Datavetenskap

Fredrik Olsson och Magnus Rosengren

Ett kodutvärderingsverktyg för Java

C-uppsats

2000:32

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Fredrik Olsson och Magnus Rosengren

Godkänd, 2000-05-31

Handledare: Martin Blom

Examinator: Stefan Lindskog

Sammanfattning

Denna uppsats är skriven på C-nivå inom ämnet Datavetenskap vid *Karlstads universitet* vårterminen –00. Bakgrunden till uppsatsen är projektet “SKUTT” som bedrivs på *Karlstads universitet*. Uppsatsens mål är att skapa ett utvärderingsverktyg för källkod. Utvecklingenhar skett i *Unix* och programspråket är C++. Programmet tar i denna version endast Javakod som indata och beräknara ett antal parametrar. Parametrarna används för att göra en subjektiv bedömning med vissa godtyckliga mått som grund. Bedömningen skall ses som en orientering beträffande kodens *“readability”, “reuseability”* och *“complexity”*.

A Code Evaluation Tool for Java

Abstract

This C-grade composition is written and given at the Department of Computer Science at Karlstad University, spring term –00. The background is a project pursued at Karlstad University, named “SKUTT”. The purpose of the composition is to crate a code evaluation tool. The development has been done in Unix and the programlanguage is C++. In this version of the program, Javacode is expected as input. After the program has determined a certain number of parameters there vill be done a subjective judgement regarding to the code’s readability, reuseability and complexity. This judgement shall be interpreted only as an orientation.

Innehållsförteckning

Figurer

List of tables

1 Inledning

1.1 Introduktion

Hur definieras ”bra” skriven källkod? Är det utifrån huruvida programmet är robust, snabbt och driftsäkert? Detta låter som ett rimligt antagande men allt eftersom har mängden programmerare och programmeringsspråk ökat och även andra aspekter kan vägas in i begreppet ”bra” källkod.

När de första programmerarna satte sig ner för att skriva ett program arbetade man nära datorn med ett så kallat ”Assemblerspråk”, dvs man gav instruktioner på processornivå i datorn. Programmeringen var inriktad på att förmå datorn att utföra önskvärda operationer. Hur dessa instruktioner sedan skrevs schematiskt var ej vedertagen praxis utan utvecklingen av programspråkens syntax (grammatik) var i centrum. Idag när de flesta på marknaden använda programmeringsspråken är tredje- och fjärde generationens språk vet man med största sannolikhet att datorn kan utföra de operationer som önskas. Programmerarens dilemma blir istället hur han/hon skall göra detta och dessutom ett skapa ett ”bra” program. Idag finns ett antal olika metoder som går att applicera på olika språk för att utveckla program. Det finns även metoder som är knutna till ett särskilt språk eller de som använder standardiserade verktyg.

Vad är då syftet med dessa metoder, tekniker och verktyg förutom att skapa tillförlitliga och snabba program? På senare år har produktionen av källkod och program ökat explosionsartat. Programmen kan exempelvis utgöra en del av ett företags lönesystem eller en fabriks realtidssystem för styrande av maskiner. Lika svårt (omöjligt) som det är att skapa en evighetsmaskin är det av olika orsaker att skriva ett program som fungerar i oändlighet. När ett program kraschar slutar också systemen att fungera, vilket kan medföra stora ekonomiska förluster. Underhållsarbetet går ofta ut på att söka igenom programmets källkod för att hitta den ”bugg” (programfel) som orsakat driftstoppet. Att leta ”buggar” kan vara som att leta efter en nål i en höstack, och alla som programmerat vet hur svårt det är att sätta sig in i någon annans kod. I samband med detta kan nämnas *readability* (läsbarhet) som en faktor i begreppet ”bra” källkod [7].

En annan aspekt av begreppet är att betrakta koden utifrån hur den går att använda, *reuseability*, dvs att koden, eller delar av den, skall kunna återanvändas i andra program.

Att bedöma ett programs standard är en subjektiv analys av källkoden där de ovan nämnda faktorerna kan ingå som parametrar.

1.2 Bakgrund

Under läsåret 99-00 erbjuder Institutionen för *Informationsteknologi* vid *Karlstads universitet* ett antal ämnesförslag till uppsats på C-nivå (tio akademiska poäng) [9]. Ett av dessa uppslag, *”A Code Evaluation Tool”* syftar till att utveckla ett redskap för utvärdering av programkod med avseende på semantisk kvalitet. Verktyget skall ”parsa” källkoden och generera värden som till exempel antal rader kod, antal klasser, antal klassanrop. Värdena skall sedan användas som parametrar för att uppskatta källkodens kvalitet med aspekter på kriterier som *”readability”, ”modularity”* och *”weakness of coupling”* m.m.

Ämnet har sitt ursprung inom ett projekt *”SKUTT”* som bedrivs av *Institutionen för Datavetenskap.* Nedan följer ett citat [12] som beskriver av projektet.

> *”The aim of the project is to develop a method that enhances software quality through improved semantic descriptions. The method, which should be based on established formal methods, should be pragmatic enough to be applied in industrial projects. The expected impact of the method is improved software quality in complex industrial projects through a reduction in the number of errors related to semantic misunderstandings. The project is funded by Nutek’s program for Complex Technical Systems (the Software Engineering Cluster) and by Ericsson Infotech AB.”*

Under åren 97/98 genomfördes en undersökning på två IT relaterade företag i Karlstadregionen. Studien visade på brister inom programutvecklingstekniken av mjukvara.

Kursen *”Projektarbete i Java”* som ges vid *Karlstad universitet* var ett första steg i ett längre samarbete mellan universitet och ett av dessa företag för att utveckla en metod där syftet var att öka kvaliteten på programvaran. Under kursen delades deltagarna in i två grupper. Grupperna fick i uppgift att skriva ett program. Programmets primära funktion var samma för båda grupperna men implementationen skilde sig åt. Den ena gruppen blev kommenderad att utveckla programmet utifrån en metod som kallas *”Metod Blå”*. Den andra gruppen använde sig *av ”Metod Gul”* vid sin programutveckling [1]. Grupperna fick inte ha någon kommunikation under projektets gång (för att förhindra spridning av lösningar) då

källkoden användes för att studera skillnader i programkvaliteten och indirekt de två metodernas effekter. Nedan följer en kort sammanfattning av de två metoderna [1].

Metod Blå

Varje metod har ett för- och eftervillkor. Förvillkoret måste vara uppfyllt innan metoden anropas. Villkoret kan ange vilka värden som är godtagbara inparametrar till metoden eller schematiskt vilka metoder som villkorslöst måste varit anropade tidigare.

Det är programmerarens uppgift att förhindra användaren skicka felaktiga parametrar som kan krascha programmet. Förvillkoret kan ses som ett virtuellt filter, dvs det anger bara villkoret i sig. Själva filtret utgörs av ett test som skall ske precis innan anropet till metoden i fråga.

Eftervillkoret skall ange metodens resultat. Detta kan vara modifikation av en variabel eller förändring av hela systemets tillstånd. Användaren skall genom att läsa ett eftervillkor bli varse om v*ad* metoden har utfört, inte *hur* det utförts. I *”Metod Blå”* används även invariant för att beskriva villkor. En invariant är ett villkor som inte får ändras och kan gälla för metoder eller klasser. Ett typiskt exempel är invarianten för variabeln *”antalPersoner”* som anger antal personer lagrade i en länkad lista:

```
”0<=antalPersoner&&antalPersoner=antal personer i listan
```

Metod Gul

I den alternativa metoden, här kallad *”Metod Gul”* används *exceptions* för felhantering.

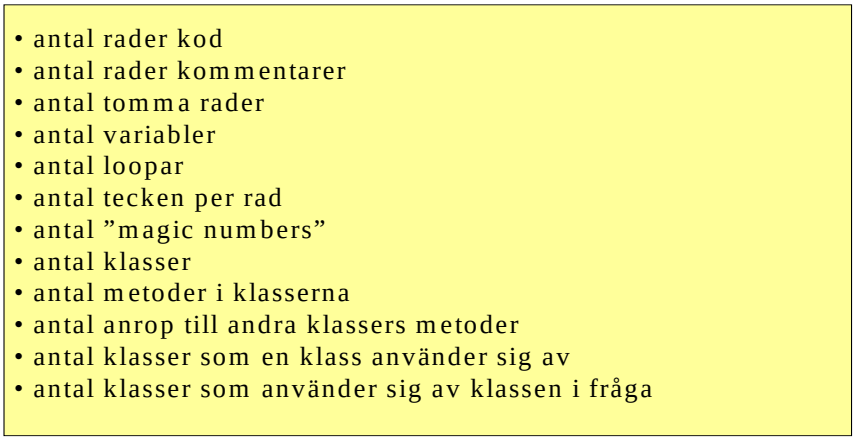
Varje metod/klass godtar alla värden som inparametrar vid ett anrop. Anropet kan även ske vid ett felaktigt tillfälle. Varje metod testar de rimliga fel som kan uppstå (alla fel kan inte förutses) och kastar en ”*exception*”. Vid metoddeklarationen är det lämpligt att ge ett beskrivande namn åt varje ”*exception*” så att det framgår vad som kan gå snett. En klass skapas med arv från grundklassen `Exception` dit alla ”*exceptions*” kastas. Denna klass skriver ut ett felmeddelande som indicerar felets art och uppkomst.

Vid utvärderingen analyserades källkoden manuellt för att se om det eventuellt fanns några skillnader i programkoden. Insamlingen av mätdata var manuell dvs analysen av koden skedde för hand eller med hjälp av verktyg som till exempel *wordcount* [4] i *Unix*. Detta är tidskrävande och därefter tillkommer tidskostnad för den subjektiva bedömningen av kriterierna.

1.3 Mål och krav

Målet med uppsatsen är att skriva att program som utvärderar ett godtyckligt Javaprogram enligt de parametrar som listas i Figur 1.1 och om möjligt utvärdera dessa parametrar till olika mått på hur koden är skriven. Ett sådant mått skulle exempelvis vara *readability* eller *complexity*. Dessa mått är helt beroende på vilka parametrar som utvärderas och arbetet med att ta fram dessa mått började först efter att parameterberäkningarna i programmet var klara.

I samarbete med handledaren och efter vad vi själva ansåg vara möjligt kom vi fram till de parametrar i Figur 1.1 som skulle vara intressanta att utvärdera i programmet.



Figur 1.1: Parametrar att utvärdera.

Följande grundläggande krav på programmet fastställdes:

- resultera i ett program användbart för att utvärdera Javakod.
- programmets ”input” skall vara en vald katalog, dvs alla dess filer.
- programmets ”output” skall vara de parametrar som erhålls vid analysen av filerna samt en utvärdering av parametrarna.
- parametrarna skall presenteras på ett för användaren begripligt sätt på skärmen.
- all utdata från programmet skall kunna sparas undan i en fil.

1.4 Förutsättningar

Uppsatsen är på 10 poäng och skrivs på C-nivå. Tillfogad tid är 20 veckor då uppsatsen skrivs på halvfart. Handledare är *Martin Blom*, universitetsadjunkt vid *Datavetenskap, Karlstads universitet*. Handledarträffar har varit genomförbara i princip varje vecka. Författarnas förkunskaper inom ämnet datavetenskap är kurser på A- och B-nivå. Eftersom inga förkunskaper fanns i Javaprogrammering har en inläsning av språket varit nödvändig [3][8]. Inläsning har även skett av de två metoder, Gul och Blå [1], som används i kursen *”Projektarbete i Java”*.

Utvecklingen sker i *Unix*miljö (*Linux*). Programspråket är *C++* och programmets skal (användargränssnitt) är *KDE* [11]. Då programutvecklingen ej är hårdvarubaserad har tillgången till datorer varit obegränsad. Tillgång till litteratur har förutom universitetets bibliotek erbjudits ur institutionen för *Datavetenskaps* litteratursamling.

2 Verktyg i programmet

I följande kapitel kommer de externa program som använts i arbetet att förklaras. Dessa är *grep, sed, awk* och *wc* [4][5][6]. De tre förstnämnda kommer att förklaras i ett gemensamt kapitel eftersom de är uppbyggda på ett liknande sätt. Sist i detta kapitel kommer ett antal exempel med förklaringar som är typiska för hur programmen har använts i arbetet.

2.1 grep, awk, sed

Awk, sed och *grep* är tre externa program som sedan *UNIX* sjunde version ingår som standard. Gemensamt för dessa verktyg är att alla använder sig av uttryck för att matcha mönster i textfiler. Följande versioner har använts i detta arbete; *GNU grep 2.3, GNU awk 3.0.3* och *GNU sed 3.02*.

Vi kommer nedan att endast beskriva kortfattat hur de olika verktygen fungerar och hur de skiljer sig åt. För att sedan ägna större utrymme åt hur de kan användas i detta program till exempel beskriva hur syntaxen för uttrycken skrivs. Den litteratur som

använts är uteslutande *Dougherty Dale* [2] samt *awk's* och *sed's* respektive *GNU*-versions manualsidor [5][6].

Samtliga har sitt ursprung ur *UNIX* original "line editor", `ed`. Eftersom programmen bygger på `ed` returnerar de hela rader, om inget annat anges, från den fil som sökningen görs i.

2.1.1 Användning

Precis som för andra programspråk måste naturligtvis en speciell syntax följas för att programmet skall "förstå" vad det skall göra. Denna syntax skiljer sig mellan de olika verktygen. Däremot är skrivsättet för det mönster som programmet skall hitta och eventuellt bearbeta detsamma. Nedan följer först en beskrivning av möjligheterna och dess syntaxer för respektive program och därefter hur de reguljära uttrycken skrivs för att hitta mönster.

2.1.2 grep

Eftersom *grep* endast har möjlighet att leta efter ett angivet mönster i en textfil, blir syntaxen för att anropa grep alltid likadan. Ett anrop ser ut som följer:

```
grep script filename
```

I *grep* består `script` av ett mönster som önskas söka efter. Eventuellt kan sökmönstret föregås av ett kommando, se Tabell 2.1.

`-c`	antal rader
`-n`	skriver till radnumret främst i respektive rad
`-v`	returnerar motsatsen, dvs det som inte matchas i sökningen

Tabell 2.1: Tabell över kommandon i grep.

`filename` är den textfil där matchningen av mönstret sker, `Input` i Figur 2.1.

`Input → Commandline → Output`

Figur 2.1: Hur grep fungerar[2].

Resultatet blir en ”output” av de träffar av det mönster som angivits, en sådan ”output” är möjlig att spara som en ny textfil. En sådan syntax skulle kunna ha utformningen:

```
grep script filename >> newfilename
```

2.1.3 sed och awk

Sed och *awk* är kraftfullare verktyg där det går att göra mer än att bara söka efter ett mönster. Syntaxen för att anrop till dessa program ser därför något annorlunda ut:

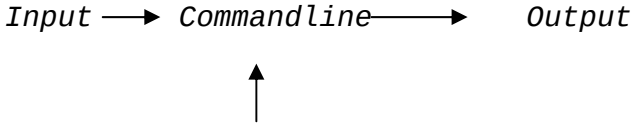
```
commando[options] script filename(>>newfilename)
```

De olika *”options”* för *sed* och *awk* är olika och vi kommer bara att ta upp de som använts i arbetet, för resterande hänvisas till *Dougherty Dale* [2]. I `script` anges vad programmet skall utföra. Till skillnad från grep finns det i *sed* och *awk* möjlighet att skriva in flera instruktioner åt gången, där en instruktion måste innehålla både ett angivet mönster och en procedur. Procedurerna i *sed* och *awk* skiljer sig åt. I sed består proceduren av olika kommandon, vanligtvis bestående endast av en bokstav, se Tabell 2.2. I *awk* däremot består proceduren av ”programming statements” och funktioner.

`i`	insert
`d`	delete
`p`	print
`s`	substitution

Tabell 2.2: Exempel på kommandon i sed

För att kunna ange ett kommando i sed måste `'-e'` anges som ”option”, denna talar om att nästa argument är ett kommando.



Script

Figur 2.2: Hur sed och awk fungerar [2].

Om *script* innehåller fler än en instruktion är det behändigt att placera instruktionerna i en egen fil, som Figur 2.2 visar. För att detta skall kunna vara möjligt måste *'-f'* anges som ”option”. Denna ”option” anger att nästa argument är en fil. Skrivsättet skulle då i *sed* bli:

```
sed -f scriptfile filename
```

2.1.4 Uttryck för sökning av olika mönster

Syftet med detta kapitel är att förklara syntaxen för de reguljära uttryck som används för att skriva in de mönster som *grep, sed* och *awk* skall leta efter i en textfil.

Ett reguljärt uttryck beskriver i detta fallet ett mönster eller en speciell sekvens av tecken. För att kunna skriva dessa uttryck finns speciella metatecken som har en speciell betydelse i programmen. Tabell 2.3 listar ett utdrag av metatecken för *grep* och *sed*. *Awk* har en något annorlunda uppsättning metatecken, men det är inget som använts i detta arbete så till dessa metatecken hänvisas läsaren till *Dougherty Dale* [2].

.	Matchar alla tecken utom nyrad.
*****	Matchar antal (inklusive noll) av det enstaka tecken som *'*'* direkt föregås av.
[…]	Matchar något av den klass av tecken som är angivna innanför hakparenteserna. Ett bindestreck *'-'* indikerar en ordnad mängd av tecken. Alla metatecken förlorar sin mening när de skrivs inuti en klass. Tecknet *'^'* inuti en klass matchar alla tecken utanför klassen.

`\`	Skrivs innan ett metatecken för att undgå dess betydelse.

Tabell 2.3: Exempel på metatecken

En punkt *'.'* kan ses som ”wildcard” för att matcha vilka tecken som helst. Skrivsättet *grep '.' filename* skulle returnera alla icketomma rader ifrån textfilen *filename.* Metatecknen *'.'* och *'*'* används ofta ihop till exempel om rader med parenteser skulle vara det som returnerades från en fil skulle det inte räcka att enbart söka efter *'()'*. Denna sökning skulle enbart returnera rader med tomma parenteser, medan *'(.*)'* även skulle returnera rader med parenteser innehållande ett antal tecken, vilka som helst, i sig.

Ett enkelt exempel för att visa metatecknets *'[…]'* betydelse är om du vill ha rader med ordet edit eller Edit returnerat från en fil. En sådant uttryck skulle se ut som *'[eE]dit'.* Observera även att rader som till exempel editor kommer att finnas med bland de rader som returneras. När ett bindestreck används innanför hakparenteserna bildas en mängd av tecken som då ingår i klassen. Ett exempel på detta är uttrycket *'[0-9]'*, som matchar alla heltal. Motsatsen, dvs det uttryck som matchar samtliga tecken utan heltalen skrivs *'[^0-9]'*.

När metatecknet ”backslash” *'\'* skrivs innan andra metatecken görs dessa om till ordinära tecken. En uttryck då tecknet *'*'* skall matchas skulle sålunda se ut på följande sätt: *'*'*. Alternativt kunde uttrycket *'[*]'* användas eftersom metatecken förlorar sin mening när de skrivs innanför hakparenteser.

2.2 wordcount, wc

Till det externa programmet *wc,* version 1.22, har vi inte studerat någon litteratur, utan har förlitat oss på den manualsida för *GNU* versionen av *wc* [4].

Det finns tre saker i *wc* som kan väljas att få information om, dessa är antal bytes, antal ord och antal rader. Skrivsättet för att ange dessa val visas i Tabell 2.4.

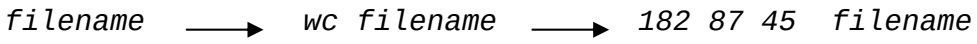
`-l`	antal nya rader som förekommer i den angivna filen
`-w`	antal ”whitespaced-separated” ord

`-c`	antal bytes (tecken)

Tabell 2.4: Tabell över valmöjligheterna i wc

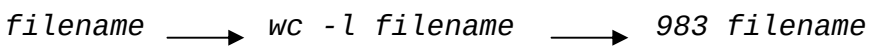
Det finns möjlighet att välja en till tre val i tabellen ovan, men för att få en retur med alla tre behövs inget alternativ anges. När endast *wc* skrivs in med efterföljande fil skrivs samtliga alternativ ut samt filnamnet.

Figur 2.3 visar vad *wc* returnerar från en fil innehållande 182 tecken, 87 ord och 45 rader när inget kommando anges.



Figur 2.3: Exempel på en inmatning i wc.

Figur 2.4 visar syntaxen och resultatet i *wc* för att endast returnera antalet rader från en fil innehållande 983 rader.



Figur 2.4: Exempel från wc där endast antalet rader returneras.

2.3 Förklarande exempel

Nedan följer ett urval av uttryck som använder antingen *wc, grep, sed* eller *awk*. I programkoden finns många fler men är i stort sett likartade varianter på de som redovisas här.

Ex. 1:

```
grep '.java' filename >> filename2
wc -l filename2 >> fiename3
```

Det skapas en ny fil, `filename2`, som innehåller de rader från `filename` där programmet *grep* hittar `'.java'`.

Ex.2:

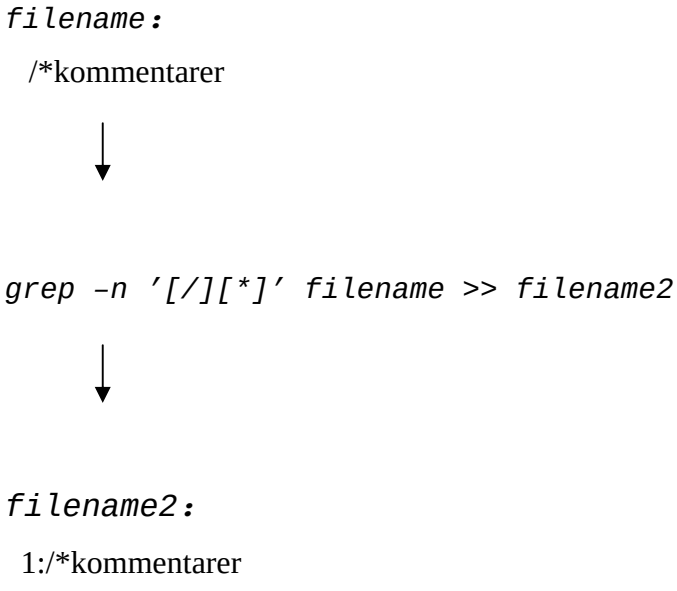
```
grep -c -v '[a-zA-Z{}/0-9]' filename >> newfilename
```

Om inte kommandot `-v` skulle funnits med skulle den nya filen innehålla antal rader från `filename` som innehöll något av de tecknen som finns innanför hakparenteserna. Nu görs istället det motsatta, dvs den nya filen innehåller antalet rader som inte innehåller något av tecknen alls.

Ex.3:

```
grep -n '[/][*]' filename >> filename2
sed -e 's/:/ /' filename2 >> filename3
awk '{print $1}' filename3 >> filename4
```

Syftet med dessa uttryck var att skapa en ny fil, i detta exempel *filename4*, som innehåller på de radnummer där `'/*'` finns i ursprungsfilen, `filename`. Första raden i exemplet skapar en ny fil som innehåller rader innehållande `'/*'` från `filename`, med tillägget att radnumret på dessa rader från `filename` läggs först i varje rad, detta visas i Figur 2.5.



Figur 2.5: Radnummer först i varje rad

För att nu komma åt radnumren med hjälp av *awk* och dess kommando `'print $1',` där *`'print $1'`* endast returnerar första fältet från inputfilen, måste radnumreringen separeras för att bli ett eget fält. Detta görs med hjälp av *sed*, andra raden i exemplet, där semikolonet ersätts med ett blankslag.

Ex. 4:

```
wc --b filename >> filename2
```

I *`filename2`* kommer information om antal bytes/tecken och namnet på den ursprungliga filen att finnas. Till exempel om *`filename`* innehåller 27 tecken kommer innehållet i *`filename2`* att bli: *27 filename.*

Ex. 5:

```
wc -w filename >> filename2
```

Som exemplet ovan fast istället för antalet tecken returnerar *wc* antalet fält som filen innehåller. Detta används som antalet ord i arbetet. Observera att till exempel följande innehåll i *`filename, heltal1=heltal2;`*, endast räknas som ett ord.

Ex. 6:

```
grep -c 'public .*(.*)' filename >> filname2
```

Syftet med uttrycket är att skapa en fil som innehåller antalet rader som innehåller deklarationer av publika funktioner. *grep* matchar nu efter ordet *`'public'`* följt av ett blankslag. Sedan kan noll eller flera tecken, vilka som helst följa som därefter måste innehålla en parentes, med eller utan innehåll.

Ex. 7:

```
awk –f variabeltyper filename >> filename2
```

Kommandot *'-f'* anger att det följs av en fil, i detta fallet *variabeltyper*. Om filens rader ser ut enligt Figur 2.6 skulle innehållet i *filename2* bli alla rader med deklarationer av typerna *int* och *float*.

```
Filename:
/int .*;/
/float .*;/
```

Figur 2.6: Innehåll i filen variabeltyper

3 Programförklaring

Följande kapitel består av två delar. Den första delen beskriver vad programmet gör i stora drag, hur det är uppbyggt. Den andra delen förklarar de olika programdelarna i *edit.cpp* mer ingående. Hela programkoden är bifogad i Bilaga A och består av följande delar*:* *edit.cpp, edit.h, list.cpp, makefile* samt textfilen *scriptfile*.

Textfilen *scriptfile* innehåller de fördeklarerade typer som finns i Java, denna fil används i funktionen som returnerar antalet variabler i koden. Om nya typer skulle deklareras i språket kan dessa lätt läggas till i denna textfil. I *list.cpp* finns enbart den lista där varje klass parametrar lagras. Huvuddelen i programkoden är *edit.cpp*. Här finns samtliga beräkningsfunktioner, de funktioner som skapar fönsterhanteringen i *KDE´n* [11] samt utskriftsfunktionerna.

3.1 Förklaring av programmet i stort

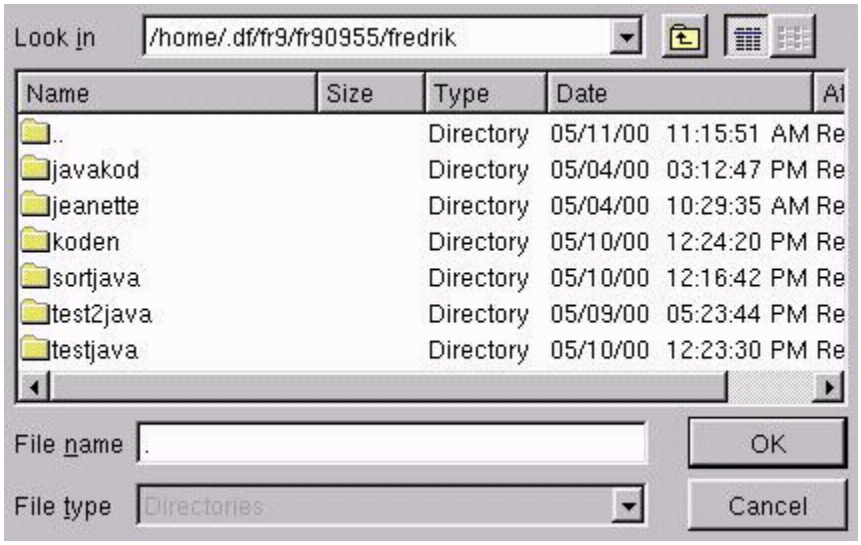
Hela programmet är byggt utifrån ett ”färdigt” skal med *KDE*-gränssnitt. Varje funktion har sedan byggts på i programmet allt eftersom. Vi har valt att beskriva programmet på två sätt, dels det som användaren kommer i kontakt med och dels vad som händer ”inuti” programkoden. Den senare i form av ett programflöde som visar funktionernas beroende och filernas existens.

3.1.1 Beskrivning av programmet ur användarsynpunkt

I det KDE-fönster som kommer upp när programmet körs ges fyra val i utgångsläget:

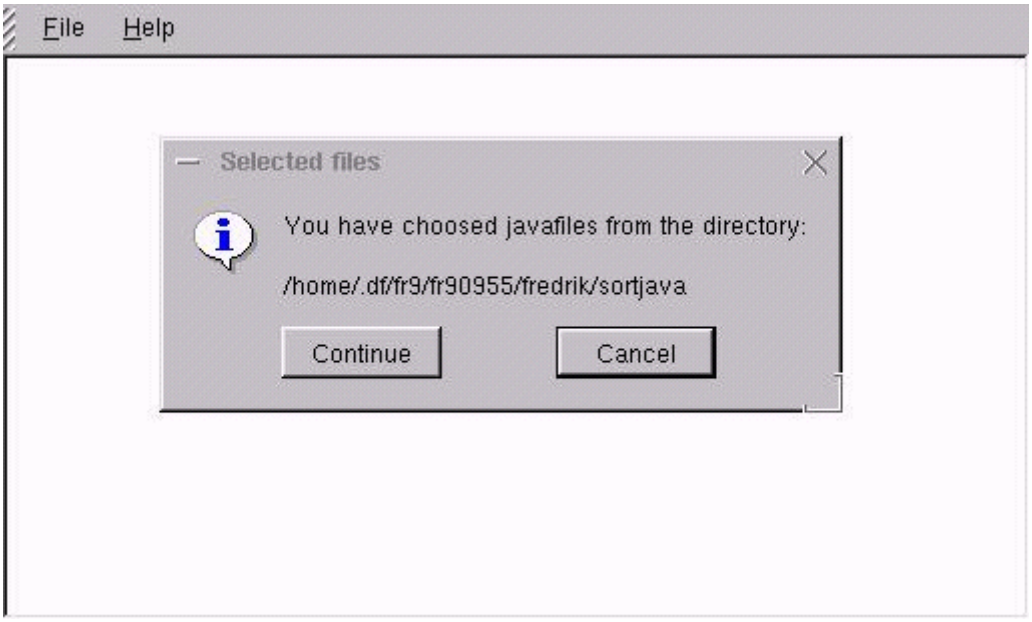
1. Att välja katalog för en ny körning.
2. Spara den utskrift som finns i KDE-fönstret som en textfil.
3. Avsluta.
4. Få kortfattad information om begränsningar/förutsättningar för att kunna använda programmet på ett bra sätt.

Om användaren väljer att göra en körning av kod kommer en dialogruta upp för att kunna välja den katalog som Javafilerna finns i enligt Figur 3.1. Observera här att endast en hel katalog kan väljas dvs att enstaka filer kan inte öppnas.



Figur 3.1: Fönsterhanteringen för att välja en katalog

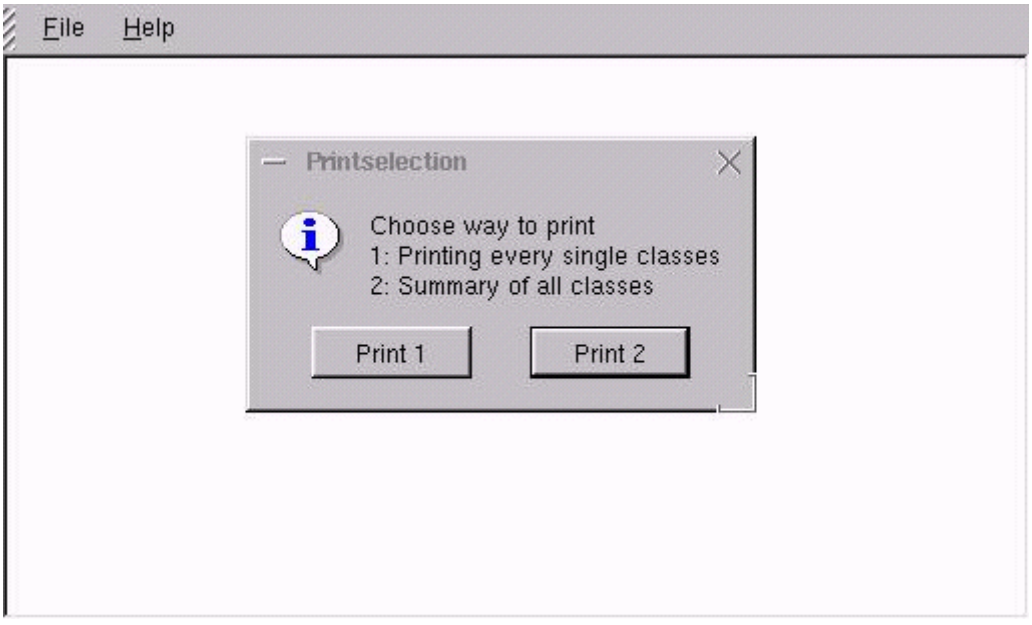
När katalogen sedan har valts kommer en informationsruta, Figur 3.2, upp som anger vilken katalog som valts, nu måste användaren välja om han/hon vill fortsätta sin körning eller avbryta den. Avbryts körningen händer ingenting utan användaren kommer endast tillbaka till utgångsläget igen.



Figur 3.2: Informationsruta om vilken katalogan som valts.

Om användaren däremot väljer att fortsätta sin körning kommer programmet att utföra sin utvärdering av koden som den valda katalogen innehåller.

När programmet utfört sin utvärdering av programkoden i den valda katalogen får användaren upp ytterligare en informationsruta, se Figur 3.3, där ett val av två utskriftsmöjligheter görs, `print1` och `print2`.



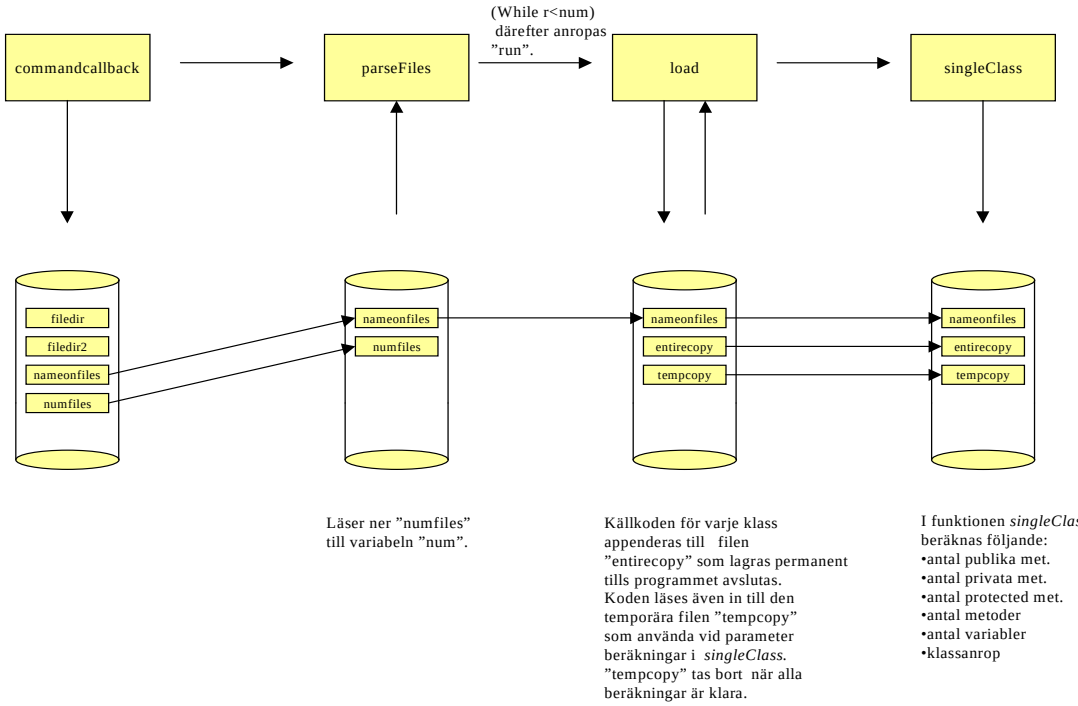
Figur 3.3: Informationsruta där val av utskrift görs.

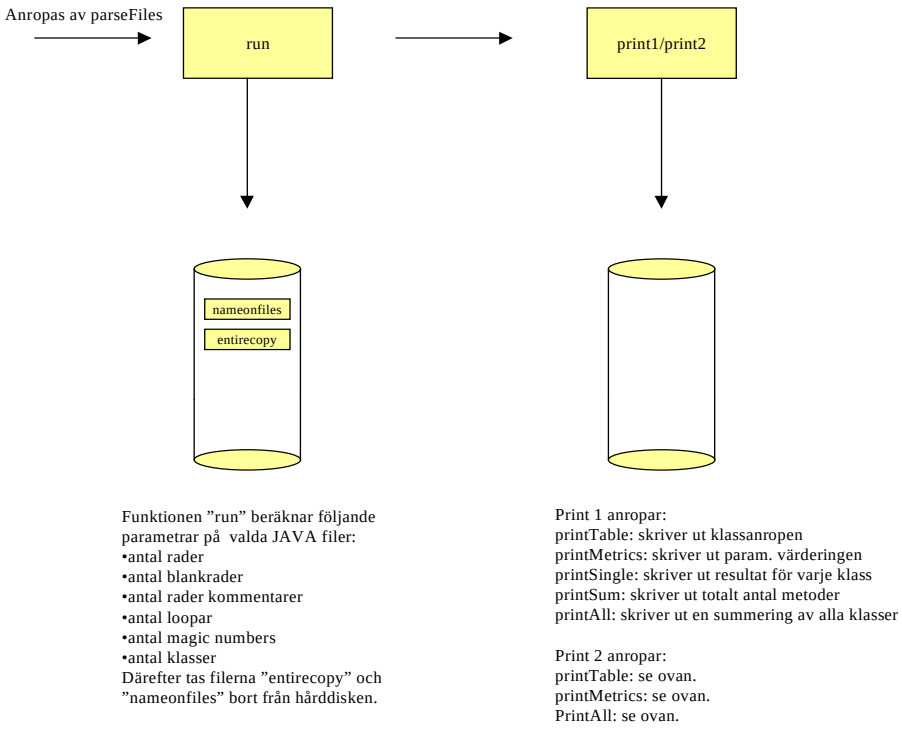
I `print1` kommer resultatet av samtliga klasser att redovisas enskilt samt att en sammanställning av klasserna redovisas. I `print2` däremot redovisas endast en sammanställning av de klasser som utvärderats. I bägge varianterna av utskrift skrivs också en tabell ut som visar antalet anrop mellan de olika klasserna.

Exempel på dessa olika utskrifter visas i Bilaga B och Bilaga C. I Bilaga D finns även diagram, gjort i *Microsoft Excel*, som är baserat på klassanropstabellen från koden som testats i kapitel fem.

3.1.2 Beskrivning av programflödet i koden.

I Figur 3.4 nedan visar grafiskt hur och när de olika funktionerna anropas. Figuren visar också när de olika filerna som programmet skapar och använder sig av existerar.





Figur 3.4: Programflöde med funktionsanrop samt var filer skapas och tas bort

3.2 Förklaringar av enskilda funktioner

I följande kapitel förklaras de enskilda funktioner som finns i *edit.cpp* mer ingående.

void Edit::commandCallback(int id_)

I funktionen finns en switchsats som är beroende på valet som görs i *KDE*-fönstret. Om en ny körning har valts läses alla existerande filer från den valda katalogen in i en fil, *filedir*. Men eftersom endast Javafiler är intressanta läggs endast Javafiler i *filedir2*. En förutsättning är att alla Javafiler måste vara angivna som *filename.java*, eftersom sökningen görs på namnet enligt:

```
grep '.java' filedir>>filedir2.
```

Följande uttryck: *sed -e 's/.java//' filedir2>>nameonfiles* syftar till att erhålla en fil där alla klassnamn nu finns angivna. Observera att här finns en begränsning, nämligen att namnet på klassen måste vara namngivet identiskt med filen, *'.java'* borttaget.

För att programmet inte skall sätta igång och göra hela körningen om katalogvalet var felaktigt kommer en dialogruta upp med hjälp av *QMessageBox::information* [11], som ”frågar” om användaren vill fortsätta körningen med den angivna katalogen.

void Edit::parseFiles(const char *dirname)

Funktionen *parseFiles* tar emot katalognamn, som egentligen är sökvägen till respektive katalog, och läser sedan in antalet Javafiler i denna katalog. Filen *filedir2* som konstruerats i funktionen *Edit* innehåller samtliga Javafiler från den valda katalogen. Varje Javafil för sig från denna fil skickas till *load(sendname,filname, num)* där *sendname* är hela sökvägen till respektive Javafil, *filename* namnet på Javafilerna och *num* är antalet Javafiler som katalogen innehåller.

Efter att varje Javafil skickats till *load* för sig anropas *run()*. Denna funktion anropar beräkningsfunktionerna som görs på hela kopian, dvs alla Javafilerna tillsammans i den valda katalogen.

void Edit::load(const char *filename,const char *linkname,int numb)

Funktionen `load()` gör i princip två saker, dels lägger till den aktuella Javafilen i en stor kopia, `entirecopy`, där till slut alla Javafiler kommer att finnas och dels skapar `tempcopy` med endast den aktuella Javafilen. Efter detta anropas funktionen `singleClass(linkname, numb)` där `linkname` endast är namnet på Javafilen.

void Edit::singleClass(const char *linkname, int numb)

Här anropas de funktioner som gör ”beräkningar” på de enskilda Javafilerna. När sedan dessa anrop är färdiga skrivs resultaten till den länkade listan genom funktionsanropet `Met.listinsert(…)).`

int Edit::countVariabel()

Funktionen returnerar antalet variabler av datatyp som är standard i Java. Dessa datatyper är listade i filen `scriptfile`, se bilaga A. Om fler datatyper läggs till i språket blir det lätt att lägga till dessa i `scriptfile`. I varje Javafil söker programmet efter dessa datatyper och alla rader med träffar läggs i en ny fil, `tempfile`.

För att sedan komma åt antalet deklarationer av variablerna, det kan finnas fler än en per rad som till exempel `’int a,b;’`, räknas förekomsten av kommatecken eller semikolon. För att kunna göra detta läses `tempfile` in till en array. I arrayen jämförs varje sedan varje tecken med `’,’` eller `’;’.` Om jämförelsen nu stämmer ökas en räknare, `int j`, upp med värdet ett. Värdet på variabeln `j` är nu antalet kommatecken och semikolon i filen.

Om en ’`int`’ deklareras i en for-sats kommer programmet att räkna upp antalet med två (två semikolon inuti parentesen). För att undvika detta tas alla for-loopar bort i `tempfile` och läggs i en ny fil, `tempfile2`. För att nu inte ”tappa” heltalsdeklarationen helt räknas antalet for-loopar med en sådan deklaration i sig, som sedan läggs till räknaren `’j’`.

Ytterligare ett fall som returnerar ett felaktigt svar är om en funktion skrivs direkt på samma rad som exempelvis följande uttryck:

```
public int fname() { return counter;}
```

varvid vi har valt att reducera filen med rader innehållande tecknet *'{'*.

int Edit::countPublic(),countPrivate(),countProtected()

Dessa funktioner returnerar antal publika, antal privata respektive antal metoder som är ”protected” i en Javaklass. Eftersom funktionerna är snarlika med endast ett undantag för ett sökord beskriver vi bara den funktion som returnerar antalet publika metoder.

Den enda sökning av ett mönster som görs är:

```
grep -c 'public .*(.*)' tempcopy>>tempfile
```

som returnerar antalet rader där ordet *public* och ett parentespar förekommer på samma rad i filen *tempcopy*.

int Edit::countMethods()

För en metod som inte är deklarerad med till exempel *public* blir det svårare att hitta ett unikt mönster för en funktion. Resultatet av detta blev att hitta ett mönster som hittar samtliga dessa metoder och sedan stegvis ”ta bort” alla andra fall som också stämmer överens med det ursprungliga sökmönstret. Den första sökning blir:

```
grep '.*(.*)' temcopy>>tempfile1
```

Eftersom funktionen inte skall returnera metoder deklarerade med till exempel *public* måste rader innehållande *public, private* och *protected* tas bort. Rader innehållande något av tecknen *'=;/.'* tas också bort då de normalt inte finns med på samma rad där en metod deklareras. Andra rader som matchar första sökningen men som inte är en deklaration av en metod är: *for, if, while, do-while, switch* och *catch*.

Ovanstående undantag bygger till största del på prövning mot olika källkoder. Vi är medvetna om att fler undantag kan finnas. Om det visar sig finnas några vanligt förekommande undantag som matchar första sökningen är dessa inte svåra att komplettera koden med.

void Edit::classCall(int vektor[], int numb)

Funktionen *classCall* skapar en heltalsvektor för varje Javafil, vars storlek är antalet Javafiler som den valda katalogen innehåller. Denna heltalsvektor innehåller antalet

anrop till andra klasser. Då ordningen på hur klasserna kommer till *classCall()* och ordningen i filen *nameonfiles* är densamma blir ordningen i heltalsvektorn korrekt.

För varje Javaklass läses ett klassnamn in, en i taget, för att kunna läggas in i vektorn. En förutsättning är här att Javafilen måste var namngiven på samma sätt som klassen. I *tempcopy* söks efter klassnamn ett efter ett, i koden motsvaras detta av:

```
ifstream FIL("nameonfiles", ios::in);
while(FIL.peek()!=EOF && count<numb){…}
```

Anropen till andra klasser sker med ett objekts namn. Vi har valt att leta efter två typer av mönster i *tempcopy* för att hitta objektens namn:

1. `klassnamn objektnamn1, objektnamn2;`
2. `klassnamn objektnamn = new f();`

En fil innehållande rader med klassens namn skapas, *tempfile*, efter sökning av klassens namn. Det första fallet för att komma åt enbart objektens namn blir att stegvis ta bort allt annat som inte är del av något objekts namn. Filen *semikfile* innehåller nu endast objektens namn för den aktuella klassen.

I andra fallet tas klassnamnet och *'='* bort och läggs i en ny fil. Efter detta returneras endast första ordet i varje rad till en fil, *newobjectfile2*, som nu är objektnamnet.

Det finns nu två filer med objektnamn, dessa slås samman till en ny fil, *searchfile*. Denna fil är den som anropen söks efter. För att hitta ett anrop söks efter objektnamnet direkt följt av en punkt.

Summan av antalet anrop för de olika objekten läggs nu in i heltalsvektorn. På varje klass respektive plats i vektorn sätts automatiskt värdet till noll eftersom ett anrop till sin egen klass inte är möjligt.

Värt att notera är att endast anrop till klasser angivna i den valda katalogen kan hittas i denna funktion.

void Edit::run()

Efter att samtliga klasser har lästs in och parametrarna för varje enskild klass har ”beräknats” anropas funktionen *run()* från *parseFiles()*, med villkoret att det finns minst en klass i listan. Funktionen anropar nu alla beräkningsfunktioner som ”arbetar” på

filen *entirecopy* och därefter *QMessageBox::information* [11] där val av utskrift görs. Beroende på valet av utskrift anropas *print1* eller *print2.*

void Edit::print1(), print2()

Beroende på vilket utskriftsval som görs anropas olika utskriftsfunktioner av de respektive funktionerna *print1* och *print2.* Dessa funktioner skriver endast ut resultaten i *KDE*-fönstret.

Även de funktioner som gör en värdering av de beräknade parametrarna anropas från dessa funktioner. Dessa funktioner kommer inte att förklaras närmare här utan läsaren hänvisas till kapitel fyra där dessa behandlas närmare.

int Edit::countLines()

Funktionen läser in enbart antal rader från filen *entirecopy* som innehåller samtliga filer från den valda katalogen.

int Edit:: countBlanks()

Denna funktion returnerar endast antalet tomma rader.

int Edit:: countComm()

Funktionens syfte är att ”beräkna” antalet rader med kommentarer. Det finns tre varianter av kommentarer i Java:

```
1. //…kommentarer…
2. /*…kommentarer…*/
3. /**…kommentarer…*/
```

Tillvägagångssättet för fall 2 och 3 är att läsa in `'/*'` och `'*/'` till två separata filer med radnumret först på varje rad. Eftersom en kommentar som börjar med `'/*'` måste följas av `'*/'` för att markera att kommentaren är slut kan radnumren på filerna jämföras parvis. Figur 3.5 innehåller ett exempel på hur dessa filer skulle kunna se ut. Antalet rader blir (5-4+1) + (15-9+1), dvs 9 rader kommentarer.

fil innehållande `'/*'` eller `'/**'`	fil innehållande `'*/'`
4	5
9	15

Figur 3.5: Figur över filer med endast radnummer

Fall 1 blir betydligt enklare att komma åt, där endast en sökning med antalet rader som innehåller `'//'` returneras enligt:

```
grep -c '//' entirecopy
```

Det värdet som funktionen returnerar är summan av de bägge sökvarianterna. En sak som bör beaktas är att om det finns kommentarer inuti andra kommentarer räknas även dessa.

int Edit:: countSigns(int lines)

Funktionen returnerar kvoten mellan antalet tecken och antalet rader från filen `entirecopy`, dvs all Javakod i katalogen. Detta blir ett mått på det genomsnittliga antalet tecken per rad.

Vi har valt att reducera antalet tecken per rad med antalet ord som varje rad innehåller. Detta för att inte räkna med de mellanslag och tabbar som annars skulle räknats med.

int Edit:: countLoops()

Metoden skall komma åt antalet loopar som finns i filen `entirecopy`. Det finns i programspråket Java tre varianter av loopar:

1. for-loop
2. while-loop
3. do-while-loop

Sökningen efter fall 2 och 3 görs samtidigt eftersom bägge innehåller mönstret `'while.*(.*)'`. Programmet `grep` används här för att returnera antalet rader där ett sådant mönster påträffas.

Även för att komma åt antalet for-loopar används `grep` enligt:

```
grep -c 'for.*(.*;.*,.*)' entirecopy
```

Funktionen returnerar summan av antalet rader av de bägge fallen.

int Edit:: countMagic()

Funktionen ”beräknar” och returnerar förekomsten av ”magic numbers”. Dock ej de tal som är deklarerade som konstanter med hjälp av `'final'` samt heltalen 0 och 1. Funktionen hittar först de rader som innehåller någon siffra med undantag för konstanterna och skapar en ny fil med dessa rader, `tempfile2`.

För att komma åt förekomsten av ”magic numbers”, det kan finnas fler än en på en rad, läses `tempfile2` in till en array, `char b[]`. Hela arrayen gås sedan igenom och för varje heltal som påträffas, dock ej 0 eller 1, ökas en räknare upp med värdet ett.

En begränsning är här att decimaltal som innehåller en punkt som exempelvis 3.14, kommer att räknas som två ”magic numbers” av programmet.

4 Värdering av parametrar

Att värdera parametrar kan ses som en subjektiv bedömning utifrån vissa godtyckliga mått. Författarna *Kemerer & Chidamber* har gjort en beskrivning av olika mått som kan användas för analys och utvärdering av objektorienterade system [7]. Måtten grundar sig på intervjuer av ett antal experter på området ”programdesign”. Följande definition av god design har *Kemerer & Chidamber* presenterat:

> *”good software design practice calls for*
> *minimizing coupling and maximizing cohesiveness”.*

Kemerer & Chidamber definierar god design som en eftersträvan att minska kopplingen mellan klasserna och att öka sammanhanget (likheten) mellan metoderna i samma klass. Med likhet menas att metodernas uppsättning av klassvariabler är enhetlig. Kopplingen mellan två klasser definieras enligt följande [7]:

”Any evidence of a method of one object using methods or instance varibles of another object constitutes coupling.”

En programdesign som har ett högt mått av koppling anses generellt vara mer komplex än en design med svag koppling.

Med definitionen av god design som grund har *Kemerer & Chidamber* skapat följande sex mått (metrics) [7]:

1. WMC: (Weighted Methods per Class) Ger en indikation på att en komplex (stor) klass med många metoder påverkar dess subklasser betydligt högre än en liten klass.
2. DIT: (Depth of Inheritance Tree) Indicerar en djup arvshierarki i systemet. Detta påverkar klasser djupt nere i hierarkin på ett oförutsägbart sätt.
3. NOC: (Number Of Children) Klasser med många subklasser indiceras som särsklit viktiga utifrån detta mått.
4. CBO: (Coupling Between Object classes) Indicerar att ett begränsat antal klassrelationer ökar återanvändbarheten av koden.
5. RFC: (Response For a Class) Beräknar och anger antalet metoder som exekverar som svar på ett meddelande (klassanrop). Högt RFC indicerar komplex klass. Klasser på botten av klasshierarkin har högre RFC än toppklasser.
6. LCOM: (Lack of COhesian in Methods) Indicerar likheten mellan två klasser. Klasser med stora olikheter bör delas in i två separata klasser för att förhindra inkapsling

Med hänsyn till vilka parametrar programmet utvärderar är följande tre mått tänkbara att använda som riktmärken i detta arbete: WMC, NOC, CBO. Vi har valt att definiera måtten som vi kallar *readability, reuseability och complexity* något annorlunda, se nedanstående kapitel om respektive mått. De övriga anser vi vara för komplicerade att beräkna med de tillgängliga verktygen i *Unix* (grep, *wc, awk, sed* mm). I Figur 4.2 finns en lista över vilka parametrar som kan utvärderas.

Då någon för ändamålet adekvat litteratur ej har påträffats är värderingen och dess bakomliggande algoritmer ett resultat av våra egna personliga bedömningar och beslut.

Enskild klass

- Antal rader
- Antal tomma rader
- Antal tecken per rad
- Antal rader med kommentarer
- Antal klasser
  - Antal loppar
  - Antal ”magic numbers”

Alla klasser

- Antal publika metoder
- Antal privata metoder
- Antal ”protected” metoder
- Antal metoder
- Antal variabler
- Klassanrop

Figur 4.1: Parametrar att utvärdera

För att beräkna dessa mått har en fyrgradig betygskala från noll till tre valts. För varje beräkning startar värdet på noll varvid måtten kontrolleras, klarar måtten ett visst kriterium ökas betyget upp med en enhet. Kriterierna som skall uppfyllas för respektive mått redovisas i nedanstående kapitel.

Användaren kommer inte att ta del av denna betygskala utan en översättning görs i programmet där betyget översätts till en textfras enligt figur 4.2.

Betyg	**Text**
0	*”Bad”*
1	*”Not so good”*
2	*”Good”*
3	*”Very good”*

Figur 4.2: Översättning från betyg till text

4.1 Readability

Readability är ett mått på en källkods läsbarhet. Aspekter på läsbarheten kan vara antal tecken per rad, antal rader med kommentarer och andelen tomma rader. Det kan vara svårt att tolka kod då den är massivt, många tecken och kommandon på samma rad, skriven. Kod som är luftigt skriven, strategiskt placerade tomrader, kan vara enklare att följa schematiskt. Tomrader är exempelvis att rekommendera mellan olika metoder och för att skilja variabeldeklarationer m.m. från metodens instruktioner. Överdriven användning av tomrader leder till motsatt effekt.

Antalet tecken per rad bör hållas på en rimlig nivå. I vissa kretsar av programmerare är det legitimt att skriva flera instruktioner på samma rad i följd. Detta är förstås tillåtet med avsikt på språkets syntax, men kan anses försvåra källkodens *readability*, då det logiska sambandet mellan instruktionerna blir svårare att följa.

I de flesta programmeringsspråk är det möjligt att skriva fortlöpande kommentarer i källkoden. Dessa skiljs från koden genom något metatecken till exempel `"//", "/*….*/"` i

C och *C++*. Antalet kommentarer i koden påverkar läsbarheten ur olika aspekter. Ett väl kommenterat program med beskrivningar av varje metod och dess uppgift kan anses öka läsbarheten då användaren endast behöver läsa kommentaren för att förstå metoden. Ur en annan synvinkel kan program med hög andel kommentarer anses få minskad läsbarhet. Det kan vara irriterande att analysera koden då kommentarerna överskuggar instruktionerna och i vissa fall är det helt enkelt svårt att skilja mellan vad som är dokumentation och vad som är kod.

För att göra en bedömning av läsbarheten använder vi oss av fyra parametrar*: antal rader, antal tomma rader, tecken/rad, antal rader med kommentarer*. Antal rader används för att kunna jämföra övriga parametrar med. Vi använder resultatet från tre algoritmer.

Nedan följer en beskrivning av algoritmerna och intervallen som krävs för att erhålla en poäng.

1. *antal rader kommentarer / antal rader* *intervall: 0.1-0.2*
2. *antal tomrader /antal rader* *intervall: 0.1-0.2*
3. *antal tecken per rad* *intervall: < 30*

Algoritm (1) har ett poängintervall som ligger mellan 10 och 20 procent. Denna bedömning är subjektiv och grundar sig på studier av befintlig Javakod. Programmet tar inte hänsyn till var i koden kommentarerna finns, dvs alla kommentarer kan ligga i ett block överst i källkodsfilen eller vara utspridd till varje funktion.

Algoritm (2) har även den ett intervall mellan 10 och 20 procent. Programmet tar på samma sätt som med kommentarerna ingen notis om vart tomraderna finns eller hur de är fördelade i koden. Den till synes höga andelen tomrader anser vi behövas för att skapa lättläst och luftigt källkod.

Den övre gränsen för antalet tecken per rad som i medelvärde bör förekomma sätts till 30. Då algoritm (3) i programmet inte räknar med antalet tomrader i nämnaren, som skulle påverka medelvärdet drastiskt, anser vi 30 tecken vara en realistisk övre gräns.

4.2 Reuseability

Ett programs *reuseability* (återanvändbarhet) anger huruvida programmet är anpassat för att kunna återanvändas i andra system. Detta kan gälla hela programmet eller delar av det, dvs

vissa metoder eller klasser. Att skapa återanvändbara program blir alltmer viktigare på dagens mjukvara marknad. De nya så kallade objektorienterade språken ökar möjligheten till detta. Tanken är att klasserna i programmen skall vara atomära med väl definierade gränssnitt då detta möjliggör att använda samma klass i andra program.

Att bedöma återanvändbarheten manuellt kräver en analys av klasskopplingen. *Enligt Kemerer & Chidamber* [7] är CBO ett mått på återanvändbarhet. Låg klasskoppling indicerar hög *reuseability* och tvärtom. De parametrar som för ändamålet anses intressanta är: *antal klasser, antal magic numbers* och *antalet klassanrop*. Dessa parametrar är relativt begränsade för att skapa sig en uppfattning varvid samma betygsskala används med undantaget att den startar med värdet ett trots att endast två kriterier kontrolleras.

Då återanvändbarheten endast beräknas för hela programmet är resultatet beroende av hur många klasser som ingår. Som en följd av detta ökar klasser med hög koppling medelvärdet och indirekt blir klasser med god ”*reuseability*” felaktigt bedömda.

Antalet ”magic numbers”, tal som står i klartext, påverkar värdet. Ett högt antal ”magic numbers” kan komplicera modifikationen av klasser som skall återanvändas. Antalet bör givetvis jämföras mot storleken på programmet, antal klasser. Ett visst antal tal i klartext bör accepteras då det är svårt att helt frångå ”magic numbers” vid programmering. Programmet räknar med alla tal i klartext, förutom de tal som ingår i en konstant deklaration samt talen 0 och 1.

Algoritmerna som programmet använder för att bedöma återanvändbarheten är följande:

1. *antalet klassanrop / antal klasser* *intervall:<4*
2. *antal ”magic numbers” /antal rader* *intervall:<0.05*

4.3 Complexity

Innan man bedömer komplexiteten i ett program måste man definiera vad som menas med ett ”komplext program”. Det går att väga in många aspekter i begreppet. *Kemerer & Chidamber* [7] menar att det tyngsta måttet på komplexiteten är klasskopplingen. Stark koppling mellan klasserna ökar komplexiteten. Sambandet mellan *reuseability* och *complexity* är enligt definitionerna uppenbart. Låg återanvändbarhet tyder på hög komplexitet. En djup arvshierarki och ett stort antal metoder i programmet anses också öka komplexiteten enligt

författarna. En klass med många metoder är i sig komplex, men påverkar även subklasser längre ner i arvshierarkin vilket ökar programmets komplexitet.

De parametrar som används vid bedömningen är: *antal loopar, antal metoder, antal variabler, antal klassanrop.*

Generellt anser vi att ett stort antal loopar ökar komplexiteten i en metod. Det kan dock vara befogat att påpeka att det är diskutabelt huruvida antalet loopar faller in under begreppet komplexitet. Vidare bör nämnas att programmet inte mäter nästlingsdjupet i for-loopar utan endast beräknar förekommande loopar till antalet.

Ett ökat antal deklarerade variabler anser vi påverka komplexiteten i någon grad. När ett stort antal variabler behandlas och programmet har en hög grad av klasskoppling ökar komplexiteten och risken för att fel skall uppstå under körning. För att bedöma komplexiteten har vi i programmet använt följande algoritmer och poängintervall:

1. *antal klassanrop / antal klasser* *intervall: <4*
2. *antal variabler / antal klasser* *intervall:<10*
3. *antal loopar /antal klasser* *intervall:<5*
4. *antal metoder totalt / antal klasser* *intervall:<8*

5 Test av program

I detta avsnitt presenteras de resultat vi erhållit under verifieringen av programmet. Att verifiera program har blivit ett allt viktigare delmoment i programutvecklings fasen. Ett väl testat program har större möjlighet att klara sig från att krascha när det är i drift. Vid verifieringen är det legitimt att utsätta systemet för kända parametrar som kan orsaka driftstopp. Dessa parametrar är ofta gränsvärden, exempelvis 0, 1 och -1 , och orsaker fel som beror på programbuggar.

Då programmet tar källkod som inparameter och denna behandlas subjektivt oberoende av dess logiska struktur kan ej detta påverka programkörningen. Med andra ord bör det inte uppstå exekveringsfel oavsett vilken källkod som matas in i programmet. Vad som kan påverkas och är av stor vikt för programmets pålitlighet är ”outputen” (resultatet). Det är

denna ”output” som vi har valt att verifiera. För att göra en rimlig bedömning av resultatet förutsätter vi att användaren känner till de tidigare nämnda begränsningar som programmet har och att den Javakod som skall behandlas är skriven enligt de vedertagna regler som finns. Vid testet har först alla parametrar som erhålls ur programmet beräknats manuellt. Resultatet jämförs sedan med programmets ”output”. Följande Java-klasser har använts vid den första provkörningen: *ListBarCanvas, ListBarObject, SortDemo, SorterAlgorithm, SortingRunner.* Klasserna ingår i ett program som grafiskt demonstrerar ett antal sorteringsalgoritmer. Koden har laddats ner från *Internet* [10].

För att förutsätta en mer detaljerad analys valde vi att först köra varje klass för sig i programmet. På så sätt erhölls följande parametrar klassvis: *antal magic numbers, antal loopar, antal kommentarer, antal variabler.* Resultatet från det första testet presenteras nedan i Figur 5.1.

Differensen anger skillnaden mellan resultatet av parametern erhållen manuellt respektive programmet. Några av resultaten kräver en noggrannare analys.

I klassen *ListBarCanvas* är differensen fyra beträffande parametern ”variabler”. Orsaken till detta finns att hitta i klassens källkod som innehåller en metod `repaint ();`. Då programmet letar igenom källkoden för att hitta variabler används en fil som innehåller aktuella sökord, `char, int, float` m.m. Denna fil matchas mot koden och programmet indikerar förekomst av sökorden. En begränsning i programmet är att det ej tar hänsyn till om sökorden uppenbarar sig i slutet av ett annat ord där det dessutom skall innehålla ett semikolon på samma rad. Vid kontroll har det visat sig att `repaint ();` förekommer fyra gånger i koden, varpå orsaken till differensen får anses hittad.

I klassen *SortDemo* hittar man fem rader med kommentarer medan programmet endast returnerar parametern tre. Under vidare kontroll av funktionen `countComm` visade det sig att vid multipel förekomst av kommentar tecknet `"/*"` returnerar programmet endast värdet ett oavsett var i koden kommentarerna finns. Efter modifikation av funktionen har ytterligare tester genomförts som verifierar korrektheten.

ListBarObject

	Manuell	*Program*	*Differens*
Tomrader:	13	13	0
Tecken/rad:	-	-	-
Kommentarer:	0	0	0
Loopar:	0	0	0
Variabler:	8	8	0
Magicnum:	0	0	0
Publica:	7	7	0
Privata:	0	0	0
Protected:	0	0	0
Metodhs:	0	0	0

ListBarCanvas

	Manuell	*Program*	*Differens*
Tomrader:	50	52	2
Tecken/rad:	-	-	-
Kommentarer:	6	6	0
Loopar:	7	7	0
Variabler:	22	26	4
Magicnum:	12	12	0
Publica:	16	16	0
Privata:	0	0	0
Protected:	3	3	0
Metodhs:	1	1	0

SortDemo

	Manuell	*Program*	*Differens*
Tomrader:	38	36	2
Tecken/rad:	-	-	-
Kommentarer:	5	3	2
Loopar:	0	0	0
Variabler:	7	8	1
Magicnum:	8	8	0
Publica:	7	7	0
Privata:	0	0	0
Protected:	0	0	0
Metodhs:	0	0	0

SorterAlgorithm

	Manuell	*Program*	*Differens*
Tomrader:	14	14	0
Tecken/rad:	-	-	-
Kommentarer:	0	0	0
Loopar:	0	0	0
Variabler:	12	11	1
Magicnum:	1	0	1
Publica:	2	2	0
Privata:	0	0	0
Protected:	0	0	0
Metodhs:	1	1	0

SortingRunner

	Manuell	*Program*	*Differens*
Tomrader:	44	44	0
Tecken/rad:	-	-	-
Kommentarer:	0	0	0
Loopar:	28	28	0
Variabler:	12	11	1
Magicnum:	13	13	0
Publica:	1	1	0
Privata:	0	0	0
Protected:	11	11	0
Metodhs:	2	3	1

Alla klasser

	Manuell	*Program*	*Differens*
Tomrader:	822	822	0
Tecken/rad:	-	-	-
Kommentarer:	11	9	2
Loopar:	40	39	1
Variabler:	92	96	4
Magicnum:	33	32	1
Publica:	33	33	0
Privata:	0	0	0
Protected:	14	14	0
Metodhs:	4	5	1

Anropen mellan klasserna enligt program

	LBO	LBC	SD	SA	SR
ListBarObject	0	0	0	0	0
ListBarCanvas	4	0	0	0	0
SortDemo	0	12	0	0	0
SorterAlgorithm	0	0	0	0	0
SortingRunner	1	0	0	0	0

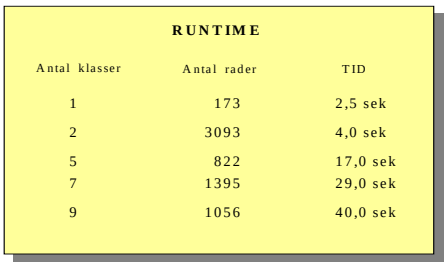
Anropen mellan klasserna manuellt

	LBO	LBC	SD	SA	SR
ListBarObject	0	0	0	0	0
ListBarCanvas	4	0	0	0	0
SortDemo	0	12	0	0	0
SorterAlgorithm	0	0	0	0	0
SortingRunner	1	0	0	0	0

Figur 5.1: Parametrar manuellt och med programmet

Genom att mata in olika källkoder, beträffande antal rader och antal klasser, har vi försökt uppskatta hur programmets ”runtime” påverkas. Med ”runtime” menas här tiden från att

programmet börjar beräkna parametrarna i klasserna tills att användaren erbjuds att skriva ut resultatet. Figur 5.2 visar olika ”runtimes” vid olika körningar där antalet klasser och antal rader varierar.



RUNTIME

Antal klasser	Antal rader	TID
1	173	2,5 sek
2	3093	4,0 sek
5	822	17,0 sek
7	1395	29,0 sek
9	1056	40,0 sek

Figur 5.2: Tider som uppmäts vid olika källkods ”input”.

Med de uppmäta tiderna som bakgrund drar vi slutsatsen att antalet klasser påverkar programmets effektivitet i högre grad än antalet rader. Vid exekvering mot två Javaklasser som innehåller 3093 rader kod är svarstiden cirka 4 sekunder. Detta skall jämföras mot ett Javaprogram innehållande 9 klasser, men endast 1056 rader kod. Svarstiden blir då 40 sekunder. Den markanta skillnaden i svarstid beror på funktionen `classCall` som beräknar hur många gånger de olika klasserna anropar varandra. Varje klass källkod parsas igenom x-1 antal gånger, där x är antalet klasser som programmet innehåller. Dvs antalet loopar som letar efter klassanrop i koden ökar kvadratiskt mot antalet klasser. I funktionen öppnas och stängs ett antal temporära filer på sekundärminnet vilket är en relativt långsam process.

Vi har av tidsskäl ej hunnit att testa alternativa lösningar, exempelvis skriva ner källkoden till en array.

6 Diskussion

Innan uppsatsens början hade vi som ambition att göra en översiktlig åskådning och analys av befintliga utvärderingsverktyg. Efter att ha letat på *Internet* under en period med magert resultat valde vi att avsluta aktiviteten. Då hade endast ett program som hittats liknade det vi sökte. Programmet fanns i två versioner, en mindre som hade begränsning i antalet rader kod som gick att analysera. Denna variant gick att beställa för en kostnad av cirka 200 dollar.

Kraven för uppsatsens programdel som tidigare nämnts anser vi vara uppfyllda. Den utvärdering som sker i programmet skall ses som en orientering eftersom de subjektiva bedömningar som gjorts ej har vetenskaplig förankring.

Det skall också nämnas att programmet ej tar någon hänsyn till kodens syntax eller semantik. Programfunktionerna har växt fram allt eftersom vi lärt oss de verktyg som varit aktuella att använda. Utvecklingsfasen har inneburit att funktionerna är i hög grad beroende av varandra och ej atomära, dvs de är svåra att återanvända.

På grund av tidsbrist har ej en analys programmets utformning i detalj gjorts för att kunna göra förbättringar beträffande effektivitet och modularitet. Det mest akuta förändrings arbetet på programmet vore att förbättra effektiviteten. Enligt de tester som genomförts erhålls svarstider på 40 sekunder redan vid en input bestående av nio klasser. Den långa svarstiden beror på att programmet öppnar och stänger temporära filer i sekundärminnet under exekvering. En alternativ lösning vore att läsa in den aktuella källkoden till primärminnet, ”arrayer”, då de långa accesstiderna ner till systemets diskminne undviks. Ett problem som medföljer denna lösning vore att de verktyg, *sed, awk, grep* m.m., som programmet använder sig av för att hitta visa parametrar ej skulle gå att använda. Den mest tids fördröjande faktorn i programmet är funktionen som beräknar klassanrop. Här sker en stor del av programmets filhantering, och eftersom anropen till funktionen ökar kvadratiskt mot antalet klasser så erhålls långa svarstider. Anledningen till att det sker ett antal filöppningar i funktionen är att sökningen efter klassanrop måste ske i flera steg. På grund av Javas syntax erhålls felaktiga resultat då koden endast matchas mot ett sökmönster. Istället sker sökningen i flera steg med ett allt finare nät som med stor sannolikhet endast resulterar i de träffar som är intressanta.

7 Referenser

[1] Blom Martin. Metodbeskrivning för pilotprojekt. *Arbetsrapport, Datavetenskap Karlstad universitet*, 1999

[2] Dougherty Dale. sed & awk. *O´Reilly & Associates, Inc*., 1990.

[3] Eriksson Hans-Erik. Programutveckling med Java. *Studentlitteratur*, 1997.

[4] GNU wc-manual, version 1.22.

[5] GNU sed-manual, version 3.02.

[6] GNU awk-manual version 3.0.3.

[7] C.F. Kemerer and S.R. Chidamber. Moose - Metrics for object oriented software engieneering. In Workshop on Processes and Metrics for ObjectOriented Software Development, OOPSLA '93, Washington, 1993

[8] Skansholm Jan. Java direkt. *Studentlitteratur*, 1998.

[9] http://www.cs.kau.se/cs/skutt/exjobb/tool.htm

[10] http://www.dd.chalmers.se/~f96hajo/sortdemo/sd_max_eng.htm

[11] http://www.doc.trolltech.com/

[12] http://www.cs.kau.se/cs/skutt/projects.html

8 Bilaga A: Koden

A.1: edit.h

```
#ifndef EDIT_H
#define EDIT_H

#include <ktopwidget.h>
#include <qmlined.h>

/*********************************************************************/
/* constants                                                         */
/*********************************************************************/
const char SPACELINE[]="                                  ";
const int U=10;
const int ITOAL=10;
const int VIEWL=40;
const int MAXSTR=100;
const int DECI=10;
const int CENTI=100;
const int TABLEWIDHT=25;

/*********************************************************************/
/* intervalls to evaluate the sourcecode                             */
/*********************************************************************/
//readability
const float UPPER = 0.2;
const float LOWER = 0.1;
const int MAXSIGN = 30;

//reuseability
const int MAXCLASS = 4;
const float MAXMAGIC = 0.05;

//complexity
const int MAXVARIABELS = 10;
const int MAXLOOP = 5;
const int MAXMET = 8;

/*********************************************************************/
/* "const char" is operations to receive metrics in the program      */
/*********************************************************************/
//commandCallback()
const char* GETJAVAFILE ="grep '.java' filedir>>filedir2";
const char* GETNUMFILE ="wc -l filedir2>>numfiles";
const char* TAKEAWAYJAVA ="sed -e 's/.java//' filedir2>>nameonfiles";
//countLines()
const char* COUNTLINES ="wc -l entirecopy>>tempfile";
//countBlanks()
const char* COUNTEMPTY ="grep -c -v '[a-zA-Z{}/]' entirecopy>>tempfile";
//countComm()
const char* COUNTCOMM1 ="grep -n '[/][*]' entirecopy>>test";
const char* COUNTCOMM2 ="sed -e 's/:/ /' test>>test2";
const char* COUNTCOMM3 ="awk '{print $1}' test2>>test3";
const char* COUNTCOMM4 ="wc -l test3>>numfiles";
const char* COUNTCOMM5 ="grep -n '[*][/]' entirecopy>>test4";
const char* COUNTCOMM6 ="sed -e 's/:/ /' test4>>test5";
const char* COUNTCOMM7 ="awk '{print $1}' test5>>test6";
```

```
const char* COUNTCOMM8 ="grep -c '//' entirecopy>>temp";
//countSigns()
const char* COUNTSIGNS ="wc --b entirecopy>>tempfile";
const char* COUNTWORDS ="wc -w entirecopy>>tempfile";
//countLoops()
const char* COUNTFOR ="grep -c 'for.*(.*;.*;.*)' entirecopy>>tempfile";
const char* COUNTWHILE ="grep -c 'while.*(.*)' entirecopy>>tempfile";
//countMagic()
const char* COUNTNUMBERS1 ="grep '[0-9]' entirecopy>>tempfile";
const char* COUNTNUMBERS2 ="grep -v 'final' tempfile>>tempfile2";
const char* COUNTNUMBERS3 ="wc --b tempfile2>>tempfile3";
//numPublic()
const char* COUNTPUBLIC ="grep -c 'public .*(.*)' tempcopy>>tempfile";
//numPrivate()
const char* COUNTPRIVAT ="grep -c 'private .*(.*)' tempcopy>>tempfile";
//numProtected()
const char* COUNTPROT ="grep -c 'protected .*(.*)' tempcopy>>tempfile";
//numMethods()
const char* COUNTMETHODS1 ="grep '.*(.*)' tempcopy>>tempfile1";
const char* COUNTMETHODS2 ="grep -v 'while' tempfile1>>tempfile2";
const char* COUNTMETHODS3 ="grep -v 'for' tempfile2>>tempfile3";
const char* COUNTMETHODS4 ="grep -v 'if' tempfile3>>tempfile4";
const char* COUNTMETHODS5 ="grep -v '[=;/.]' tempfile4>>tempfile5";
const char* COUNTMETHODS6 ="grep -v 'switch' tempfile5>>tempfile6";
const char* COUNTMETHODS7 ="grep -v 'public ' tempfile6>>tempfile7";
const char* COUNTMETHODS8 ="grep -v 'private ' tempfile7>>tempfile8";
const char* COUNTMETHODS9 ="grep -v 'protected ' tempfile8>>tempfile9";
const char* COUNTMETHODS10 ="grep -v 'catch' tempfile9>>tempfile10";
const char* COUNTMETHODS11 ="wc -l tempfile10>>tempfilerun";
//numVariables()
const char* COUNTVARIABLE1 ="awk -f scriptfile tempcopy>>tempfile";
const char* COUNTVARIABLE2 ="grep -v 'for.*(.*;.*;.*)' tempfile>>tempfile2";
const char* COUNTVARIABLE3 ="grep -v '{' tempfile2>>tempfile4";
const char* COUNTVARIABLE4 ="wc --b tempfile4>>tempfile3";
const char* COUNTVARIABLE5 ="grep -c 'for.*(.*;.*;.*)' tempfile>>tempfile5";
//classCall()
const char* COUNTCLASSCALL1 =" [0-9A-Za-z, ]*;' tempcopy>>tempfile";
const char* COUNTCLASSCALL2 ="sed -e 's/.*";
const char* COUNTCLASSCALL3 =" //' tempfile>>objectfile";
const char* COUNTCLASSCALL4 ="sed -e 's/,/ /' objectfile>>commfile";
const char* COUNTCLASSCALL5 ="sed -e 's/;/ /' commfile>>semikfile";
const char* COUNTCLASSCALL6 ="cat semikfile>>searchfile";
const char* COUNTCLASSCALL7 =" .*=.*;' tempcopy>>tempfile2";
const char* COUNTCLASSCALL8 =" //' tempfile2>>objectfile2";
const char* COUNTCLASSCALL9 ="sed -e 's/=/ /' objectfile2>>objectfile3";
const char* COUNTCLASSCALL10 ="awk '{print $1}' objectfile3>>newobjectfile2";
const char* COUNTCLASSCALL11 ="cat newobjectfile2>>searchfile";
const char* COUNTCLASSCALL12 ="wc -w searchfile>>numfiles";
const char* COUNTCLASSCALL13 ="[.].*' tempcopy>>hitfile";


class Edit : public KTopLevelWidget
{
   Q_OBJECT

 public:
   Edit();
   ~Edit();
   void itoa(char* str, int number);
   void parseFiles(const char *dirname);
   void load(const char *filename,const char *linkname,int numb);
   int countLines();
   int countBlanks();
   int countComm();
   int countSigns(int lines);
   int countLoops();
   int countMagic();
   void enableCommand(int id);
```

```
    void disableCommand(int id);
    void run();
    int evalRead(int line,int blank,int sign,int com);
    int evalReuse(int kl,int magi,int line);
    int evalComp(int kl,int loop);
    char* translate(int para);
    void printTable(int kl);
    void printMetrics(int line,int blank,int sign,int com,int kl,int loop,int magi);
    void printSingle(int kl);
    void printAll(int line,int blank,int sign,int com,int kl,int loop,int magi);
    void printSum(int kl);
    void print1(int line,int blank,int sign,int com,int kl,int loop,int magi);
    void print2(int line,int blank,int sign,int com,int kl,int loop,int magi);
    void singleClass(const char *linkname,int numb);
    int countPublic();
    int countPrivate();
    int countProtected();
    int countMethods();
    int countVariabel();
    void classCall(int vektor[], int numb);
  public slots:
    void commandCallback(int id_);

  private:
    KMenuBar *menu;
    QMultiLineEdit *view;
    QString filename_;
    void initMenuBar();
    list<int> Met;
};
#endif
```

A.2: edit.cpp

```
#include "list.cpp"
#include <qfiledlg.h>
#include <qfile.h>
#include <kapp.h>
#include <kmenubar.h>
#include <qstring.h>
#include <qmsgbox.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qlayout.h>
#include <stddef.h>
#include <qbutton.h>
#include <qpushbutton.h>
#include <iostream.h>
#include <fstream.h>
#include <signal.h>
#include "edit.h"
#include "edit.moc"

#define ID_OPEN 100
#define ID_EXIT 101
#define ID_ABOUT 102
#define ID_SAVE 103

/*****************************************************************************/
/* PRE: number is a positive or negative integer                             */
/* POST: number is returned as an ASCII-array                                */
/*****************************************************************************/
void Edit::itoa(char* str, int number)
{
            int radius=U;
            int size=0;
            int num=number;
            while(num)
            {
                        size++;
                        num/=U;
            }
            num=number;
            radius=U;
            for(int i=0;i<size;i++)
            {
                        str[size-i-1]=(char)(num%radius+48);
                        num/=U;
            }
            if(number == 0)
            {
                        str[size]='0';
                        size++;
            }
            str[size]='\0';
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: A new KDE window is created                                         */
/*****************************************************************************/
Edit::Edit()
  : filename_()
{
  initMenuBar();
  view= new QMultiLineEdit(this, "Main View");
```

```
  setView(view);
  show();
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: Destructur                                                          */
/*****************************************************************************/
Edit::~Edit()
{

}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The users choice is executed                                        */
/*****************************************************************************/
void Edit::commandCallback(int id_)
{
  QString dirname;
  char printarr[MAXSTR],command[MAXSTR];

  switch(id_) {
  case ID_OPEN:
            dirname= QFileDialog::getExistingDirectory();
            if (!dirname.isEmpty())
            {
                        strcpy(command, "ls ");
                        strcat(command, dirname);
                        strcat(command, ">>filedir");
                        system(command);
                        system(GETJAVAFILE);
                        system(GETNUMFILE);
                        system("rm filedir");

                        system(TAKEAWAYJAVA);

                        if(Met.listlength()==0)
                                    view->clear();

                        strcpy(printarr,"You have choosed javafiles from the
                                    directory:\n\n");
                        strcat(printarr,dirname);

                        switch( QMessageBox::information(this, "Selected
                                    files",printarr,"Continue","Cancel",0,1)){
                                    case 0:
                                                parseFiles(dirname);
                                                break;
                                    case 1:
                                                system("rm filedir2");
                                                system("rm numfiles");
                                                break;
                        }
            }
            break;
  case ID_ABOUT:
            QMessageBox::about(this,"About Edit","This program is written by
Olsson, Fredrik and
Rosengren, Magnus.\n\nLast updated: 2000-05-08\n\nTo run this program the are
things you should
know about, these are listed below:\n
- you must select a directory, not only a single file
- takes only files named with '.java' in the selected directory
- the classes must be named as the javafiles
- lines with comments within comments will count twice
- number of methods is methods thoose not are public, private or protected
- number of lines is lines with blanks excluded
```

```
- magic numbers counts integers except 0,1 and integers declared as final
- a line content only a space is not an empty line");
            break;
  case ID_EXIT:
            exit(0);
            break;
  case ID_SAVE:
            QString f=QFileDialog::getSaveFileName("java_eval.txt","", this);
            fstream FIL(f, ios::out);
            FIL << view->text();
            FIL.close();
            break;
  }
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: All files in the chosen directory is sent to "load"                 */
/*****************************************************************************/
void Edit::parseFiles(const char *dirname)
{
            char filename[MAXSTR];
            char sendname[MAXSTR];
            int num,r=0;

            ifstream FIL2("numfiles", ios::in);
            FIL2>>num;
            FIL2.close();
            system("rm numfiles");

            ifstream FIL("filedir2", ios::in);
            while (r<num)
            {
                        strcpy(sendname,dirname);
                        FIL>>filename;
                        strcat(sendname,"/");
                        strcat(sendname,filename);
                        load(sendname,filename,num);

                        r++;
            }
            FIL.close();
            system("rm filedir2");

            if(num!=0)
                        run();
            else
                        QMessageBox::information(this, "No javafiles","No matcing
                                    javafiles in your selected directory");
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: All parameters on "entirecopy" is collected and way to print is choosed */
/*****************************************************************************/
void Edit::run()
{
            int lines,emptylines,com,classes,numloop,magic,tprad;

            lines=countLines();
            emptylines=countBlanks();
            lines=lines-emptylines;
            com=countComm();
            tprad=countSigns(lines);
            numloop=countLoops();
            magic=countMagic();
            classes=Met.listlength();
```

```
            switch( QMessageBox::information(this, "Printselection","Choose way to
print\n1: Printing every single classes\n2: Summary of all classes","Print
1","Print 2",0,1)){

            case 0:
            print1(lines,emptylines,tprad,com,classes,numloop,magic);
            break;
            case 1:
            print2(lines,emptylines,tprad,com,classes,numloop,magic);
            break;
            }
            system("rm entirecopy");
            system("rm nameonfiles");
            for(int i=classes;i>0;i--)
                        Met.listdelete(i);
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of all lines is checked                                  */
/*****************************************************************************/
int Edit::countLines()
{
            int x;

            system(COUNTLINES);

            ifstream FIL("tempfile", ios::in);
            FIL>>x;
            FIL.close();
            system("rm tempfile");

            return x;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of empty lines is checked                                */
/*****************************************************************************/
int Edit::countBlanks()
{
            int y;

            system(COUNTEMPTY);

            ifstream FIL("tempfile", ios::in);
            FIL>>y;
            FIL.close();
            system("rm tempfile");

            return y;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of lines with comments is checked                        */
/*****************************************************************************/
int Edit::countComm()
{
            int one,two,w,num,res=0,i=0;

            system(COUNTCOMM1);
            system(COUNTCOMM2);
            system(COUNTCOMM3);
            system(COUNTCOMM4);
            ifstream FIL4("numfiles", ios::in);
            FIL4>>num;
            FIL4.close();
```

```
            system("rm numfiles");

            system(COUNTCOMM5);
            system(COUNTCOMM6);
            system(COUNTCOMM7);

            system(COUNTCOMM8);
            ifstream FIL3("temp", ios::in);
            FIL3>>w;
            FIL3.close();
            system("rm temp");

            ifstream FIL("test3", ios::in);
            ifstream FIL2("test6", ios::in);

            while(i<num)
            {
                        FIL>>one;
                        FIL2>>two;
                        res=res+two-one+1;
                        i++;
            }
            FIL.close();
            FIL2.close();
            system("rm test*");
            res=res+w;
            return res;

}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of signs per line is checked                             */
/*****************************************************************************/
int Edit::countSigns(int numlines)
{
            int sign,result,word;

            system(COUNTSIGNS);

            ifstream FIL("tempfile", ios::in);
            FIL>>sign;
            FIL.close();
            system("rm tempfile");

            system(COUNTWORDS);

            ifstream FIL2("tempfile", ios::in);
            FIL2>>word;
            FIL2.close();
            system("rm tempfile");

            sign=sign-word;
            result=sign/numlines;

            return result;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of for, while and do-while loops is checked              */
/*****************************************************************************/
int Edit::countLoops()
{
            int y,z,x;

            system(COUNTFOR);
```

```
            ifstream FIL("tempfile", ios::in);
            FIL>>y;
            FIL.close();
            system("rm tempfile");

            system(COUNTWHILE);

            ifstream FIL2("tempfile", ios::in);
            FIL2>>z;
            FIL2.close();
            system("rm tempfile");

            x=z+y;
            return x;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of magic numbers except 0 and 1 is checked               */
/*****************************************************************************/
int Edit::countMagic()
{
        int arraysize,m=0,i=0,j=0;

            system(COUNTNUMBERS1);
            system(COUNTNUMBERS2);
            system("rm tempfile");
            system(COUNTNUMBERS3);

            ifstream FIL("tempfile3", ios::in);
            FIL>>arraysize;
            FIL.close();
            system("rm tempfile3");

            char b[arraysize];

            ifstream FIL2("tempfile2", ios::in);
            while (FIL2.peek()!=EOF)
            {
                        FIL2.read((char*)&b[m],1);
                        m++;
            }
            FIL2.close();
            system("rm tempfile2");

            while(i<arraysize)
            {
                        if(b[i]<='9' && b[i]>='0')
                        {
                                    if(b[i]=='0' || b[i]=='1')
                                    {
                                                if(b[i+1]<='9' && b[i+1]>='0')
                                                            j++;
                                                while(b[++i]<='9' && b[++i]>='0')
                                                            i++;
                                    }
                                    else
                                    {
                                    j++;
                                                while(b[++i]<='9' && b[++i]>='0')
                                                            i++;
                                    }
                        }
                        else
                                    i++;
            }

            return j;
```

```
}


/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The metrics based on "readability" is evaluated                     */
/*****************************************************************************/
int Edit::evalRead(int line,int blank,int sign,int com)
{
            int readability=0;
            float qvote1,qvote2;

            qvote1=(float)blank/line;
            qvote2=(float)com/line;

            if(qvote1<=UPPER && qvote1>=LOWER)
                        readability++;
            if(qvote2<=UPPER && qvote1>=LOWER)
                        readability++;
            if(sign<=MAXSIGN)
                        readability++;

            return readability;

}


/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The metrics based on "reuseability" is evaluated                    */
/*****************************************************************************/
int Edit::evalReuse(int kl,int magi,int line)
{
            int reuseability=1;
            float qvote1,qvote2;
            int* vektor;
            int sum=0;

            for(int i=0;i<kl;i++)
            {
                        vektor=Met.getVektor(i+1);
                        for(int j=0;j<kl;j++)
                                    sum=sum+vektor[j];
            }

            qvote1=(float)sum/kl;
            qvote2=(float)magi/line;

            if(qvote1<=MAXCLASS)
                        reuseability++;
            if(qvote2<=MAXMAGIC)
                        reuseability++;

            return reuseability;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The metrics based on "complexity" is evaluated                      */
/*****************************************************************************/
int Edit::evalComp(int kl,int loop)
{
            int complex=0;
            float qvote1,qvote2,qvote3,qvote4;
            int* vektor;
            int sumvekt=0,sumvar=0,summet=0;

            for(int i=0;i<kl;i++)
```

```
            {
                        vektor=Met.getVektor(i+1);
                        for(int j=0;j<kl;j++)
                                    sumvekt=sumvekt+vektor[j];
                        sumvar=sumvar+Met.getVariabel(i+1);
                        summet=summet+Met.getMethods(i+1);
                        summet=summet+Met.getPublic(i+1);
                        summet=summet+Met.getPrivate(i+1);
                        summet=summet+Met.getProtected(i+1);
            }
            qvote1=(float)sumvekt/kl;
            qvote2=(float)sumvar/kl;
            qvote3=(float)summet/kl;
            qvote4=(float)loop/kl;

            if(qvote1<=MAXCLASS)
                        complex++;
            if(qvote2<=MAXVARIABELS && qvote4<=MAXLOOP)
                        complex++;
            if(qvote3<=MAXMET)
                        complex++;

            return complex;
}

/*****************************************************************************/
/* PRE:  para >=0 && para <4                                                 */
/* POST: The value of para is translated into text                           */
/*****************************************************************************/
char* Edit::translate(int para)
{
            if(para==0)
                        return ("BAD");
            else if(para==1)
                        return ("NOT SO GOOD");
            else if(para==2)
                        return ("GOOD");
            else if(para==3)
                        return ("VERY GOOD");
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The table over "classbounding" is printed on the screen             */
/*****************************************************************************/
void Edit::printTable(int kl)
{
            char* name;
            char q[ITOAL];
            int* vektor;
            int temp;

            view->insertLine("****** CLASSBOUNDING ******(lines calling
                                    columns)\n\n");
            int ins=TABLEWIDHT;
            for(int i=0;i<kl;i++)
            {
                        name = Met.getName(i+1);
                        view->insertAt(SPACELINE,3,ins);
                        view->insertAt(name,3,ins);
                        ins=ins+TABLEWIDHT;
            }

            ins=TABLEWIDHT;
            for(int i=0;i<kl;i++)
            {
                        name = Met.getName(i+1);
                        vektor=Met.getVektor(i+1);
```

```
                        view->insertLine(name);
                        for(int j=0;j<kl;j++)
                        {
                                    temp=vektor[j];
                                    itoa(q,temp);
                                    view->insertAt(SPACELINE,i+4,ins);
                                    view->insertAt(q,i+4,ins);
                                    ins=ins+TABLEWIDHT;

                        }
                        ins=TABLEWIDHT;
            }
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The evaluated metrics is printed on screen                          */
/*****************************************************************************/
void Edit::printMetrics(int line,int blank,int sign,int com,int kl,int loop,int
magi)
{
            int readability,reuseability,complex;
            char* answ1;
            char* answ2;
            char* answ3;
            char rdkomm[VIEWL],rukomm[VIEWL],cokomm[VIEWL];

            readability=evalRead(line,blank,sign,com);
            answ1=translate(readability);
            strcpy(rdkomm,"Readability: ");
            strcat(rdkomm,answ1);

            reuseability=evalReuse(kl,magi,line);
            answ2=translate(reuseability);
            strcpy(rukomm,"Reuseability: ");
            strcat(rukomm,answ2);

            complex=evalComp(kl,loop);
            answ3=translate(complex);
            strcpy(cokomm,"Complexity: ");
            strcat(cokomm,answ3);

            view->insertLine("\n****** METRICS ******");
            view->insertLine(rdkomm);
            view->insertLine(rukomm);
            view->insertLine(cokomm);
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The parameters for every single class is printed on screen          */
/*****************************************************************************/
void Edit::printSingle(int kl)
{
            char* name;
            char pc[VIEWL],pe[VIEWL],pd[VIEWL],vr[VIEWL],md[VIEWL];
            char a[ITOAL],b[ITOAL],c[ITOAL],d[ITOAL],e[ITOAL];

            view->insertLine("\n****** SINGLE CLASSES ******");

            for(int i=0;i<kl;i++)
            {
                        name = Met.getName(i+1);
                        strcpy(pc,"Number of public methods: ");
                        strcpy(pe,"Number of private methods: ");
                        strcpy(pd,"Number of protected methods: ");
                        strcpy(md,"Number of methods: ");
                        strcpy(vr,"Number of variables: ");
```

```
                        int pub=Met.getPublic(i+1);
                        int priv=Met.getPrivate(i+1);
                        int prot=Met.getProtected(i+1);
                        int met=Met.getMethods(i+1);
                        int var=Met.getVariabel(i+1);

                        itoa(a,pub);
                        itoa(b,priv);
                        itoa(c,met);
                        itoa(d,var);
                        itoa(e,prot);

                        strcat(pc,a);
                        strcat(pe,b);
                        strcat(pd,e);
                        strcat(md,c);
                        strcat(vr,d);
                        strcat(vr,"\n");

                        view->insertLine(name);
                        view->insertLine(pc);
                        view->insertLine(pe);
                        view->insertLine(pd);
                        view->insertLine(md);
                        view->insertLine(vr);
            }
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The parameters checked for all code is printed on scrren            */
/*****************************************************************************/
void Edit::printAll(int line,int blank,int sign,int com,int kl,int loop,int magi)
{
        char a[ITOAL],b[ITOAL],c[ITOAL],d[ITOAL],e[ITOAL],f[ITOAL],g[ITOAL];
        char lines[VIEWL],blanks[VIEWL],signs[VIEWL];
        char comm[VIEWL],rund[VIEWL],numb[VIEWL],cla[VIEWL];


            itoa(a,line);
            itoa(b,blank);
            itoa(c,com);
            itoa(d,kl);
            itoa(e,loop);
            itoa(f,magi);
            itoa(g,sign);

            strcpy(lines,"Number of lines (without blanks): ");
            strcpy(blanks,"Number of empty lines: ");
            strcpy(signs,"Number of signs per line: ");
            strcpy(comm,"Number of lines with comments: ");
            strcpy(rund,"Number of loops: ");
            strcpy(numb,"Magic numbers: ");
            strcpy(cla,"Number of classes: ");

            strcat(lines,a);
            strcat(blanks,b);
            strcat(signs,g);
            strcat(comm,c);
            strcat(cla,d);
            strcat(rund,e);
            strcat(numb,f);

            view->insertLine("\n****** ALL CLASSES ******");
            view->insertLine(lines);
            view->insertLine(blanks);
            view->insertLine(signs);
```

```
            view->insertLine(comm);
            view->insertLine(cla);
            view->insertLine(rund);
            view->insertLine(numb);
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: A summary of parameters to all classes is printed on screen         */
/*****************************************************************************/
void Edit::printSum(int kl)
{
            char a[ITOAL],b[ITOAL],c[ITOAL],d[ITOAL],e[ITOAL];
            char totpub[VIEWL],totpriv[VIEWL],totprot[VIEWL];
            char totmet[VIEWL],totvar[VIEWL],name[VIEWL];
            int pubsum=0,privsum=0,protsum=0,metsum=0,varsum=0;


            for(int i=0;i<kl;i++)
            {
                        int pub=Met.getPublic(i+1);
                        int priv=Met.getPrivate(i+1);
                        int prot=Met.getProtected(i+1);
                        int met=Met.getMethods(i+1);
                        int var=Met.getVariabel(i+1);
                        pubsum=pubsum+pub;
                        privsum=privsum+priv;
                        protsum=protsum+prot;
                        metsum=metsum+met;
                        varsum=varsum+var;
            }

            itoa(a,pubsum);
            itoa(b,privsum);
            itoa(c,protsum);
            itoa(d,metsum);
            itoa(e,varsum);

            strcpy(name,"Summary of all classes");
            strcpy(totpub,"Number of public methods: ");
            strcpy(totpriv,"Number of private methods: ");
            strcpy(totprot,"Number of protected methods: ");
            strcpy(totmet,"Number of methods: ");
            strcpy(totvar,"Number of variables: ");
            strcat(totpub,a);
            strcat(totpriv,b);
            strcat(totprot,c);
            strcat(totmet,d);
            strcat(totvar,e);

            view->insertLine(name);
            view->insertLine(totpub);
            view->insertLine(totpriv);
            view->insertLine(totprot);
            view->insertLine(totmet);
            view->insertLine(totvar);
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The result of choice "print1" is executed                           */
/*****************************************************************************/
void Edit::print1(int line,int blank,int sign,int com,int kl,int loop,int magi)
{

            printTable(kl);
            printMetrics(line,blank,sign,com,kl,loop,magi);
            printSingle(kl);
```

```
            printSum(kl);
            printAll(line,blank,sign,com,kl,loop,magi);
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The result of choice "print2" is executed                           */
/*****************************************************************************/
void Edit::print2(int line,int blank,int sign,int com,int kl,int loop,int magi)
{
            printTable(kl);
            printMetrics(line,blank,sign,com,kl,loop,magi);
            printAll(line,blank,sign,com,kl,loop,magi);
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of public methods in a single class is checked           */
/*****************************************************************************/
int Edit::countPublic()

{
            int num;

            system(COUNTPUBLIC);

            ifstream FIL("tempfile", ios::in);
            FIL>>num;
            FIL.close();
            system("rm tempfile");

            return num;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of private methods in a single class is checked          */
/*****************************************************************************/
int Edit::countPrivate()
{
            int num;

            system(COUNTPRIVAT);

            ifstream FIL("tempfile", ios::in);
            FIL>>num;
            FIL.close();
            system("rm tempfile");

            return num;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of protected methods in a single class is checked        */
/*****************************************************************************/
int Edit::countProtected()
{
            int num;

            system(COUNTPROT);

            ifstream FIL("tempfile", ios::in);
            FIL>>num;
            FIL.close();
            system("rm tempfile");

            return num;
```

```
}


/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of unnamned methods in a single class is checked         */
/*****************************************************************************/
int Edit::countMethods()
{
            int num;

            system(COUNTMETHODS1);
            system(COUNTMETHODS2);
            system(COUNTMETHODS3);
            system(COUNTMETHODS4);
            system(COUNTMETHODS5);
            system(COUNTMETHODS6);
            system(COUNTMETHODS7);
            system(COUNTMETHODS8);
            system(COUNTMETHODS9);
            system(COUNTMETHODS10);
            system(COUNTMETHODS11);
            ifstream FIL("tempfilerun", ios::in);
            FIL>>num;
            FIL.close();
            system("rm tempfile1");
            system("rm tempfile2");
            system("rm tempfile3");
            system("rm tempfile4");
            system("rm tempfile5");
            system("rm tempfile6");
            system("rm tempfile7");
            system("rm tempfile8");
            system("rm tempfile9");
            system("rm tempfile10");
            system("rm tempfilerun");


            return num;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of variables in the code is checked                      */
/*****************************************************************************/
int Edit::countVariabel()
{
            int arraysize,numbfor,m=0,i=0,j=0;
            system(COUNTVARIABLE1);
            system(COUNTVARIABLE2);
            system(COUNTVARIABLE3);
            system(COUNTVARIABLE4);
            system(COUNTVARIABLE5);

            ifstream FIL3("tempfile5", ios::in);
            FIL3>>numbfor;
            FIL3.close();
            system("rm tempfile5");

            ifstream FIL("tempfile3", ios::in);
            FIL>>arraysize;
            FIL.close();
            system("rm tempfile3");
            system("rm tempfile2");
            system("rm tempfile");

             char b[arraysize];
```

```
            ifstream FIL2("tempfile4", ios::in);
            while (FIL2.peek()!=EOF)
            {
                        FIL2.read((char*)&b[m],1);
                        m++;
            }
            FIL2.close();
            system("rm tempfile4");

            while(i<arraysize)
            {
                        if(b[i]==',' || b[i]==';')
                        {
                                    j++;
                        }
                        i++;
            }
            j=j+numbfor;
            return j;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The number of calls to other classes is checked and stored in the list  */
/*****************************************************************************/
void Edit::classCall(int vektor[], int numb)
{

        int tempnum=0,i=0,count=0,counter=0,result=0,sum=0;
        char name[VIEWL],object[VIEWL];
        char comm[MAXSTR],comm2[MAXSTR],comm3[MAXSTR],comm4[MAXSTR],comm5[MAXSTR];

            ifstream FIL("nameonfiles", ios::in);
            while(FIL.peek()!=EOF && count<numb)
            {
                        FIL>>name;
                        strcpy(comm,"grep '");
                        strcat(comm,name);
                        strcat(comm,COUNTCLASSCALL1);
                        system(comm);

                        strcpy(comm4,COUNTCLASSCALL2);
                        strcat(comm4,name);
                        strcat(comm4,COUNTCLASSCALL3);
                        system(comm4);
                        system("rm tempfile");

                        system(COUNTCLASSCALL4);
                        system("rm objectfile");
                        system(COUNTCLASSCALL5);
                        system("rm commfile");

                        system(COUNTCLASSCALL6);
                        system("rm semikfile");

                        strcpy(comm3,"grep '");
                        strcat(comm3,name);
                        strcat(comm3,COUNTCLASSCALL7);
                        system(comm3);

                        strcpy(comm2,"sed -e 's/");
                        strcat(comm2,name);
                        strcat(comm2,COUNTCLASSCALL8);
                        system(comm2);
                        system("rm tempfile2");
                        system(COUNTCLASSCALL9);
                        system("rm objectfile2");
                        system(COUNTCLASSCALL10);
```

```
                        system("rm objectfile3");

                        system(COUNTCLASSCALL11);
                        system("rm newobjectfile2");

                        system(COUNTCLASSCALL12);

                         ifstream FIL3("numfiles", ios::in);
                        FIL3>>tempnum;
                        FIL3.close();
                        system("rm numfiles");

                        ifstream FIL2("searchfile", ios::in);
                        while(counter<tempnum)
                        {
                                    FIL2>>object;
                                    strcpy(comm5,"grep -c '");
                                    strcat(comm5,object);
                                    strcat(comm5,COUNTCLASSCALL13);
                                    system(comm5);
                                    ifstream FIL4("hitfile", ios::in);
                                    FIL4>>result;
                                    sum=result+sum;
                                    system("rm hitfile");
                                    counter++;
                        }
                        FIL2.close();
                        system("rm searchfile");

                        vektor[i]=sum;
                        i++;
                        count++;
                        tempnum=0;
                        counter=0;
                        sum=0;
            }
            FIL.close();
            int q=Met.listlength();
            vektor[q]=0;
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The parameters of the class is written to the list                  */
/*****************************************************************************/
void Edit::singleClass(const char *linkname, int numb)
{
            int antpub,antpriv,antmet,antvar,antprot;
            char name[MAXSTR];
            int* vektor=new int [numb];

            strcat(name,linkname);

            antpub=countPublic();
            antpriv=countPrivate();
            antprot=countProtected();
            antmet=countMethods();
            antvar=countVariabel();
            classCall(vektor, numb);

            Met.listinsert(name,antpub,antpriv,antprot,antmet,antvar,Met.listlength
(),vektor);
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The code of the class is written to a large file, "entirecopy", and a   */
/*       temporary, "tempcopy"                                               */
```

```
/*****************************************************************************/
void Edit::load(const char *filename,const char *linkname,int numb)
{
            char command[MAXSTR],command2[MAXSTR];

            strtok(linkname,".");

            strcpy(command, "cat ");
            strcat(command, filename);
            strcat(command, ">>entirecopy");
            system(command);

            strcpy(command2, "cat ");
            strcat(command2, filename);
            strcat(command2, ">>tempcopy");
            system(command2);

            singleClass(linkname, numb);
            system("rm tempcopy");
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: The KDE menubar is created                                          */
/*****************************************************************************/
void Edit::initMenuBar()
{
  QPopupMenu *file = new QPopupMenu;

  file->insertItem(klocale->translate("Select your directory"), ID_OPEN );
  file->insertItem(klocale->translate("Save as..."), ID_SAVE );
  file->insertSeparator();
  file->insertItem(klocale->translate("Exit....."), ID_EXIT);

  QPopupMenu *help = new QPopupMenu;
  help->insertItem( klocale->translate("About..."), ID_ABOUT);

  menu = new KMenuBar( this );
  CHECK_PTR( menu );
  menu->insertItem( klocale->translate("&File"), file );
  menu->insertSeparator();
  menu->insertItem( klocale->translate("&Help"), help );
  menu->show();
  setMenu(menu);

  connect (file, SIGNAL (activated (int)), SLOT (commandCallback
(int)));
  connect (help, SIGNAL (activated (int)), SLOT (commandCallback
(int)));
}

/*****************************************************************************/
/* PRE: "id" is a defined ID_X                                               */
/* POST: The Command is made enabled for the user                            */
/*****************************************************************************/
void Edit::enableCommand(int id)
{
  menu->setItemEnabled(id, true);
}

/*****************************************************************************/
/* PRE: "id" is a defined ID_X                                               */
/* POST: The Command is made disabled for the user                           */
/*****************************************************************************/
void Edit::disableCommand(int id)
{
  menu->setItemEnabled(id, false);
}
```

```
/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: True                                                                */
/*****************************************************************************/
void usage()
{
  printf("edit " EDIT_VERSION_STRING "(c) Richard J. Moore 1997\n");
  printf( "Released under GPL see the file LICENSE for details\n");
}

/*****************************************************************************/
/* PRE: True                                                                 */
/* POST: True                                                                */
/*****************************************************************************/
int main(int argc, char **argv)
{
  KApplication *app;
  Edit *toplevel;

  if ((argc == 2) && (strcmp(argv[1], "-h") == 0))
  {
    usage();
    exit(0);
  }
  app= new KApplication(argc,argv,"edit");
  toplevel= new Edit();
  app->setMainWidget(toplevel);
  toplevel->show();
  app->exec();
}
```

8.1 A.3: list.cpp

```
#include<stddef.h>

template <class T>
struct listnode;

template <class T>
class list
{
 public:
   list();
   ~list();
   void listinsert(char name[],T pub,T priv,T prot,T met,T var,int pos, int
                   vektor[]);
   int listlength();
   void listdelete(int pos);
   T& getPublic(int n);
   T& getPrivate(int n);
   T& getProtected(int n);
   T& getMethods(int n);
   T& getVariabel(int n);
   char* getName(int n);
   int* getVektor(int n);
 private:
   int numb;
   listnode<T>* head;
   listnode<T>* pointfinder(int N);
};

template <class T>
struct listnode
{
            char *sokvag;
            int* vect;
            T publ;
            T priv;
            T protect;
            T method;
            T variabel;
            listnode<T>* next;
            listnode();
};

/*************************************************************************/
/* PRE: True                                                             */
/* POST: A listnode is created                                           */
/*************************************************************************/
template <class T>
listnode<T>::listnode(): next(NULL)
{
}

/*************************************************************************/
/* PRE: True                                                             */
/* POST: A list is created                                               */
/*************************************************************************/
template <class T>
list<T>::list():head(NULL),numb(0)
{
}

/*************************************************************************/
/* PRE: There is a list                                                  */
/* POST: The list is terminated                                          */
```

```
/*************************************************************************/
template <class T>
list<T>::~list(){}

/*************************************************************************/
/* PRE: There is a list                                                  */
/* POST: The number of element is returned                               */
/*************************************************************************/
template <class T>
int list<T>::listlength()
{
            return numb;
}

/*************************************************************************/
/* PRE: True                                                             */
/* POST: The parameters is inserted in the list and numb++              */
/*************************************************************************/
template <class T>
void list<T>::listinsert(char name[],T pub,T priv,T prot,T met,T var,int pos,int
vektor[])
{

            listnode<T>* newPtr = new listnode<T>;
            newPtr->sokvag=new char[strlen(name)];
            strcpy(newPtr->sokvag, name);

            newPtr->publ = pub;
            newPtr->priv = priv;
            newPtr->protect = prot;
            newPtr->method = met;
            newPtr->variabel = var;
            newPtr->vect = vektor;

            if(pos==0 || !listlength())
            {
                        newPtr->next = head;
                        head = newPtr;
            }
            else
            {
                        listnode<T>* cur = pointfinder(pos);
                        newPtr->next = cur->next;
                        cur->next = newPtr;
            }
        numb = listlength() + 1;
}

/*************************************************************************/
/* PRE: At least one element in the list                                 */
/* POST: The element is deleted and numb--                               */
/*************************************************************************/
template <class T>
void list<T>::listdelete(int pos)

{
            listnode<T>* cur;

            if(pos==1)
            {
                        cur = head;
                        head = head->next;
            }
            else
            {
                        listnode<T>*  prev = pointfinder(pos-1);
                        cur = prev->next;
                        prev->next = cur->next;
```

```
            }
            cur->next = NULL;
            delete cur;
            cur = NULL;
            numb--;
}

/*************************************************************************/
/* PRE: At least one element in the list                                 */
/* POST: The parameter "public" is returned                              */
/*************************************************************************/
template <class T>
T& list<T>::getPublic(int n)
{
            listnode<T>* cur = pointfinder(n);
            return cur->publ;
}

/*************************************************************************/
/* PRE: At least one element in the list                                 */
/* POST: The parameter "private" is returned                             */
/*************************************************************************/
template <class T>
T& list<T>::getPrivate(int n)
{
            listnode<T>* cur = pointfinder(n);
            return cur->priv;
}

/*************************************************************************/
/* PRE: At least one element in the list                                 */
/* POST: The parameter "protected" is returned                           */
/*************************************************************************/
template <class T>
T& list<T>::getProtected(int n)
{
            listnode<T>* cur = pointfinder(n);
            return cur->protect;
}

/*************************************************************************/
/* PRE: At least one element in the list                                 */
/* POST: The parameter "methods" is returned                             */
/*************************************************************************/
template <class T>
T& list<T>::getMethods(int n)
{
            listnode<T>* cur = pointfinder(n);
            return cur->method;
}

/*************************************************************************/
/* PRE: At least one element in the list                                 */
/* POST: The parameter "variabels" is returned                           */
/*************************************************************************/
template <class T>
T& list<T>::getVariabel(int n)
{
            listnode<T>* cur = pointfinder(n);
            return cur->variabel;
}

/*************************************************************************/
/* PRE: At least one element in the list                                 */
/* POST: The name of the class is returned                               */
/*************************************************************************/
template <class T>
char* list<T>::getName(int n)
```

```
{
            listnode<T>* cur = pointfinder(n);
            return cur->sokvag;}

/*************************************************************************/
/* PRE: At least one element in the list                                 */
/* POST: The vector containing numbers of calls to other classes is returned*/
/*************************************************************************/
template <class T>
int* list<T>::getVektor(int n)
{
            listnode<T>* cur = pointfinder(n);
            return cur->vect;
}

/*************************************************************************/
/* PRE: At least one element in the list                                 */
/* POST: Position or 0 is returned                                       */
/*************************************************************************/
template <class T>
listnode<T>* list<T>::pointfinder(int N)
{
            listnode<T>*  cur;

            cur = head;
            for(int i = 1;i<N;++i)
                        cur = cur->next;
            return cur;
}
```

8.2 A.4: scriptfile

```
/char .*;/
/String .*;/
/boolean .*;/
/byte .*;/
/short .*;/
/int .*;/
/long .*;/
/float .*;/
/double .*;/
```

8.3 A.5: makefile

```
.PHONY: all clean

KDEDIR= /usr
QTDIR=  /usr
MOC=    /usr/bin/moc
CXX=    g++
INCDIR= -I$(KDEDIR)/include -I$(QTDIR)/include/qt
CXXFLAGS=       -g -Wall $(INCDIR) -DEDIT_VERSION_STRING="\"0.1\""
LIBDIR= -L$(KDEDIR)/lib -L$(QTDIR)/lib -L/usr/X11R6/lib
LIBS=   -lkdeui -lkdecore -lqt -lXext -lX11


OBJ=    edit.o

META=   edit.moc

all:    edit

clean:
           rm -f *.o
           rm -f *.moc
           rm -f *~
           rm -f nameonfiles
           rm -f entirecopy
           rm -f tempcopy
           rm -f core
           rm -f filedir*
           rm -f tempfile*

%.o:    %.cpp
           $(CXX) -c $(CXXFLAGS) -o $@ $<

%.moc:  %.h
           $(MOC) -o $@ $<

edit:   $(META) $(OBJ)
           g++ -o edit $(OBJ) $(LIBDIR) $(LIBS)
```

Bilaga B: Print1 som utskrift

```
****** CLASSBOUNDING ******

                        ListBarCanvas           SortDemo
ListBarCanvas           0                       0
SortDemo                12                      0

****** METRICS ******
Readability: GOOD
Reuseability: NOT SO GOOD
Complexity: BAD

****** SINGLE CLASSES ******
ListBarCanvas
Number of public methods: 16
Number of private methods: 0
Number of protected methods: 1
Number of methods: 26
Number of variables: 3

SortDemo
Number of public methods: 7
Number of private methods: 0
Number of protected methods: 0
Number of methods: 8
Number of variables: 0

Summary of all classes
Number of public methods: 23
Number of private methods: 0
Number of protected methods: 3
Number of methods: 1
Number of variables: 34

****** ALL CLASSES ******
Number of lines (without blanks): 367
Number of empty lines: 88
Number of signs per line: 28
Number of lines with comments: 11
Number of classes: 2
Number of loops: 12
Magic numbers: 20
```

Utskriften är bara en del av det program som testats i kapitel fem, så själva resultaten är inte intressanta i ovanstående utskrift.

I detta utskriftsval visas överst en tabell som visar klassanropen, klasserna radvis anropar klasserna kolumnvis. I tabell ovan går alltså att utläsa att klassen *SortDemo* anropar klassen

ListBarCanvas 12 gånger. Därefter följer en redovisning av de mått som utvärderats. Varje klass redovisas sedan var för sig inklusive en sammanställning av klasserna. Även den del som ”beräknar” parametrar på alla klasser presenteras.

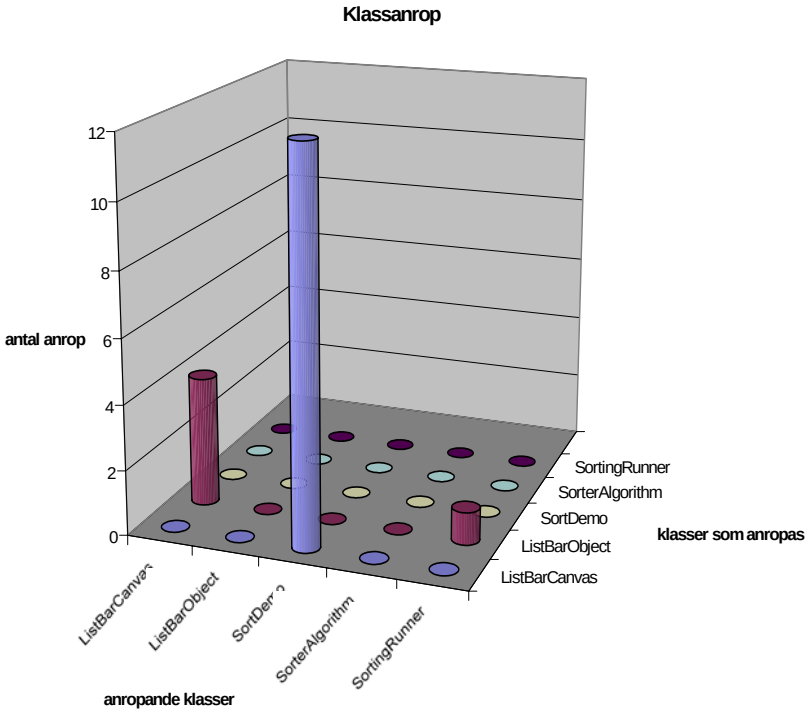
Bilaga C: Print2 som utskrift

```
****** CLASSBOUNDING ******

                        ListBarCanvas           SortDemo
ListBarCanvas           0                       0
SortDemo                12                      0

****** METRICS ******
Readability: GOOD
Reuseability: NOT SO GOOD
Complexity: BAD

****** ALL CLASSES ******
Number of lines (without blanks): 367
Number of empty lines: 88
Number of signs per line: 28
Number of lines with comments: 11
Number of classes: 2
Number of loops: 12
Magic numbers: 20
Summary of all classes
Number of public methods: 23
Number of private methods: 0
Number of protected methods: 3
Number of methods: 1
Number of variables: 34
```

Utskriften är bara en del av det program som testats i kapitel fem, så själva resultaten är inte intressanta i ovanstående utskrift.

Överst i utskriften visas en tabell över hur klasserna anropar varandra. Klasserna radvis är de klasser som anropar och klasserna kolumnvis är de klasser som blir anropade. Efter detta följer en redovisning av de mått som utvärderats med hjälp av parametrarna. Sist redovisas en samman-ställning av parametrarna över samtliga klasser.

Bilaga D: Diagram över klassanrop



Diagrammet ovan visar en illustration över klassanropen. Diagrammet är skapat i *Microsoft Excel* med tabellen tagen från den kod som testats i kapitel fem. Ur diagrammet kan man till exempel avläsa att klassen *SortDemo* anropar klassen *ListBarCanvas* 12 gånger.