TECHNICAL REPORT NO. 06

Indledende gennemgang af KMS' transformationsbibliotek.

Thomas Knudsen

DANISH MINISTRY
OF THE ENVIRONMENT

National Survey
and Cadastre

Indhold

Thomas Knudsen

Indledende gennemgang af KMS' transformationsbibliotek.

National Survey and Cadastre—Denmark, technical report series number 06
ISBN 978-87-92107-28-2
Technical Report
Published 2009-12
This report is available from http://www.kms.dk

1 Indledning

1.1 Opgaven

I sensommeren 2009 havde jeg en interessant opgave: jeg skulle danne mig et overblik over koden til Kort & Matrikelstyrelsens transformationsbibliotek, og (især) overveje mulige veje for den fremtidige vedligeholdelse af koden. Da jeg havde fået al koden leveret fra Karsten Engsager (som de sidste ca. 15 år har været transformationsbibliotekets ankermand) sad jeg hen over et par sensommerdage og kiggede materialet igennem ved brug af almindeligt tilgængeligt open source værktøj, på en helt almindelig bærbar pc, med det GNU/Linuxbaserede styresystem Ubuntu 9.04.

De følgende sider giver en oversigt over hvad jeg foretog mig. Teksten er taget direkte (eller kun let redigeret) fra min logfil over processen, hvor jeg noterede de kaldte kommandoer, deres output, og mine egne overvejelser og reflektioner over proces og resultater. Der er med andre ord tale om en personlig, yderst subjektiv, næsten *stream of consciousness*-agtig stil, som ikke er sædvanlig for en teknisk rapport.

Når jeg alligevel har valgt at benytte den subjektive stil er det fordi adskillige kolleger som gennemlæste den uredigerede logfil, udtrykte at den på en både klar og lettilgængelig måde, ledte læseren gennem et scenarie i simpel kodeeksploration, som de havde haft stor nytte af.

Så håbet om at logfilen (i en let typografisk tilpasning) kan være til endnu mere nytte for andre, er vel hovedgrunden til at rapporten har fået den personlige form den har.

1.2 Kodebasen

KMS' transformationsbibliotek har en lang historie: koden er skrevet over en periode på 30-40 år af flere forskellige programmører (dog med Knud Poder og Karsten Engsager som de faste ankermænd). Koden er velafprøvet og robust: den har i mange år været rygraden i KMS' bearbejdning af koordinatdata. Den er armeret med al den panser og plade der kommer af at have rejst gennem tre generationer af programmeringssprog (Fortran, Algol, C), endnu flere generationer af operativsystemer (BOSS, Unix SysV, Vax VMS, MSDOS, MSWindows, Solaris, Linux), hardwareplatforme (GIER, RC-4000, RC-8000, Digital Vax, Sun SPARC, Intel 386, etc) og tilhørende compilersystemer.

Men koden har aldrig været sendt en tur gennem den vridemaskine af teamunderstøttende hjælpesystemer som gennem de sidste ca. 25 år, er drevet frem inden for open source udviklingskulturen.

Vi står imidlertid over for et generationsskifte mht. vedligeholdelse af koden. Derfor giver det god mening dels at undersøge koden nærmere, dels at overveje hvordan vi kan reducere den kodemængde der skal vedligeholdes af KMS og vores direkte samarbejdspartnere: hvis der er dele af koden der kan indarbejdes i eksisterende open source projektions- og transformationsbiblioteker (fx proj og CSmap) kan vi drage nytte af den (store) bruger- og udviklerkreds, og den gode infrastruktur, disse biblioteker råder over – og dermed på en gang reducere mængden af kode vi selv skal vedligeholde, og øge kvaliteten af den udskilte kode.

1.3 Rapporten

Denne rapport prætenderer som antydet ovenfor, ikke at være objektiv: Dels er den baseret på et personligt arbejdspapir, dels ligger der en klar personlig holdning bag arbejdet – nemlig den at kode bliver bedre af at blive brugt (flere øjne på koden og flere use cases reducerer restfejlmængden).

Så for at få reduceret fejlmængderne og øget koderobustheden er det, i det omfang der er politisk opbakning til det, fagligt interessant for KMS at få lukket op for koden bag nogle af vore systemer: dermed bliver vores indsats mere værd for samfundet, og vores kode får øget kvalitet og bedre mulighed for at være normsættende. Men når jeg i det følgende ofte argumenterer for maksimal åbenhed i kildekode, er der altså (i rapportens ånd) tale om en personligt/faglig vurdering, som ikke nødvendigvis afspejler hverken noget praktisk realiserbart, eller Kort & Matrikelstyrelsens officielle holdning.

2 TRLIB – indledende gennemgang af koden

Nedenstående er en rent teknisk gennemgang af kode modtaget fra Karsten Engsager.

Kommentarer og overvejelser er sat i et sans-serif skriftsnit. Gengivelser fra computerskærmen er sat i et `monospatieret` skriftsnit.

`Monospatierede` linjer der indledes med strengtegn (`$`), og linjeblokke afgrænset af "`$$$`", er kommandoer afgivet fra kommandoprompten. Øvrige `monospatierede` linjer er output fra kommandoer.

NB: output er ofte gengivet i redigeret form for at fokusere på det væsentlige.

HISTORIK:

2009-08-11 (Thomas Knudsen) Kapitel 1-4

2009-08-12 (Thomas Knudsen) Kapitel 5: doxygen Kapitel 6: koden

2009-08-13 (Thomas Knudsen) Kapitel 6: koden (fortsat) Kapitel 7: udfordringen

2009-08-13 (Thomas Knudsen) Enkelte rettelser og præciseringer

2009-12-14 (Thomas Knudsen) Voldsom reformatering til pdfLaTeX/KMS teknisk rapport. Enkelte rettelser og tilføjelser.

2.1 Overblik

Strukturen – hvordan er den?

```
$ pwd
/home/tk/Documents/trlib
$ ls
include/ src/ trlib.pad
```

Hvor mange kode- og headerfiler har vi

```
$ ls include | wc -l
215
$ ls src | wc -l
145
```

. . . og hvor mange kodelinjer?

```
$ cat src/*.c|wc -l
38481
$ cat include/*.h|wc -l
12696
```

Vi står altså med et projekt, der indeholder ca. 350 filer, og ca. 50000 linjer kode (50 kloc, hvis man vil lyde popsmart).

Mange af headerfilerne er i det væsentlige dokumentation af kodefilerne (eksempel: filen bshlm1.h dokumenterer funktionen bshlm1, hvis kode ligger i filen bshlm1.c).

Denne type headerfiler kan integreres med kodefilerne, hvis vi går over til at bruge doxygen (eller lignende) som dokumentationsværktøj.

Dermed kommer vi et stykke under 300 filer, og får mere sikkerhed for at dokumentation og kode er i overensstemmelse.

2.2 Åbenlyse fejl og mangler

Dele af koden er ikke i daglig brug, og uhensigtsmæssigheder fra forne tiders kodepraksis stikker ind imellem frem.

Den slags kan analyseres med en statisk kodechecker af linttypen, fx "splint". Men før vi kører det tunge skyts i stilling kan vi lige bruge GNU C compilerens fejlfindingsudstyr til at få et første overblik:

```
$$$
cd src
```

```
gcc -c -I../include *.c 2>gccerr.out
gcc -c -I../include -Wall *.c 2>gccerr.wall
gcc -c -I../include -Wall -pedantic *.c 2>gccerr.pedantic
cd ..
$$$

$ grep -c error src/gccerr.*
src/gccerr.out:15
src/gccerr.pedantic:15
src/gccerr.wall:15

$ grep -c warning src/gccerr.*
src/gccerr.out:50
src/gccerr.pedantic:142
src/gccerr.wall:80
```

OK – op til 15 deciderede fejl (men se nedenfor: så mange er der ikke reelt!), og et sted mellem 50 og 142 warnings afhængig af hvor nærtagende man beder compileren om at være.

De fleste warnings er af formen

```
warning: format '%4ld' expects type 'long int', but argument 5 has type 'unsigned int'
```

De er altså ikke umiddelbart alvorlige, men lette at rette.

Lad os kigge lidt mere på Fejlene:

```
$ grep error src/gccerr.out

fgetshpprj.c:92: error: 'ptrdiff_t' undeclared (first use in this function)
fgetshpprj.c:92: error: (Each undeclared identifier is reported only once
fgetshpprj.c:92: error: for each function it appears in.)
fgetshpprj.c:92: error: expected ')' before 'p_txt1'
fgetstec.c:226: error: invalid storage class for function 'fpr_stec_hd_1'
fgetstec.c:626: error: conflicting types for 'fpr_stec_hd_1'
fgetstec.c:405: error: previous implicit declaration of 'fpr_stec_hd_1' was here
sgetst.c:213: error: 'ptrdiff_t' undeclared (first use in this function)
sgetst.c:213: error: (Each undeclared identifier is reported only once
sgetst.c:213: error: for each function it appears in.)
sgetst.c:213: error: expected ')' before 'p'
sgetst.c:570: error: expected ')' before 'st_ad'
sgetst.c:571: error: expected ')' before 'p'
sgetst.c:592: error: expected ')' before 'ps'
sgetst.c:602: error: expected ')' before 'p'
```

Umiddelbart ser det ud til at være en manglende "`#include <stddef.h>`" [1] i 3 filer, så definitionen af `ptrdiff_t` kommer til at mangle. Resten af fejlene er formodentlig afledt af dette. Det er trivielt at rette.

2.3 Indføring af distribueret versionsstyring

Før jeg tager fat på den lille fejlretning, giver det god mening af få noget versionsstyring indført. Og da koden nok fremover skal vedligeholdes af en *gruppe* af mennesker, så skal det være distribueret versionsstyring (DVCS).

For tiden står kampen på DVCSmarkedet, mellem Bazaar (bzr) og Git (git). Hver har sine fordele. Jeg vælger Bazaar - primært pga. det hjælpsomme udvikler og brugercommunity (se fx Emma Jane Hogbins open week netcast [2])

```
$$$
bzr whoami "Thomas Knudsen"
bzr init
bzr add
bzr commit -m "Initial import of code from Karsten Engsager"
$$$
```

Det var det – så er trlib under distribueret versionsstyring!

2.4 Første rettelser

fgetshpprj.c, sgetst.c og fgetstec.c manglede "`#include <stddef.h>`". Efter at have indført dem tester vi:

```
$$$
cd src
gcc -c -I../include *.c 2>gccerr.out
cd ..
grep error src/gccerr.out
$$$

fgetstec.c:227: error: invalid storage class for function 'fpr_stec_hd_1'
fgetstec.c:627: error: conflicting types for 'fpr_stec_hd_1'
fgetstec.c:406: error: previous implicit declaration of 'fpr_stec_hd_1' was here
```

OK – det hjalp, men nu springer en ny type fejl i øjnene. Det ser ud som endnu en manglende headerfil. Lad os finde `fpr_stec_hd_1`

```
$ grep -lc fpr_stec_hd_1 src/*.c include/*.h
src/fgetstec.c
```

aha – `fpr_stec_hd_1` findes altså kun i filen fgetstec.c

```
$ grep -n fpr_stec_hd_1 src/fgetstec.c

227:static short fpr_stec_hd_1();
406: alarm = fpr_stec_hd_1(stdout,
416: alarm = fpr_stec_hd_1(stdout,
432: alarm = fpr_stec_hd_1(stdout,
438: alarm = fpr_stec_hd_1(stdout,
457: alarm = fpr_stec_hd_1(stdout,
467: alarm = fpr_stec_hd_1(stdout,
477: alarm = fpr_stec_hd_1(stdout,
483: alarm = fpr_stec_hd_1(stdout,
506: (void) fpr_stec_hd_1(stdout,
512: (void) fpr_stec_hd_1(stdout,
519: (void) fpr_stec_hd_1(stdout,
531: (void) fpr_stec_hd_1(stdout,
595: (void) fpr_stec_hd_1(stdout,
605: (void) fpr_stec_hd_1(stdout,
611: alarm = fpr_stec_hd_1(stdout, "tabelfejl", stec_hd);
627:static short fpr_stec_hd_1(
```

AHA! – en uvane fra de glade pre-ansi-C-dage: I linje 227 forwarddeklareres funktionen med tom parameterliste (dvs. void i ansi/iso-C termer), men i linje 627 defineres funktionen så med en kilometerlang parameterliste. Dette er med andre ord en ca. 20 år gammel fejl. Fin reminder om at checke at testkode faktisk dækker koden (gcov is your friend!).

Jeg sletter linje 227, retter til med en fuld prototype i toppen af filen, og rekompilerer!

```
$$$
cd src
gcc -c -I../include *.c 2>gccerr.out
cd ..
grep error src/gccerr.out
$$$
```

Yep – så compiler den fejlfrit. Om koden så fungerer er en helt anden sag. Det kan vi tage fat på senere!

Der er dog en warning tilbage, som er bekymrende:

```
set_grs.c:583:21: warning: multi-character character constant
```

Kodelinjen, den refererer til er denne:

```
if (qr != 'EOF') qr = '"';
```

Hvor `qr` er en returværdi fra `fgetc`. Der skulle naturligvis have stået `EOF` og ikke `'EOF'`. Den retter vi lige (og tester).

```
$ grep -c warning src/gccerr.out
33
$ grep warning src/gccerr.out|grep -vc format
0
```

OK – nu er der 33 warnings tilbage. De er alle simple format warnings. Situationen bliver dog værre når man bruger options Wall og pedantic, men vi må udrydde de vorter hen ad vejen – lige nu handler det om at få koden til at kompilere forholdsvist sikkert, og at få lavet en kodebase der dækker alle platforme. Første skridt er at committe ændringerne:

```
$ bzr commit -m "Corrected evident, compiler detectable, errors"

Committing to: /home/tk/Documents/trlib/
modified trlib.pad
modified include/geo_dat.h
modified src/fgetshpprj.c
modified src/fgetstec.c
modified src/gccerr.out
modified src/set_grs.c
modified src/sgetst.c
Committed revision 2.
```

2.5 Første version af linkbare biblioteker

Så bygger vi biblioteker - både shared og static (se [3,4] for de intrikate detaljer...)

```
$$$
cd src
gcc -I../include -O2 -c *.c
ar rcs libtr.a *.o
gcc -I../include -O2 -c -fPIC *.c
gcc -I../include -O2 -shared -Wl,-soname,libtr.so.1 -o libtr.so.1.0.1 *.o
cd ..
$$$

$ ls -l src/lib*
-rw-r--r-- 1 tk tk 654418 2009-08-11 15:00 src/libtr.a
-rwxr-xr-x 1 tk tk 472136 2009-08-11 15:01 src/libtr.so.1.0.1
```

Det så ud til at gå fint, så nu prøver vi også med krydskompilering til windows:

```
$$$
cd src
i586-mingw32msvc-gcc -I../include -O2 -c *.c
i586-mingw32msvc-ar rcs libtrwin.a *.o
i586-mingw32msvc-gcc -I../include -O2 -c -fPIC *.c
i586-mingw32msvc-gcc -I../include -O2 -shared -Wl,-soname,libtrwin.so.1...
                     -o libtrwin.so.1.0.1 *.o
cd ..
$$$

$ ls -l *lib*
-rw-r--r-- 1 tk tk 654418 2009-08-11 15:00 libtr.a
-rwxr-xr-x 1 tk tk 472136 2009-08-11 15:01 libtr.so.1.0.1
-rw-r--r-- 1 tk tk 579974 2009-08-11 15:11 libtrwin.a
-rwxr-xr-x 1 tk tk 835198 2009-08-11 15:11 libtrwin.so.1.0.1
```

Det så jo også ud til at gå - men er det så et DLL eller er det noget andet, der kom ud af det? Vi spørger "file":

```
$ file libtrwin.so.1.0.1
libtrwin.so.1.0.1: PE32 executable for MS Windows (DLL) (console) Intel 80386 32-bit
```

smukt - det ser ud til at være et DLL.

Der kan stadig være masser af skjulte problemer: vi har ikke gjort andet end at sikre at koden kompilerer. Men det er da også et stykke ad vejen!

2.6 Doxygen

Doxygen dokumentationsværktøjet blev kørt via en GUIfrontend "doxywizard", som skriver opsætningsfilen "`doxygen_log.txt`".

Doxygen resulterer i en samling af htmlfiler med krydsreferencer for alle filer, symboler, makroer, etc., og samme informationer i en pdf-fil (på en bagatel af 850 sider!), med tilsvarende links på kryds og tværs.

Det har gjort det væsentligt lettere at navigere i koden og giver derfor både bedre overblik og bedre forståelse.

Doxygen genererer også kaldegrafer og inverse kaldegrafer for alle funktioner (grafiske oversigter over "hvor bliver denne funktion kaldt fra, og hvilke andre funktioner bliver kaldt herfra?").

Kaldegraferne har især bekræftet min eksisterende mistanke om at specielt koden til håndtering af labelsystemet er voldsomt kompliceret. Det undrer mig betragteligt, fordi vi har en beskrivelse af systemets opbygning på Backus/Naurform (BNF) - og det ser ikke synd(erl)igt kompliceret ud.

Jeg regner derfor med at det vil være realistisk at renovere koden (fx ved at skrive tokenisering/parsingen i lex/yacc, eller lignende. Eller (hvis det er vigtigt at holde koden i ren C), gentænke det i lex/yacc-termer (hen ad vejen). Som det ser ud nu tror jeg det vil være meget vanskeligt at vedligeholde (omend jeg ikke vil udelukke at det er et problem man kan dokumentere sig ud af: måske er det ikke så kompliceret når man ser det fra det rette perspektiv).

3 Bemærkninger

3.1 Koden

Umiddelbare tanker ved gennemgang af koden:

Headerfiler Reducer antallet af headerfiler: primært en for hele biblioteket: trliblow.h, og en for de *brugerkaldbare* dele (APIet): trlib.h

Informationsindkapsling Opaque objects: det er ikke nødvendigt for højniveaukode at kende indholdet af structs som de bare skal håndtere en pointer til. Derfor giver det god mening at definere structs i en header for sig, og så bare forwarddeklarere dem i trlib.h - så er man sikker på at brugerkode ikke uautoriseret kommer til at ændre i interne dele.

Automatisk konsistenskontrol af headerfiler Headerfilerne bør nok `#include`s i kodefilerne, så vi kan få compileren til at checke at der er overensstemmelse mellem deklaration og definition.

Dokumentationsvedligeholdelse Dokumentation fra headerfiler flyttes evt. til doxygenkommentarer i kodefilerne.

Kodeoprensning `#include "blabla.h"` bør nok erstattes med `#include <blabla.h>`, da vi ikke har c og h filer i samme bibliotek.

Kodeoprensning Sidenummerering: det er mange år siden det har været nødvendigt at foretage manuel opdeling af kildekodeudskrifter. De fleste koderedigeringsværktøjer kan lave elegante, farvekodede udskrifter af kildekode – og ellers kan man bruge Enscript eller A2ps til formålet. Derfor bør vi efter min mening fjerne sideinddelingskommentarerne efterhånden som vi kommer igennem koden: de er mere til forstyrring end til oplysning.

Tegnsætproblemer I `geo_lab.h` er en del ubehageligheder: platformsspecifikke hjælpemakroer som burde øge portabiliteten, men reelt hurtigt kan blive et problem: koden antager at vi kører på en iso 8859-1 tegnsæt/encoding med dansk kollation, men reelt vil den oftere og oftere blive kørt i et utf-8 miljø. Derudover antager koden nedenfor at char er signed. Det er jeg ikke "sikker på at man kan være sikker på".

Dette er et VANSKELIGT problem (se fx [5]), og med til at bestyrke min mistanke om at vi må gøre noget alvorligt ved labelsystemet. Der skal tænkes! Her er djævelen virkelig på spil i detaljerne (fx ved vi ikke nødvendigvis hvilket tegnsæt/kollation tidligere gemte labels står gemt i).

```
(fra geo_lab.h)

/* machine dependent functions */

#ifdef _WIN32

#define C_DIGIT(c) ((c)!='Æ' && (c)!='Ø' && (c)!='Å' &&
                    (c)!='æ' && (c)!='ø' && (c)!='å' && isdigit(c))

#define C_LOWER(c) ((c)!='Æ' && (c)!='Ø' && (c)!='Å' &&
                   ((c)=='æ' || (c)=='ø' || (c)=='å' || islower(c)))

#define C_UPPER(c) ((c)!='æ' && (c)!='ø' && (c)!='å' &&
                   ((c)=='Æ' || (c)=='Ø' || (c)=='Å' || isupper(c)))

#define C_ALPHA(c) (C_LOWER(c) || C_UPPER(c))

#define C_SPACE(c) ((c) != -89 && (c)!='Æ' && (c)!='Ø' && (c)!='Å' &&
                    (c) != 'æ' && (c)!='ø' && (c)!='å' && ((c)>0 && isspace(c)))

#define C_PUNCT(c) ((c) != -89 && (c)!='Æ' && (c)!='Ø' && (c)!='Å' &&
                          (c)!='æ' && (c)!='ø' && (c)!='å' && ispunct(c))

#else

/* machine rekfs1 */

#define C_DIGIT(c) (isdigit(c))

#define C_LOWER(c) ((c)=='æ' || (c)=='ø' || (c)=='å' || islower(c))

#define C_UPPER(c) ((c)=='Æ' || (c)=='Ø' || (c)=='Å' || isupper(c))

#define C_ALPHA(c) (C_LOWER(c) || C_UPPER(c))

#define C_SPACE(c) ((c)!='Æ' && (c)!='Ø' && (c)!='Å' && isspace(c))

#define C_PUNCT(c) ((c) != -89 && (c)!='Æ' && (c)!='Ø' && (c)!='Å' &&
                          (c) != 'æ' && (c)!='ø' && (c)!='å' && ispunct(c))
#endif
```

3.2 Udfordringen

Vi kommer ikke uden om at vedligeholde trlib. Det er en enestående kodesamling, som er essentiel for KMS' rolle som kustode for historiske data. Der findes ikke noget (kendt) lignende bibliotek af kode som kan tage sig af alle aspekter af projektioner og transformationer - fra fladtrådte datumløse ting, til fuldt 4D datumskift.

Hvorfor er det lige et problem? der findes jo en del projektionsbiblioteker...

Ja, men kortprojektioner er rene matematiske entiteter, som kan behandles i et miljø, hvor vi er på sikker teoretisk grund.

Datumskift er generelle transformationer, geofysisk (under)bestemte, og dermed prediktionsproblemer.

Poder og Engsager [6] behandler emnet meget klart i deres indledende afsnit. Det afsnit er sørgeligt underciteret i verdenslitteraturen. Det bør udvides med overvejelser om tidsvarierende effekter og genudgives med henblik på at skabe forståelse for at trlib er så meget mere end "bare endnu et projektionsbibliotek".

De problemer trlib løser, er vanskelige. Vedligeholdelse og anvendelse kræver stor indsigt fra både udviklere og brugeres side. Teamet Poder/Engsager sad/sidder inde med en helt enestående kombination af praktisk sans og viden om numerik, geodæsi, og datalogi. En kombination vi ikke kan forvente at finde samlet på så få hoveder fremover. Så når koden skal vedligeholdes må det gøres under mere åbne former, og for at tiltrække anvender- og udviklerressourcer ("grab mindshare"), må vi sikre en øget interoperabilitet mellem KMS' labelsystem og mere udbredte metadatamodeller, som EPSG og gdal/proj.

Men først og fremmest: få koden ud i den åbne verden – vi kan ikke vedligeholde den meningsfuldt uden konkret input fra brugere: vi kan simpelthen ikke tromme tilstrækkelig megen viden sammen in-house i kms/dtu-space til at sikre kodens langtidsoverlevelse.

TRlib på launchpad.net under en åben, permissiv licens – hellere i morgen end i overmorgen!

4 Videre læsning

4.1 Litteraturhenvisninger og fodnoter

1 stddef.h: http://en.wikipedia.org/wiki/Stddef.h

2 Emma Jane Hogbin: OpenWeek - Bazaar for docs http://pastebin.ubuntu.com/160234/

3 http://www.adp-gmbh.ch/cpp/gcc/create_lib.html Lynhurtig oversigt over hvordan man fremstiller og anvender dynamisk og statisk linking med gcc. Matthew Perry har også noget at sige om sagen i [4], hvor han fokuserer på Pythonaspektet.

4 Dynamisk linkning i Python: http://www.perrygeo.net/wordpress/?cat=6

5 Anonym: Using mbcs and wide functions http://www.archivum.info/comp.lang.c/2006-05/04410/using_mbcs_and_wide_functions_(was:_Re:_8_bit_character_string_to_16_bit_character_string)

6 Poder, K., and K.Engsager: Some conformal mappings and transformations for geodesy and cartography. KMS pubs. 4. series, vol. 6, KMS 1998

7 König R, and K. H. Weise: Mathematische Grundlagen der Höheren Geodäsie und Kartographie, Erster Band, Springer, Berlin/Göttingen/Heidelberg, 1951

4.2 Relevante webresourcer

Peter Toft om Doxygen http://www.version2.dk/artikel/9173-kode-vaerkstedet-1-doxygen
http://www.version2.dk/artikel/8006-doxygen-elegant-overblik-over-ccjava-kode
http://sslug.dk/foredrag/pto/doxygen_2.pdf

Peter Toft om test og testdækning http://www.version2.dk/artikel/9444-kode-vaerkstedet-2-gcc-og-kode-daekning

Peter Toft om fejlsøgning etc. http://sslug.dk/foredrag/pto/valgrind_4.pdf
http://www.version2.dk/artikel/10412-foss-aalborg-1

Matthieu Brucher om Valgrind http://matt.eifelle.com/2009/04/07/profiling-with-valgrind/

Poul-Henning Kamp om Lint http://www.version2.dk/artikel/8993-bedre-kode-998

National Survey and Cadastre
Rentemestervej 8
2400 Copenhagen NV
Denmark

http://www.kms.dk