avsnitt 2.4.

Pakkene Brukergrensesnitt, Estatica og statistics er de pakkene som *vi* skal utvikle. Java biblioteket er selvfølgelig allerede utviklet og tilgjengelig. Operativsystemet og Java biblioteket tar seg av filsystemet, men vi må naturligvis implementere funksjonalitet for å lagre til og hente fra fil.

Pilene i diagrammet illustrerer avhengighetsforhold i systemet. Pakken Brukergrensesnitt har en slik uselvstendighet til Estatica; dette, derimot, gjelder ikke andre veien. Grensesnittet til bruker kan derfor enkelt forandres uten at det går ut over logikken for øvrig. Statistics skal være helt uavhengig av Estatica, og vil bare ha et avhengighetsforhold til Java biblioteket. Dette medfører ikke bare at Estatica kan gjennomgå store forandringer uten at logikken i statistics vil bli rammet, men andre applikasjoner kan benytte seg av statistics.

Slik oppdeling av systemet støtter objektorienterte prinsipper og har flere fordeler:

- God struktur i design av løsning.
- Støtter gjenbruk, der pakker kan være gjenstand for gjenbruk i andre prosjekter.
- Enklere å videreutvikle – pakker kan forandres uten at det berører andre pakker.
- Støtter dagens ”populære” utviklingsmodeller der utviklingen foregår i iterasjoner og inkrementer (dette gjelder også RUP som vi benytter oss av).

Kapittel 4

4 GUI-DESIGN OG BRUKERINTERAKSJONER

Painting is only a bridge linking the painter's mind with that of the viewer.
Eugene Delacroix

One picture is worth ten thousand words.
Chinese proverb

A picture shows me at a glance what it takes dozens of pages of a book to expound.
Ivan Sergeyecich

Treat nature in terms of the cylinder, the sphere, the cone, all in perspective.
Paul Cezanne

4.1 Grafisk brukergrensesnitt

Etter mange timer nært arbeid med oppdragsgiver ble vi enige om et tilfredsstillende brukergrensesnitt som oppfyller kravene (jfr. avsnitt 2.3). Vi har hatt en kontinuerlig prosess med utvikling av ulike prototyper som har ledet oss fram til det endelige resultatet. Dette arbeidet har foregått i Delphi, som har vært et praktisk verktøy å benytte seg av i denne sammenheng.

Vi har kommet fram til følgende vinduer som vår applikasjon må bestå av:

- Hovedvindu (Første vindu brukeren ser når applikasjonen startes)
- Velg fordeling dialog (Modal dialog som ber brukeren velge en fordeling)
- Grafisk vindu for envariable datasett
- Grafisk vindu for tovariable datasett
- Grafisk vindu for simulering av konfidensintervall

Disse vinduene vil bli vist i figurer under. Viktige GUI-komponenter vil være merket med bokstaver, slik at vi kan henvise til disse utover i kapitlet. Ettersom vi benytter oss av standard UML-notasjon, har vi laget reelle Use Case, og disse skal inneholde henvisninger til ”screenshots”.

4.1.1 Hovedvindu

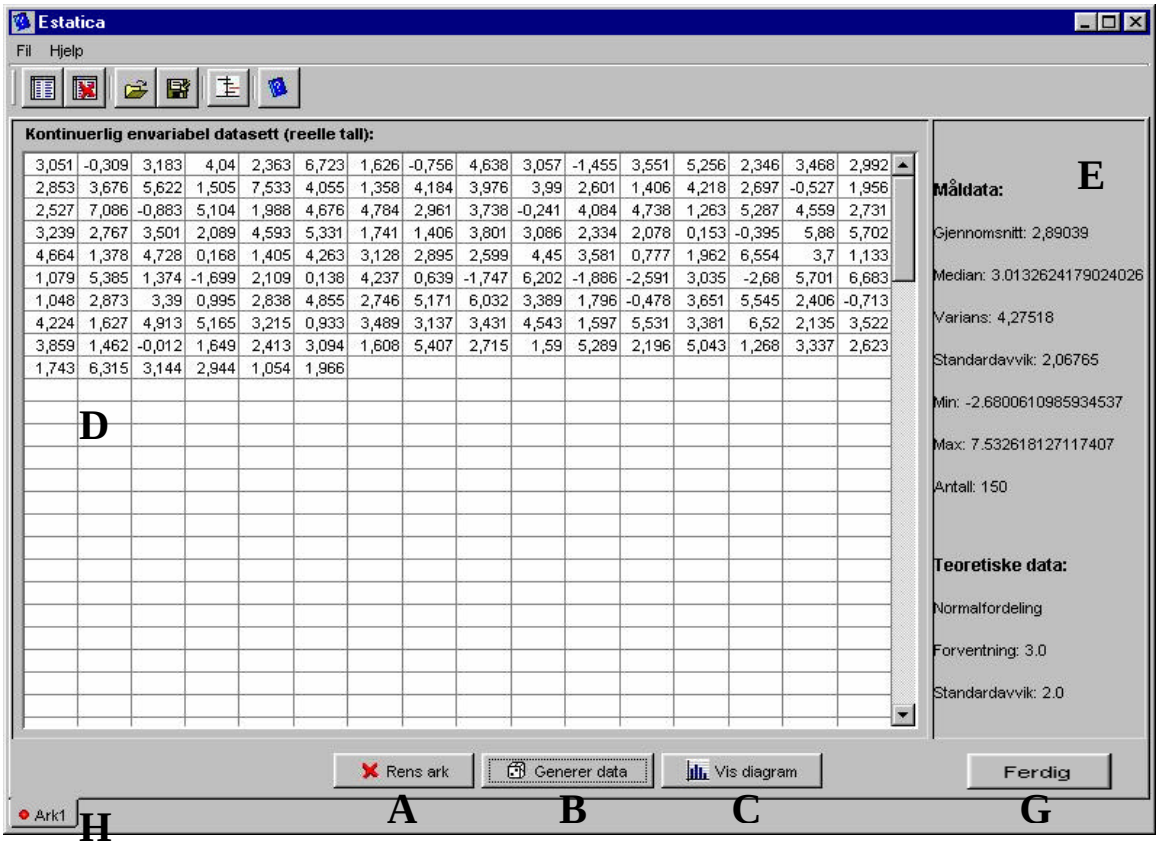
Dette vinduet skal være det som displayes når applikasjonen starter. Vinduet vil da være tomt - ingen ark/tabell har ennå blitt opprettet. En hovedmeny, verktøylinje og en tom editor er det eneste synlige for brukeren, som nå har flere muligheter:

- Bruker kan opprette en editor (et envariabelt eller tovariabelt ark). Dette skal kunne gjøres enten ved å opprette et nytt ark (jfr. figur 4.1.**A**) eller å åpne et datasett fra fil (jfr. figur 4.1.**C**).
- Bruker skal få simulert konfidensintervall ved å åpne et nytt konfidensintervall – vindu (jfr. figur 4.1.**E**).
- Bruker kan be om hjelp ved å åpne ”Hjelp” – dialogen (jfr. figur 4.1.**F**).

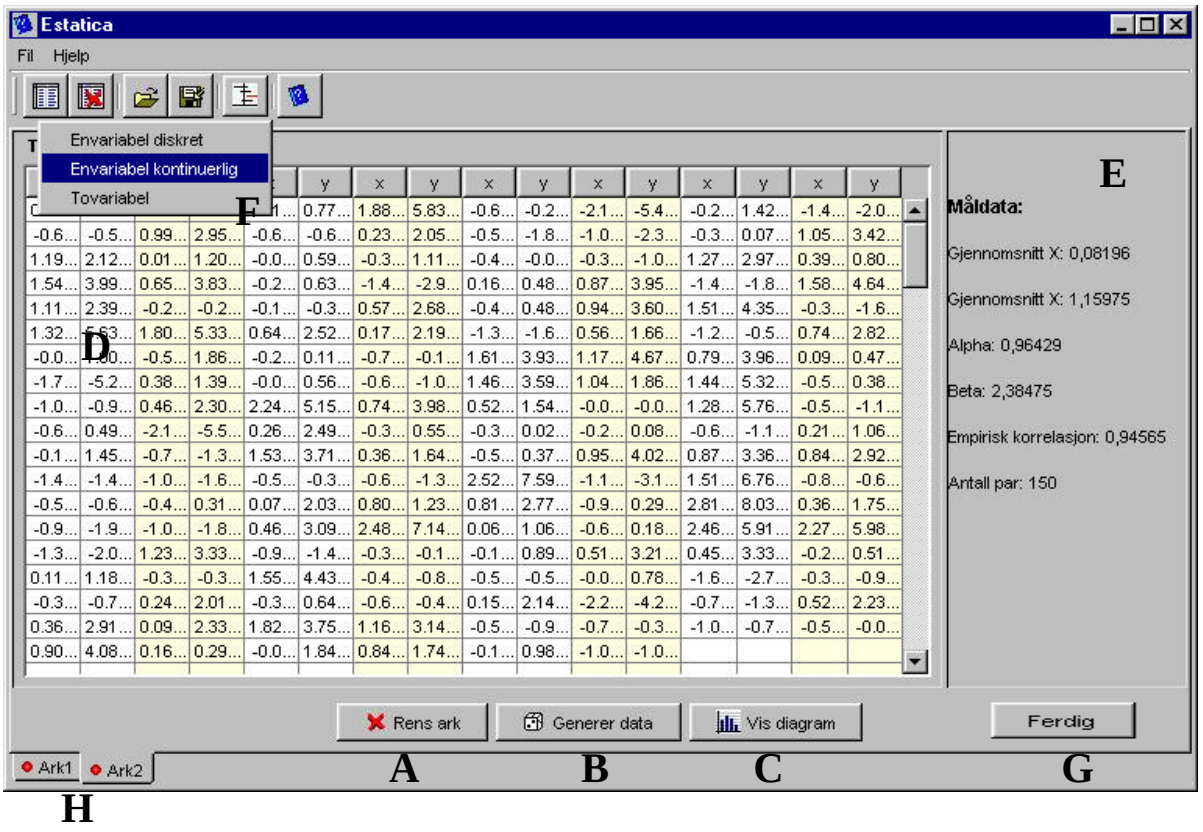
Alle disse operasjonene kan gjøres via verktøylinja:

Figur 4.1 Verktøylinje for hovedsiden, Estatica

Hovedsiden vil se slik ut etter at et envariabelt ark (figur 4.2) har blitt åpnet og tovariabelt ark (figur 4.3):



Figur 4.2 Hovedside med et envariabelt ark



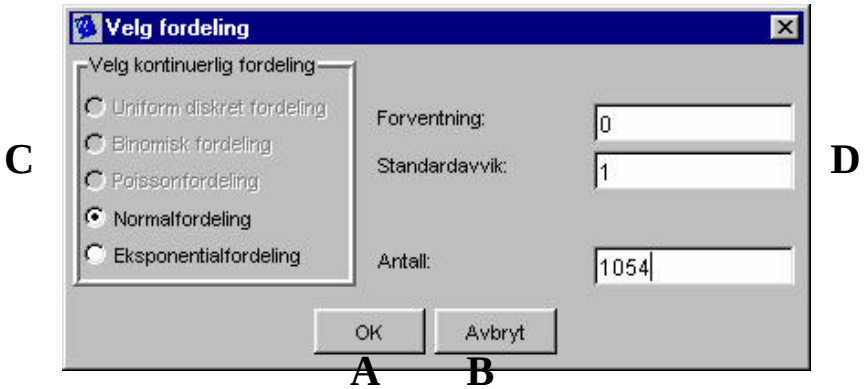
Figur 4.3 Hovedside med et tovariabelt ark

4.1.2 Velg fordeling dialog

Velg fordeling dialogen skal benyttes når bruker ønsker å generere et datasett. Dialogen bør være et modalt vindu som skal opprettes hver gang knappen ”Generer data” trykkes (jfr. figur 4.2.**B**). Her må bruker velge aktuell fordeling og sette verdier i aktuelle parametre. Når knappen ”OK” trykkes, skal dataene bli generert (jfr. figur 4.4).

Denne dialogen vil også bli benyttet når bruker ønsker å få tegnet en teoretisk modell i grafikkvindu for envariable data. Den eneste forskjellen vil være at feltet ”Antall” blir skjult.

Dialogen vi ble enige om:



Figur 4.4 Velg fordeling dialog

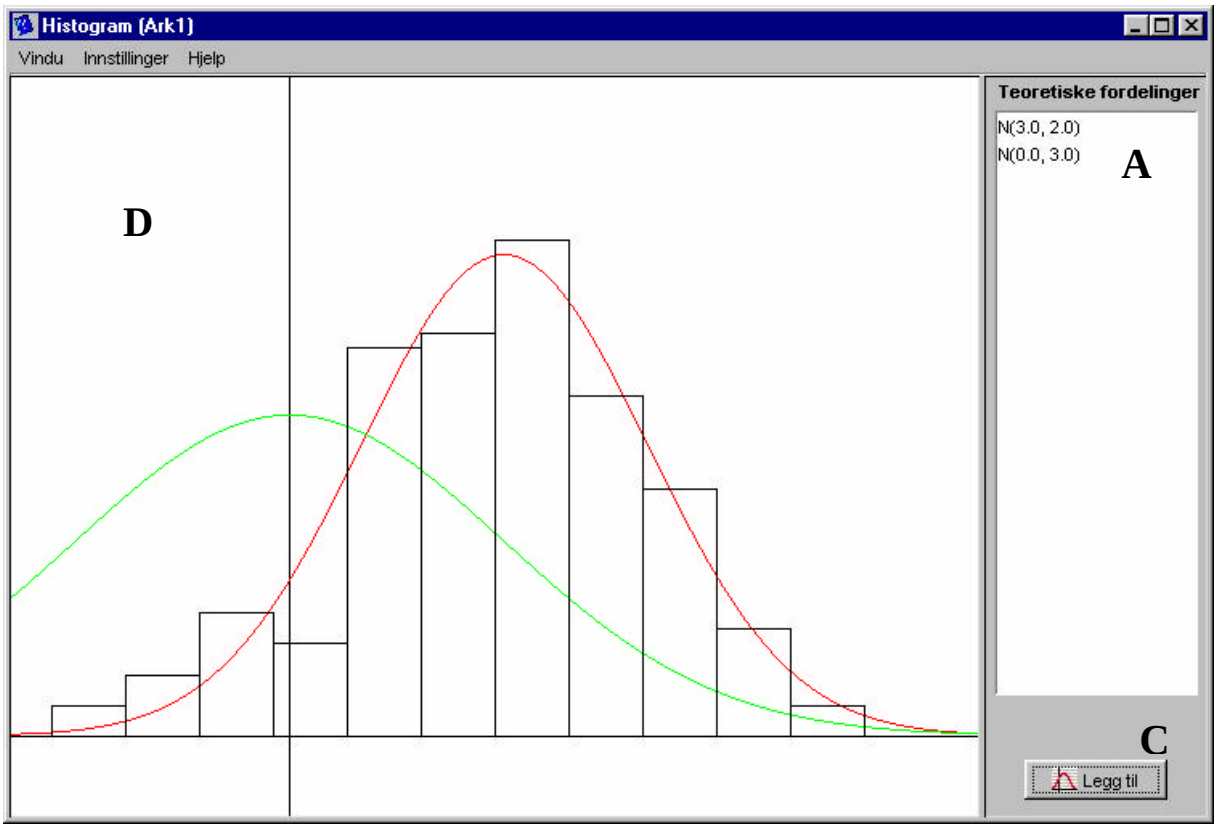
4.1.3 Grafisk vindu for datasett envariabel

Dette vinduet skal displaye grafiske representasjoner for envariable datasett. Det skal være mulig å få tegnet både teoretiske modeller og grafiske tolkninger av datasett. For hvert datasett vil det være kun en grafisk tolkning, mens det kan tegnes mange teoretiske modeller oppå en slik tolkning.

For diskre data vil den grafiske tolkningen bestå av et stolpediagram, mens for kontinuerlige data vil dette representeres som et histogram.

Vinduet skal inneholde en liste med de teoretiske modeller som etter hvert tegnes (jfr. figur 4.5.**A**). For å legge til en modell må bruker trykke på knappen "Legg til" (jfr. figur 4.5.**C**). Da vil bruker få opprettet en "velg fordeling" – dialog hvor han/hun velger ønsket fordelingsmodell (jfr. figur 4.4). Det vil også være mulig å fjerne elementer (modeller) fra denne listen.

Vinduet vi ble enige om:

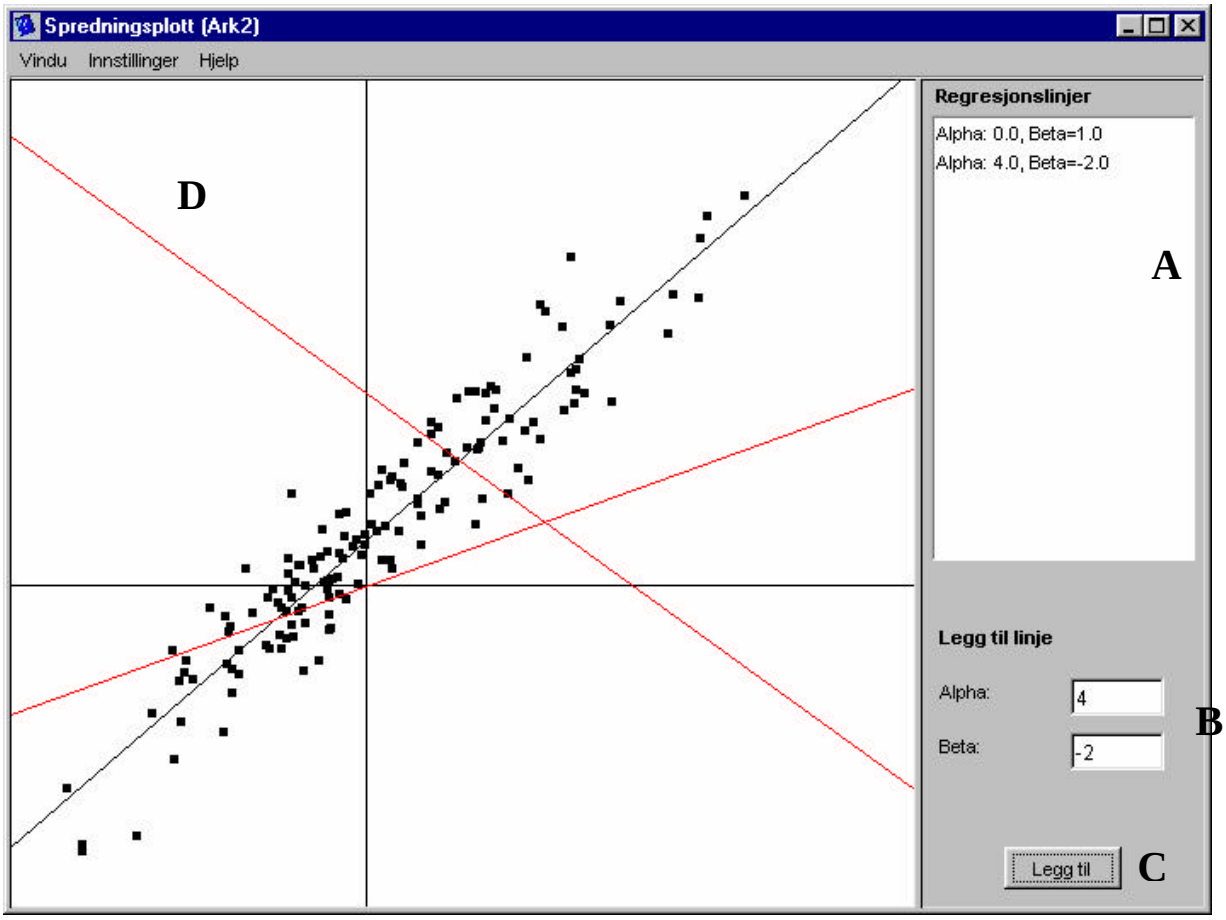


Figur 4.5 Grafisk vindu for datasett envariabel (her med kontinuerlige data)

4.1.4 Grafisk vindu for datasett tovariabel

Dette vinduet skal brukes for å representere grafisk tolkning av et tovariabelt datasett. Tolkningen representeres i form av en regresjonslinje og et spredningsplott. Det skal også være mulig å legge til teoretiske regresjonslinjer over den virkelige linja. Dette vil foregå ved at bruker setter ønskede verdier for ”alpha” og ”beta” og trykker på knappen ”Legg til” (jfr. figur 4.6.**C**). De teoretiske linjene vil legges inn i en liste som finnes på høyre siden i vinduet (jfr. figur 4.6.**A**). Det skal også være mulig å slette elementer (linjer) i fra listen.

Vinduet vi ble enige om:



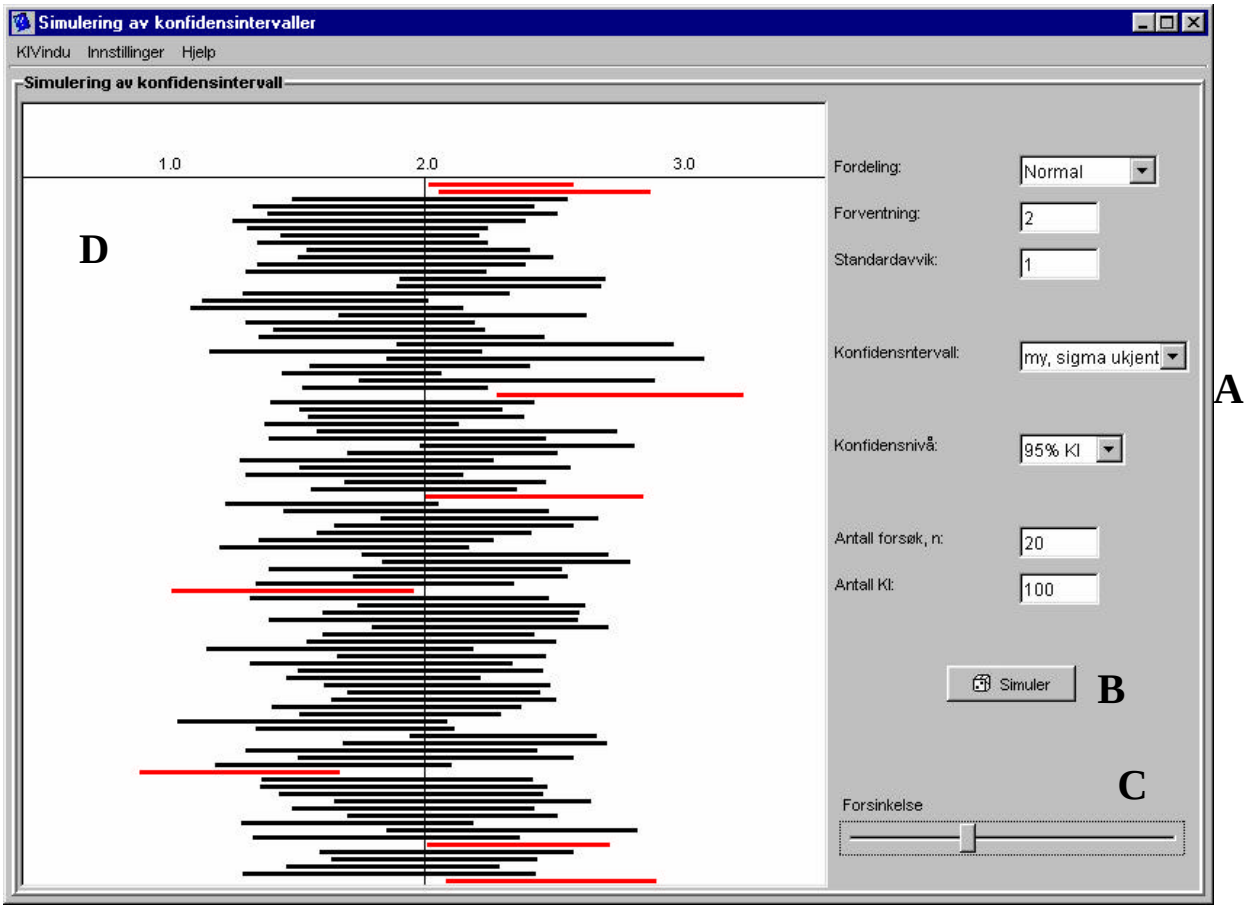
Figur 4.6 Grafisk vindu for datasett tovariabel

4.1.5 Grafisk vindu for simulering av konfidensintervall

Konfidensintervallvinduet vil ikke være avhengig av at det har blitt generert datasett. Det skal åpnes direkte i fra hovedside ved å trykke på hurtigknappen ”Simuler konfidensintervall” (jfr. figur 4.1.**E**). For å få simulert intervallene må bruker velge type konfidensintervall, sette tilhørende parametre og trykke på knappen ”Simuler” (jfr. figur 4.7.**B**).

Det er viktig å få frem hvilke intervaller som treffer (inneholder forventningsverdi) og hvilke som bommer. Vi har derfor bestemt oss for å skille på disse intervallene ved at de som treffer skal være svarte, mens de som bommer skal være røde. Det skal være mulig å legge inn forsinkelse på konfidensintervallene etter hvert som de genereres (jfr. figur 4.7.**C**).

Vinduet vi ble enige om:



Figur 4.7 Grafisk vindu for simulering av konfidensintervaller

4.2 Brukerinteraksjoner

For å illustrere brukerinteraksjoner har vi valgt å benytte systemmodellene reelle Use Case, system sekvensdiagram og kontrakter. I avsnittene under (avsnitt 4.2.1 til 4.2.3) har vi tatt med noen eksempler som illustrerer disse modellene. Etter samtale med veileder ble vi enige om at de ble såpass detaljerte, at vi trekker de øvrige ut av selve rapporten og legger dem med som vedlegg **B** og **C**. Disse vil allikevel være en del av designdokumentet.

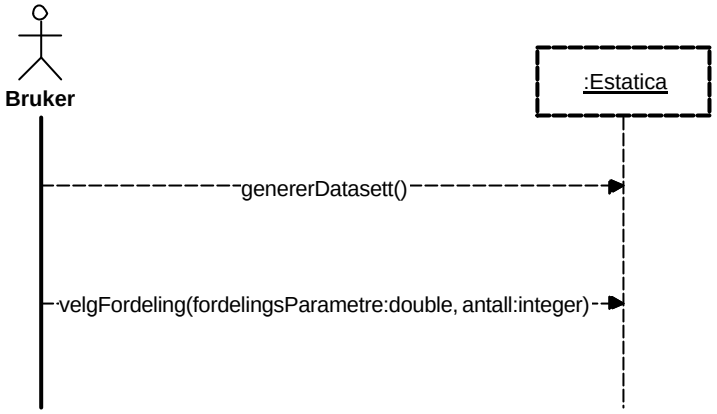
System sekvensdiagram lages på bakgrunn av foreliggende Use Case beskrivelser (jfr. avsnitt 2.3.4) . De forteller *hva* bruker kan utføre på systemet. Diagrammene gir oss klarhet i hendelsene som bruker foretar seg mot systemet, det vil si at de viser detaljene i all input. Systemet svarer på disse hendelsene ved å utføre en tilhørende operasjon.

For hver hendelse i et system sekvensdiagram har vi skrevet en kontrakt. Disse kontraktene beskriver effekten av hver systemoperasjon. Her skal det fremgå hvilke endringer man har hatt på tilstander i systemet, og ikke hvilke handlinger som utføres.

Reelle Use Case er første steg i applikasjonen som sier noe om *hvordan* løsningen skal bli. De beskriver på en god måte hvordan brukeren kommuniserer med systemet, og hvordan systemet skal svare på brukerhenvendelser. I Use Case beskrivelsene som følger, vises det til ”screenshot’ene” i avsnitt 4.1. I tillegg henviser vi til kravene i kapittel 2. Når disse Use Casen designes og implementeres, vil formodentlig kravene være oppfylt og problemstillingen løst. Å henvise på denne måten i reelle Use Case er standard UML notasjon.

4.2.1 Eksempel på systemmodeller for Use Caset ”Automatisk generere datasett”

System sekvensdiagram



Kontrakter

Navn:	**genererDatasett()**
Ansvar:	Systemet skal generere et datasett for bruker.
Type:	System.
Kryss referanser:	K1, K1.4
Pre-betingelser:	• Det må foreligge *Tabell* og *Ark*.
Post-betingelser:	• En *VelgFordelingDialog* ble opprettet.

Navn:	**velgFordeling(antall : Integer, parametre : Double)**
Ansvar:	Bruker skal fortelle systemet hvilken fordeling og antall forsøk han/hun ønsker.
Type:	System.
Kryss referanser:	K1, K1.4
Unntak:	Bruker avbryter ved å trykke ”Avbryt” i *VelgFordelingDialog*.
Pre-betingelser:	• Det må foreligge *Tabell* og *Ark*. • En *VelgFordelingDialog* må være opprettet.
Post-betingelser:	• Eventuelt gammelt *Datasett* ble fjernet. • En *RandomGenerator* ble opprettet. • Et *Datasett* ble opprettet. • *Datasett* ble assosiert med *Ark*. • En *Fordeling* ble opprettet. • Parametre for *Fordeling* ble satt. • *Fordeling*en ble assosiert med *Datasett*. • *RandomGenerator*en ble fjernet. • *VelgFordelingDialog*en ble fjernet.

Real UseCase-beskrivelse

Aktør:	Bruker
Hensikt:	Generere et datasett automatisk utfra en gitt fordeling.
Sammendrag:	Det foreligger et tomt ark (tabell).Bruker ønsker å få generert et datasett. Bruker ber systemet generere et datasett med gitt fordeling og tilhørende parametre. Datasettet lagres i systemet for videre bruk.
Type:	Primært og real use case
Kryss referanser:	K1, K1.2, K1.2.1, K1.2.2, K1.2.3, K1.4, K1.5, K1.6, K1.7 K3, K3.1, K3.2

Forventet hendelsesforløp:

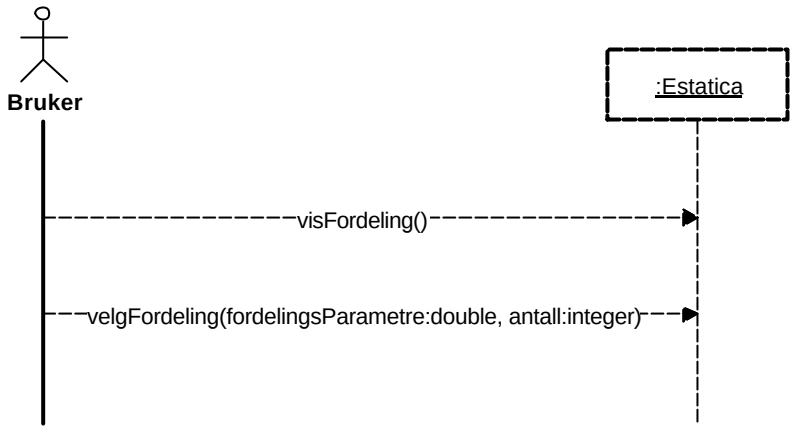
Aktør hendelser	**System hendelser**
1. Dette Use Caset starter når bruker tykker på knappen ”Generer data” – fig.4.2.**B** /fig.4.3.**B**.	
	2. Systemet displayer et lite vindu med ulike fordelinger og tilhørende parametere – fig4.4.
3. Bruker velger aktuell fordeling og verdier for tilhørende parametere – fig.4.4.**C** / fig.4.4.**D**. Han/hun velger fordeling ved å trykke på knappen ”OK” – fig.4.4.**A**.	
	4. Systemet lukker vinduet.
	5. Systemet genererer tall ut fra gitte opplysninger og skriver disse ut til skjerm – fig.4.2.**D** / fig.4.3.**D**.
	6. Systemet lagrer datasettet, regner ut ulike målverdier. Disse verdiene skrives til skjerm – fig.4.2.**E** / fig.4.3.**E**.

Alternative forløp:

- Linje 3: Bruker velger å avbryte ved å trykke på knappen ”Avbryt” - fig.4.4.**B**.
- Linje 3: Bruker har tastet inn ugyldige parametre, han/hun må få melding om dette. Feil/mangler må rettes opp for å kunne gå videre.

4.2.2 Eksempel på systemmodeller for Use Caset "Få grafisk representasjon av modeller"

System sekvensdiagram



Kontrakter

Navn:	**visFordeling()**
Ansvar:	Systemet skal gi bruker en grafisk representasjon av en teoretisk modell.
Type:	System.
Kryss referanser:	K5
Pre-betingelser:	• Det må foreligge *Tabell* og *Ark*.
Post-betingelser:	• En *VelgFordelingDialog* ble opprettet.

Navn:	**velgFordeling()**
Ansvar:	Bruker skal fortelle systemet hvilken fordeling han/hun ønsker.
Type:	System
Kryss referanser:	K5
Pre-betingelser:	• Det må foreligge *Tabell* og *Ark*. • Det må foreligge *GrafikkVindu* og et *GrafikkPanel*. • En *VelgFordelingDialog* må være opprettet.
Post-betingelser:	• En *Fordeling* med initialiserte parametre ble opprettet. • *Fordeling*en ble assosiert med *GrafikkPanel*et. • *VelgFordelingDialog*en ble slettet.

Real UseCase-beskrivelse

Aktør: Bruker
Hensikt: Få displayet en teoretisk modell.
Sammendrag: Bruker ønsker å få grafisk representasjon av en teoretisk modell. Bruker velger fordeling med ønskede parametre. Systemet displayer modellen.
Type: Primært og real use case
Kryss referanser: K4, K4.1, K4.2, K4.3

Forventet hendelsesforløp:

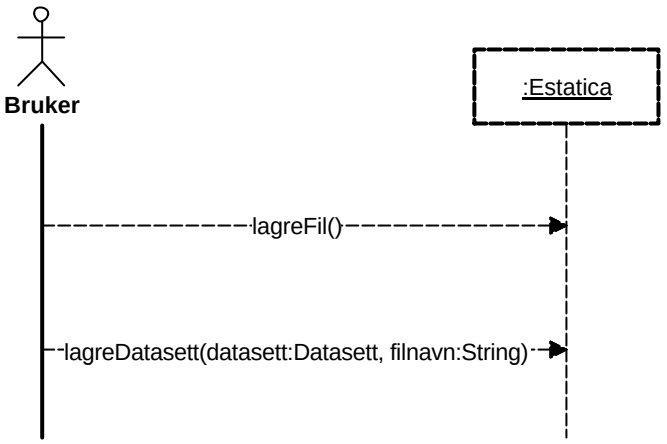
Aktør hendelser	**System hendelser**
1. Dette Use Caset starter når bruker ønsker å få displayet en teoretisk modell ved å trykke på knappen "Legg til" – fig.4.5.**C** / fig.4.6.**C**. Videre forløp er avhengig av type datasett: **a)** Envariabelt: fortsett i punkt **2**. **b)** Tovariabelt: Bruker må ha lagt inn ønskede verdier for "alpha " og "beta". Fortsett i punkt **6**.	
	2. Systemet displayer en "Velg Fordeling"-dialog – fig.4.4.
3. Bruker velger aktuell fordeling med tilhørende parametre – fig.4.4.**C** / fig.4.4.**D**.	
4. Når bruker har valgt fordeling, trykker han /hun på knappen "OK" – fig.4.4.**A**.	
	5. Systemet lukker "Velg fordeling" – dialogen.
	6. Systemet leser inn aktuelle parameter verdier og displayer den teoretiske modellen - fig.4.5.**D** / fig.4.6.**D**.

Alternative forløp:

- Linje 4: Bruker velger å avbryte ved å trykke på knappen "Avbryt" – fig.4.4.**B**.
- Linje 6: Bruker har lagt inn ugyldige verdier for parametrene. Systemet må gi melding om dette.

4.2.3 Eksempel på systemmodeller for Use Caset "Legge datasett på fil"

System sekvensdiagram



Kontrakter

Navn:	**lagreFil()**
Ansvar:	Bruker ber om å få lagre et datasett på fil. Systemet displayer en fil-browser.
Type:	System
Kryss referanser:	K2, K2.1
Merknader:	Hvis filen ikke finnes opprettes en ny fil.
Pre-betingelser:	• Det må foreligge et *Ark* og et *Datasett*
Post-betingelser:	• En fil-browser ble displayet.

Navn:	**lagreDatasett(datasett : Datasett, filnavn : String)**
Ansvar:	Systemet lagrer datasett på fil.
Type:	System
Kryss referanser:	K2, K2.1
Unntak:	Hvis det ikke foreligger et datasett, gi bruker melding om dette.
Pre-betingelser:	• Det må foreligge et datasett. • En fil må ha blitt valgt.
Post-betingelser:	• Et *Datasett* ble lagret på fil. • Fil-browseren ble lukket.

Real UseCase-beskrivelse

Aktør: Bruker
Hensikt: Lagre et datasett på fil til senere bruk.
Sammendrag: Det foreligger et datasett, og bruker ønsker å lagre dette datasettet på fil. Bruker får mulighet til å navngi filen og plassering, og systemet lagrer filen på disk.
Type: Primært og real use case
Kryss referanser: K2, K2.1

Forventet hendelsesforløp:

Aktør hendelser	**System hendelser**
1. Dette Use Caset starter når bruker trykker på hurtigtasten for lagring på fil eller velger tilsvarende operasjon på fil-menyen – fig.4.1.**D**.	
	2. Systemet displayer et standard fillagrings-dialog til bruker.
3. Bruker navigerer i filstrukturen på disken, og velger mappe som filen skal lagres i. Brukeren skriver inn ønsket filnavn og trykker OK.	
	4. Systemet lukker fillagringsdialogen.
	5. Systemet lagrer filen på angitt plass og med gitt navn.
	6. Systemet setter navnet på arket/datasettet til navnet på filen - fig.4.2.**H** / fig.4.3.**H**.

Alternative forløp:

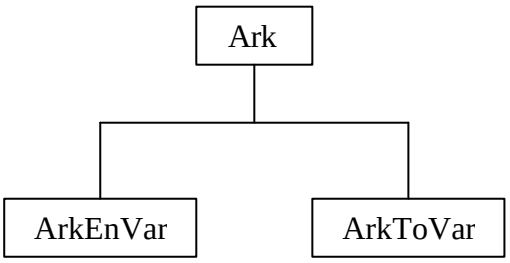
- Linje 1: Det foreligger ikke noe datasett. Systemet gir melding til bruker og avslutter Use Caset.
- Linje 3: Bruker velger å avbryte ved å trykke knappen ”Avbryt”.
- Linje 3: Hvis dette arket er åpnet fra en fil (eller har blitt lagret til fil mens det har vært oppe) skjer lagringen automatisk.
- Linje 5: Bruker prøver å navngi filen med samme navn som et annet ark. Systemet gir melding til bruker, og gir mulighet for å skrive nytt navn.

4.3 Design av pakken Brukergrensesnitt

Pakken Brukergrensesnitt må inneholde de vinduer og dialoger som er beskrevet i avsnitt 4.1. I Java vil et vindu være definert som en klasse, og vi må definere følgende klasser:

- **Hovedside:** Systemets hovedside – vinduet som kommer opp når applikasjonen startes.
- **ArkEnVar:** Et ark som inneholder tabellen bruker kan legge et envariabelt datasett i.
- **ArkToVar:** Et ark som inneholder tabellen bruker kan legge et tovariabelt datasett i.
- **GrafikkVinduEnVar:** Grafikkvindu som tegner envariable datasett.
- **GrafikkVinduToVar:** Grafikkvindu som tegner tovariable datasett.
- **KIGrafikkVindu:** Vindu der bruker kan simulere konfidensintervaller.
- **Hjelp:** Vindu som displayer hjelp til bruker.
- **VelgFordelingDialog:** Vindu der bruker kan velge en fordeling og skrive inn parametre.

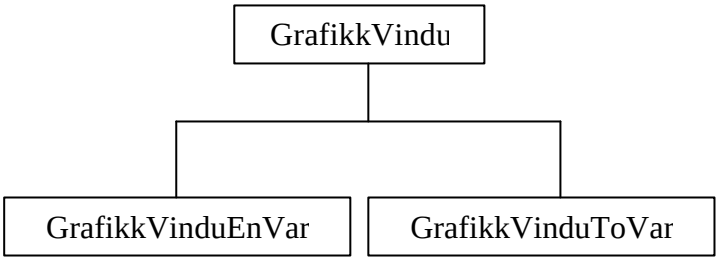
Vi trenger bare en hovedside, men arkene er avhengig av type datasett som bruker ønsker å jobbe med. Derfor må vi lage flere typer ark, og dette kan vi representere ved å definere følgende klasser:



Figur 4.8 Klassediagram for *Ark*

Ark vil være en abstrakt klasse, mens *ArkEnVar* og *ArkToVar* vil henholdsvis inneholde envariabelt og tovariabelt datasett. Et ark vil også være knyttet til et av de to grafikkvinduene (GrafikkVinduEnVar og GrafikkVinduToVar). På denne måten kan bruker be om å få en grafisk representasjon av datasettet som ligger i arket.

De to Grafikkvinduene vil ha samme struktur som arkene:



Figur 4.9 Klassediagram for *GrafikkVindu*

GrafikkVindu vil, som *Ark*, være en abstrakt klasse, der *GrafikkVinduEnVar* og *GrafikkVinduToVar* henholdsvis vil tegne envariable og tovariable diagrammer.

Hovedside vil være knyttet til et *KIGrafikkVindu*, fordi man kan åpne et slikt vindu fra hovedsiden via verktøylinja.

Klassene beskrevet i dette avsnittet vil nok etter hvert utgjøre bare en liten del av de klassene som må implementeres i pakken Brukergrensesnitt. Allikevel vil nok disse klassene være de aller viktigste, og det holder å nevne bare disse klassene for å illustrere design og struktur i denne omgang.

Kapittel 5

5 SYSTEMDESIGN

Classes struggle, some classes triumph, others are eliminated.
Mao Zedong

Say not you know another entirely, till you have divided an inheritance with him.
Johann Kasper Lavater

Good as it is to inherit a library, it is better to collect one.
Augustine Birrel

Is it a world to hide virtues in?
William Shakespear, Twelth Night

Have no friends not equal to yourself.
Confucius

What we experience of nature is in models, and all of nature's models are so beautiful.
Richard Buckminster Fuller

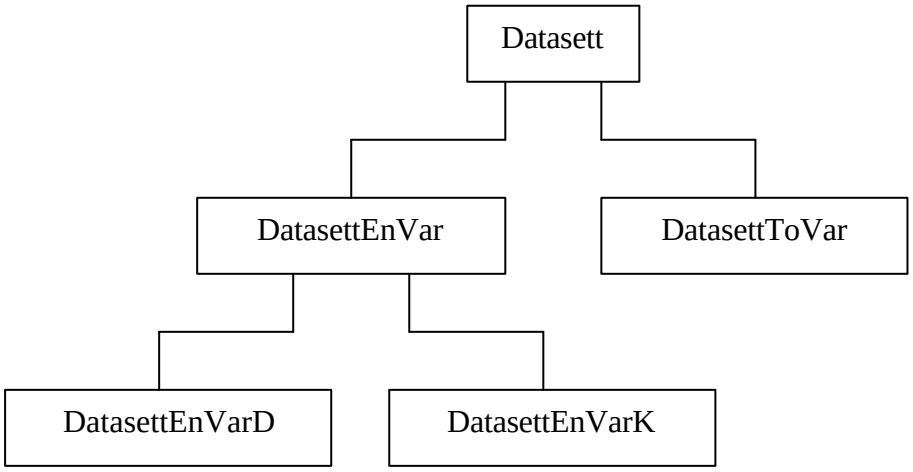
5.1 Design Estatica

Pakken Estatica vil være selve systemet som brukeren ”ser og føler” (se figur 3.1). Estatica er selv delt opp i pakker. Vi ser det svært fordelaktig å splitte funksjonalitet og konsepter. Vi har identifisert følgende pakker som må ligge i Estatica:

- **Brukergrensesnitt:** Dette er den pakken som er beskrevet i avsnitt 3.1.1. Denne pakken ligger fysisk i pakken Estatica, men kan logisk trekkes ut som et eget lag i systemet (jfr. figur 3.1).
- **Datasett:** Denne pakken inneholder funksjonalitet for å ta vare på og prosessere et datasett. Denne pakken bør ha en god struktur (jfr. krav K3) som støtter effektiv og rask behandling av store mengder data.
- **Fordeling:** Vi må på en eller annen måte representere en fordeling, og vi lager en egen pakke som har ansvaret for å håndtere de ulike fordelingene.
- **Konfidensintervall:** Funksjonalitet for å behandle konfidensintervaller plasseres i denne pakken.
- **Regresjon:** Om vi rekker å implementere tovariabelmodeller, vil all funksjonalitet for å behandle regresjonslinjer ligge her.

5.1.1 Pakken datasett

Vi må på en eller annen måte representere et datasett i maskinen. Det er naturlig å tenke oss en klasse *Datasett*. Representasjonen av et datasett er også avhengig av om datasettet er envariabelt eller tovariabelt. For et envariabelt datasett må vi også skille mellom kontinuerlige og diskrete data. Vi har funnet det hensiktsmessig å dele inn i klasser på denne måten:

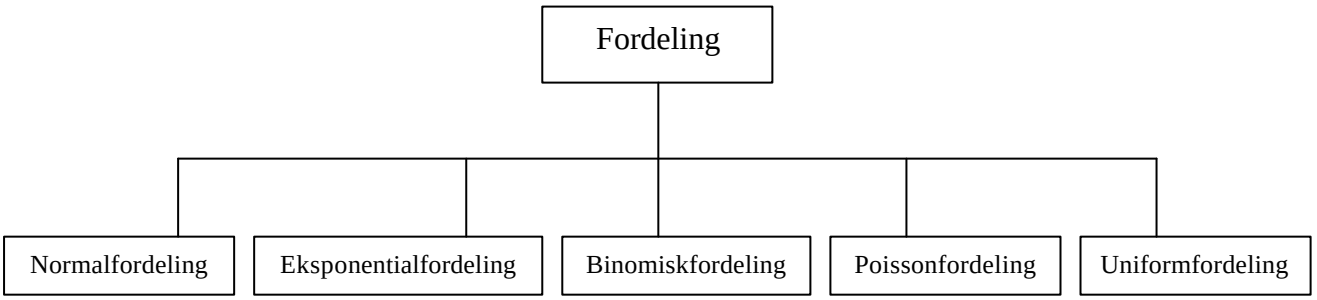


Figur 5.1 Klassediagram for pakken Datasett

DatasettEnVarD vil være den konkrete klassen som representerer et envariabelt diskret datasett. *DatasettEnVarK* er dermed den klassen som kan representere et envariabelt kontinuerlig datasett. Disse to klassene arver fra den abstrakte klassen *DatasettEnVar*. Det er ingen hensikt i å skille mellom diskrete og kontinuerlige datasett for tovariabelmodeller, og derfor vil *DatasettToVar* være en konkret klasse som arver fra *Datasett*.

5.1.2 Pakken fordeling

Vi er nødt til å finne en måte å representere en fordeling. En fordeling vil statistisk sett beskrive seg selv. Med det mener vi at en fordeling vet om den selv er kontinuerlig eller diskret. Derfor er det ingen vits å skille mellom kontinuerlig og diskrete fordelinger (det gjør de selv), og vi kan representere en fordeling svært enkelt:

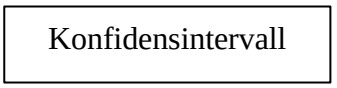


Figur 5.2 Klassediagram for pakken Fordeling

På denne måten kan en fordeling representeres som et objekt i systemet. *Fordeling* vil være en abstrakt klasse som alle de konkrete fordelingene arver fra.

5.1.3 Pakken konfidensintervall

Et konfidensintervall kan representeres på en enda enklere måte. Det finnes mange typer konfidensintervaller, men vi trenger ikke ta hensyn til dette i Estatica. Beregning av konfidensintervaller vil foregå i den utenforstående statistikkpakken, og dette medfører at alt vi trenger er *en* klasse.



Figur 5.3 Klassediagram for pakken Konfidensintervall

Denne klassen kan representere et hvilket som helst konfidensintervall, og vil inneholde grensene i intervallet.

5.1.4 Pakken regresjon

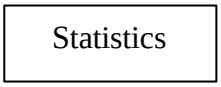
For tovariable datasett skal det regnes ut regresjonslinjer, og det må da selvfølgelig finnes en mulighet for å beskrive en slik linje.

En linje er av formen y = a + bx, og det finnes ikke flere typer linjer. Representasjonen blir da en enkelt klasse:

Figur 5.4 Klassediagram for pakken Regresjonslinje

5.2 Design statistics

Statistics skal som sagt innbefatte funksjonalitet for statistiske og matematiske beregninger. I Java finnes en klasse, java.lang.Math, som inneholder matematiske funksjoner. Vi har bestemt oss for å lage en klasse som har samme struktur som java.lang.Math. Dette medfører at eventuelle systemutviklere som ønsker å benytte pakken statistics, er noenlunde kjent med strukturen. Klassen Math i Java er en final klasse med statiske funksjoner. Vi vil derfor komme til å definere en final klasse med statiske funksjoner:



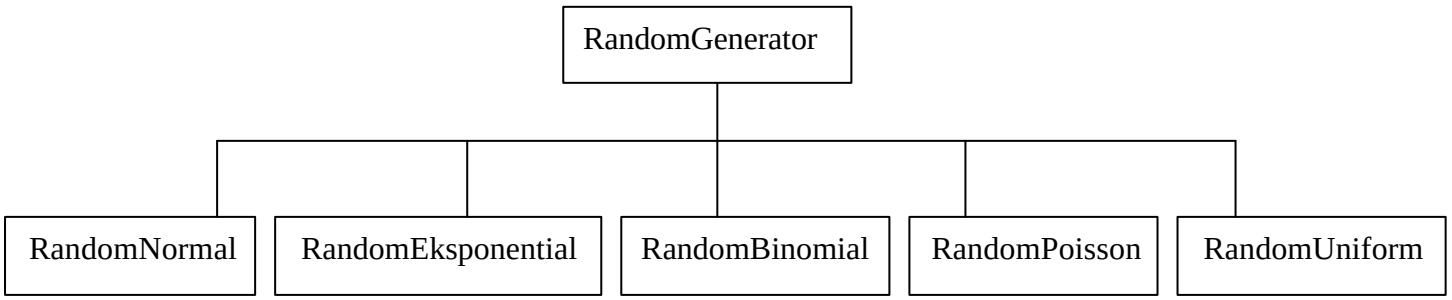
Figur 5.5 Klassediagram for final klasse Statistics

Dette vil ikke være nok, så vi har identifisert flere pakker som skal finnes i pakken statistics.

- **randomgenerators:** Det må utvikles randomgeneratorer for hver av fordelingene som skal benyttes. I Java finnes allerede en randomgenerator, men denne må utvides til å generere tall fra en ønsket fordeling. Vi må derfor designe og implementere funksjonaliteten for denne pakken.
- **graphics:** Denne pakken skal inneholde funksjonalitet for å tegne diagrammer (histogrammer og stolpediagrammer) og kurver (teoretiske fordelinger).
- **confidenceinterval:** Det må finnes muligheter for å beregne konfidensintervaller. Denne pakken vil inneha dette ansvaret.

5.2.1 Pakken randomgenerators

En randomgenerator skal produsere tall ut ifra en gitt fordeling med tilhørende parametre. Det vil være hensiktsmessig å skille de ulike fordelingene. En struktur som følger vil kunne representere en generator for en fordeling:



Figur 5.6 Klassediagram for pakken randomgenerators

Objekter av for eksempel *RandomBinomial* skal kunne generere binomisk fordelte tall. Alle de konkrete randomgeneratorene arver fra den abstrakte klassen *RandomGenerator*.

5.2.2 Pakken graphics

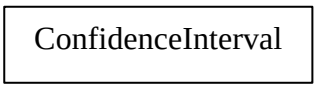
Det vil virke logisk å skille opptegning av diagrammer og grafiske representasjoner av teoretiske fordelinger. For å følge strukturen som er brukt i Java for opptegning av objekter, vil vi komme til å lage to final klasser som inneholder statiske funksjoner for å tegne ønsket diagram eller kurve. Vi definerer da to klasser som følger:

Figur 5.7 Klassediagram for pakken graphics

Disse to klassene trenger ikke kjenne til hverandre. Diagram har ansvaret for opptegning av histogrammer og stolpediagrammer ut ifra et eller annet datasett. GraphicsDistribution vil inneholde funksjonalitet for opptegning av teoretiske fordelinger.

5.2.3 Pakken confidenceinterval

Vi følger strukturen til java.lang.Math i Java også her, så vi lager en final klasse med statiske funksjoner for de ulike typer konfidensintervaller. En enkelt klasse er nødvendig:



Figur 5.8 Klassediagram for pakken confidenceinterval

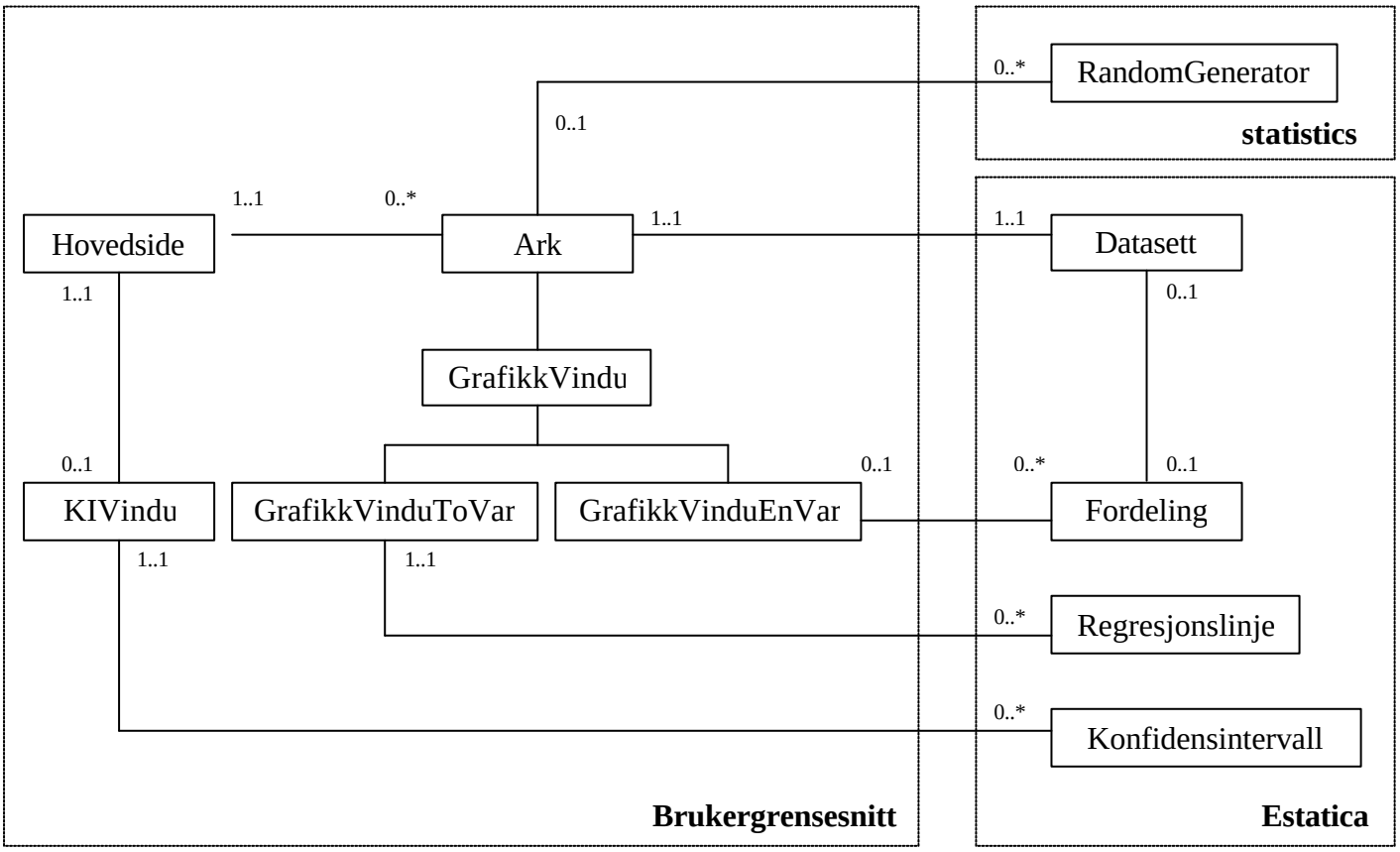
Ved å kalle de statiske funksjonene i denne klassen, vil programmereren få returnert korrekt utregnet konfidensintervall.

5.3 Design av hele systemet

Nå som vi har vært inni de ulike pakkene som må utvikles, kan vi se på hvordan disse pakkene og klassene skal kommunisere med hverandre.

5.3.1 Grovt klassediagram av hele systemet

For å forenkle klassediagrammet mest mulig, har vi sløyfet noe av arven i dette diagrammet. Når vi for eksempel tegner klassen *Datasett*, mener vi alle de underliggende konkrete klassene (se figur 5.1).



Figur 5.9 Grovt klassediagram av hele systemet

Det må understrekes at dette diagrammet bare er halve sannheten. Vi vil måtte definere mange flere klasser, og vi forventer å identifisere nye klasser også i implementasjonsfasen. Vi ser ingen grunn til å tegne opp alle klassene i dette diagrammet; det vil være mer forvirrende enn beskrivende. Klassediagrammet forsøker bare å illustrere strukturen på systemet, og hvordan pakkene kommuniserer med hverandre. De stiplede linjene omringer de tre pakkene. Vi har jo allerede demonstrert at for eksempel statistikkpakken inneholder flere klasser enn det man ser av diagrammet. Men disse andre klassene er statiske klasser og kan ikke kreeres og dermed knyttes opp i mot andre klasser. Det er jo heller ikke ønskelig.

5.3.2 Kommentarer til klassediagram

Strukturen beskrevet over vil tilfredsstille uavhengigheten mellom pakkene slik figur 3.1 i kapittel 3 legger opp til. Statistikkpakken vil være fullstendig uavhengig av de andre pakkene, da det bare er *Ark* som *bruker* randomgeneratorene. Estatica-pakken vil heller ikke ha noe avhengighetsforhold til Brukergrensesnitt, ettersom det er brukergrensesnittet som benytter seg av og sender informasjon til Estatica. Dette tilfredsstiller også kravene beskrevet i kapittel 2.

Kapittel 6

6 IMPLEMENTERING/KODING

And now we must use our little gray cells
Agatha Christie's Hercule Poirot

Den lange, lange sti over myrene og ind i skogene hvem har trakket op den?
Knut Hamsun, Markens Grøde

All hope abandon, ye who enter here.
Dante Alighieri

The spider's touch, how exquisitely fine!
Feels at each thread, and lives along the line.
Alexander Pope

The power of the visible is the invisible.
Marianne Moore

6.1 Implementeringsfasen

Denne fasen har vært en stor utfordring for oss. Vi følte at vi hadde god kontroll under designfasen, og at vi fant en god struktur der krav til systemet ble oppfylt. Ulike systemmodeller, diagrammer og beskrivelser som vi har utviklet i henhold til UML-notasjon, har virkelig hjulpet oss i implementeringen av systemet. Det ville vært fullstendig uoverkommelig å implementere applikasjonen hvis vi ikke hadde gjennomgått en god kravspesifisering- og designfase.

Ettersom systemmodellene har tatt for seg krav til systemet, har vi forhåpentligvis greid å oppfylle prosjektets mål ved å fullbyrde arbeidet vi gjorde før implementasjonsfasen.

Det har vært en del tunge stunder under kodingen og implementeringen. Enkelte ganger har vi følt at vi har låst oss fast, mens andre ganger har alt løsnet. Det har blant annet vært en utfordring å oppfylle krav til hurtighet. Vi har måttet finne raske og effektive algoritmer, og mange av dem har vi laget selv. God datastruktur har også vært et ubetinget krav for at applikasjonen skal være rask nok.

Det har vært en del tung matematikk (i våre øyne) som vi har måttet realisere. Dette gjelder spesielt da utviklingen av statistikkpakken og randomgeneratorene. Et annet moment som har beslaglagt mye tid, er selve opptegningen av diagrammer og kurver. Vi har blitt tvunget til å implementere funksjonalitet for skalering av akser, slik at grafikken havner innenfor synlig tegneområde. Dette har ikke vært lett.

En av løsningene på dette problemet har vært å normalisere tallene i et datasett (subtrahere gjennomsnitt og dividere med standardavvik). Bruker trenger ikke vite at datasettet egentlig er normalisert – for brukeren vil datasettet fremtone som det opprinnelige datasettet. På denne måten har for eksempel et normalfordelt datasettet vært standard normalfordelt, og vi har fått anledning til å operere på en kjent fordeling. Dette har hjulpet oss i stor grad, men har også ført til at visse ting måtte gjøres på litt spesielle måter.

6.2 Valg av verktøy og utviklingsmiljø

Vi har som kjent benyttet oss av Java i implementeringen av systemet. I tillegg har vi i stor grad benyttet oss av Delphi for å lage GUI-prototyper, slik at vi enklere har identifisert krav og ønsker. Når vi da endelig ble fornøyd med prototypen, implementerte vi løsningen i Java.

Det er flere grunner til at vi har brukt Java i den endelige løsningen:

- Ved videre arbeid på applikasjonen, vil det være en mulig utvidelse (ønske fra oppdragsgiverne) å legge deler av applikasjonen ut på WEB. Dette betyr at Java er det naturlige valget.
- En av oppdragsgiverne har noe kunnskaper i Java, og kan være til hjelp med å løse statistiske problemstillinger i Java.
- Ved bruk av Java oppnår vi plattformuavhengighet. Mange studenter benytter seg av Linux – maskiner, og ønsker å kjøre applikasjonen der.

Det må bemerkes at vi begynte implementeringen med Java Development Kit 1.3.1. Underveis i prosjektet utgav Sun Java Development Kit 1.4.1. Vi var dermed raske til å installere den nye versjonen, og vi har benyttet den med hell siden. Applikasjonen benytter seg derfor av den aller nyeste versjonen av Java.

Helt i starten av implementeringen, utforsket vi Java Builder – et utviklingsmiljø i Java. Vi kunne ikke se oss fornøyd med dette programmet. Mye unødvendig kode ble generert i Java Builder, men vår applikasjon krevde optimalisert og rask kode. Vi gikk derfor raskt bort fra å utvikle i Java Builder, og derfor er all kildekode for vårt system produsert av oss – dette gjelder også oppbygging av GUI. På denne måten har vi hatt full kontroll på koden, og vi kunne produsere optimalisert kode som passet vårt prosjekt.

6.3 Prinsipper og standarder

Vi bestemte tidlig i prosjektet de standarder og metoder vi skulle bruke i implementeringen. Vi har benyttet oss av java-dokumentasjon, men vi la også stor vekt på å kommentere og dokumentere kode underveis, slik at andre systemutviklere kan forstå hva som er blitt gjort. Hver java-fil inneholder følgende header:

```
/*****************************************************
*Navn: Navn på fil                                   *
* Klasse: Klassenavn                                 *
* Versjon: Versjonsnummer                            *
* Dato: Dato                                         *
* Forfattere: Navn på forfattere                     *
* Kommentarer: Hva gjør denne klassen                *
*****************************************************/
```

Vi stilte også andre krav til koden:

- Variabel og funksjonsnavn navn begynner med små bokstaver. Når navnet inneholder flere ord, har de neste ordene store bokstaver. Variabelnavn – da spesielt GUI-komponenter - har i tillegg oppsatte prefix-forkortelser.
  Eksempelvis: JLabel jlblTotalSum.
- Alle ord i et klassenavn begynner med store bokstaver.
  Eksempelvis: class BinomiskFordeling
- Det blir lagt stor vekt på oversiktlighet i koden for øvrig.
- Kildekoden vil bli dokumentert ved hjelp av javadoc – funksjonaliteten. Dette er en kjent måte å dokumentere javakode på.
- All kode og dokumentasjon har ligget på 2 maskiner (PC som står hjemme). I tillegg har vi benyttet lagringsmuligheter på skolen.
- Backup på mindre filer har blitt tatt fortløpende på diskett.
- Hver uke tok vi backup av all dokumentasjon og kildekode på CD.
- Distribusjon av backup har foregått via CD og email.

I designfasen benyttet vi oss som sagt av UML-standard. Under utvikling av design, baserte vi oss på kjente Patterns/prinsipper (GRASP), og dette ble selvfølgelig videreført i implementasjonen. Avvik fra krav- og designdokumentet har blitt dokumentert og skrevet om i denne rapporten.

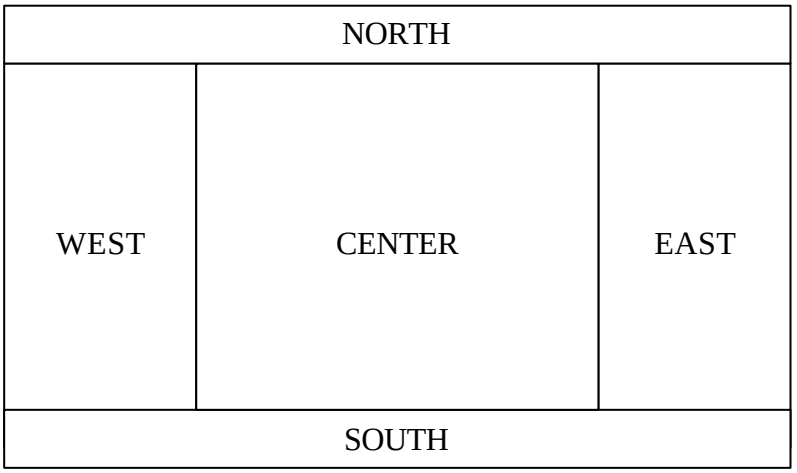
6.4 Implementering av pakken Brukergrensesnitt

Implementering av pakken Brukergrensesnitt sørger for at de prototypene som oppdragsgiver ble fornøyd med i designfasen blir frembrakt (jfr. avsnitt 4.1). I tillegg til at det grafiske brukergrensesnittet blir implementert i denne pakken, vil den også sørge for at det er mulig å kommunisere med Estatica pakken. Dis se filene/klassene måtte vi implementere for å oppfylle kravene:

- Hovedside.java
- Ark.java, ArkEnVar.java, ArkToVar.java
- GrafikkVinduEnVar.java, GrafikkPanelEnVar.java, GrafikkVinduToVar.java, GrafikkPanelToVar.java
- KIGrafikkVindu.java, KIGrafikkPanel.java
- ToVarCellRenderer.java
- VelgFordelingDialog.java, VelgFordelingDialogToVar.java
- Andre mindre klasser: InputSjekker.java, Hjelp.java, SplashEstatica.java

Sammenhengen mellom de mest essensielle klassene finnes i figur 5.9.

I avsnittene under tar vi for oss implementering av de viktigste vinduene og panelene med tilhørende funksjonalitet. Alle vinduene bruker den ferdig definerte layouten ”BorderLayout” som finnes i Java. Det vil være området i senter som tilpasser seg de andre områdene.



Figur 6.1 BorderLayout, layout definert i Java

Hovedprinsippene rundt implementasjon av de ulike vinduene er de samme. Alle vinduene vil arve fra *JFrame* som er en ferdig definert klasse i Java. De vil ha en standard filmeny som bruker lett kan identifisere seg med. I tillegg vil *Hovedside* ha en verktøylinje som legges i nord (under filmenyen). Til hvert element i hovedmenyen og verktøylinja har vi lagt til en *"ItemListener"* som lytter til hendelser. Når en hendelse trigges (bruker trykker for eksempel på en knapp i verktøylinja), tar klassen seg av dette ved å utføre tilhørende operasjon.

```
public class Vindu extends JFrame{
        ….
        private …               //Her defineres private komponenter og
        ….                      //attributter som klassen har behov for

        public Vindu(){                         //Construktor
                super("Estatica");              //Setter tittel på vinduet

                //Her bygges GUI'en opp og eventuelle
                //parametre initialiseres
        }

                //Her defineres nødvendige funksjoner

        private class HovedmenyHendelser implements ActionListener{
                public void actionPerformed(ActionEvent e){

                        //Inner class som håndterer hendelser
.                       // i hovedmenyen
                }
        }
}
```

Figur 6.2 Kodestrukturen av en klasse som "er" et vindu

6.4.1 Hovedside

Klassen *Hovedside* inneholder mest GUI funksjonalitet . Vi har implementert hovedmeny, en verktøylinje i nord, og en *JTabbedPane* i center. *JTabbedPane* er et område hvor det kan legges til flere paneler oppå hverandre. Bruker kan navigere mellom disse panelene ved hjelp av tabber. I en tab må det legges et panel som inneholder ønskede GUI komponenter. Dette vil i vårt tilfelle være et ark og derfor må ark være et panel. Kommer mer tilbake til *Ark* i avsnitt 6.4.2.

I *Hovedside* måtte vi på en eller annen måte holde orden på arkene. Vi valgte å bruke en hashtabell slik at vi lett kunne assosiere en *key* (navnet til arket) med selve arkobjektet. Dette er en effektiv datastruktur som gjør det raskt å aksessere de ulike objektene. Fordi navnet til arket er en nøkkel i hashtabellen, kan ingen ark ha samme navn. Vi implementerte funksjonalitet som sørget for å legge inn eller ta ut objekter fra hashtabellen.

Funksjonaliteten som har vært nødvendig å implementere i denne klassen kan sees i samsvar med verktøylinja og hva den har å tilby (jfr. figur 4.1). Vi har laget funksjoner som tar seg av disse ulike oppgavene:

- Funksjonalitet for legge til et ark.
  Her har vi gjort slik at det opprettes et nytt ark av riktig type avhengig av hvilken type bruker valgte:

  ```
  private Ark opprettNyttArkEnVar(String navn, int type){…}
  private Ark opprettNyttArkToVar(String navn){…}
  ```

  Stringen ”navn” vil være nøkkelen som blir brukt for å legge det nye arkobjektet i hashtabellen. Variabelen ”type” sier om det envariable arket skal godta heltall eller reelle tall.
- Funksjonalitet for å fjerne et ark.
- Vi må fjerne gjeldende tab i *JTabbedPane,* og tilhørende arkobjekt må fjernes fra hashtabellen.
- Funksjonalitet for å åpne et ark (åpne fil).
  Når bruker skal åpne en fil, må vi vite hvilket type ark filen skal åpnes i. Dette løste vi ved at det vises en popupmeny når hurtigtasten ”Åpne” trykkes. I denne menyen må bruker velge type ark han/hun ønsker å åpne filen i.
- Funksjonalitet for å lagre et ark (lagre til fil)
  I tillegg til standard lagringsfunksjonalitet, har vi gjort det slik at det synes på tabben om et ark er lagret eller ikke (jfr. figur 4.2 og figur 4.3). Rødt ikon betyr at arket har blitt modifisert og ikke lagret, blått betyr at siste oppdatering er lagret.
- Funksjonalitet for å opprette et konfidensintervall vindu
- Funksjonalitet for å få hjelp

Vi viser ikke eksempler på disse funksjonene, men henviser isteden til kildekoden og javadokumentasjonen.

6.4.2 Ark

Klassen *Ark* må være et panel, det vil si at den arver fra klassen *JPanel* som er ferdig definert i Java. Dette begrunnet vi i avsnitt 6.4.1. Dette panelet benytter også BorderLayout i likhet med vinduene. Her har vi lagt en tabell (*JTable*) i center, et panel med knapper i sør og et panel med måldata i øst (jfr. figur 4.2 og figur 4.3). I tabellen kan det legges inn tall.

Ark er en abstrakt klasse som har to underklasser (jfr. figur 4.8) og vi ønsker forskjellig utseende på tabellen avhengig av type ark:

- *ArkEnVar*
  Her valgte vi å bruke standard ”LookAndFeel” på tabellen.
- *ArkToVar*
  Vi ønsket å på en eller annen måte skille mellom tallpar i tabellen. Dette løste vi ved å definere vår egen *TableCellRenderer* som satte farge på kolonnene og som la til tabellheadere (vekselsvis x og y).

Et *Ark* har ett og bare ett datasett. Type datasett er avhengig av hvilket ark som har blitt opprettet – dette ordnes i de konkrete arkene. Dersom bruker ønsker å generere et datasett automatisk, vil *Ark* benytte seg av klassen *RangomGenerator* for å opprette et nytt datasett.

Om bruker velger manuelt å legge inn tall i tabellen, har vi gjort det slik at måldata oppdateres automatisk for hver gang han/hun oppdaterer en celle i tabellen. Dette er et avvik fra kravene, men positivt i så måte. Det står mer om dette i avsnitt 6.5.1.

Funksjonaliteten som har vært nødvendig å implementere i denne klassen kan sees i samsvar med de knappene som ligger sør i panelet. Vi har laget funksjoner som håndterer de ulike hendelsene som trigges :

- Rens ark
  Her har vi implementert funksjonalitet som sørger for at hele tabellen renses for tall.
- Generer data
  Når denne knappen trigges, vil bruker få displayet en VelgFordelingDialog (jfr. figur 4.4) hvor han/hun velger ønsket fordeling som skal genereres.
- Vis diagram
  Når denne knappen trigges, får bruker displayet et *GrafikkVindu*

Annen funksjonalitet i *Ark* er filhåndtering som skjer ved at *Hovedside* kaller funksjoner i denne klassen etter at hendelsene har blitt trigget i *Hovedside* sin verktøylinje eller hovedmeny.

6.4.3 GrafikkVindu

Klassen *GrafikkVinduEnVar* og *GrafikkVinduToVar* innholder mest GUI funksjonalitet . Vi har implementert hovedmeny, et *GrafikkPanelEnVar/GrafikkPanelToVar* i center og en liste med teoretiske modeller i øst.

Grafikkpanelene er egne klasser som arver i fra *JPanel*. De inneholder en funksjon

```
public void paintComponent(Graphics g){…}
```

som vi har overstyrt (overloaded), og den har som oppgave å tegne grafikken ut til skjerm. I tillegg har hvert panel en vektor med teoretiske modeller som har ansvaret for å tegne seg selv. Dette skjer ved at panelet går i gjennom vektoren og kaller funksjonen tegn(…) som er definert i modellene:

```
while(enum.hasMoreElements())          //Går igjennom hele vektoren
        enum.nextElement().tegn(…);    //Ber en og en modell tegne seg på
                                       //panelet
```

Vi skiller mellom de teoretiske modellene som er tilgjengelig for *GrafikkVinduEnVar* og *GrafikkVinduToVar*:

- Envariabel teoretisk modell: Fordeling
- Tovariabel teoretisk modell: Regresjonslinje

Grafikkpanelene har funksjonalitet som støtter oppdatering av vektoren med de teoretiske modellene. Forandringer i lista (som finnes øst i grafikkvinduene) vil da enkelt bli oppdatert når en ny modell legges til eller slettes. Det eneste vi måtte gjøre var å legge vektoren inn i lista på ny ved hver oppdatering. Grafikkvinduene inneholder funksjonalitet for hvordan man kan legge til nye elementer i lista. Denne hendelsen trigges ved at man trykker på knappen ”Legg til” som finnes under lista (jfr. figur 4.5 og figur 4.6).

6.4.4 KIGrafikkVindu

Klassen *KIGrafikkVindu* innholder mest GUI funksjonalitet . Vi har implementert hovedmeny, et *KIGrafikkPanel* i center og et panel med nødvendige tekstfelter for input i øst (jfr. figur 4.7).

Grafikkpanelet er, med likhet av grafikkpanelene, en egen klasse som arver i fra *JPanel*. Den inneholder også funksjonen

```
public void paintComponent(Graphics g){…}
```

som vi har overstyrt (overloaded), men her har den ansvaret for å tegne de simulerte konfidensintervallene ut til skjerm. Dette foregår på samme måte som i grafikkpanelene ved at panelet har en vektor med intervallene som har ansvaret for å tegne seg selv. For hver simulering opprettes det et konfidensintervallobjekt som legges inn i denne vektoren.

Øvrig funksjonalitet som er implementert i denne klassen:

- Funksjonalitet som støtter forsinkelse ved simulering.
  Vi har lagt til en *JSlider* nederst i høyre hjørne av vinduet (jfr. figur 4.7.**C**). Når bruker beveger på denne, kan han/hun selv bestemme hastigheten intervallene skal tegnes ut til skjerm (et etter et). Dette skjer ved hjelp av et *Timer*-objekt, som trigges hvert n'te millisekund.
- Funksjonalitet for å lese inn og sette nødvendige parametre gitt av bruker (jfr. figur 4.7.**A**). Denne informasjonen sier noe om type og antall intervaller.

6.4.5 Øvrige klasser i pakken Brukergrensesnitt

VelgFordelingDialog

Vi måtte implementere to forskjellige dialoger, en for envariable data og en for tovariable data. Dette var nødvendig fordi ved genering av tovariable data måtte også alpha og beta angis av bruker. I tillegg var det ikke behov for å skille mellom diskrete og kontinuerlige fordelinger. I VelgFordelingDialog for envariable data kan vi skille mellom disse fordelingene ved å enable radiobuttons for den ene typen, se figur 4.4 hvor bare de kontinuerlige fordelingene er enabled.

InputSjekker

Denne klassen inneholder funksjoner som benyttes ved testing av input fra bruker. Hvis input er ugyldig, displayes et modalt vindu til bruker med en egnet feilmelding. Dette gjør at kravene om robusthet og brukervennlighet imøtekommes.

Hjelp

Kravet om en hjelpedialog løste vi ved å implementere et vindu som leser inn en tekstfil. Det finnes en slik fil for hver av de største vinduene (Hovedvindu, GrafikkVinduEnVar, GrafikkVinduToVar og KIGrafikkVindu). Avhengig av hvilket vindu som har åpnet hjelpen, vil tilhørende tekstfil leses inn i dialogen.

SplashEstatica

Selv om det ikke er noe krav om å implementere en Splash, synes vi det var behørig med en.

6.5 Implementering av Estatica

Implementeringen av pakken Estatica følger beskrivelsen av design i avsnitt 5.1. Som forventet ble virkeligheten noe mer kompleks, men hovedprinsippene gjelder fortsatt.

Følgende filer og klasser har blitt implementert:

- Pakken datasett: DatasettDatasett.java, DatasettEnVar.java, DatasettEnVarD.java, DatasettEnVarK.java, Tallpar.java
- Pakken fordeling: Fordeling.java, Normalfordeling.java, Eksponentialfordeling.java, Binomiskfordeling.java, Poissonfordeling.java, Uniformfordeling.java
- Pakken konfidensintervall: Konfidensintervall.java
- Pakken regresjon: Regresjonslinje.java
- TabellModell.java, TabellModellEnVar.java, TabellModellToVar.java

6.5.1 Datasett

Vi måtte finne en god struktur for å representere et datasett. Klassene ble implementert som vist i figur 5.1. Det har blitt stilt store krav til hurtighet, og det har vært en stor utfordring å finne gode og fornuftige algoritmer for beregninger av måldata.

Et *Datasett*-objekt må ha kontroll over alle tallene som ligger i et datasettet, og tilhørende måldata som gjennomsnitt, median, varians osv. Men dette er faktisk ikke helt sannferdig. I Java benyttes klassen *JTableModel* for å holde orden på informasjonen som ligger i cellene i en JTable (Javas implementasjon av en tabell). Default vil disse dataene ligge i en vektor i tabellmodellen, men vi har implementert en egen tabellmodell for å holde orden på tallene i tabellen. Vi har definert en egen og effektiv struktur, som har vist seg å oppfylle krav om hurtighet og robusthet. Vi implementerte en 16*125 *array* av Object'er istedenfor en *vektor* av Object'er. På denne måten kan tabellmodellen holde orden på alle tallene, og det er enkelt og raskt å aksessere en slik array. Vi definerte en tabellmodell for envariable datasett og en for tovariable datasett. Disse to tabellmodellene arver fra den abstrakte klassen *TabellModell* (se siste punktet ovenfor), som igjen utvider *JTabelModel*.

Når vi nå har laget en egen tabellmodell, og denne inneholder de tallene som finnes i tabellen, er det faktisk ikke nødvendig at datasett-objektene omfatter sine tall – det blir faktisk overflødig. Derfor vil et datasett-objekt bare ha informasjon om sine respektive måldata. Dette er dog ikke helt korrekt når det gjelder *DatasettEnVar*. I denne klassen må vi ha et sortert tallsett for å kunne beregne medianen.

Datasettet må jo på en eller annen måte kunne regne ut sine måldata, og disse måldataene vil være forskjellig avhengig av om datasettet er envariabelt eller tovariabelt. For tovariable datasett er jo et tall egentlig et tallpar. Derfor har vi også definert en klasse, *Tallpar,* som representerer et tallpar. For å representere et tall for envariabel datasett, holder det å benytte Javas ferdig definerte *Integer* og *Double*.

Nå som strukturen og prinsippene er forklart, kan vi se nærmere på hvordan datasettene ble implementert. Alle de ulike *Datasett*-klassene følger samme prinsipp:

```
public class Datasett… extends Datasett{
        …
        private …                       //Her defineres de ulike måldata
        private …                       //som hører til dette datasettet.
        …

        public void leggInnTall (Object element){
                //Her ligger funksjonalitet for å legge inn et tall i datasettet.
                //Et element kan være en Integer, Double eller Tallpar
                //avhengig av datasettets type.
                //Datasettet trenger som sagt ikke å ta vare på dette elementet,
                //men bruker elementet til å oppdatere sine måldata.
        }
        public boolean erTom(){
                //Funksjon som returnerer true hvis datasettet er tomt.
        }
        public void slett(){
                //Funksjon som kalles når datasettet skal slettes. Alle måldata
                //blir nullstilt.
        }
        public void tegn(…){
                //Funksjon som tegner en grafisk representasjon av datasettet;
                //histogram hvis datasettet er kontinuerlig, stolpediagram hvis
                //datasettet er diskret og spredningsplott hvis det er tovariabelt.
        }
}
```

Figur 6.3 Implementasjon av datasett

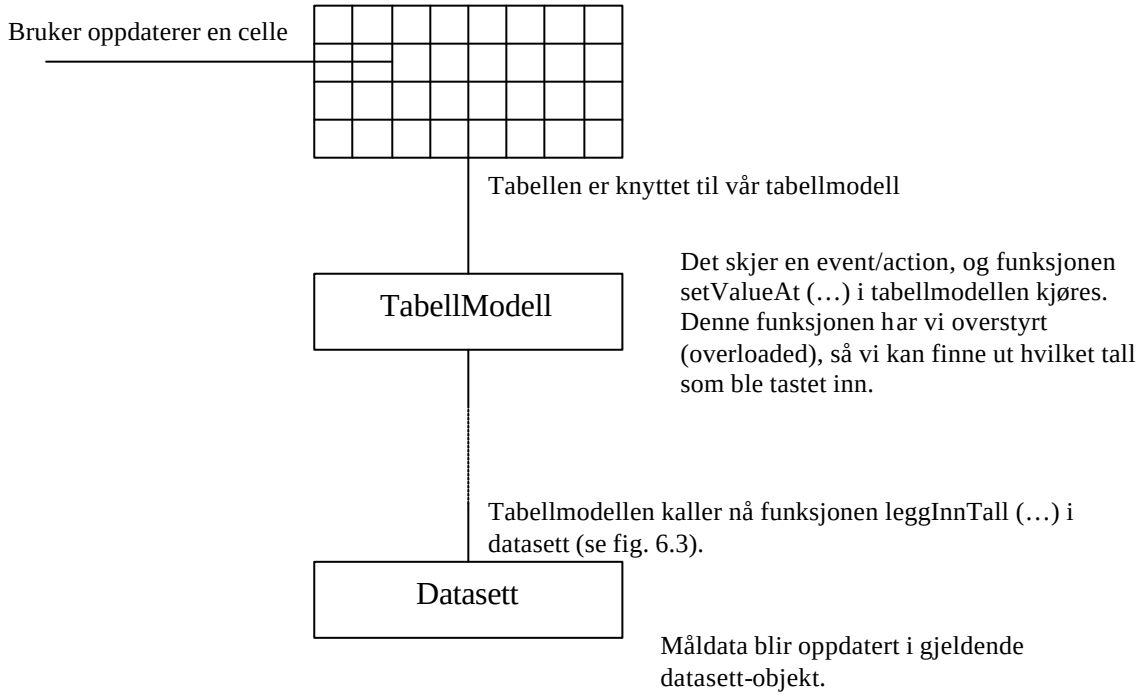
Datasett-objektene inneholder selvfølgelig mer enn dette som her er vist, men disse metodene vil være de som beskriver hvordan et objekt av type *Datasett* kan brukes av andre objekter. Når funksjonen *leggInnTall(element)* kalles, vil alle måldata bli oppdatert i henhold til det nye tallet/tallparet som ble lagt til.

Å oppdatere måldata er ikke trivielt. Vi har implementert egne metoder i *Datasett*-objektene som gjør denne jobben, men disse kan ikke nåes utenfra og er definert som *private.* For å for eksempel beregne ny varians etter at nytt tall er lagt til, må vi definere en funksjon som beregner ny varians med bakgrunn i gammel varians og det nye tallet. Slike funksjoner

må defineres for alle de ulike måldataene i et datasett. Beregninger på disse måldataene må følge krav definert i vedlegg **A**. Vi tar ikke med noen eksempler på disse metodene her, men vi henviser til kildekode for gjeldende klasser. Selv om flere av metodene er kompliserte, har de vist seg å være raske og effektive.

Bruker taster inn sine tall i tabellen som ligger i et ark. Som sagt i avsnitt 6.4, fant vi en god løsning på krav om hurtighet og brukervennlighet for inntasting av tall. Tanken dreide seg først om at bruker skulle taste inn sine tall, og trykke på en knapp, "Ferdig", for å få beregnet måldata for datasettet. Dette ville ha fungert greit og kravene ville vært oppfylt, men vi mener å ha funnet en bedre løsning som eliminerer problemene med hurtighet. Brukervennligheten økes også betraktelig. Når bruker taster inn sine tall, blir nå måldata beregnet *underveis.*

Når en celle i tabellen blir oppdatert (bruker taster inn et nytt tall, fjerner et tall eller forandrer tallet i en celle), skjer det en event/action. Denne event'en kaller en funksjon i tabellmodellen (*JTableModel*) til gjeldende tabell (*JTable*). Dette skjer fordi det er tabellmodellen som vet hvordan en celle i tabellen skal oppdateres. Nå er det jo vi som har implementert tabellmodellen (vi bruker ikke Javas default tabellmodell), så vi har full kontroll på nettopp denne event'en. Vi kan derfor for hver slik event, legge det nye tallet/tallparet inn i datasettet ved å kalle funksjonen *leggInnTall(element).*



Figur 6.4 Implementasjon av dynamisk oppdatering av *Datasett*

På denne måten kan vi dynamisk oppdatere datasettet, slik at målverdiene blir kontinuerlig oppdatert mens bruker taster inn tall – denne oppdateringen skjer momentant. Figuren over er en forenkling av virkeligheten , men prinsippet blir belyst.

Nå kan det jo hende at det ikke blir lagt inn et nytt tall, men en celle kan ha blitt oppdatert eller et tall ble fjernet fra tabellen. Metoden beskrevet i figur 6.4 vil ikke kunne håndtere dette problemet. Vi har derfor utvidet klassen *Datasett* med følgende metode:

```
public void fjernTall (Object verdi){
        //Denne funksjonen kalles når et
        //tall/tallpar skal fjernes fra datasettet.
        //Funksjonen må derfor oppdatere måldata.
}
```

Figur 6.5 Metoden som fjerner et tall i et datasett

Denne funksjonen gir oss mulighet til å fjerne et tall fra et datasett. Når en celle oppdateres og event'en kjøres, kan vi i tabellmodellen finne ut om det var et tall i cellen fra før eller ikke. Dermed vet vi hvordan vi skal kunne holde datasettet oppdatert:

Et nytt tall/tallpar, X, ble lagt til. Tabellmodellen gjør følgende:

```
datasett.leggInnTall(X);
```

Et tall/tallpar, Y, ble fjernet fra tabellen. Tabellmodellen gjør følgende:

```
datasett.fjernTall(Y);
```

Et tall/tallpar, Y, ble byttet ut med et annet, X. Tabellmodellen gjør følgende:

```
datasett.fjernTall(Y);
datasett.leggInnTall(X);
```

Figur 6.6 Tabellmodellen oppdaterer datasettet

Denne metoden vil fungere uavhengig av om datasettet er envariabelt eller tovariabelt. Om datasettet er envariabelt diskret, vil vi sende med et *Integer*-objekt til datasettet. Envariable kontinuerlige datasett mottar *Double*-objekter. For tovariable datasett vil vi benytte klassen *Tallpar* (som vi har implementert). Her vil tabellmodellen finne ut hvilke to tilhørende celler som har blitt oppdatert, og kreere et *Tallpar*-objekt som sendes til *Datasett*-objektene. Resten vil funksjonene i *Datasett* ta seg av.

Av figur 6.3 går det frem at alle *Datasett*-objektene har en funksjon, tegn(). Denne funksjonen har som oppgave å tegne en grafisk representasjon av datasettet. Ettersom hver objekt av type *Datasett* selv vet om det er et envariabelt diskret, envariabelt kontinuerlig eller tovariabelt, kan denne funksjonen tegne riktig type diagram.

Det ligger en del mer funksjonalitet i *Datasett*, men det vil være unødvendig å ta med alt her. Blant annet finnes det funksjonalitet for å normalisere et datasett og beregne frekvenstabell (relative frekvenser). Vi henviser til javadokumentasjonen og kildekode.

6.5.2 Fordeling

De klassene som skal representere en fordeling har samme struktur som i figur 5.2. Denne strukturen fører til at vi enkelt kan skille ulike typer fordelinger, og det vil være en lett operasjon å utvide med flere fordelinger senere. En fordeling trenger attributter som beskriver fordelingen, fordi de ulike fordelingene har ulike parametre. Svært forenklet vil en *Fordeling* se slik ut:

```
public class …fordeling extends Fordeling{
        …
        private..                       //Her defineres parametrene til
        …                               //fordelingen.

        …fordeling( parametre ){        //Konstruktor som
                                        //initialiserer en fordeling.
        }
        …                               //Her defineres funksjoner
        …                               //for å hente og sette parametrene

        tegn(…){                        //Funksjon som tegner denne
                                        //fordelingen til skjerm.
        }
}
```

Figur 6.7 Implementasjon av en fordeling

Vi går ikke særlig nærmere på denne klassen. Klassens hovedfunksjon er å representere en fordeling. Eksempelvis vil *DatasettEnVar* ha et *Fordeling*-objekt hvis datasettet stammer fra en eller annen fordeling. *GrafikkVinduEnVar* inneholder også flere fordelinger (de fordelingene som skal tegnes til skjerm). *GrafikkVinduEnVar*-objektet kan helt enkelt be fordelingene sine tegne seg selv til skjerm ved å kalle funksjonen tegn() (se figur over). Dette vil være naturlig, for det er bare fordelingen selv som vet hvordan den skal representere seg selv grafisk.

Vi kan for eksempel kreere en normalfordeling ved å opprette et *Normalfordeling*-objekt på denne måten:

```
Fordeling nf = new Normalfordeling(forventningsverdi, standardavvik);
```

Vi kan be dette objektet tegne seg selv ved å kalle funksjonen tegn():

```
nf.tegn(…);
```

Øvrig funksjonalitet for å behandle fordelinger finnes i statistikkpakken.

6.5.3 Konfidensintervall

Et objekt av type *Konfidensintervall* har samme hensikt som et objekt av type *Fordeling*; vi trenger en representasjon av et konfidensintervall. *KIGrafikkVindu* skal simulere konfidensintervaller, og dette vinduet må på en eller annen måte holde orden på de konfidensintervallene som skal tegnes til skjermen. Vi har implementert denne klassen:

```
public class Konfidensintervall {
        …
        private..                       //Her defineres grensene
        …                               //for dette intervallet.

        Konfidensintervall ( grenseverdier ){
                                        //Konstruktor som
                                        //initialiserer et intervall.
        }
        …                               //Her defineres funksjoner
        …                               //for å hente og sette grensene.

        tegn(…){                        //Funksjon som tegner dette
                                        //konfidensintervallet til skjerm.
        }
}
```

Figur 6.8 Implementasjon av et konfidensintervall

Også her finnes en metode som tegner intervallet til skjerm. KIGrafikkVindu kan dermed bare kalle funksjonen tegn() for et *Konfidensintervall*-objekt for å få tegnet intervallet.

Øvrig funksjonalitet for beregning av flere typer konfidensintervaller finnes i statistikkpakken.

6.5.4 Regresjon

GrafikkVinduToVar trenger å holde på flere regresjonslinjer. En regresjonslinje kan enkelt implementeres, fordi en linje er av typen $y = \alpha + \beta x$. Vi definerer derfor en klasse som tar vare på α og β, og denne klassen vil også ha en funksjon tegn() som tegner linjen til skjerm.

```
public class Regresjonslinje {
        …
        private..                       //Her defineres α og β
        …                               //for denne linjen.

        Regresjonslinje (α og β){
                                        //Konstruktor som
                                        //initialiserer ei linje.
        }
        …                               //Her defineres funksjoner
        …                               //for å hente og sette α og β.

        tegn(…){                        //Funksjon som tegner denne
                                        //regresjonslinja til skjerm.
        }
}
```

Figur 6.9 Implementasjon av en regresjonslinje

6.6 Implementering av pakken statistics

Denne pakken følger beskrivelsen av design i avsnitt 5.2. Det stilles selvfølgelig store krav til matematiske beregninger, og disse kravene er beskrevet i vedlegg **A**. For å lage en fullstendig statistikkpakke som Estatica kan benytte seg av, har vi implementert følgende klasser og filer:

- Statistics.java
- Pakken graphics: Diagram.java, GraphicsDistribution.java
- Pakken confidenceInterval: ConfidenceInterval.java
- Pakken randomgenerators: RandomGenerator.java, RandomNormal.java, RandomEksponential.java, RandomBinomial.java, RandomPoisson.java, RandomUniform.java

6.6.1 Statistics

Denne klassen skal, som beskrevet i avsnitt 5.2, ha samme struktur som klassen java.lang.Math i Java biblioteket. Statistics inneholder derfor generelle funksjoner for statistiske beregninger. Her har vi altså lagt funksjonalitet for beregning av sannsynlighet i et gitt punkt for de ulike fordelingene. Metoder for å beregne ulike måldata har som sagt blitt lagt direkte i *Datasett*-klassene, fordi de ikke ble generelle nok til å ligge i statistikkpakken.

Vi definerte følgende klasse i filen Statistics.java:

```
public final class Statistics{

        public static double probabilityNormal(…){ … }
        public static double probabilityEksponential (…){ … }
        public static double probabilityBinomial(…){ … }
        public static double probabilityPoisson(…){ … }
        public static double probabilityUniform(…){ … }

                //Definering av andre hjelpefunksjoner
}
```

Figur 6.10 Implementering av klassen Statistics

Vi ønsker for eksempel å få sannsynligheten for normalfordeling med forventning lik 3 og standardavvik lik 2 i punktet x ved å si:

```
sannsynlighet = Statistics.probabilityNormal (x, 3, 2);
```

Kravene beskrevet i SK 3 vil dermed være innfridd.

Alle disse funksjonene regner ut sannsynlighet i henhold til de matematiske krav beskrevet i vedlegg **A**. Vi går ikke inn på hver av disse funksjonene, men holder oss til å illustrere et eksempel – utregning av sannsynlighet for binomisk fordeling i et gitt punkt, x:

```
public static double probabilityBinomial(int x, int n, double p){
        return binomial(n,x)*Math.pow(p,x)*Math.pow((1-p),(n-x));
    }

public static double binomial(int a, int b){          //Funksjon for å regne binomial
                                                      //koeffisienten a over b
    if(a == 0 || b==0 || (a == b)) return 1;
    int ab = a - b;
    double temp=1;
    for( ; a > 0; a-- ){
        temp *= ((double)a / (double)(ab * b));
        if(ab > 1) ab--;
        if(b > 1) b--;
    }
    return temp;

}
```

Figur 6.11 Eksempel på utregning av sannsynlighet for binomisk fordeling

Her ser vi et eksempel på en hjelpefunksjon som probabilityBinomial() må benytte seg av. Funksjonen binomial(), returnerer binomialkoeffisienten a over b. Vi måtte utvikle denne algoritmen selv, for kravet til hurtighet er stor. Denne algoritmen har en effektivitet på n = a, og dette er også ”WorstCase”. O-notasjonen blir derfor lineær, O(n), der n er lik a i binomialkoeffisienten a over b. Dette tilfredsstiller våre krav for en slik algoritme.

Vi har prøvd flere algoritmer, men dette har gått utover hurtigheten. I tillegg har tidligere algoritmer ikke taklet store tall. I formelen for binomialkoeffisienten inngår blant annet utregning av n!, der n er lik a i binomialkoeffisienten a over b. For store tall blir mellomregningene uhåndterlige (allerede ved 20! spøker det for de primitive datatypene vi har til rådighet). Algoritmen over takler n på over 400!

For de andre funksjonene, henviser vi til javadokumentasjonen og kildekode.

6.6.2 Pakken graphics

Denne pakken skal inneholde funksjonalitet for å tegne diagrammer og teoretiske fordelinger. Pakken følger strukturen som beskrevet i avsnitt 5.2.2. Vi har implementert to klasser, Diagram (ansvar for opptegning av histogrammer og stolpediagrammer) og GraphicsDistribution (ansvar for opptegning av teoretiske fordelinger).

```
public final class Diagram{

        public static void drawBarGraph(…){ … }          //Tegner stolpediagram

        public static void drawHistogram(…){ … }         //Tegner histogram

}
```

Figur 6.12 Implementasjon av klassen Diagram

Hver av disse funksjonene mottar nok informasjon til å tegne et diagram: frekvenstabell (relative frekvenser) og hvor den skal tegne diagrammet.

Klassen GraphicsDistribution har tilsvarende struktur:

```
public final class GraphicsDistribution{

        public static void drawNormal (…) { … }          //Tegner normalfordeling
        public static void drawEksponential (…) { … }    //Tegner eksponentialfordeling
        public static void drawBinomial (…) { … }        //Tegner binomisk fordeling
        public static void drawPoisson (…) { … }         //Tegner poissonfordeling
        public static void drawUniform (…) { … }         //Tegner uniform fordeling

}
```

Figur 6.13 Implementasjon av klassen GraphicsDistribution

Hver av disse funksjonene mottar argumenter som forteller hva som skal tegnes og hvor det skal tegnes. Funksjonene benytter seg av klassen *Statistics* for å beregne sannsynlighet i hvert punkt.

Vi går ikke nærmere inn på funksjonene i Diagram og GraphicsDistribution, men henviser til javadokumentasjonen og kildekode.

6.6.3 Pakken confidenceInterval

Pakken inneholder bare en klasse, ConfidenceInterval. Denne klassen innehar funksjonalitet for å beregne ulike typer konfidensintervaller med gitte konfidens nivåer (jfr. krav i vedlegg **A**).

```
public final class ConfidenceInterval{

        public static double[] zIntervall (…) { … }

        public static double[] tIntervall (…) { … }

        public static double[] sIntervall (…) { … }

                //Definering av andre hjelpfunksjoner

}
```

Figur 6.14 Implementasjon av klassen *ConfidenceInterval*

Disse funksjonene beregner de konfidensintervaller som applikasjonen skal håndtere. I tillegg har vi implementert konfidensintervaller for sannsynligheten p og raten lambda, men applikasjonen vår skal ikke håndtere disse intervallene.

Funksjonene mottar nok informasjon til å beregne et konfidensintervall, og de returnerer en array som inneholder grensene i intervallet.

6.6.4 Pakken randomgenerators

Pakken randomgenerators inneholder de klasser som er beskrevet i avsnitt 5.2.1. Disse klassene skal benyttes for å generere tall ut ifra en gitt fordeling, og de må implementeres på forskjellige måter. Dette er, i motsetning til de andre klassene i pakken statistics, konkrete klasser som det kan opprettes objekter av.

Alle randomgeneratorene benytter seg av Javas ferdig definerte klasse, java.util.Random. Denne randomgeneratoren benytter seg av en metode som kalles "Linear Congrential Method", introdusert av D. Lehmer i 1951 og beskrevet av Donald E. Knuth i *"The Art of Computer Programming,* Volume 2: *Seminumerical Algorithms"*. Denne algoritmen for å produsere pseudo-random-tall er den mest kjente og brukte. Denne er beskrevet i dokumentasjonen for Java, og vi forklarer ikke metoden nærmere her. Det er uansett vår oppgave å bruke denne randomgeneratoren til å produsere tall for en gitt fordeling med tilhørende parametre.

De randomgeneratorene vi har implementert ser generelt slik ut:

```
public class Random… extends RandomGenerator{

        public int/double next…(){ … }          //Returnerer neste randomgenererte tall

        public Object nextRandom(){             //Returnerer neste randomgenererte tall,
                return new Integer/Double(next… ());   //men i form av et Object
        }

                //Flere funksjoner må også defineres, men vi tar ikke med disse her

}
```

Figur 6.15 Implementasjon av klassene i pakken randomgenerators

Vi har implementert en funksjon next…( ) for hver av de ulike fordelingene (nextNormal(), nextBinomial() osv.). I tillegg har vi implementert funksjonen nextRandom(), som returnerer et Object. Dette har vi gjort fordi vi ikke alltid vet hvilken randomgenerator vi bruker. Vi kan da bruke *RandomGeneraor* – referanser til å referere til de andre generatorene. På grunn av *polymorphism* og *dynamic method binding* blir korrekt metode kjørt, og vi får returnert en *Object*- referanse vi kan forholde oss til. Selve logikken ligger i de ulike next…() – metodene, og de har ansvaret for å generere tall ut ifra gitt fordeling.

Hvis jeg ønsker meg en randomgenerator som generer poissonfordelte tall med $\lambda = 2$ og $t = 3$, oppretter jeg følgende objekt ved å si:

```
RandomGenerator rg = new RandomPoisson(2, 3, seed);
```

Jeg kan også selv sette seed'en til randomgeneratoren. Nå trenger jeg bare kalle funksjonen nextPoisson() eller nextRandom() så mange ganger jeg ønsker:

```
int tall = nextPoisson();
```

tall vil her være et poissonfordelt tall med parameter $\lambda = 2$ og $t = 3$.

Det blir altfor detaljert om vi her skulle gå inn på alle metodene for alle de ulike randomgeneratorene. Det holder å illustrere med et eksempel – generering av et eksponentialfordelt tall med gitt lambda (λ).

Sannsynlighet i et punkt, t, for eksponentialfordelingen med gitt lambda (λ) kan beskrives som:

$$f(t) = \lambda e^{-\lambda t}$$

Den kumulative fordelingsfunksjonen blir da:

$$F(t) = \int_0^t \lambda e^{-\lambda t}$$

Utregning av dette integralet ved substitusjon gir:

$$F(t) = 1 - e^{-\lambda t}$$

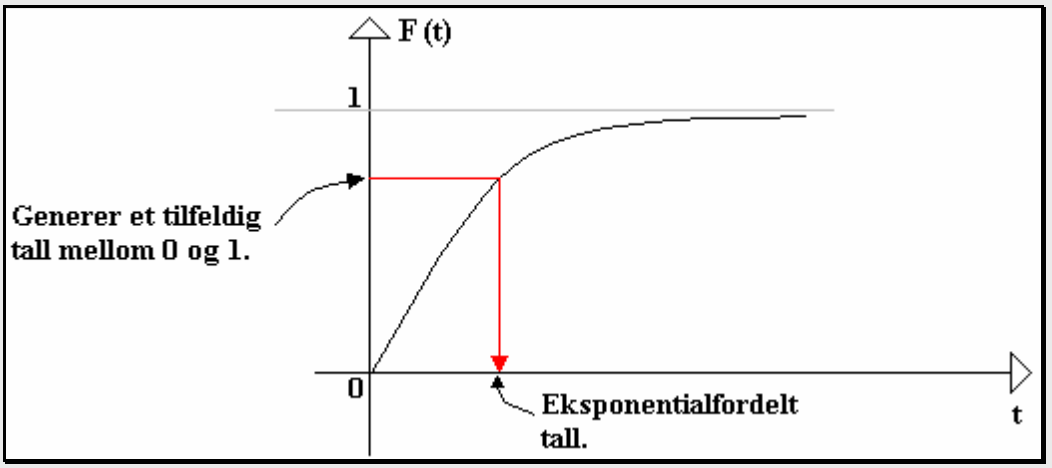
Denne funksjonen kan inverteres, og dette gir at:

$$t = -\frac{1}{\lambda} \ln(1-y)$$

Figur 6.16 Utregning av den inverterte fordelingsfunksjon for eksponentialfordelingen

Vi kan bruke den inverterte fordelingsfunksjonen for å trekke et eksponentialfordelt tall hvis vi har en randomgenerator som produserer tall mellom 0 og 1 med uniform sannsynlighet. Og det har vi jo! Dette vil foregå på følgende måte – et diagram vil illustrere prinsippet:

Kumulative fordelingsfunksjon ser slik ut:



Ved å bruke den inverterte fordelingsfunksjon, har vi en funksjon av t. Vi kan da bare sette inn et tilfeldig tall mellom 0 og 1, og vi får et eksponentialfordelt tall med parameter λ.

Figur 6.17 Diagram som viser hvordan generere et eksponentialfordelt tall

Vi implementerer derfor nextEksponential() på følgende måte:

```
public double nextEksponential(){
        return (-1/lambda)*Math.log(1-randomGenerator.nextDouble());
        }
```

Nå har det seg slik at vi ikke kan finne den inverterte fordelingsfunksjonen for alle de andre fordelingene – vi klarer for eksempel ikke engang å regne ut fordelingsfunksjonen for normalfordelingen, til det er integralet for komplisert. For disse andre fordelingene benytter vi oss av helt andre metoder. Vi går ikke nærmere inn på flere av disse metodene her, men henviser til javadokumentasjonen og kildekoden.

6.7 Andre momenter omkring implementering

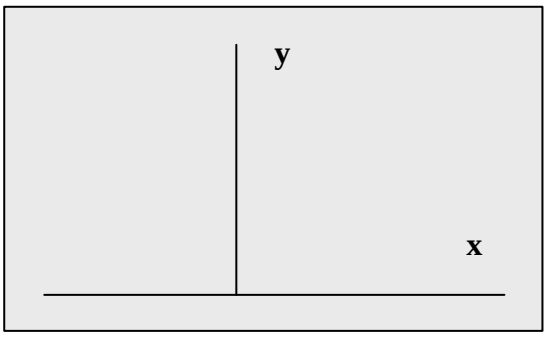
6.7.1 Opptegning av grafiske representasjoner

Opptegning av grafikk har foregått i det vanlige kartesiske koordinatsystem som vi er vant til. I Java vil dette koordinatsystemet ha origo i øvre venstre hjørne:

Figur 6.18 Koordinatsystemet i Java

Vi måtte derfor, før hver opptegning, utføre visse transformasjoner. Ved hjelp av en lineær transformasjon, flyttet vi origo til midten av arket (*affine transformation*). I tillegg vendte vi y-aksen ved hjelp av følgende matrise:

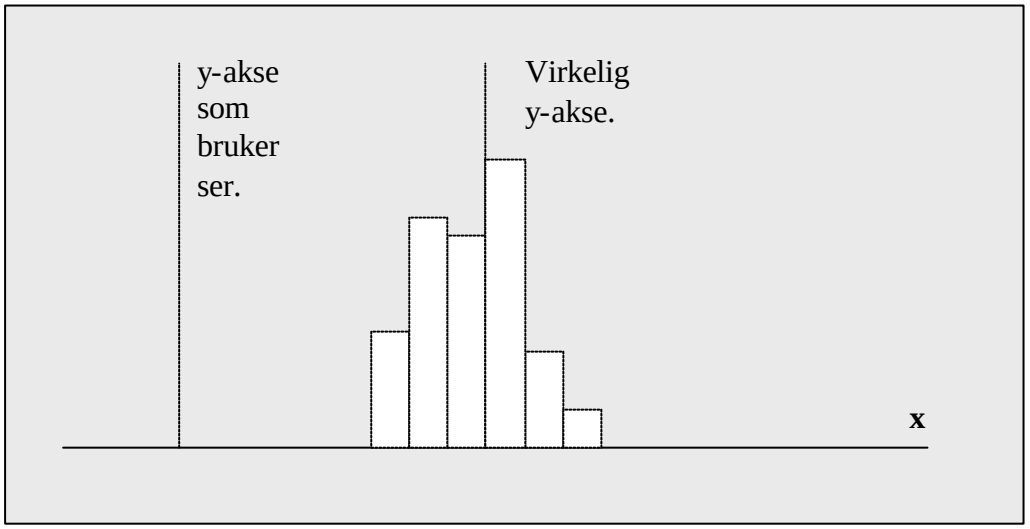
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Figur 6.19 Resulterende koordinatsystemet

Ved opptegning benyttet vi oss av Javas primitive objekter (piksler, linjer, rektangler osv.), definert i java.awt.Graphics og java.awt.Graphics2D. Ved hjelp av disse kunne vi konstruere histogrammer, stolpediagrammer, spredningsplott og kurver og linjer.

Dette var en stor utfordring som vi brukte mye av tiden på. Altfor ofte havnet grafikken utenfor bildet. Dette problemet ble løst ved å normalisere datasettene (trekke fra gjennomsnitt og dividere med standardavviket). På denne måten kunne vi jobbe med et koordinatsystem som brukeren ikke ser. Brukeren trenger ikke å få vite at et datasett er normalisert – brukeren får presentert sitt diagram som om datasettet var det opprinnelige. Dette førte også til at et diagram alltid blir tegnet midt på skjermen.



Figur 6.20 Normalisering av et datasett

Ved normalisering av et datasett, havner forventningsverdien i origo (= 0). For brukeren kan jo dette fortone seg noe rart hvis forventningen egentlig er 4,3. Datasettet (histogrammet eller stolpediagrammet), tegnes i forhold til den virkelige y-aksen som vist i figuren over. Forventningen havner altså midt på y-aksen. For at brukeren skal få illusjonen om at det er det virkelige datasettet han/hun ser, forskyves y-aksen som vist på figuren. Denne forskyvningen vil være (0-gjennomsnitt)/standardavvik. I tillegg måtte vi finne fornuftige metoder for å regne ut hvor mange piksler som skal tilsvare 1 enhet på aksene.

Ved tovariable modeller, brukte vi en litt annen metode. Også her måtte vi finne en metode for å regne ut enhetene på aksene. Dette gjorde vi ved å ta utgangspunkt i maksimumsverdiene og minimumsverdiene. Slik kunne vi regne ut hvor mange piksler en enhet kunne være for å få plass til hele diagrammet.

I tillegg til å gjøre disse operasjonene som nevnt over, måtte vi også ta hensyn til hvor stort panelet vi tegnet på var. Dette skulle vise seg å bli noe innviklet, men vi har nå implementert dette på en slik måte at skaleringen tilpasses når brukeren forandrer størrelsen på vinduet.

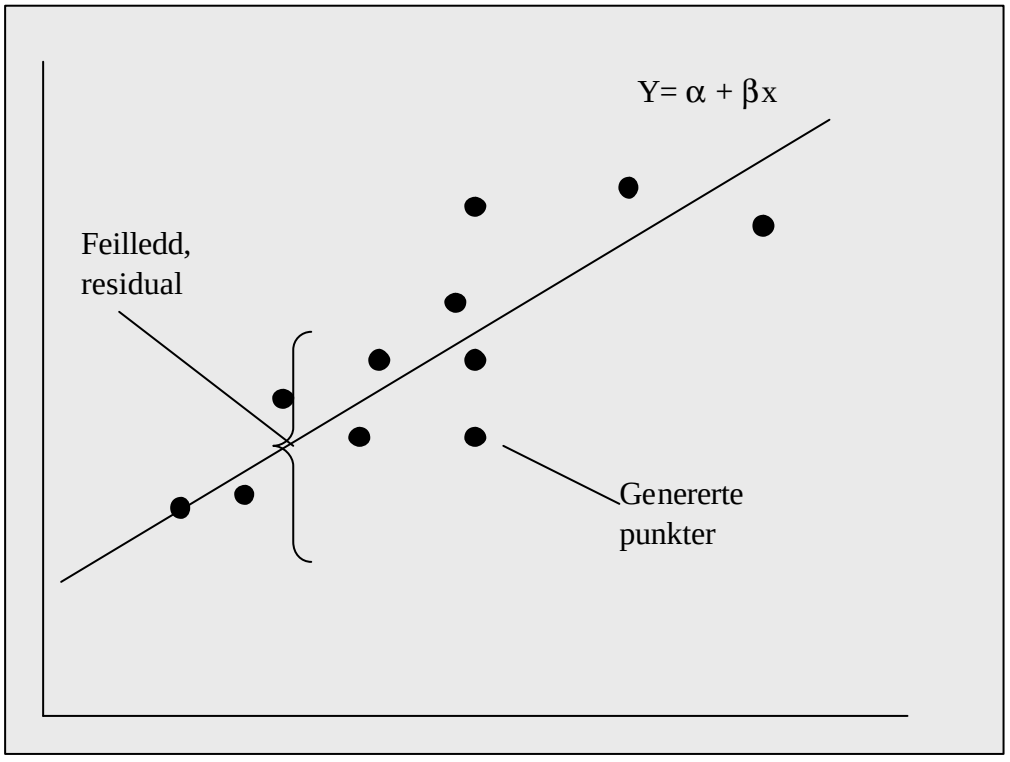
Det er mange andre aspekter vi gjerne skulle ha belyst ved opptegninger av grafiske representasjoner. Men vi går ikke enda nærmere på de ulike metodene vi har benyttet; det blir for detaljert. Vi henviser isteden til kildekode for flere detaljer.

6.7.2 Generering av nye datasett

Ved automatisk generering av data benytter *Ark*-objektene seg av randomgeneratorene. Ettersom vi har valgt strukturen som allerede er blitt beskrevet, er det en smal sak å generere et datasett. Arket vil slette det datasettet som allerede finnes, og klargjøre for et nytt. Via *VelgFordelingDialog*en har arket fått nok opplysninger til å velge riktig randomgenerator og hvor mange tall som skal produseres. Slik kan ark be randomgeneratoren produsere et visst antall tall, og legge disse inn i datasettet tall for tall:

```
for(antall tall som skal produseres){
        tall = randomgenerator.next…();
        datasett.leggInnTall(tall);
}
```

For tovariable modeller blir dette noe annerledes. Her måtte vi benytte oss av to randomgeneratorer (se avsnitt 8.3.2 for problemer rundt nettopp dette). Bruker måtte her velge ønsket α og β for linjen $y = \alpha + \beta x$. Den ene randomgeneratoren genererer x'ene i formelen for y. Dette fører selvsagt til at alle genererte punkter havner på linjen y. Derfor må vi også generere et residual (feilledd). Dette feilleddet genereres ved hjelp av normalfordelingen, der forventningen er lik 0 og standardavvik valgt av bruker.



Figur 6.21 Generering av tallpar

Feilleddene (e) representerer altså feil i y-retning. Et tallpar genereres derfor som følger:

(x, y)

Der x er generert ut ifra en normalfordeling med forventning og standardavvik gitt av bruker, og der $y = \alpha + \beta x + e$. Feilleddet, e, er normalfordelt med forventning = 0 og standardavvik gitt av bruker.

6.7.3 Simulering av konfidensintervallene

I *KIGrafikkVindu* kan bruker simulere konfidensintervaller. Etter at bruker har valgt fordeling, type konfidensintervall, konfidensnivå og antall intervaller, har vi nok informasjon til å starte simuleringen. *KIGrafikkVindu* benytter seg da av en av randomgeneratorene som beskrevet i avsnittet ovenfor. Når et datasett er generert, ber vi *ConfidenceInterval* i statistikkpakken beregne et konfidensintervall for oss. Vi får da returnert en array som inneholder grensene i intervallet. Da kan vi kreere et *Konfidensintervall*-objekt, og denne legges da inn i vektoren som finnes i *KIGrafikkPanel,* som da ber hvert av konfidensintervallene tegne seg selv til skjerm.

6.7.4 Filhåndtering

Når bruker ønsker å lagre eller hente et datasett til eller fra fil, skjer det en event i hovedvinduet (en knapp ble for eksempel trykket på verktøylinja). *Hovedside* vil da kalle en funksjon i gjeldende ark. Dette arket har da ansvaret for å lagre sitt datasett til fil. Vi går ikke nærmere inn på hvordan dette skjer, men henviser til javadokumentasjon og kildekode.

Filformatet er en vanlig tekstfil. Dette er fordi oppdragsgiverne ønsker mulighet for å lage en egen tekstfil og importere denne filen i Estatica. Tekstfilen består av tall, der space (’ ’) fungerer som skilletegn. Bruker kan selv bestemme om en fil skal åpnes i et envariabelt diskret, envariabelt kontinuerlig eller tovariabelt ark. Om bruker prøver å åpne en fil med flyttall i et envariabelt ark, blir alle tallene gjort om til heltall. For tovariable datasett, vil to og to tall utgjøre et tallpar. Derfor bør det være et partall antall tall i en fil som åpnes i et tovariabelt ark. Hvis dette ikke er tilfelle, sløyfes det siste tallet og bruker for melding i form av en advarsel.

Hvis bruker skulle prøve å åpne en fil som har samme navn som et allerede eksisterende ark, får bruker spørsmål om han/hun ønsker å åpne det lagrede datasettet eller ikke foreta seg noe. Det er også implementert flere feilsjekker for filhåndtering, og dette skal nå være robust og sikkert.

Kapittel 7

7 Kvalitetssikring og testing

Learn to labor and to wait.
Jonathan Wadsworth Longfellow

And oftentimes excusing of a fault
Doth make the fault the worse by the excuse.
William Shakespear

It is common sense to make a method and try it.If it fails, admit it frankly and try another. But above all, try something.
Franklin Delano Roosevelt.

O! Throw away the worser part of it,
And live the purer with the other half.
William Shakespear

7.1 Kvalitetssikring

Nå har det dessverre seg slik at et hovedprosjekt som dette ikke kan gjennomgå en fullverdig avslutningsfase. Vi har ingen muligheter til å følge opp produktet etter release, men det er vel heller ikke meningen. Et systemutviklingsprosjekt bør følges opp etter release, og prosjektet går inn i en vedlikeholdsfase for å videre kvalitetssikre og utvide.

Vi har allerede snakket om kvalitetssikring i rapporten. Et godt forprosjekt har hjulpet oss i å utvikle gode og fornuftige standarder og rutiner, spesielt da med tanke på føring av dokumentasjon, dokumentasjon av kode og kodestandarder. Vi har underveis i prosessen sørget for å dokumentere det vi har gjort. Derfor har også skrivingen av denne rapporten gått relativt smertefritt. Vi har vært bevisste på å følge en kontrollert prosess, som har bunnet ut i den velprøvde systemutviklingsmodellen RUP. På denne måten føler vi at vi har hatt en kvalitetssikret prosess, noe som er essensielt for å få et kvalitetssikret produkt.

Det er mange årsaker til dårlig kvalitet, og vi har prøvd å eliminert så mange som mulig:

- Dårlig systemutviklingsprosess
- Dårlige brukergrensesnitt
- Ufullstendig testing
- Ufullstendig dokumentasjon
- Svakt design
- Mangelfull videreføring av spesifikasjon
- Feiltolkning av oppdragsgivers krav og ønsker
- Feil og ufullstendige krav
- Undervurdering av ”uutrykte” krav

Testing av produktet er selvfølgelig med på å sikre kvaliteten. Men testing er mer enn å teste systemet for feil. Kravspesifikasjonen skal brukes under testingen, og dette fører til at vi må stille krav til kravene våre. Kravene må være:

- valide (Er de riktige kravene satt opp?)
- konsistente (Er noen av kravene motstridende?)
- komplette (Er kravene dekkende?)
- realistiske (Er kravene realistiske?)
- verifiserbare (Kan man teste om kravene er oppfylt?)

Under utviklingen av kravspesifikasjonen, måtte vi kontinuerlig sørge for at kravene oppfylte punktene over. Derfor har vi utformet kravspesifikasjonen på en slik måte at den ble *problemorientert*. På denne måten har vi gjennom de senere kapitler prøvd å løse kravspesifikasjonen som om det var en oppgave og et problem. Kravene har også blitt formet på en slik måte at vi kan teste om kravene er oppfylt. Vi henviser til loggbok i vedlegg **K** for tidspunktene til disse ulike testene.

Testing har blitt utført i følgende trinn:

- Testing av krav som forklart over.
- Prototyping, GUI.
- Testing av moduler.
- Testing etter integrasjon.
- Testing av hele systemet.

7.2 Prototyping, GUI

Dette har vi allerede snakket om, men er verdt å nevne under dette kapitlet. Ved å lage prototyper av brukergrensesnittet, har vi kunnet sørge for at oppdragsgiverne har fått et brukergrensesnitt som de kan identifisere seg med. Prototypene ble altså testet direkte opp imot oppdragsgiverne. Dette har vært en kontinuerlig prosess gjennom hele prosjektet, og vi føler at vi har hatt stort utbytte av denne måten å jobbe på. Mot slutten av prosjektet gikk vi bort ifra prototypene og testet betaversjoner mot oppdragsgiverne.

7.3 Testing av moduler

Vi har som RUP legger opp til, utviklet i iterasjoner. Dette betyr at vi har utviklet et og et inkrement og integrert det inn i systemet. Det har derfor vært viktig å utføre testing av modulen før vi integrerte den i systemet. Dette har foregått både individuelt og med hele gruppen. Følgende måtte utføres:

- Valg av tester: Sørge for at testen måler kravene og at testen er pålitelig.
- Funksjonell testing, *BlackBox*: Se på modulen som en *BlackBox*, der vi tester alle mulige input og output ved hjelp av *ekvivalens partisjonering* (går ut på å velge fornuftige inputs slik at alle muligheter blir prøvd). På denne måten har vi kunnet oppdage mulige feil i systemet, og vi har eliminert ulovlige input ved å sjekke på input og eventuelt gi feilmelding til bruker. Her har vi også sjekket output fra systemet opp i mot kravene (da spesielt opp mot de matematiske kravene i vedlegg **A**).
- Funksjonell testing, WhiteBox: Se på modulen som en WhiteBox, der vi har gått detaljert inn på koden og testet mulige veier i koden (*path-testing*).
- Operasjonell testing: Testing av de operasjonelle kravene. Her har vi prøvd å sette systemet ”på prøve” med tanke på hurtighet og robusthet. Vi har gitt systemet ekstreme verdier som input for å finne begrensninger i systemet. Deretter har vi gått inn i koden og prøvd å eliminere begrensningene. Et godt eksempel på denne typen testing, er testingen av funksjonen som beregner binomialkoeffisienten a over b (jfr. Fig. 6.11). Vi begynte med en treg algoritme som taklet en *a* på 20, og endte opp med en algoritme som takler en *a* på over 400 og som har en lineær effektivitet.

Vi skulle ønske vi kunne gjennomføre enda bedre tester av moduler enn det vi har rukket. Tidsrammen har vært streng. Innenfor tidsrammen føler vi allikevel at vi har greid å utvikle robuste moduler som tilfredsstiller krav.

7.4 Testing etter integrasjon

Etter testing av en modul, må denne modulen integreres inn i systemet (et system som består av andre ferdigutviklede komponenter). Etter integreringen måtte vi selvfølgelig teste om systemet "hadde godt av" oppgraderingen. Vi testet da systemet som helhet, og sørget for å oppdage mulige feil som følge av integreringen av en ny modul.

Vi erfarte at denne testingen forløp seg relativt enkel. Når modulen hadde gått gjennom en individuell testing, viste det seg ofte at integreringen ikke skapte noen store problemer i systemet som helhet.

7.5 Testing av hele systemet

Dette er vel det punktet man aldri kan si seg ferdig med. Hele tiden kommer man på hvordan man skulle løst et problem noe annerledes. Vi har allikevel blitt enige om et tidspunkt for når vi sier at applikasjonen er ferdig utviklet. Testing av hele systemet har blitt gjennomført, og vi føler at produktet, slik det nå foreligger, er klart for innlevering. Denne testingen har blitt utført ved nøye og detaljert gjennomgåelse av systemet. Vi har for eksempel tatt for oss ett og ett vindu i applikasjonen, og testet så mange mulige brukerinteraksjoner som mulig. Nå har det seg slik at denne type testing også bør fortsette etter at produktet har blitt tatt i bruk av sluttbrukerne, men et hovedprosjekt setter jo begrensninger på et slikt type vedlikehold.

Kapittel 8

8 Diskusjon av resultater / avslutning

The weel is come full circle.
William Shakespeare, King Lear

Now og write it before them in a table, and note it in a book,
That it may be for the time to come for ever and ever.
The Old Testament

The longest part of the journey is said to be the passing of the gate.
Marcus Terentius Varro

Me want cookie!
The Cookie Monster, Sesame Street

I love being a writer. What I can't stand is the paperwork.
Peter De Vries

Push on – keep moving.
Thomas Morton

Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentence.
William Strunk, Jr.

8.1 Resultater

8.1.1 Prosessen

Vi føler nå at vi endelig har avsluttet et hardt stykke arbeid. Ettersom vi bare er to på gruppen, har vi måttet jobbe ekstra hardt for å få prosjektet i land.

Vi har jobbet tett opp imot våre oppdragsgivere. Kravspesifikasjonen har fremtrådt som klar og enkel å jobbe mot. Under designfasen var det naturlig å se på kravspesifiseringen som et problem vi skulle løse. Slik skulle designfasen gi svar på *hvordan* vi skulle løse oppgaven og likedan skulle implementasjonen forklare i detalj hvordan løsningen ble frambrakt. En løsning er formet, og der er på tide å se om løsningen er dekkende og tilfredsstillende.

Det er fort gjort å bli villedet inn på et spor som ikke fører fram noensteds. Vi har prøvd å være bevisste på valg av spor. Vi har hele tiden spurt oss selv; vil dette føre frem? Finnes det alternativer? Oppfyller dette kravene? Det har vært viktig å være kritisk til egne meninger og valg. Man må hele tiden være bevisst på at ens løsninger og oppfatninger ikke nødvendigvis er de eneste og beste. Derfor kan andre kilder, som litteratur og fagpersoner (veileder og oppdragsgiverne), bidra med mye hvis man bare er villig til å søke. Men dette betyr slettes ikke at man ikke skal være kritisk til denne type kilder også. Vi har satt oss inn i mye ny og fremmed teori, men bare en brøkdel av dette har vært avgjørende og nyttig.

Vi har fulgt en planlagt og kontrollert prosess. Utvikling Gantt-skjema har gjort det enklere å styre prosessen mot ønsket retning. Vi har holdt oss innenfor ulike milepæler og tidsfrister (jfr. Loggbok vedlegg **K** og statusrapporter vedlegg **G**). Dette betyr ikke at det ikke har vært tidspress. Blant annet fikk vi en eksamen *før* påske, som vi trodde vi skulle ha *etter*. Dette skapte noen uforutsette problemer i fremdriften.

Utviklingsmodellen vi har benyttet oss av har vært klar og enkel å bruke. Denne har hjulpet oss med å holde prosjektet innenfor satte rammer. Vi definerte milepæler på bakgrunn av RUP. Utviklingen foregikk i iterasjoner, og det har vært gode muligheter for å gå tilbake i prosessen å gjøre endringer der det har vært nødvendig. RUP støtter i stor grad en slik iterativ prosess. Det legges også vekt på Use Case-drevet utvikling der planleggbarhet og styring er ivaretatt. Selv om vi går programutviklingsretningen i ingeniørstudiet, har vi også fulgt Systemutvikling2-faget i vårt andre år.

Møter med oppdragsgivere og veileder har vært svært nyttige. Møtereferater er å finne i vedlegg **H** og **I**. Disse møtene har gitt os verdifulle tilbakemeldinger og tips. Kommunikasjon har foregått i møter og via elektronisk post. Vi har også holdt web-siden for prosjektet noenlunde oppdatert med betaversjoner, rapporter og referater.

8.1.2 Produktet

Omsider foreligger det en applikasjon. Denne har vært gjennom mange oppdateringer og forandringer, og vi har endelig et produkt gruppen kan si seg fornøyd med. Det er klart at det er ting vi kunne gjort annerledes eller ting vi skulle ønske vi hadde rukket. Men det er jo en del av oppgaven å sette begrensninger og rammer – kravene må tross alt være realistiske.

Systemet, slik vi ser det, oppfyller kravene beskrevet i kapittel 2. Utviklingen har foregått svært målrettet i forhold til kravspesifikasjonen, og har forhåpentligvis ført til at sluttproduktet er tilfredsstillende. Men det må slås fast at systemet og de ulike pakkene kan være gjenstand for videreutvikling.

Også dokumentasjonen er en del av produktet. Vi har fulgt standard UML-notasjon i dokumenteringen, og det vil være til stor hjelp for eventuelle senere utvidelser og oppdateringer. Kode er også godt dokumentert ved hjelp av javadokumentasjonen.

8.2 Alternative løsninger

Systemet har hatt mange former og skikkelser. Vi har vært gjennom mange alternative løsninger før vi falt på den forhåpentligvis beste.

Det har ikke vært mange avvik fra kravene. Etter at vi gjennomførte en stor omstrukturering av datastrukturen, førte dette til at flere dører åpnet seg for oss. Dette førte blant annet til at vi kunne gjøre applikasjonen mer brukervennlig ved at måldata for et datasett ble oppdatert mens bruker tastet inn tall. Omstruktureringen førte også til at vi fikk tid til å utvikle tovariable modeller.

Omstruktureringen gikk ut på å generalisere flere av klassene, slik at vi fikk et bedre arvehierarki enn først planlagt (dette gjelder spesielt *Datasett*-klassene). Vi gikk også bort ifra Javas ferdig definerte tabellmodeller, og utviklet vår egen. Dette førte til at vi raskt kunne oppdatere et datasett under editering.

Tidlig i prosjektet hadde vi også en alternativ løsning på hvordan editoren skulle utvikles. Vi hadde først tenkt at inntasting av tall skulle foregå i en vanlig teksteditor (uten ved bruk av tabell). Vi fant ut at dette ikke ville være særlig brukervennlig, og det ville ha blitt mer komplisert å utvikle.

8.3 Problemer som oppstod

8.3.1 Prosessen

Prosessen har i og for seg gått relativt smertefritt. Vi har jobbet godt sammen, og fordelingen av arbeidsoppgaver har vært upåklagelig. Det har ikke vært noen problemer i gruppen, og hvert medlem har til enhver tid hatt arbeidsoppgaver.

Når sant skal sies hadde vi noen småproblemer i den spede begynnelse. Det var vi som fikk ideen til å gjennomføre et slikt prosjekt. Vi gikk da naturlig nok til de forelesere vi har hatt i statistikk., og begge disse stilte seg gledelig positivt til et slikt prosjekt. Men da vi presenterte ideen videre for ansvarlige for hovedprosjektene, følte vi oss noe motarbeidet. Vi kan godt forstå at egendefinerte oppgaver må gjennom en helt annen prosess en oppgaver definert av skolen. Men allikevel hadde vi håpet og trodd at slikt initiativ skulle bli ønsket velkommen – det ble det imidlertid ikke før vi fikk tildelt en veileder. Vi retter i så måte en stor takk til Harald Liodden, som hadde troen på oss og ga oss klarsignal.

Det bør også her nevnes at vi hadde tidsproblemer rundt påske. Dette kom av at en eksamen vi skulle ha etter påske ble flyttet fram (se statusrapportene). Dette førte til at vi måtte bruke deler av påsken for å hente oss inn igjen. Dette gikk imidlertid greit.

8.3.2 Produktet

Det har vært en del utfordringer å ta tak i. Dette gjelder da spesielt med tanke på implementeringen av løsningen. Til tider har det vært vanskelig å realisere matematikken bak systemet. I begynnelsen brukte vi en tungvinn datastruktur og trege algoritmer. Applikasjonen var rett og slett for treg. Dette førte til at vi midt i prosjektet gjorde en fullstendig omstrukturering av hele designet. Etter denne omstruktureringen løsnet det meste, og prosjektet fikk en rask og jevn fremgang. Dette var en modningsprosess vi måtte gjennom, og vi samlet opp en del bitre erfaringer. Men da skal det også sies at tilfredsstillelsen var desto større da en god løsning begynte å ta form.

Vi har hatt noen problemer med å finne gode algoritmer og metoder for ulike beregninger. Vi brukte mye tid på research, men fant ikke altfor mye som kunne passe inn i vårt prosjekt. Derfor er vi blitt tvunget til å spandere mye tid på små detaljer i koden.

Opptegning av grafiske representasjoner burde vært et nærmest eget kapittel. Dette punktet har voldt oss mye hodebry og frustrasjon. Skalering av aksene må utføres, og det må utføres korrekt. Vi laget mange ulike løsninger før vi fant en som fungerte. En måte å tegne på, passet kanskje ikke for en annen PC med annen skjermoppløsning. Nå skal løsningen fungere for skjermstørrelser over 800*600. Dette var jo også minstekravet.

Det som var hovedproblemet med grafiske representasjoner, var å få ulike diagrammer til å ”passe” oppå hverandre. Eksempelvis skulle normalfordelingen passe oppå et histogram. Vi slet lenge med å få denne skaleringen til å stemme. Samme problemet oppstod da vi skulle plassere to stolpediagrammer oppå hverandre.

Randomgeneratorene har også være en stor utfordring å utvikle. I starten gikk det dessuten altfor tregt å generere et antall tall. Flere randomgeneratorer så også ut til å ikke produsere riktig fordelte tall. Og løsningen på problemet var ofte bare en liten detalj!

Vi hadde spesielt et problem med en randomgenerator som genererte tall som ikke så ut til å være random i det hele tatt. Dette skulle være en randomgenerator som var ferdig utviklet. For generering av tallpar i tovariable ark, må det genereres et tall, x, langs en linje, men også et residual (feil) som er normalfordelt med forventning lik 1 og et standardavvik som bruker velger. For hver genererte x, skulle det også genereres et residual. Når et slikt residual benyttes, fører dette til at punktene (tallparene) som blir generert ikke blir liggende på ei linje (det ville jo ikke være særlig random). Men uansett hva vi gjorde, så alle punktene ut til å ville ligge langs ei rett linje (residualene så bare ut til å påvirke stigningskoeffisienten på den rette linja) – en slik randomgenerator kunne vi ikke se oss fornøyd med. Etter detaljert gjennomgåelse av randomgeneratoren, fant vi omsider ut at problemet ikke var selve randomgeneratoren. Vi benyttet oss av to randomgeneratorer for å produsere tallparene, og det viste seg at disse to randomgeneratorene ble kreert omtrent samtidig, så randomgeneratorene fikk *samme seed*; og seed’en beregnes ved hjelp av systemklokka. Løsningen ble dermed å utnytte den tiden bruker benytter for å velge fordeling. På denne måten vil ikke randomgeneratorene bli kreert samtidig og de vil få forskjellig seed.

Nå skal vi ikke gå inn på alle problemer vi har vært borti, men har bare illustrert noen eksempler. Vi har også skrevet om flere av problemene i statusrapportene og møtereferatene.

8.4 Forslag til videre arbeid

Vi håper at andre studenter kan ta opp tråden og videreutvikle systemet. Etter hvert finner nok de ulike sluttbrukerne svakheter ved applikasjonen, og kanskje oppstår flere ønsker og behov. Også vi har flere ting vi skulle ønske vi hadde rukket, og det vil være naturlig å foreslå videre arbeid:

- Tall på akser for opptegning av diagrammer (Dette er nok ønskelig for oppdragsgiverne, men var ikke spesielt kritisk i dette prosjektet).
- Utvide med flere fordelinger (først og fremst rektangulær fordeling, som vi dessverre ikke rakk å gjennomføre til det hele. Funksjonalitet for rektangulær fordeling finnes i statistikkpakken, men vi har ikke rukket å gjennomføre dette i Estatica. Det er ikke mye arbeid som gjenstår på dette punktet).
- Utvide med flere typer konfidensintervaller (vi har utviklet funksjonalitet for dette i statistikkpakken, men det var ikke et krav å ha med disse i Estatica).
- Legge deler av applikasjonen på WEB i form av for eksempel applet'er.
- Utvidelse av tovariable modeller (simulering av regresjonslinjer og konfidensintervaller).
- Hypotesetesting.
- Tidsrekker (Shewart-diagrammer)

Her er det i grunn oppdragsgiverne som kan finne ønsker og behov for videre utvikling. Vi har i hvert fall illustrert noen mulige utvidelser. Etter hvert vil det nok også dukke opp muligheter for forbedringer til arbeidet vi har gjort. For å oppdage disse, bør nok systemet brukes en del først.

Selv om prosjektet nå avsluttes som hovedprosjekt ved Høgskolen i Gjøvik, er ikke det dermed sagt at vi ikke fortsetter å jobbe med prosjektet. Vi har svært lyst til å fortsette å utvikle statistikkpakken. Slik utvikling kan forresten være svært aktuelt med hensyn på videre studier på NTNU. Men dette får tiden vise.

8.5 Konklusjon

Resultatet er et produkt som vi ser oss fornøyd med. Applikasjonen tilfredsstiller kravene som er blitt definert. Om dette også oppfyller kravene som oppdragsgiverne har hatt til systemet gjenstår å se, men vi håper virkelig at applikasjonen kan brukes til det formålet det er ment. Foreleserne i statistikkfagene ved Høgskolen i Gjøvik, har vist stor velvilje til å bidra med å utvikle et slikt verktøy for bruk i undervisning. I faget har det allerede vært benyttet applikasjoner via WEB (applets), som studentene har kunnet gått inn på og eksperimentere. Dette kan tyde på et behov for et slikt program vi nå har utviklet.

Applikasjonen har vært gjennom planlagte tester, og skal nå fungere som forventet i henhold til krav. Brukervennligheten, der kompleksiteten skjules for bruker, har vært et viktig moment i utviklingen av dette systemet. Vi håper å ha tilfredsstilt dette kravet gjennom intensiv prototyping der tilbakemeldinger fra oppdragsgiverne har vært viktigste kilde til forbedringer. Applikasjonen skal i tillegg tilfredsstille krav om hurtighet og robusthet, slik at verktøyet kan benyttes i undervisningen.

Systemet er ment å være gjenstand for senere utvidelser. Derfor har vi forsøkt å lage en god dokumentasjon i form av denne rapporten og tilhørende vedlegg, kildekode og tilhørende kommentarer og javadokumentasjon. Programmet er delt opp i klare lag og pakker, der individuelle moduler kan utvides uavhengig av andre.

Det er selvfølgelig en del ting vi ønsker vi hadde rukket å gjennomføre, men som ikke har vært innenfor de tidsrammer som er satt. Vi har derfor i denne rapporten skissert mulige utvidelser av systemet. Vi anbefaler andre studenter sterkt vurdere å ta opp igjen tråden, og om oppdragsgiverne er like ivrige som de har vært med oss, bør det være kø bak eventuelle senere prosjekter.

Å gjennomføre et stort prosjekt som dette gir en mye. Man lærer å arbeide målrettet og strukturert, og vi føler stort utbytte av å ha gjennomført prosjektet. Bitre erfaringer og tilfredsstillende "aha-opplevelser" gjør at læringseffekten er stor. Selv om vår gruppe bare består av to personer, føler vi allikevel at vi har fått erfaringer med å jobbe sammen med andre. Prosjektarbeid er hevet over enhver tvil en interessant måte å jobbe på, der man får føle at man utretter noe som en gruppe så vel som individ. I tillegg får man et stort faglig utbytte, for man kan ikke lære bedre det man lærer seg selv.

9 Litteraturliste

Trykt og skreven litteratur:

- Java How To Program – 3rd ed. (Deitel & Deitel)
- Enterprise Java *with* UML (CT Arrington)
- John Zukowski's Definitive Guide to Swing for Java 2 (Zukowski)
- Mastering DELPHI 6 (Marco Cantu)
- Algorithms in C++ (Robert Sedgewick)
- GUI – guiden (Laura Arlov)
- Applying UML and Patterns (Larman)
- Software Engineering (Sommerville)
- Prosjektarbeid (Harald Westhagen)
- Statistikk – for universitet og høgskoler (Gunnar G. Løvås).
- An introduction to Mathematical Statistics and Its Applications – 2nd ed. (Richard J. Larsen, Morris L. Marx)
- Introductory Statistics – 3rd ed. (Weiss/Hassett)
- Calculus – 5th ed. (Edwards & Penney)

Øvrige kilder:

- java.sun.com
- API spesification for the Java 2 Platform, Standard Edition, version 1.4
- Notater og kompendium fra undervisning ved Høgskolen i Gjøvik.