

Doc: Dokumentationsværktøj for CPH STL

Jens Peter Svensson

*Datalogisk Institut, Københavns Universitet
Universitetsparken 1, 2100 København Ø*

Resumé. Denne rapport beskriver tilblivelsen af værktøjet ”Doc” som kan bruges til generering af dokumentation. Dette værktøj anvendes af CPH STL buildsystem så dokumentationen for programbiblioteket bliver genereret automatisk. Værktøjet ”Doc” blev implementeret som en udvidelse af Doxygen. Denne rapport indeholder forklaringer på hvordan jeg indsamlede krav til funktionaliteten i ”Doc” og hvordan jeg testede prototypen af ”Doc”.

1. Introduktion

Formålet med projektet var at forenkle arbejdsgange relaterede til vedligeholdelse og generering af dokumentation for kundens kodebase. Et andet formål var at få implementeret features der gjorde onlineversionen af dokumentationen mere brugervenlig for den individuelle bruger. Kunden for det færdige produkt var CPH STL personificeret af Jyrki Katajainen.

Resultatet af dette projekt var værktøjet ”Doc”, der er en udvidelse af Doxygen [1]. Dette værktøj implementerer nogle af indsamlede featureforslag, der automatiserer dele af dokumentationskrivningen og gør den færdige dokumentation mere brugervenlig. Et andet resultat var en udvidelse af CPH STL buildsystem så det kan sættes til automatisk at genere dokumentationen af releases af kundens kodebase. Et tredje resultat var en implementering af onlinefunktionaliteter til at individualisere dokumentation for den enkelte bruger samt lade brugerne udvide onlinedokumentationen vha. kommentarer.

Til denne rapport findes der tre appendikser; disse er tilgængelige i separate elektroniske dokumenter.

SRS for Doc [2]: Dette appendiks er ”Software Requirement Specificationen” og indeholder alle de forslag til features som jeg indsamlede og har dokumenteret. Når der henvises til en feature, vil jeg bruge featurens tag til at henviser til dem i dette dokument.

Kildekoden for Doc [3]: Dette appendiks indeholder, som navnet indikerer, kildekoden skrevet under dette projekt. Det vil dog kun indeholde kode, som jeg har skrevet eller rettet i. Dette er for at mindske dets størrelse og fremhæve den kode jeg har skrevet. Til kildekoden hører filer fra Doxygens kildekode som jeg har rettet i. Disse kan findes under sektion 1 i appendikset. Filer, der skal lægges på webserveren,

kan findes under sektion 2 i appendikset og filer, der ikke rigtigt hører til nogen af stederne, kan findes under sektion 3 i appendikset.

Featurespørgeskemaresultat for Doc [4]: Dette appendiks er et spørgeskema med forskellige featureforslag. De svar, som jeg fik fra de interviewede personer, er skrevet ind i skemaet. Hver interviewede person er identificerede af et tal. Placeringen af tallet indikerer hans svar på et spørgsmål. Hvis et tal mangler ved et spørgsmål, blev den interviewede enten ikke spurgt dette spørgsmål eller der blev ikke givet et svar. Hvis et spørgsmål ikke blev stillet den interviewede var dette ikke en fejl, men en konsekvens af svar den interviewede havde givet til foregående spørgsmål som gjorde et uddybende spørgsmål senere unødvendig at stille.

2. Projektforløb

Projektet blev naturligt delt op i faser, hvor en fase var den periode der gik mellem to møder med kunden.

2.1 Møder

Hver anden uge under projektforløbet, fra 7 september 2009 til 1 december 2009, blev der holdt et møde med kunden. Under disse møder blev der forklaret hvor langt projektet var nået i den foregående fase og der blev aftalt hvad der blev forventet i den efterfølgende fase.

2.2 Indsamling af features

I den første fase af projektet indsamlede jeg features og beskrev disse features i et "Software Requirement Specification" (SRS) dokument. Dette dokument indeholder kun features der beskriver hvad produktet gør eller hvordan en bruger interagerer med produktet. Der vil ikke findes nogen tekniske krav til systemet i dette dokument; disse skulle have været skrevet i et "System Requirement Specification" dokument.

Jeg brugte flere metoder til at indsamle featureforslag til SRS-dokumentet. En metode jeg brugte til at indsamle features på var interviews af kunden og dennes medarbejdere, hvor jeg spurgte til hvilket features de kunne tænke sig at have i deres dokumentationsværktøj. Under interviewne kom der flere interessante featureforslag frem, blandt andet kan følgende nævnes.

dokumentation.template.argumenter: Denne feature var et ønske om at gøre det muligt at se hvilke template argumenter en klasse eller funktion tager, læse forklaringen af argumenterne samt gøre det muligt for programmøren at dokumentere disse argumenter så de altid bliver forklaret og vist ens i dokumentationen.

dokumentation.metodens.kompleksitet: Denne feature er ligesom den foregående, men handler om kompleksiteten af en funktion.

En anden metode jeg havde tænkt mig at bruge til at indsamle features med, var ved at foretage interviews af mine medstuderende om hvilke features de syntes manglede i Doxygen. Denne ide opgav jeg dog relativt hurtigt da hele baggrunden for ideen var en antagelse om at mine medstuderende havde en hvis erfaring med at læse dokumentation lavet af Doxygen eller selv bruge Doxygen. Der tegnede sig dog hurtigt et billede af at mine medstuderende ikke kender Doxygen eller er klar over hvilket dokumentationsværktøj der har lavet de dokumentationer de brugte hvis nogen. Derfor kunne de ikke udtale sig om features de syntes manglede.

For ikke helt at miste input fra mine medstuderende lavede jeg i stedet for et "multiple choice" spørgeskema over featureforslag. Featureforslagene var dels mine egne hentede med stor inspiration fra dokumentationen på www.php.net og dels dem jeg havde indsamlet fra kunden og dennes medarbejdere. Dette skema er vedlagt som et appendiks. Antallet af interviewede personer som jeg brugte dette skema på var 7. Dette er næppe nok til at mene, at deres svar kan bruges til at indikere den gennemsnitlige persons indstilling til en feature, men jeg mener dog at jeg som minimum kan udlede hvilke features der har lidt til ingen interesse. Da et skjult formål med skemaet var at få endnu flere featureforslag, var der mange af spørgsmålene der var "open-ended" i den forstand at en af svarmulighederne i spørgsmålet var "Andet" hvorpå det var forventet at den interviewede specificerede hvad dette andet var.

Ud fra svarene i dette spørgeskema vurderede jeg at de adspurgte studerende storset alle var helt uinteresserede i at kunne se en versionshistorik i dokumentationen og at de også alle var uinteresserede i at kunne se statistik over koden i dokumentationen. Jeg vurderede også at de alle var interesserede i kunne skrive kommentarer i dokumentationen. Denne feature er beskrevet i SRS-dokumentet som kravene identificerede af følgende tags:

- `dokumentation.brugerkommentar.vis`
- `dokumentation.brugerkommentar.tilføj`
- `administration.brugerkommentar.slet`
- `administration.brugerkommentar.accepter`

Den sidste metode jeg fik nye features på var, at kunden ved hvert møde jeg holdt med ham, kom med nye featureforslag som han kunne tænke sig. Disse forslag ville jeg så skrive ind i SRS-dokumentet efter det pågældende møde.

3. SRS-dokumentet

Jeg vil i denne sektion beskrive SRS-dokumentet som jeg lavede og vedligeholdte under dette projekt. Jeg vil også komme ind på mine erfaringer med at gøre dette i OpenOffice Writer (OOW), som ledte mig til visse features som jeg mener et "Requirement Management Tool" bør have.

3.1 Dokumentet

Som nævnt lavede og vedligeholdte jeg et SRS-dokument i OOW. Jeg vil ikke sige at jeg valgte at skrive et SRS-dokument i dette program, men derimod valgte at lave et eksisterende OpenOffice-dokument, der havde en liste af featurebeskrivelser, om til et SRS-dokument. Dette skete efter mit andet møde med kunden. Til udformningen af SRS-dokumentet brugte jeg som grundlag det eksempel som findes i appendikset af [5]. Dette eksempel bruger tags til at identificere en feature med. Et tag er en liste af ord adskilt med punktummer. Bogen fortæller at det er vigtigt, at tags er unikke og at de, hvis featuren ikke bliver slettet, ikke bliver brugt til andre features. Dette førte til at mit SRS-dokument har en sektion med slettede featuretags. Bogen siger også at man kan bruge forskellige former for nummereringer til at identificere en feature, men jeg fandt det mere sandsynligt at jeg ville kunne genkende en feature via et tag end via et nummer og ved at genkende en feature kan man spare sig selv for at skulle slå dem op. For at øge associationsgraden mellem tags og features kan man give featureerne tags så de bliver grupperede ind i kategorier i SRS-dokumentet. Som eksempel kan nævnes featuren "kodeviser.synlig.filliste" fra SRS-dokumentet. Ved første ord i dette tag ved jeg med det samme at featuren hører til kodeviserdelen af projektet eftersom at alle featuretags i den del af projektet starter med "kodeviser". Denne gruppering er bedre end bare at give features tilfældige tags.

Dokumentet har også en prioritetsliste af featurene efter hvor vigtige jeg mener kunden fik dem til at lyde. I dette projekt er valget af hvilke, features der blev implementeret, baseret på hvor lette jeg mente de var at implementere. Dette valg blev taget under den antagelse at jeg så kunne nå at teste flere features. I et reelt projekt skulle jeg have baseret valget af features efter deres prioriteret.

Dokumentet har en ændringshistorik så både kunden og jeg kunne følge med i hvad der blev ændret i dokumentet.

Alle features er beskrevet ud fra en skabelon som jeg til dels selv har lavet. Denne skabelon indeholder en kort beskrivelse af en feature, en titel på featuren, hvordan featuren forbedrer produktet, hvem der foreslog featuren, hvilke andre features der på en måde kan siges at være relaterede, features prioritret og forslag til hvordan jeg foreslår en feature kan implementeres. Alle features har alle felter selv når disse felter er tomme. Grunden til dette er at jeg læste i bogen at man skulle lade tomme sektioner i SRS-dokumentet blive, så læseren ikke spekulere på om de er blevet glemt. Om dette også skulle gælde for felter i min skabelon er uvist. Jeg mener at man kunne argumentere for at manglen af et felt burde være bedre, da det så viser jeg har husket at slette et tom felt fra min featureskabelon. Til at beskrive, hvordan en feature forbedrer et produkt, bruger jeg tre kategorier som jeg mener en feature kan forbedre en internetside. At dele beskrivelsen op i disse kategorier valgte jeg fordi de fleste features omhandler internetsider. Kategorierne er:

Fremhævning: En feature kan forbedre en internetside ved at fremhæve vigtig information på siden. Dette kan gøres f.eks. ved, at formatere tekst så det, der ønskes fremhævet, stikker mere ud end resten af teksten. Et eksempel på dette ville være `kodeviser.spacing.kodeelement` der beskriver spacing mellem kodeelementer indsat som en værdi af systemet. Denne feature kan f.eks. tydeliggøre udtrykket `"a+b*c+d"` til `"a + b*c + d"` fordi den kan indsætte et mellemrum på begge sider af `"+"`. Der er dog lidt snyd i dette eksempel, da `"Doc"` ikke kan indsætte mellemrum på begge sider af `"*"` overhovedet, se forklaring i sektion 6.1. En anden måde at fremhæve information for brugeren er ved at skjule anden information. Et eksempel på dette er featuren `kodeviser.skjul.kommentarer` som omhandler hvordan brugeren kan få systemet til at folde eller fjerne kommentarer fra kildekoden. Dermed kan brugeren koncentrere sig om koden uden at skulle forstyrres af kommentarerne.

Navigation: En feature kan forbedre navigationen på en internetside ved at mindske antallet af handlinger, som brugeren skal foretage for at komme frem til en given information på siden. Disse handlinger består, som jeg ser det, enten af klik med musen, som regel for at få en ny side vist, eller scroll på siden, fordi det som brugeren vil se, ikke bliver vist i browseren når sidens størrelse er mere end browservinduet. Den største forbedring jeg mener man kan foretage sig i denne kategori er dermed helt at fjerne antallet af klik og mindske scrolllængden. Et eksempel på en feature, der opnår begge disse, er `kodeviser.synlig.filliste`. Denne feature mindsker antallet af klik ved at brugeren ikke længere skal til siden i dokumentationen, der har fillisten. Featuren mindsker scroll fordi listen følger med brugeren ned ad siden, så brugeren ikke skal scrolle til menuen i toppen af siden.

Andet: Denne kategori er her ud fra den observation, at ikke alle forbedringer naturligt kan forklares med fremhævning og navigation. En feature kunne f.eks. forbedre sikkerheden ved et system eller gøre en tidligere manuel handling automatisk.

Forklaringerne af en features forbedringer af et system kunne bruges til at værdisætte og dermed prioritere en feature. Eksempelvis vil jeg mene at en feature, hvis forbedring ikke kan forklares og beskrives, ikke har nogen værdi og dermed får ingen prioritet.

3.2 Erfaringer med SRS-dokument i OOW

Mine erfaringer med at vedligeholde SRS-dokumentet i OOW har ledt mig til den konklusion at OOW mangler en del features til at være fornuftigt valg at bruge som `"Requirement Manegement Tool"`. Det kan dog, som vist med SRS-dokumentet, godt bruges. De største problemer man løber ind i er at man i OOW kun kan se en sektion af dokumentet af gangen og at intet opdateres automatisk. Jeg har f.eks. lavet og opdateret prioritetslisten og ændringshistorikken manuelt.

Udfra mine erfaringer med OpenOffice vil jeg mene, at et "Requirement Management Tool" skal som minimum kunne gøre følgende.

- Det skal give brugeren mulighed for at se og rette flere featurebeskrivelser samtidig; helst så mange denne ønsker.
- Det skal give brugeren mulighed for at erklære et view af features kun med udvalgte dataer. Disse kunne f.eks. være titlen og beskrivelsen af en feature. Dette kunne bruges til at lave et helt SRS-dokument, der skjuler noget data som f.eks. løsningsforslagene der ikke havde nogen reel mening for kunden. Denne feature havde OOW ikke; derfor så kunden alle dataer for en feature.
- Det skal kunne lave en automatisk prioritetsliste, dette er i grunden næsten dækket ind med skjulning af feature data, men der mangler at man så skulle kunne sortere features baseret på deres dataer.
- Det skal kunne lave en automatisk ændringshistorik med opsamling af ændringer i bestemte datoer, således at kunden kunne se en ændringshistorik mellem to datoer, mens forfatteren af SRS-dokumentet kan se historikken ændring for ændring.
- Det skal kunne sættes til at bruge tags som featureid'er. OOW har ikke denne feature. De tags der står i SRS-dokumentet er lavet ved at erklære dem som krydshenvisninger.
- Det skal kunne navigeres til et krav når dettes tag bliver valgt andet steds. Da tags i OOW blev erklæret som krydshenvisninger, kunne man navigere til kravet ved at klikke på tagget i SRS-dokumentet.
- Det skal give mulighed for at rollback et krav til en foregående tilstand. OOW giver kun mulighed for at fortryde ændringer lavet i dokumentet siden det blev åbnet.
- Det skal kunne vise brugte id'er. OOW giver mulighed for dette da man kan se en liste over alle krydshenvisninger.
- Det skal give mulighed for at indskrive andet end krav i systemet, som f.eks læsevejledning, ordbog og templatebeskrivelser.
- Det skal give mulighed for at erklære link mellem to krav og blive advaret om når disse brydes, f.eks som konsekvens af at det ene af kravene slettes.
- Det skal være umuligt at skrive invalide tags. I OOW var det muligt at skrive invalide tags da det bare tog det jeg markerede som en krydshenvisning.

4. Valg af tredjepartssoftware

Jeg valgte at bruge tredjepartssoftware til de elementer af projektet hvor jeg vurderede, at det var lettere at finde et tredjepartssoftware end selv at lave koden. I valg af tredjepartssoftware var det eneste, der var vigtig for mig, at jeg havde lov til at distribuere og modificere softwaren som jeg havde lyst til.

4.1 Doxygen

Eftersom at lave et helt dokumentationssystem fra bunden ville tage væsentlig længere tid end projektperioden, var det allerede valgt fra projektets start at bruge Doxygen og bare bygge prototyper af features ind i dette. Doxygen blev ikke valgt fordi det nødvendigvis er det bedste dokumentationssystem til at dokumentere C++ kode som kundens kodebase er kodet i, men blev valgt fordi det var det eneste C++ dokumentationssystem jeg kendte til ved projektetsstart. Doxygen kan hentes gratis fra dets internet-side <http://www.stack.nl/~dimitri/doxygen/>. GPL-licensen giver mig lov til at bruge, ændre og distribuere koden og programmet.

4.2 GeSHi : Generic Syntax Highlighter

Da featurene `documentation.brugerkommentar.vis` og `documentation.brugerkommentar.tilføj` blev implementerede fandt jeg det naturligt, at brugerne af disse features formentlig ville skrive C++ kode i deres kommentarer. Da det er min overbevisning at kode bliver mere læsbar, når den syntaksfarves, fandt jeg GeSHi der kan tage C++ kode og give denne kode farve. GeSHi er ikke nødvendigvis det bedste valg, men det var det første jeg fandt der var skrevet i PHP ved at søge på Google. GeSHi kan hentes gratis fra dets internetside <http://qbnz.com/highlighter/>. Licensen til GeSHi er også GPL-licensen.

Et alternativ jeg kunne have prøvet var at tage den del af Doxygen der visualiserer kildekoden og lægge dette i en eksekverbar fil, og så fra PHP kalde denne fil med koden i brugerens kommentar. Dette skulle da producere HTML-uddata, som vil være det der blev vist, når brugerne læste kommentaren. Fordelen ved dette ville have været at de features jeg implementerede til visualisering af kode i Doxygen ville blive brugt på kode skrevet i brugernes kommentarer, som f.eks. indrykning bestemt af brugeropsætningen. Jeg vurderede dog at denne mulighed ville tage væsentligt længere tid at implementere.

4.3 *changeCSS*

Dette software har ikke et egentlig navn; det består kun af en Javascript funktion der kan ændre på attributterne i en CSS-klasse. Jeg brugte denne software til at implementere featuren `documentation.skjul.metodervariabler`, der bruges til at skjule og vise elementer i dokumentationen baseret på om de er `public`, `private` eller `protected`. Funktionen kan kopieres fra http://www.shawnolson.net/scripts/public_smo_scripts.js. Det eneste krav forfatteren af funktionen stillede, var at jeg gav ham "credit" for at have lavet funktionen; dette mener jeg at er gjort ved ikke at have fjernet hans kommentar i toppen af koden.

4.4 CPH STL buildsystem

Eftersom at kunden udtalte et ønske om at generering af dokumentation skulle være noget der skete så automatisk som muligt, ændrede jeg i kundens buildsystem således at det kaldte "Doc" med den udgave af CPH STL som blev bygget med buildsystemet. CPH STL buildsystem kan findes under "CPHSTL/Script/Build-system".

5. Vertikal evolutionær prototype

I min beskrivelse af den nuværende prototype anvender jeg terminologien fra [5].

"Doc" er en vertikal prototype, fordi de features, som jeg valgte at implementere fra SRS-dokumentet, blev kodet så de faktisk er en fungerende del af "Doc". De vil derfor blive brugt på en hvilken som helst programkode som "Doc" afvikles på. Dermed er den prototype jeg lavede faktisk brugbar og ikke bare visuel.

Det modsatte af en vertikal prototype er en horisontal prototype. Den slags prototype viser kun hvordan brugergrænsefladen til det færdige program vil se ud. I mit tilfælde ville jeg, hvis jeg havde valgt at implementere en sådan, kunne have nøjedes med at vise brugergrænsefladerne til de forskellige brugertyper der interagerer med produktet. Som jeg ser det, er der kun tre brugertyper til mit produkt.

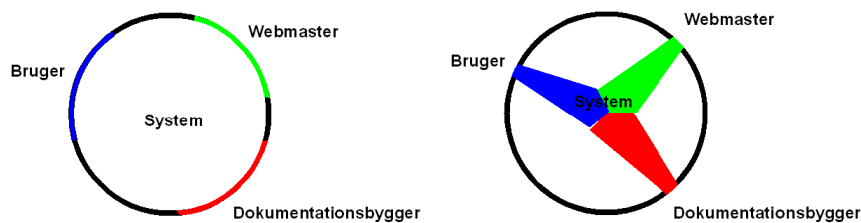
Dokumentationsbyggeren: personen der har ansvar for at dokumentation bliver genereret over programkoden. Ved en horisontal prototype kunne jeg have nøjedes med at vise denne person hans grænseflade til "Doc". Da denne person bruger CPH STL buildsystemet, ville denne grænseflade være hvordan han installerede en kompileret udgave af "Doc" og indbyggede kald til denne i buildsystemet. Ved min vertikale prototype skal dokumentationen også laves når buildsystemet bliver brugt.

Webadministratoren: personen der har ansvar for at lægge dokumentationen på nettet. Ved en horisontal prototype kunne jeg have nøjedes med at fortælle denne person hvordan han lagde filerne på sin webserver og hvilke filer han derefter skulle afvikle for at installere og administrere dokumentation, samt vise ham hans administrationsgrænseflade. Ved min vertikale prototype skulle en udgave af dokumentationen installeres under installationen og administratoren skulle kunne administrere dokumentationen.

Brugerne af dokumentation: personerne der bruger dokumentationen på internetsiden. Ved en horisontal prototype ville det have været nok at vise hvordan dokumentationen så ud for en bruger med en forudvalgt brugeropsætning. Brugeropsætningen kunne skrives ind i dokumentation lavet med Doxygen. Ved min vertikale prototype skal brugeren selv kunne sætte sin brugeropsætning og se hvordan dokumentationen ændrer sig som konsekvens af dette.

Det burde være klart at en horisontal prototype ville have som minimum vist de samme grænseflader som min vertikale prototype gør, men den ville have været hurtigere at lave, fordi den havde krævet at jeg skulle kode features ind i Doxygen, som tager tid. Dog tager det selvfølgelig også tid at skrive en brugsopsætning ind færdig lavet dokumentation.

Den tid jeg havde sparet ved en horisontal prototype kunne være blevet brugt til at visualisere flere features. Forskellen på de to slags prototype har jeg prøvet illustrere med figur 1. Cirklen symboliserer systemet, kanten er grænsefladen og området inde i cirklen er systemet selv.



Figur 1. Horisontal og vertikal prototype; farverne indikerer hvilke dele af systemet der er prototype.

Jeg anser også "Doc" for at være en evolutionær prototype, da jeg ikke finder det særligt sandsynligt at kunden på noget tidspunkt vil smide min prototype væk og starte helt forfra med en ren udgave af Doxygen. Jeg finder det dog sandsynligt og det er da også min forståelse, at kunden har tilsinds at få andre personer/studerende til at arbejde videre med "Doc".

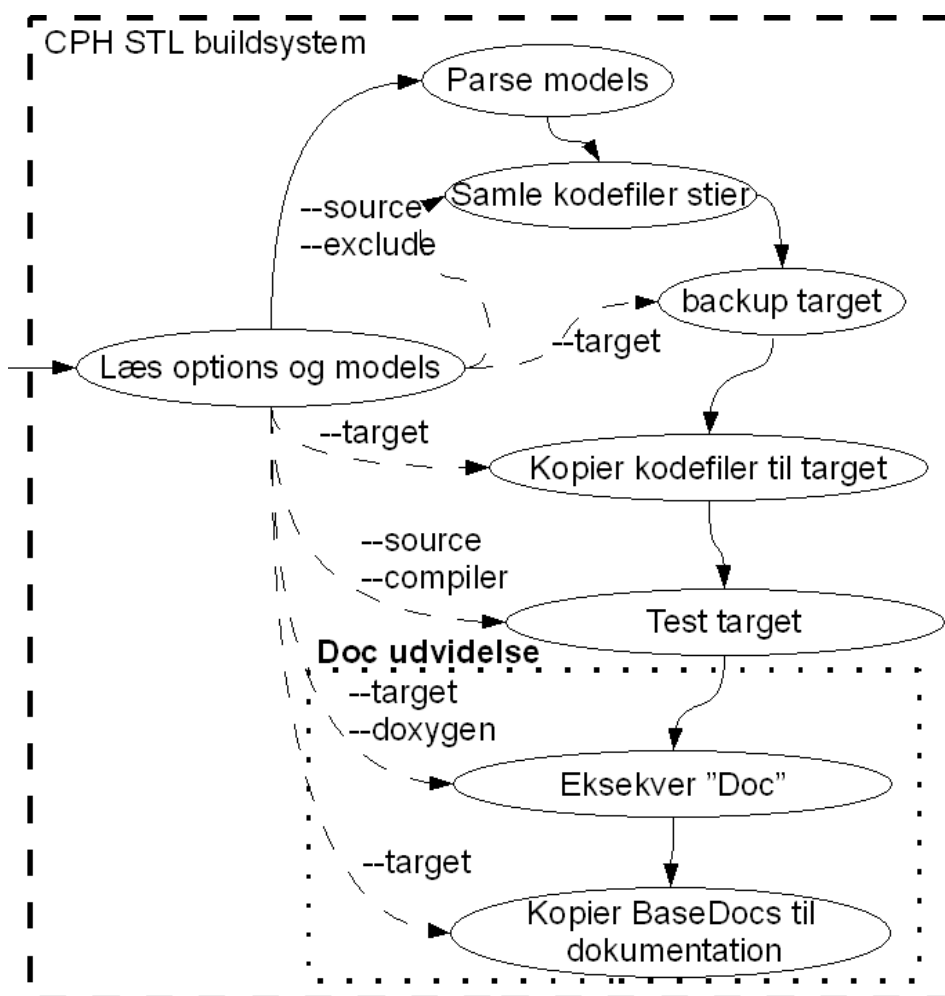
6. Designvalg

Jeg har valgt at skrive denne sektion om de valg jeg har foretaget under implementationen af "Doc" for at efterfølgende studenter, der ønsker at arbejde videre med "Doc", kan få et kort overblik over hvordan jeg implementerede "Doc". Jeg vil forklare valg foretaget i implementeringen af features i Doxygen og valg foretaget i implementering af serversiden af online-dokumentationen.

For at forklare interaktionen mellem CPH STL buildsystem og "Doc" lavede jeg en illustration af CPH STL buildsystemet, hvordan dette system fungerer såvidt jeg kan læse dets Python koden, efter at jeg implementerede dokumentationsgeneration i det. Illustrationen kan ses i figur 2.

Som det ses af figuren, bliver dokumentationen af et build først lavet efter testen af buildet. Dette er fordi systemet i testfasen fjerner filer hvis test fejler.

Generering af dokumentationen for et build foregår i to skridt. Først bliver "Doc" kørt på kodefilerne i buildet og derefter bliver filerne i "BaseDocs" folderen kopieret ind i dokumentationsfolderen.



Figur 2. CPH STL buildsystem.

6.1 Doxygen

I implementationen af features i "Doc" prøvede jeg at sørge for at de features jeg implementerede blev implementeret så sent som muligt i dokumentationsgenereringen, som regel først i HTML-generatoren. Grunden til dette er et ønske om at en feature skulle have få sideeffekter som der skulle tages højde for.

Et andet valg foretaget i implementeringen af features var at lade "Doc" læse implementeringstekster ind fra en ekstern fil og senere skrive dem til uddata. Der er f.eks. en implementeringstekst for hvordan kommaer skrives når "Doc" skriver koden for en fil, så i stedet for at "Doc" skriver kommaet skrives implementeringsteksten. Denne kunne være " , " der vil sætte et mellemrum på hver side af kommaet. Der var to grunde til dette; dels var

kaldt til generatorens uddata mens andre kun skriver HTML-tags til uddata og lader `codify` funktionen udskrive koden.

Denne måde at generere kodevisninger på har den konsekvens at generatoren ikke kender noget til hvordan koden ser ud eller hvad den betyder. Dette gjorde at de features der blev implementerede for at gøre kodevisninger mere dynamiske, blev implementeret ved enten at stoppe generatoren i at skrive uddata og stedet huske den kode den skulle have skrevet, indtil den var sikker på at koden var speciel, så den skulle skrives anderledes til uddata end andet kode. Et eksempel på dette er at kodelæseren kalder en funktion i generatoren hver gang et keyword starter og en anden hver gang keywordet slutter. Da en af subfeaturene i `codeviser.spacing.prototype` er at indsætte en specificeret spacing efter "if" keywordet stopper den første funktion generatoren i at skrive uddata, når den anden funktion bliver kaldt testes det fangede uddata og hvis det er "if" sættes generatoren til ikke at udskrive whitespace indtil den efterfølgende "(", i stedet indsættes en tekst fra implementationsfilen, i begge tilfælde udskrives det fanget keyword først. I eksemplet bliver koden "if" fanget og genkendt som værende speciel, men keywordet kunne også have været f.eks. "return", generatoren vil ikke finde dette keyword speciel og derfor udskrives "return" til uddata i "Doc" på samme måde som "Doxygen" gør det kun med syntaksfarvning.

Det burde være tydeligt at denne teknik til at generere kodevisninger på har en kraftig effekt på hvilke features der kan implementeres. Helst skal den kode en feature kræver være både unik og lokal. Ved unik menes der at kodestumpen helst skal have en og kun en betydning. Et eksempel på dette er keywords med undtagelse af "while", som også bruges som slutningen af en "do" statement; disse har kun en betydning i koden. Med lokal menes der at den kode der udgør stumpen skal være ubrudt. Keywords er også et eksempel på dette. Et eksempel på kode der ikke er lokal er "statement" blokke der dækker flere linier af kode og dermed ville blive brudt af mere lokale features i disse blokke. Hvis koden ikke er unik som f.eks. + og ++ skal generatoren for at kunne indsætte +-spacing, huske det første plus og kun indsætte spacing hvis det næste tegn ikke er et plus. Endvidere skal det også antages at + kun bliver brugt infix notation i koden. Kode som ikke er lokal, som f.eks. * der godt nok kan bruges som et normalt gangetegn, men som også bruges til at dereferere en pointer, kan der ikke implementeres features for i generatoren uden at denne udvides til at huske tilstande. I tilfældet med * tegnet skulle generatoren kunne huske dels om den var i en expression og dels om tilstanden af denne expression tillod gange eller en dereferering. Hvis man lavede en feature på * tegnet, ville denne blive brugt ved "c=a*b", men også ved "c=*b". Så hvis der f.eks. blev indsat mellemrum på begge sider af et * tegn fik man "c=a * b" og "c= * b'".

Jeg overvejede og eksperimenterede i starten af projektet kort med at bruge et syntakstræ i stedet for tekststrømmen til at lave kodevisninger. Fordelen ved syntakstræer er at de fortæller mere om kodens struktur. Det ville derfor være lettere at implementere features der er baserede på kodens struktur ud fra et træ. Ideen gik ud på at konvertere alle noder i et træ til funktionskald i

det valgte implementeringssprog. Brugeren kunne da lave udskiftningsregler for alle noder i form af simple tekster med navngivet udskiftningssteder, se figur 4. Jeg kunne godt få ideen til at virke på enkle syntakstræer, men jeg opgav at implementere den af to grunde. Først og fremmest vurderede jeg at det ikke kunne gøres inden for projektperioden, men endnu vigtigere vurderede jeg også at jeg ikke bare kunne bruge en eksisterende parser. Grunden til dette var at det ikke var sproget C++ som jeg ønskede at lave en visualisering af, men hele teksten programmøren havde skrevet som var både C++ koden, kommentarene og makroerne.

Syntakstræ til funktionskald



Funktionskald til tekst

```

Funktion plus(venstre,højre){
  Hent kopi bruger plus string, f.eks. "%venstre + %højre"
  Udskift "%venstre" med venstre i plus string
  Udskift "%højre" med højre i plus string
  Returnerer plus string
}
  
```

plus(3,5) —————> 3 + 4

Figur 4. Syntaksnode til kodevisning for plusnoden.

6.2 Server

Jeg valgte at implementereserver siden af dokumentationen i PHP, da jeg havde erfaring med dette sprog. Valget blev dermed udelukkende taget for at nedbringe arbejdstiden ved at gøre brug af min eksisterende kendskab til PHP.

Jeg valgte at bruge klasser hvis funktioner og variabler alle er erklæret `static`. Dette gør at dataerne og funktionerne er klassefunktioner og klasse-dataer. Grunden til dette valg er at jeg til alle tider ønskede kun at have en instans(objekt) af en klasse. Der er f.eks. altid kun en aktiv brugeropsætning. Da jeg kun havde en instans, kunne jeg bruge en klasse lavet som tidligere beskrevet til at være denne instans og dermed aldrig lave instanser af klasser. En fordel ved dette valg er at, eftersom klasser er globalt tilgængelige, så er deres klassefunktioner og dataer det også. Dermed behøvede jeg ikke

sende objekter til funktioner for at få tilgængelig data, men kunne tilgå dataen direkte i klassen. Dette kunne godt lyde som et singleton pattern, men er det ikke da dette pattern kræver en instans af klassen. Der er ikke noget øvre loft på hvor mange instanser man kan lave af klasserne. Det giver bare ikke mening at lave nogen da alle funktioner og dataer som sagt er erklæret `static`.

Jeg valgte at bruge PDO som databasegrænseflade. Grunden til dette valg var dels at kunden havde udtrykt ønske om at kunne udskifte sit database-administrationssystem med andre sådanne systemer, uden at skulle rette i koden og dels et ønske om at øge sikkerheden. PDO er en data tilgangsgrænseflade i PHP, der i.flg. dokumentation sikrer mod SQL-injektioner.

The parameters to prepared statements don't need to be quoted; the driver automatically handles this. If an application exclusively uses prepared statements, the developer can be sure that no SQL injection will occur (however, if other portions of the query are being built up with unescaped input, SQL injection is still possible).

<http://www.php.net/manual/en/pdo.prepared-statements.php>

Da PDO kun er en datatilgangsgrænseflade betyder det at den SQL jeg har skrevet ikke bliver ændret af PDO til korrekt SQL for den databasedriver der er i brug, men kun at de funktioner jeg bruger til at kalde databasedriveren med vil være de samme. Dette betyder at hvis driveren ændres fra MySQL, som er den nuværende driver, til en anden driver virker SQL-forespørgelserne ikke nødvendigvis med den nye driver. Som konsekvens af dette lavede jeg en klasse der indeholder alle SQL-forespørgelserne og interaktionen med PDO. Ideen med klassen er at, hvis man beslutter sig for at ændre til en anden driver, skal man kun rette i en klasse. En sideeffekt af dette er at klassen er en "God"-klasse i den forstand at den skal kunne gemme, hente, slette og oprette alle slags dataentiteter, i dette tilfælde kun brugeropsætninger og brugerkommentarer.

7. Test

I denne sektion vil jeg beskrive den test som jeg foretog af "Doc" og resultatet af testen. Jeg vil også kort komme ind på hvordan jeg mener SRS-dokumentet blev testet ved uformelle reviews.

7.1 Uformelle reviews

De indsamlede featureforslag blev testet ved "informal reviews". Med dette mener jeg at jeg ved hvert møde med kunden, hvor vi talte om hvad jeg havde arbejdet på siden sidste møde, viste kunden SRS-dokumentet. Kunden kunne i dette dokument så læse de forskellige featureforslag, se når de ændrede tilstand og når der kom nye til. Kunden kunne også ved disse møder afprøve en udgave af dokumentationen med implementerede features. Dette

kunne han ved at jeg dagen forinden havde lagt en udgave af dokumentation genereret med "Doc" på nettet som kunden så kunne afprøve.

Eftersom at både SRS-dokumentet og dokumentationseksemplet var elektroniske, havde kunden have haft mulighed for et nærmere studie af begge efter hvert møde.

7.2 Brugertest

Imod slutningen af projektet blev der foretaget en brugertest. Dette blev gjort ved at jeg skrev en række opgaver som jeg mente at en af systemets brugertyper skulle kunne udføre med systemet. Den eneste bruger i denne test var kunden, som derfor agerede allerede de tre brugertyper.

Mens brugeren foretog testen observerede jeg hvad han foretog sig, for at se hvad der voldte problemer. Jeg hjalp kun brugeren, når jeg vurderede at han var på vej i en forkert retning eller hvis han sad fast et sted.

7.2.1 Dokumentationsbyggeren

Som dokumentationsbygger var det kundens opgave at installere "Doc" i CPH STL buildsystem og lave dokumentation for et build. Til at hjælpe ham med dette havde jeg skrevet en kort guide til installation af "Doc". Denne kan ses i kildekode appendikset 3.4. Kunden fulgte guiden, men da det viste sig i et skridt af guiden at det var uklart, hvilke filer og hvortil han skulle kopiere dem, var jeg nød til at hjælpe ham. Det lykkedes heller ikke for kunden at finde en udgave af Graphviz programmet til dennes OS. Graphviz er ikke strengt nødvendigt, da det kun bruges til at lave diagrammer med. Et andet problem som opstod var at den distribution jeg havde lavet ikke kunne afvikles på kundens OS. Dette sidste problem betød at kunden som dokumentationsbygger var nød til at tage "Doc" kildekoden, som jeg havde medbragt, og kompilere den på hans maskine. Til dette fandtes der ingen guide.

Resultatet af brugertesten for dokumentationsbyggeren blev at jeg ændrede teksten i guiden til installation af en distribution, således at det er klarere hvilke filer og hvortil de skal kopieres og at jeg skrev en guide til kompilation af "Doc". Disse kan ses i kildekode appendikset 3.6.

7.2.2 Webadministrator

Som webadministrator var det kundens opgave at lægge den genererede dokumentation på websiden. Til dette var der en guide, der kan ses i kildekode appendikset sektion 3.5. Kunden havde problemer med at udføre denne opgave, da han ikke kunne huske de MySQL informationer der gav root access til databaseserveren. Disse var nødvendige da installationen af dokumentationen forsøger at oprette en database. Dette problem blev ikke løst under testen, men jeg havde dagen forinden uploadet testdokumentation til min egen server. Testen fortsatte med denne dokumentation. En anden opgave

som webadministratoren har er at acceptere eller slette brugernes kommentarer. Måden systemet var bygget op på var at administratoren skulle tage stilling til hver kommentar en gang; efterfølgende kunne han ikke se kommentaren. Dette var en god ide da administratoren derfor kun skulle tage stilling til nye kommentarer, men kunden gjorde mig opmærksom på at han ikke var ufejlbarlig og at det derfor var nødvendigt at kunne slette kommentarer, som han fejlagtigt havde accepteret.

Resultatet af brugertesten for webadministratoren blev at administrationsdelen af dokumentationen blev udvidet med muligheden for at se og slette allerede accepterede kommentarer.

7.2.3 Bruger

Som bruger af dokumentation havde kunden ingen tilgængelig guide fordi der ikke er en tilgængelig i dokumentationen for rigtige brugere. I denne fase af testen prøvede kunden at se på de forskellige dele af dokumentationen. Ved at gøre dette lykkedes det ham at finde relativt store fejl i kodeviseren, bl.a. at denne fejlagtigt fangede et ":" efter scope keywords. Konsekvensen af dette var at den fjernede alle efterfølgende blanke linier og skrev den første ikke blanke linie på samme linie som kolonnet med dennes linies linienummer først. Dette kan ses på figur 5. Kodeviseren skulle kun have fanget ":" i C++ konstruktionen "(COND)?EXP:EXP;"

```
00032     class double_stack_heap_store {
00033         public : 00035         typedef C comparator_type;
00036         typedef typename A::template rebind<P>::other component_allocator_type;
```

Figur 5. Forkert fjernelse efter : i filen double-stack-heap-store_8h_09_09_source.php.

Under testen kom brugeren med en række nye featureforslag baserede på hans oplevelse af dokumentationen. Et af disse var ikke at udfylde med 0'er i kodeviseren når linienumret ikke var 5 cifre langt, i figur 5 kan det ses hvordan linienumret så ud i "Doc" under testen. Brugeren udtrykte også interesse for se matematiske symboler i stedet for de normale kodesymboler ==, !=, osv. En sidste feature han udtrykte interesse for var at kunne ændre flere opsætningsmuligheder af gangen, i stedet for at skulle ændre dem hver for sig, som vist i figur 6.

Resultatet af brugertesten blev bl.a. tilføjelsen af featurene kodeviser.matematiske.symboler, bruger.opsætning.symbol.valg og bruger.opsætning.grupper til SRS-dokumentet og implementering af disse i "Doc". De to første features lader brugeren vælge om denne ønsker at bruge de normale kodesymboler i koden, se figur 7, eller de tilsvarende matematiske symboler, se figur 8.

Den tredje feature lader brugeren ændre i flere opsætninger på en gang ved at ændre på opsætningsværdien for en hel gruppe af opsætninger, se figur 9.

Den fejlagtige fjernelse af blanke linier blev også rettet og nuller i starten af linienumre blev fjernet, se figur 10.

Symbol	Before	After	Description
__	0	1	A positive natural number or zero to indicate the number spaces placed before and after ','
;	2	2	A positive natural number or zero to indicate the number spaces placed after a ','; note this does not apply to the first character
+	2	2	A positive natural number or zero to indicate the number spaces placed before and after '+'
+=	2	2	A positive natural number or zero to indicate the number spaces placed before and after '+='
<<=	2	2	A positive natural number or zero to indicate the number spaces placed before and after '<<='
>>=	2	2	A positive natural number or zero to indicate the number spaces placed before and after '>>='
&=	2	2	A positive natural number or zero to indicate the number spaces placed before and after '&='
=	2	2	A positive natural number or zero to indicate the number spaces placed before and after ' ='
%=	2	2	A positive natural number or zero to indicate the number spaces placed before and after '%='
=	2	2	A positive natural number or zero to indicate the number spaces placed before and after '='

Figur 6. Uden grupper i file:usersettings.php.

```

97         if (s != 0 && (*first_pair).height() == (*s).height()) {
98             (*first_pair).successor_pair() = jump;
99         }

```

Figur 7. Matematiske symboler slået fra i file double-stack-heap-store_8h_09_09.source.php.

7.2.4 Guideværktøj

Mens kunden udførte testen som wedadministrator og som dokumentationsbygger observerede jeg at han brugte meget tid på at finde den rigtige guide. Dette var dels fordi der var to separate guider, men også fordi guiderne bare var lange tekster skrevet i enten ren tekst eller HTML. De lange tekster gjorde det svært for brugeren at finde den rigtige del af guiden. For at forbedre dette er der nu kun en guide, som en yderligere forbedring bliver der kun vist en del af guiden af gangen og en indholdfortegnelse til resten af guiden.

For at lave denne guide lavede jeg et lille guideskrivningsværktøj i Javascript. Dette tool tager teksten fra body-tag i en HTML-fil. For hver linie i teksten, der starter med "@", laves der et nyt indeks med titel svarende til resten af teksten i linien og for hver linie der starter med "-" laves der et nyt skridt i det nuværende indeks. Teksten til skridtet bliver teksten som den er skrevet i file indtil den næste linie der starter med "-" eller "@" eller enden af file. Dette tool lavede jeg for at guideforfatteren, som var mig selv, ikke behøvede at tænke på hvilke HTML-tags han skulle bruge. En anden fordel ved dette værktøj er at det ikke kræver specielle programmer til hverken skrive eller læse en guide. En guide kan læses i en hver browser, selv hvis Javascript er slået fra. Den kan også læses i en teksteditor da de eneste tegn der ikke hører til er "@" og "-" og få HTML-tags i starten og slutningen af file. Ligeledes kan den skrives i en almindelig teksteditor.

Den nye guide og tool kan findes i kildekode appendikset under sektionerne 3.6 og 3.7

```

97         if ($s != 0 && (*first_pair).height() == (*s).height()){
98             (*first_pair).successor_pair() = jump;
99         }

```

Figur 8. Matematiske symboler slået til i filen double-stack-heapstore_8h_09_09_source.php.

For all mathmathical operators		Before: <input type="text"/>	The number of spaces to u
		After: <input type="text"/>	
__+__:	Before: <input type="text"/>	A positive natural number or zero to indicate the number spaces placed b	
	After: <input type="text"/>		
__%__:	Before: <input type="text"/>	A positive natural number or zero to indicate the number spaces placed l	
	After: <input type="text"/>		
__/_:	Before: <input type="text"/>	A positive natural number or zero to indicate the number spaces placed b	
	After: <input type="text"/>		
For all assignments		Before: <input type="text"/>	The number of
		After: <input type="text"/>	
__=__:	Before: <input type="text"/>	A positive natural number or zero to indicate the number spaces placed b	
	After: <input type="text"/>		
__+=__:	Before: <input type="text"/>	A positive natural number or zero to indicate the number spaces placed	
	After: <input type="text"/>		
	Before: <input type="text"/>		

Figur 9. Med grupper i filen:usersettings.php, ved at skrive værdier tekstfelterne for gruppen ændres værdierne i alle tekstfelter i gruppen.

7.3 Browsertest

Da dokumentation skulle vises i en webbrowser, er det en klar mangel at jeg ikke som minimum testede om systemet fungerede i de mest udbredte browserer. Både kunden og jeg brugte Firefox som browser. Dette er derfor den eneste browser som jeg har set dokumentationen i.

8. Konklusion

I dette projekt lærte jeg at det ikke er nok at kende interviewteknikker til at få informationer fra personer. Det er også vigtigt at vurdere om en teknik passer til den person man skal interviewe. Man skal f.eks. ikke vælge at bruge en teknik der kræver at den interviewede har viden om et emne, hvis denne person ikke har det.

Jeg lærte at arbejde med tredjepartssoftware og at integrere forskellige softwareprodukter med hinanden for at lave et større produkt. Der blev i projektet gjort brug af fire tredjepartssoftwareprodukter og jeg har skrevet kode i følgende sprog C++, CSS, HTML, Javascript, MySQL, PHP og

```
32     class double_stack_heap_store {
33     public:
34
35         typedef C comparator_type;
36         typedef typename A::template rebind<P>::other component_allocator_type;
```

Figur 10. Ingen fjernelse efter : ved scope keywords og ingen unødvendige 0'er i linienumre i filen double-stack-heap-store_8h_09_09_source.php

Python.

Jeg lærte at featureforslag til et produkt kan findes på mange måder. Fra brugerne og kunden gennem via interviews, men også ved at lade dem teste prototyper af produktet. Man kan bruge sin egen viden om gode features ved tilsvarende produkter.

Ved test lærte jeg også at det ikke er nok bare at forberede sig på at alt til testen virker som det skal. Man skal også have backup planer for alt det der kan gå galt, ideelt set planer der gør at testen kan foresætte. Uden disse kan man i værstefald risikere ikke at kunne udføre en test overhovedet og man vil derfor blive nød til at arrangere en ny test.

Der blev udviklet et tool "Doc" der kan lave HTML-dokumentation over C++ kode med en mulighed for at vise koden som brugeren ønsker at få den vist. Der blev også lavet et lille tool der kan lave bedre guider end rene tekstguider, men som ikke gør det meget sværere at skrive dem.

9. Tilkendegivelser

Jeg vil godt takke Jyrki Katajainen for at have været vejleder på dette projekt og for at lært mig at skrive bedre rapporter. Jeg vil også takke ham for at agere kunde og bruger i projektet. Jeg vil sige tak til Bo Simonsen, Claus Jensen og andre studerende for at have taget sig tid til at blive interviewet til projektet. Til sidst vil jeg takke folkene bag Doxygen, GeSHi, CPH STL og til Shawn Olson for at have udviklet tredjepartssoftware produkter brugt i projektet.

Litteratur

- [1] D. van Heesch, Doxygen: Source code documentation generator tool, Worldwide Web Document (1997–2009). Available at <http://www.stack.nl/~dimitri/doxygen/>.
- [2] J. P. Svensson, *SRS for Doc*. (Fil <http://cphstl.dk/Tool/Doc/Report/SRS.odt>.)
- [3] J. P. Svensson, *Kildekoden for Doc*. (Fil <http://cphstl.dk/Tool/Doc/Report/sourceCode.pdf>.)
- [4] J. P. Svensson, *Featurespørgeskemaresultat for Doc*. (Fil <http://cphstl.dk/Tool/Doc/Report/questionnaire.odt>.)
- [5] K. E. Wiegers, *Software Requirements*, Microsoft Press, Redmond, Washington (2003).