Automatisk generering af unittests fra Equivalence Partitioning Boundary-Value Analysis(EP-BVA) ark

Diplomingeniørprojekt

Asbjørn Normann Stourup
Juni 2015

DTU Space
Institut for Rumforskning og -teknologi

Indholdsfortegnelse

Forord

Dette er et afgangsprojekt udarbejdet af Asbjørn Normann Stourup, elektroingeniørstuderende ved DTU på diplomlinien, vurderet til 15 ECTS points. Det er udviklet i samarbejde med Siemens Flow Instruments med Jesper Esbensen som virksomhedsvejleder og Peter Brauer som DTU vejleder.

SFIs software afdeling skal have et kæmpe tak for den tålmodighed og støtte jeg har modtaget i løbet af dette projekt.

Indledning

Der er stigende efterspørgsel på gennemtestning af embedded software. Det sker pga. opmærksomheden omkring omkostningerne ved at sende utestet eller delvist testet firmware på markedet. Ved at release ugennemtestet FW skabes en høj risiko for, at man sender et produkt på markedet med opstartsproblemer. Disse kan være utroligt dyre at afvikle, da det fx. kan kræve review/debug/test af gammel kode eller endda tilbagerulning af produkter. Ved at sørge for grundige tests af FW helt fra unittests på klasse niveau over hardware-software integrationstests til system tests, minimeres sandsynligheden for fejl og derved de omkostninger som opstår senere i forløbet af produktudviklingen.

I Siemens Flow Instruments udvikles flowmålere, der måler gennemstrømning af bl.a. spildevand, olie og gas. Da fejl i elektronikken i værste fald kan resultere i generering af gnister, er det ekstremt vigtigt at der sker robust fejlhåndtering. Disse flowmålere kan være installeret i fx. olieraffinaderier og i værste fald kan det betyde antændelse olie/gas og derved eksplosionsfare. For at kunne sælge disse flowmålere kræves der derfor et højt niveau af sikkerhed. Dette garanteres ved at opnå certificering efter IEC standarden IEC61508, som definerer SIL inden for kategorierne hardware safety integrity og systematic safety integrity. SIL 3 stiller en række krav til bl.a. traceability af krav, kodereview, test og testreview. Denne certificering garanterer derved en katastrofefrekvens på $\geq 10^{-8}$ til $< 10^{-7}$ per time [1 s. 34], hvilket svarer til 1 katastrofe per 1140 år ved kontinuerlig operation. Mængden af tests og review af samme er stor, og der bruges mange arbejdstimer på dette.

Af disse to hovedårsager er det interessant at kigge på minimering af fejl i definering og implementering af tests. Da der i forvejen hos SFIs SW afdeling er fokus på automatisering gennem Continuous Integration er det derfor oplagt at undersøge muligheden for at kunne generere tests automatisk ud fra et stykke dokumentation. Det kan give en optimering af arbejdsprocessen, især forbundet med test delen.

Akronymliste

SW - Software

FW - Firmware

SFI – Siemens Flow Instruments

SIL – Safety Integrity Level

API – Application Programmable Interface

CFT – Class Function Test

CMPT – Component Test

FT – Functional Testing

SMT – State Machine Testing

Projekt

Projektbeskrivelse

Der ønskes en evaluering og eventuel revidering af det nuværende format af EP-BVA sheets i excel format, som udgangspunkt for at kunne lave en applikation der genererer Class Function Tests for klasser i Rhapsody. Genereringen af CFTs kan deles op i generering af test setup, der indeholder formatet af CFTs og fx. stubbe og afhængigheder, og genereringen af test cases, der indbefatter oprettelsen af hver funktions enkelte test cases.

Projektet udarbejdes i samarbejde med SFI. Den applikation, som implementeres bliver udviklet mhp. en videreudvikling af SFIs SW afdeling. Det vil sige at arbejdsmetoder og værktøjer er begrænsede til dem der benyttes her.

Værktøjer

Til udvikling af applikationen der skal læse EP-BVA ark og generere CFTs findes nogle værktøjer herunder plug-in Application Programmable Interfaces. APIer er SW biblioteker bestående af rutiner, protokoller oa. beregnet til udvikling af SW applikationer. Plug-in APIer gør det muligt for applikationer at kommunikere med andre typer applikationer.

Visual studio

Selve applikationen implementeres i udviklingsmiljøet ydet af Microsoft Visual Studio 13 i sproget C#. Den implementeres som en Windows Form, hvilket bl.a. giver en simpel mulighed for at specificere stien til det EP-BVA ark som CFTen skal genereres fra, og integreret mulighed for kommunikation med Microsoft Excel.

Excel API

Visual Studio besidder en API, der gør det muligt at tilgå informationen i EP-BVA arkene i Excel formatet ".xlsx". Denne API tilføjes som reference i Visual Studio projektet for at tilgå funktionaliteten.

Rhapsody

Rational Rhapsody er et UML modelbaseret udviklingsværktøj til embeddede systemer, der i øjeblikket udvikles af IBM. Det er her CFTerne skal oprettes og placeres korrekt i den eksisterende projektstruktur. Programmeringssproget brugt er C++.

Rhapsody API

IBM har lavet en API der gør det muligt at gå ind i en et Rhapsody projekt og manipulere med struktur og indhold. Denne API refereres som Excel APIen i projektet for at kunne tilgå funktionaliteten.

Cantata

Cantata er et dynamisk unittesting miljø. De test cases som skal genereres kommer primært til at bestå af Cantata macroer.

Planlægning

Udviklingsmetode

Agile software development er en iterativ udviklingsmetode, der giver muligheden for adaptiv planlægning, hvor større opgaver brydes ned i mindre tasks, som fordeles ud over iterationer (sprints). Da projektet kan deles op i 3 primære delopgaver, som kan fordeles på 3 sprints af 3 uger er det et passende format (se tidsplan nedenfor). Derudover indebærer agile metoden Scrum[1] møder, hvilket i dette projekt indebærer korte møder hver formiddag ved et Kanban board (se bilag 1). Kanban boardet benyttes til at overskueliggøre fremgang i projektet og hænger naturligt sammen med sprint planlægningen. Denne metode vælges da det er metoden SFIs SW afdeling følger, og derfor kan udviklingen af dette projekt køre parallelt med de øvrige projekter i afdelingen. Derudover har der fra start været en interesse i projektet og pga. Scrum møder hver dag får alle i afdelingen et indblik i projektet.

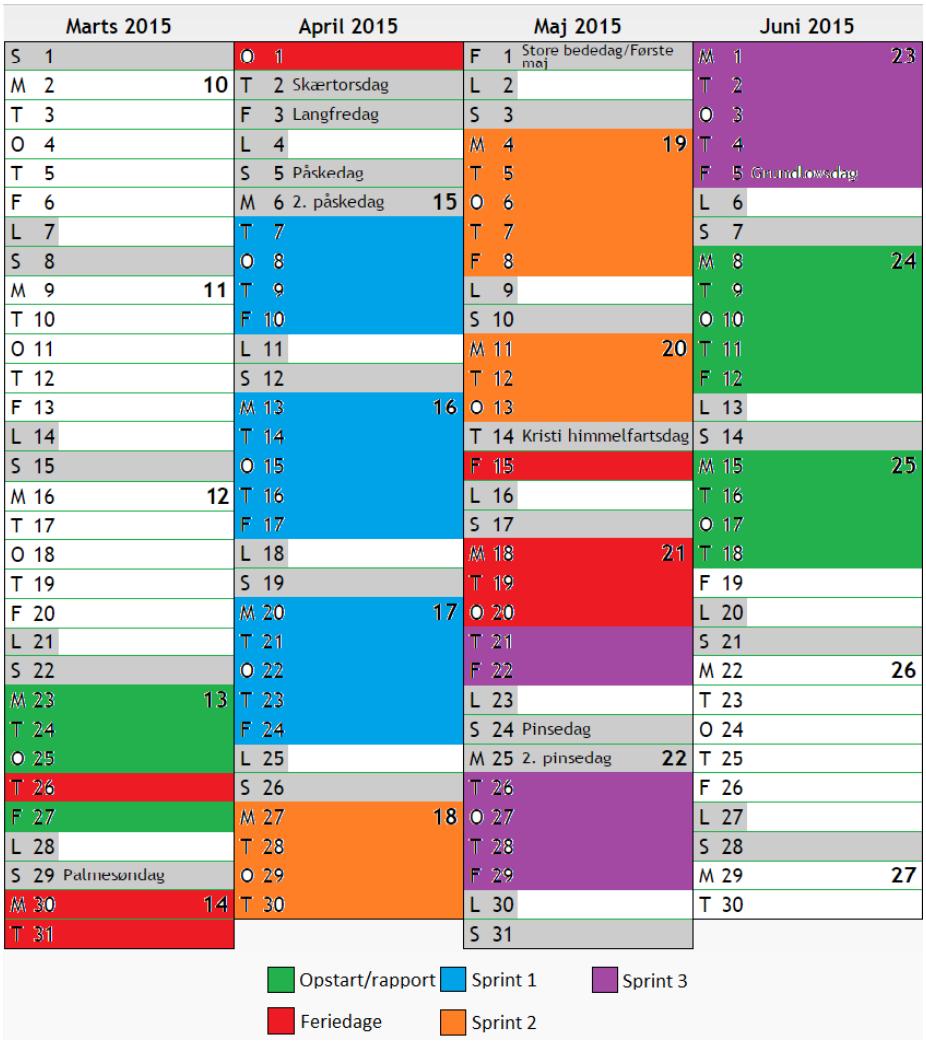
Tidsplan

De 50 dage afvikles over 12 uger. De 12 uger starter med årets uge 13 værende projektets uge 1. Der planlægges 3 sprints af 3 uger, med 1 uges opstart og 2 ugers rapportskrivning.

Projektuge 1 benyttes til indledende research og indsamling af de nødvendige værktøjer.
Projektuge 2-4 (Sprint 1) benyttes til øvelse i SFIs SW afdelings testproces, herunder udarbejdning af et EP-BVA ark til brug af implementering af projektet.
Projektuge 5-7 (Sprint 2) benyttes til øvelse i udviklingsværktøjer herunder API'er benyttet i implementeringen af projekt SW, og udvikling af Test Setup.
Projektuge 8-10 (Sprint 3) benyttes til udvikling af Test Cases.
Projektuge 11-12 benyttes til rapportskrivning.

Se plan på figur 1 nedenfor.

[1] Mødekoncept inden for Agile, hvor man hver dag mødes 15 minutter til en hurtig statusmelding over ens arbejdsopgaver inden for udviklergruppen



Figur 1 Projektplan

Sprintplanlægning

Herudover planlægges mindre opgaver til hver sprint. En dags arbejde defineres som 5 timer, grunden til dette er, at der skal være indberegnet tid til møder oa. i arbejdsdagen. Da jeg ikke som sådan deltog i møder skal de 5 timer nærmere betragtes som en skala mere end et reelt timetal, hvilket burde ligge nærmere 7 timer per dag. Det vil sige at en opgave med estimatet 5 timer helst skulle kunne løses i løbet af en dag, hvis der ikke opstår større komplikationer. Hvis en opgave ikke løses inden for et sprint overgår denne til næste sprint sammen med de nye opgaver. Se figur 2 for sprintplan efter endt forløb, nedenfor.

	A	B	C	D	E	F	G	H
7	**78**		Analysis of EP-BVA sheets (NOG/KHE comparison)	EXP	1		10	
8	**79**		Redesign of EP-BVA sheet	EXP	1		10	
9	**80**		Write report	EXP	3		10	
10				Sprint 1 task hours:			70	
11	**Exam project - Asbjørn, Sprint 2**							
12	**73**		Rhapsody API practice	EXP	1		15	
13	**74**		Test setup implementation	EXP	2		30	
14	**75**		Analysis of EP-BVA sheets (NOG/KHE comparison)	EXP	1		2	
15	**76**		Redesign of EP-BVA sheet	EXP	2		7	
16	**77**		Write report	EXP	3		10	
17				Sprint 2 task hours:			64	
18	**Exam project - Asbjørn, Sprint 3**							
19	**78**		Implement inheritance (generalizations, test setup continua	EXP	1		2	
20	**79**		Investigate tester priorities for test case generation	EXP	2		1	
21	**80**		EP-BVA draft presentation	EXP	2		1	
22	**81**		Implement test case generation	EXP	3		40	
23	**82**		Project result presentation	EXP	4		1	
24	**83**		Write report	EXP	4		10	
25				Sprint 3 task hours:			55	
26	**Exam project - Asbjørn, last 1½ week**							
27	**#**		Write report	EXP	1		40	
28								
29				Total task hours:			249	
30								

Figur 2 Endelig sprintplan
NB: Nogle tasks er overgået til næste sprint, bl.a. analyse/redesign af EP-BVA og implementering af nedarvning i test setup

Projektafgrænsning

EP-BVA syntaks:

Idet EP-BVA arkene laves af forskellige testere kan der være en stor varians i hvorledes disse udfyldes. Det vil sige for at kunne implementere CFTs fuldstændigt kræver det et omfattende regelsæt for EP-BVA syntaksen. Der foreligger ikke et sådan regelsæt, og det vil blive udeladt i dette projekt af tidsmæssige hensyn. Dette betyder desuden at CFTs ikke vil blive fuldstændigt implementeret fra applikationen, men nærmere skelettet af disse.

Funktionelle tests og state machine testing:

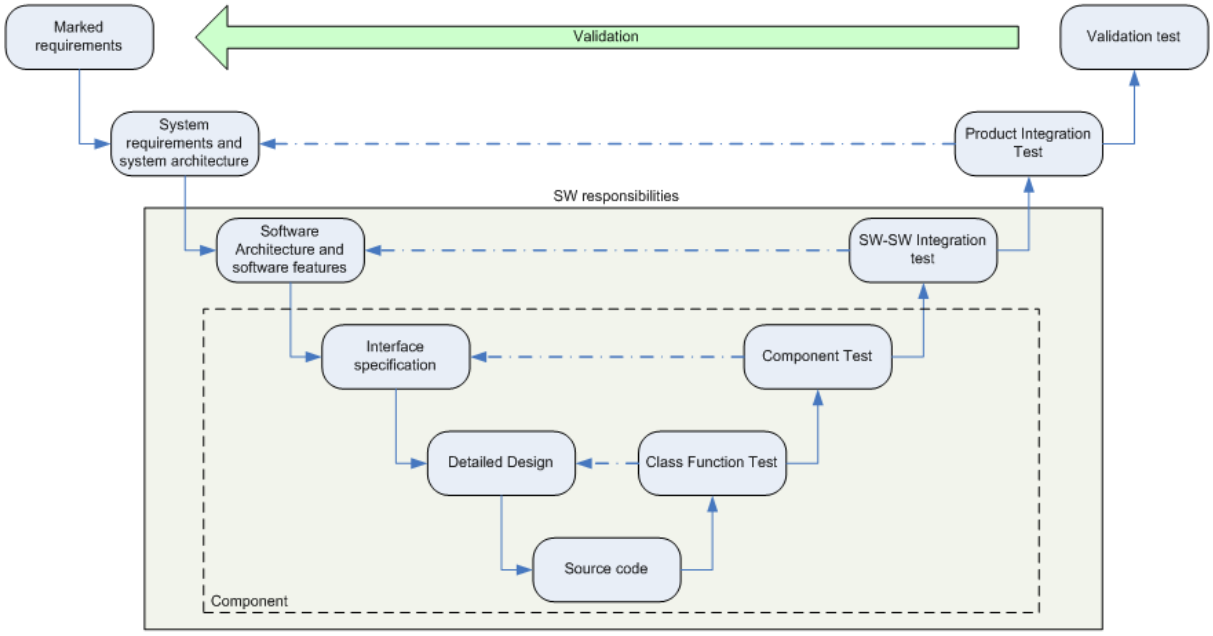
FT og SMT er en del af formatet på de eksisterende EP-BVA ark og vil blive udeladt af projektet af tidsmæssige hensyn.

Indledende forberedelse

For at kunne implementere en applikation der indgår i testprocessen kræver det indsigt i samme. Følgende afsnit vil afdække testprocessen samt øvelserne løst i forbindelse med erhvervelsen af viden om emnet.

Unit testing

Unit testing er test af SW moduler på lavest mulige niveau. Det vil sige test af klasser på funktionsniveau, altså CFT. Disse danner grundlag for de mere omfattende component tests, der dog ikke vil blive berørt i dette projekt. Udviklingen hos SFI følger V-modellen, hvor unittest(CFT) på figur 3 herunder ses som det laveste niveau på test siden.



Figur 3 V-modellen

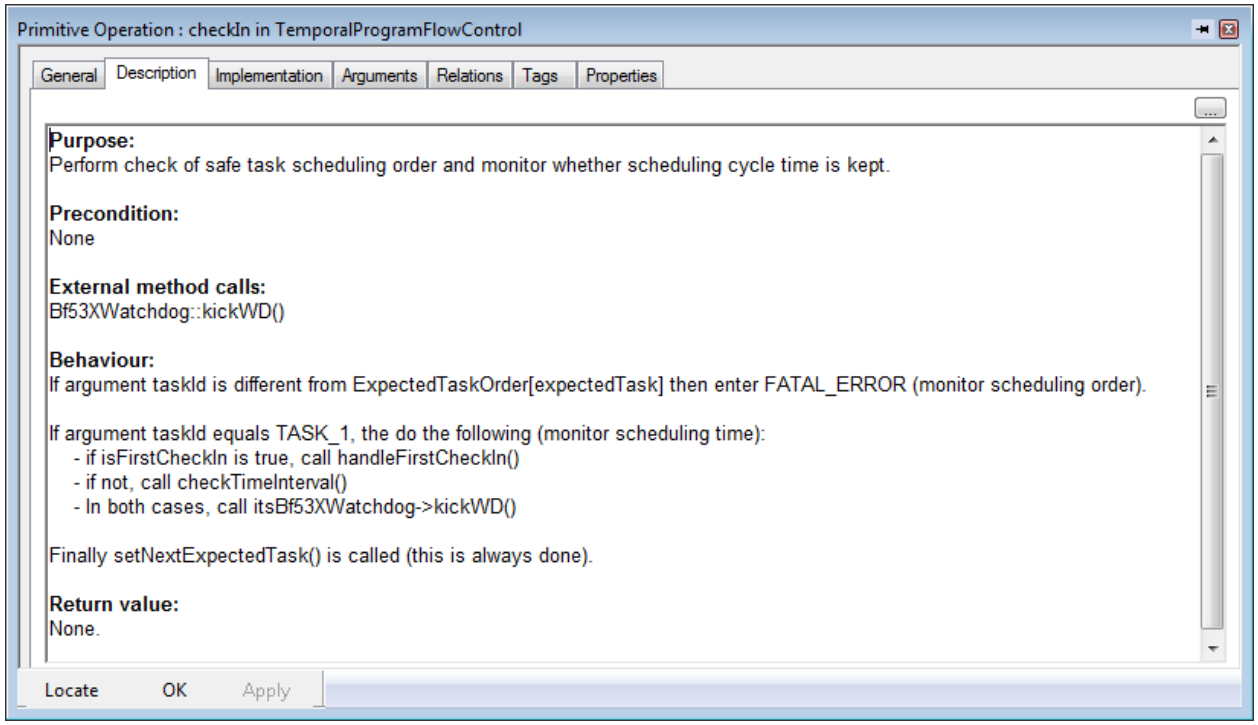
CFT er en effektiv måde at opnå høj code coverage, hvilket er krævet pga. det ønskede SIL 3. Disse CFTer er implementeret som black box testing, dette vil sige at man anser koden i klasserne for ukendt og tester denne kode ved at teste med et vilkårligt input og sammenligne med det forventede output.

Testproces

SW udviklingen i forbindelse med V-modellen hos SFI er test-dreven. Det vil sige klasser og komponenter udvikles mhp. at skulle gennemgå prædefinerede tests. Fra beskrivelser af funktionerne (se figur 4) på disse klasser laves EP-BVAerne (Se bilag 2 for TemporalProgramFlowControl EP-BVA). Fra disse EP-BVAer designes CFTerne. Følgende er et eksempel på den beskrevne proces med udgangspunkt i funktionen checkIn på klassen TemporalProgramFlowControl.

EP-BVA fra beskrivelse

På figur 4 herunder ses beskrivelsen af funktionen checkIn. Første linje i behaviour afsnittet repræsenterer i dette tilfælde EP 2, for input parameteren taskId.



Figur 4 Beskrivelse af funktionen checkIn på klassen TemporalProgramFlowControl

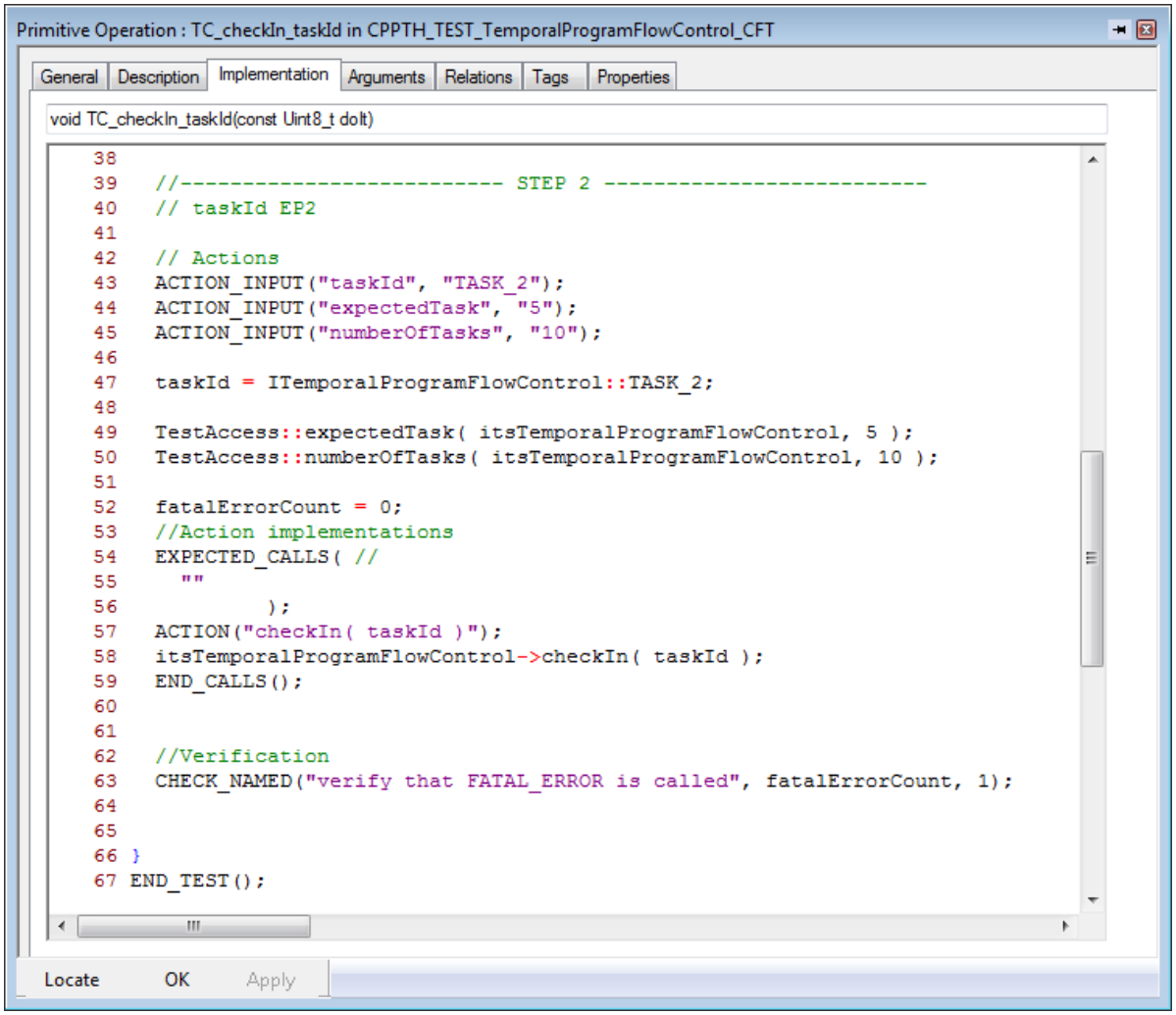
De første 2 celler giver sig selv. Type cellen angiver at parameteren er et input argument på funktionen, og EP cellen angiver at det er den anden EP for funktionen checkIn med test parameteren taskId. EP Low cellen udfyldes med den test condition som er beskrevet i funktionsbeskrivelsens første linje under behaviour, altså hvis taskId er forskellig fra den forventede task. Da der ikke skal laves en BVA for denne EP efterlades EP Upper cellen tom. Test Value cellen sættes til en arbitrær gyldig værdi, i dette tilfælde TASK_2. Dependency cellen angiver hvilke forhold der skal være tilstede for at testen overhovedet kan lykkes, i dette tilfælde refereres der til dependency for EP1(Se bilag 2). Til sidst findes Behaviour cellen der beskriver det forventede output, her en simpel FATAL_ERROR.

3	Function	Parameter	Type	EP	EP Low	EP Upper	Test Value	Dependency
30	checkIn	taskId	Argument	EP2	taskId != ExpectedTaskOrder[expectedTask]		TASK_2	same as taskId EP1
31	Behaviour				FATAL_ERROR			

Figur 5 EP1 for taskID variablen af TemporalProgramFlowControl EP-BVA

CFT fra EP-BVA

Step 1 på figur 6 herunder viser test casen for den beskrevne EP i figur 5. Test casen består af Cantata macroer og kommentarer. ACTION_INPUT angiver de beskrevne dependencies. Resten af Actions blokken indebærer funktionskald til stubbe og variabel angivelse. CHECK_NAMED angiver kontrol af det forventede output.



Figur 6 TemporalProgramFlowControl checkIn test case uddrag

Design

Eksisterende EP-BVA layout

SFIs EP-BVA sheets (se figur 7) er overordnet delt op i en EP og en BVA del. EP delen i venstre side der udledes ved analyse af klassernes funktionsbeskrivelser, og BVA delen i højre side der udledes af EP delen. Klassens EP-BVA af funktionerne indtastes nedefter efter hinanden.

2	Class:	SignalSearch							BVA for EP Low			BVA for EP Upper		
3	Function	Parameter	Type	EP	EP Low	EP Upper	Test Value	Dependency	BVL	BVM	BVU	BVL	BVM	BVU
22	countWindows													
23	countWindows	winCounter[4]	Indirect	EP1	*	4	1	winToBeAnalyzed = 2 path = 4		*		3	4	
24	Behaviour				winToBeAnalyzed = 1 winCounter[4] = 0					EP1		EP1	EP1	EP2
25	countWindows	winCounter[4]	Indirect	EP2	5	*	30	same as winCounter[4] EP1	4	5	6		*	
26	Behaviour				winToBeAnalyzed = 5 winCounter[4] = 25				EP1	EP2	EP2		EP2	

Figur 7 SFI NOG EP-BVA layout

EP:

Den sorte header angiver hvilke værdier der skal indtastes i de tilsvarende felter nedefter og klassens navn er angivet i toppen. Hver anden række angivet med behaviour angiver de forventede testresultater for hver enkelt EP.
Den gule header angiver hvilken funktion der behandles, og er implementeret for at øge læsbarheden af formatet. Hele EP-BVAen for klassen er lavet i en tab, og fortsætter derfor nedefter.

BVA:

Her analyseres de respektive EPers behaviour ved grænseværdierne. Værdierne der indsættes er de for EP Lower og Upper omkringliggende værdier.

Der foreligger EP-BVA sheet i et andet layout (SFI Karlsruhe). Dette bliver brugt til delvis autogenerering af testkode. Da det ville kræve flere ressourcer at sætte sig ind i hvordan layoutet fungerer og hvordan et 3. parts program konverterer dette format til testkode i Rhapsody forkastes dette og et EP-BVA layout vælges baseret på SFI NOG layoutet.

Nyt EP-BVA layout

Overvejelserne ifm. et redesign af EP-BVA layoutet gik primært på overskuelighed og genkendelighed og sekundært på, at layoutet skulle ligge direkte op til implementeringen af applikationen.

Da der kan forekomme klasser med flere hundrede linjers EP-BVA splittes funktionerne ud på individuelle tabs (Se bilag 3 for nyt layout) for at øge læsbarheden. Det betyder at funktionsnavnet erstatter klassenavnet i toppen af den sorte header (se figur 8). Den gule funktionsheader erstattes med angivelse af EP. De to ting til sammen betyder at søjlerne for funktionsnavn og EP udgår. For at lette implementering af applikationen tilføjes en søjle i forlængelse af BVA sektionen, der angiver om der skal oprettes test case steps for BVA.

	A	B	C	D	E	F	H	I	J	K	L	M	N
1	**Equivalence partitions**						**Boundary Value Analysis**						
2	**Function**	init					**BVA for EP Low**			**BVA for EP Upper**			
3	**Parameter**	**Type**	**EP Low**	**EP Upper**	**Test Value**	**Dependency**	**BVL**	**BVM**	**BVU**	**BVL**	**BVM**	**BVU**	**BVA testing**
4				**EP1**									
5	numTasks	Argument	*	20	15	Bf53X::getCCLK() returns 200.000.000 numberOfTasks = 0 cycleLimitMinTime = 0 cycleLimitMaxTime = 0		*		19	20	21	X
6	Behaviour			numberOfTasks = numTasks cycleLimitMinTime = 1950000 cycleLimitMaxTime = 2050000				EP1		EP1	EP1	EP2	
7				**EP2**									
8	numTasks	Argument	21	*	100	same as numTasks EP1	20	21	22		*		X
9	Behaviour			numberOfTasks = 20 cycleLimitMinTime = 1950000 cycleLimitMaxTime = 2050000			EP1	EP2	EP2		EP2		

Figur 8 Nyt EP-BVA layout

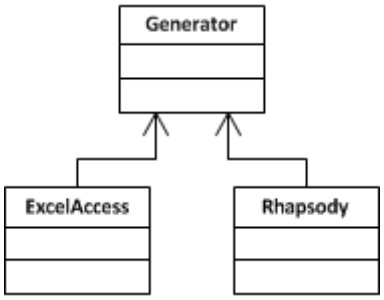
Syntaks

Ifm. implementering af applikationen er der tilføjet få syntaktiske regler, der sørger for at inputtet kan tolkes korrekt på funktions arkene:

1) Celle A1(Header): Der *skal* stå Equivalence Partitions. Dette bliver brugt til at kontrollere om der er flere funktioner der skal inkluderes i CFTen.
2) Celle B1(Function): Det præcise funktionsnavn på den funktion der testes. Bruges til opretning af test case operationer i Rhapsody.
3) Celle A5(Parameter): Det præcise navn på den parameter der testes i funktionen. Bruges til opretning af test case operationer i Rhapsody.
4) Celle F5(Dependency): Variabel afhængigheder skal indeholde '=' for at skelne variabel navne fra værdi. Operations afhængigheder skal indeholde "returns" for at skelne operations navne fra retur værdien.
5) Celle D6(Behaviour): Samme som celle F5.

Applikations SW

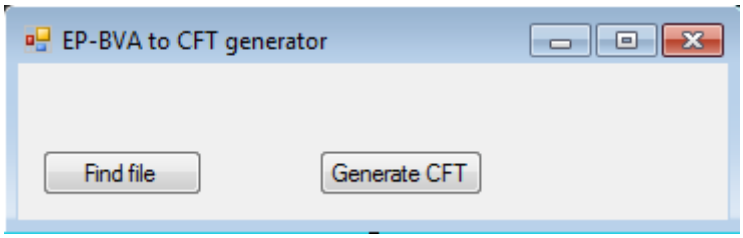
Applikationen designes som en Windows Form. Dette gør at funktionaliteten til finde EP-BVA filens sti er lettilgængelig. Derudover tilføjes en konsol til bl.a. fejlbeskeder. Det er muligt at erstatte dette med en logfil, hvilket ville være oplagt ifm. automatiseringsprocessen. Der oprettes en klasse til hver API der står for at håndtere kommunikation med hhv. Excel og Rhapsody (Se figur 9). Klassen Generator administrerer disse.



Figur 9 Forholdet mellem kommunikationsklasserne ExcelAccess og Rhapsody, og Generator klassen.

Implementering

Applikationen implementeres i udviklingsmiljøet ydet af Microsoft Visual Studio 13 i sproget C#. Den implementeres som en meget simpel Windows Form, med en filebrowser knap og en genererings knap (se figur 10).



Figur 10 Windows form brugeren bliver præsenteret for

ExcelAccess

Kommunikationen med EP-BVA arkene faciliteres af ExcelAccess klassen.

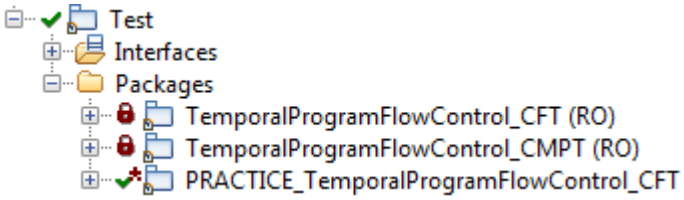
Funktionen *openExcelWorkbook(String path)* modtager en filsti som string argument og åbner i en try catch, forbindelsen til dette dokument. Derefter sættes et aktivt worksheet og funktionen returnerer true, hvis der ikke er forekommet en exception.

Funktionen *closeExcel()* lukker forbindelsen.

Funktionen *getSheetCellData(int row, int row, int sheetIndex)* modtager række, søjle og ark indeks som int argumenter og returnerer teksten fra den pågældende celle. Hvis ark indekset er 0 udtrækkes celletekst fra Stubs arket.

Rhapsody

Kommunikation med Rhapsody modellen faciliteres af Rhapsody klassen. For at få forbindelse til Rhapsody modellen kræver det at den pågældende model der skal arbejdes med er åbnet i Rhapsody udviklingsmiljøet med administrator rettigheder. Normalt i SW udviklingen hos SFI forekommer der striks versionskontrol ved ClearCase. Dette omgås ved at oprette en test mappe med præfixet ”PRACTICE” (se figur 11), som der arbejdes med.



Figur 11 Rhapsody testmappen i folderstrukturen

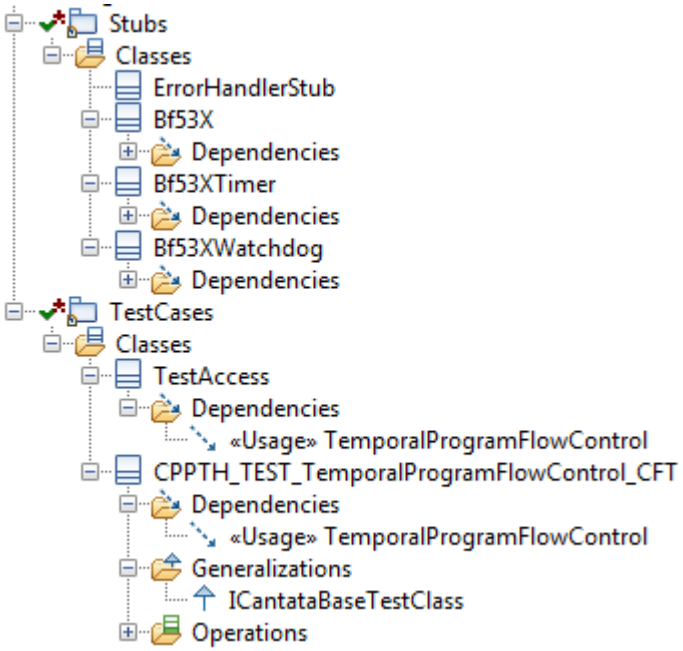
Derudover bemærkes det at samtlige kald til Rhapsody modellen foretages i try-catches. Dette sker da Visual Studio kaster en exception bl.a. hvis man forsøger at oprette elementer der findes i forvejen, dvs. samme type elementer på samme niveau i folderstrukturen, med ens navne.

Funktionen *openProject()* forsøger at forbinde til den åbne Rhapsody model og gemmer et projekt objekt der arbejdes på. Returnerer true hvis der ikke kastes exception.

Funktionen *makeClass(IRPPackage classPackage, String className)* forsøger at oprette en klasse, med navnet className, i den classPackage den modtager som input argument. Den nyoprettede klasse returneres. Bruges til at oprette stubbe og test cases.

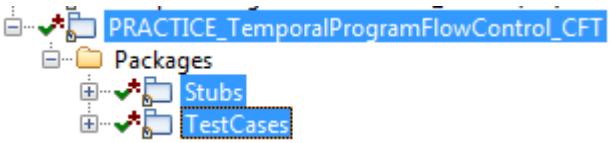
Funktionen *makeDependency(IRPClass derivedClass, String baseClassName)* forsøger at oprette en dependency mellem en klasse med navnet baseClassName og klassen derivedClass. Bruges i oprettelse af stubbe og TEST samt testAccess klasserne (se figur 12).

Funktionen *makeGeneralization(IRPClass derivedClass, String testInterfaceName)* forsøger at oprette en generalization mellem derivedClass og et test interface med navnet angivet i den string angivet som inputargument. Bruges i oprettelse af stubbe og TEST samt testAccess klasserne.



Figur 12 Implementerede dependencies og generalizations

Funktionen *confirmMakeCFTPackages(String className)* forsøger at fastslå om klassen der skal testes CFT mappe eksisterer. Derudover oprettes Stubs og TestCases mapperne (se figur 13), hvis de ikke eksisterer.



Figur 13 Folderstrukturen for klassen under tests CFT mappe

Funktionen *getStubsOrTCPackage(String className, String packageName)* forsøger at finde en mappe med navnet packageName i CFT mappen. Bruges i forbindelse med oprettelse af stubbe og test cases.

Generator

Generator klassen benytter excelAccess til at hente celledata fra EP-BVA arkene og benytte Rhapsody klassen til at generere elementer i Rhapsody modellen ud fra disse.

Funktionen *generateCFTs()* kalder funktionerne *buildDictionary()*, *generateStubs(int row)* og *generateTestCases()*. Den bliver eksekveret ved tryk på Generate CFT knappen på popup vinduet (se figur 10).

Test setup

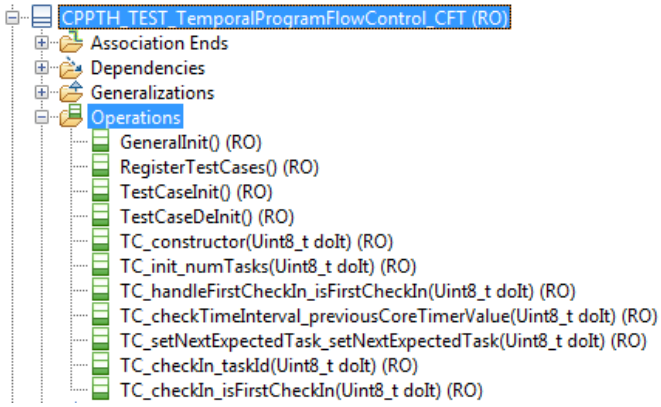
Test setuppet indbefatter generering af stubs. Dette sker ved funktionen *generateStubs(int row)*. Denne funktion benytter en while løkke til at gå igennem felterne i søjlerne A og B startende fra rækken angivet ved inputargumentet *row*, som bliver inkrementeret ved hvert gennemløb for at sikre linjeskift i celle aflæsningen. En if blok i while løkken tjekker når værdien #END (se figur 14) indlæses for at afslutte løkken.

	A	B
1	**Stubs to generate**	
2	**Test class:**	TemporalProgramFlowControl
3	**Component name:**	**Stub name:**
4	ErrorHandler	ErrorHandlerStub
5	Bf53X	Bf53X
6	Bf53XTimer	Bf53XTimer
7	Bf53XWatchdog	Bf53XWatchdog
8	#END	

Figur 14 Arket benyttet til stubgenerering

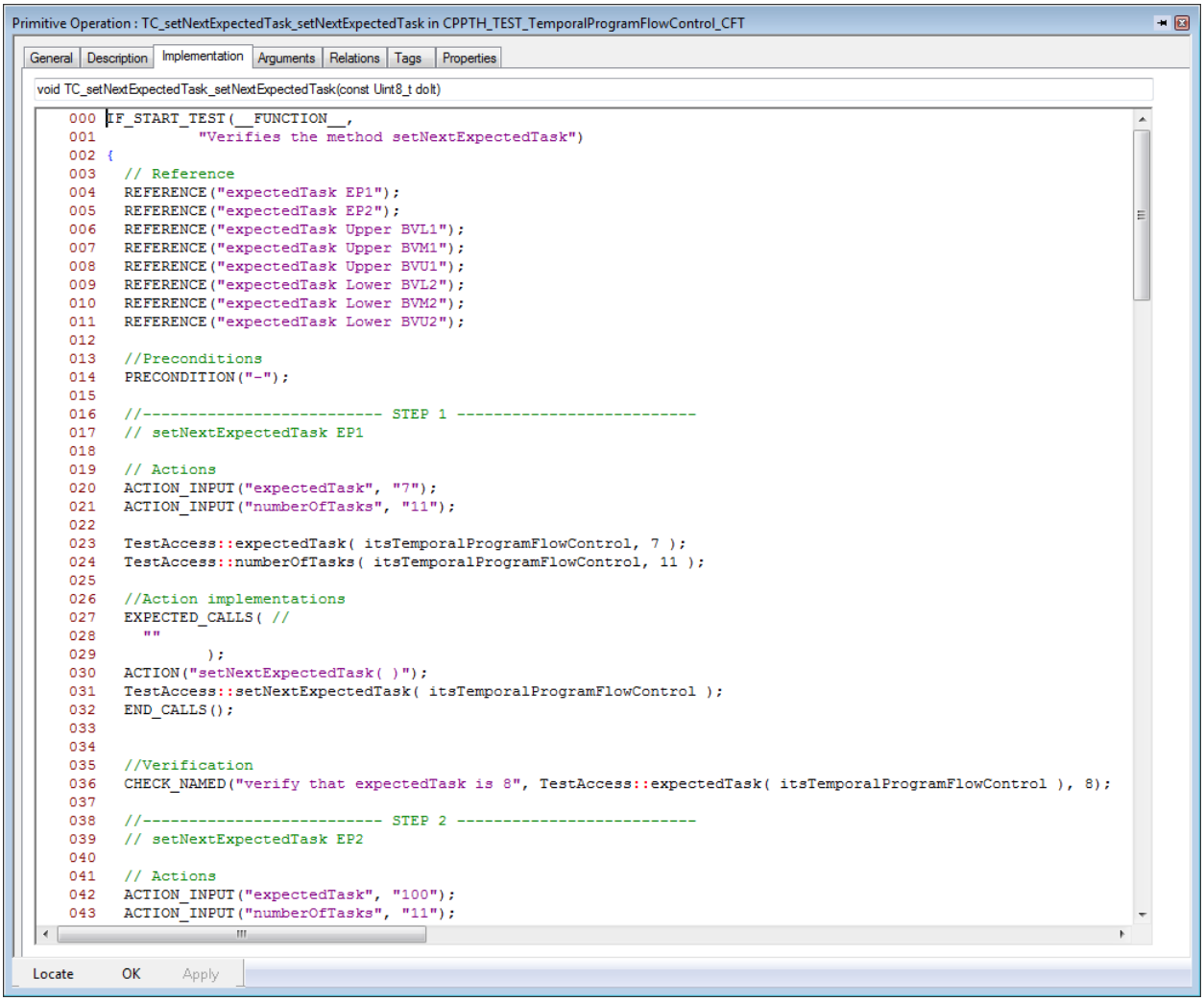
Test case

Funktionen *generateTestCases()* står for generering af test cases. Dette sker ved funktionen *makeTCOperations(int row, int col)*. Test cases oprettes som operationer på den pågældende test klasse i Rhapsody modellen (se figur 15).



Figur 15 Placering af test case operationer

Disse test cases har et format som skal overholdes (se figur 16) og det sker ved at sætte strings sammen for til sidst at lave en stor string der angives som implementering for den aktuelle test case.



Figur 16 Format af test case. Bemærk opdelingen i initiering blok og efterfølgende steps.

Navigering i EP-BVA arkets celler sker ved at benytte et prædefineret dictionary med cellenavn som nøgle og celle objekter som data der indeholder et offset i koordinater. Disse offset koordinater angiver hvor cellen findes i arket i forhold til hvor den aktuelle EP header findes.

Funktionen *makeTCOperations(int row, int col)* fungerer ved at først at oprette operationer i Rhapsody modellen med navnene GeneralInit, testCaseInit og testCaseDeInit. Disse tre operationer oprettes statisk hver gang. Herudover oprettes operationen RegisterTestCases, som er en operation der opremser samtlige test cases. Denne implementeres sidst i programmet når alle test cases er oprettede. Selve dataudtrækningen fra EP-BVA arkene foregår i en while løkke. I starten af løkken testes den overordnede header for strengen ”Equivalence partitions”, hvis denne ikke findes antages det at det aktuelle ark ikke er et EP-BVA ark og løkken afsluttes. Herefter testes om den aktuelle EP header indeholder strengen ”EP” hvis ikke så antages det at alle EPer er gennemgået og programmet inkrementerer en variabel for at komme videre til næste EP-BVA ark.
Hvert gennemløb i while løkken gennemgår en EP og opretter en test case ud fra denne. Herunder oprettes steps til BVAerne hvis de tilsvarende genereringsceller er markeret (se figur 17).

H	I	J	K	L	M	N
Boundary Value Analysis						
	BVA for EP Low			**BVA for EP Upper**		
BVL	**BVM**	**BVU**	**BVL**	**BVM**	**BVU**	**BVA testing**
	*		17	18	19	X
	EP1		EP1	EP1	EP2	
18	19	20		*		X
EP1	EP2	EP2		EP2		

Figur 17 BVA layout i EP-BVA arkene

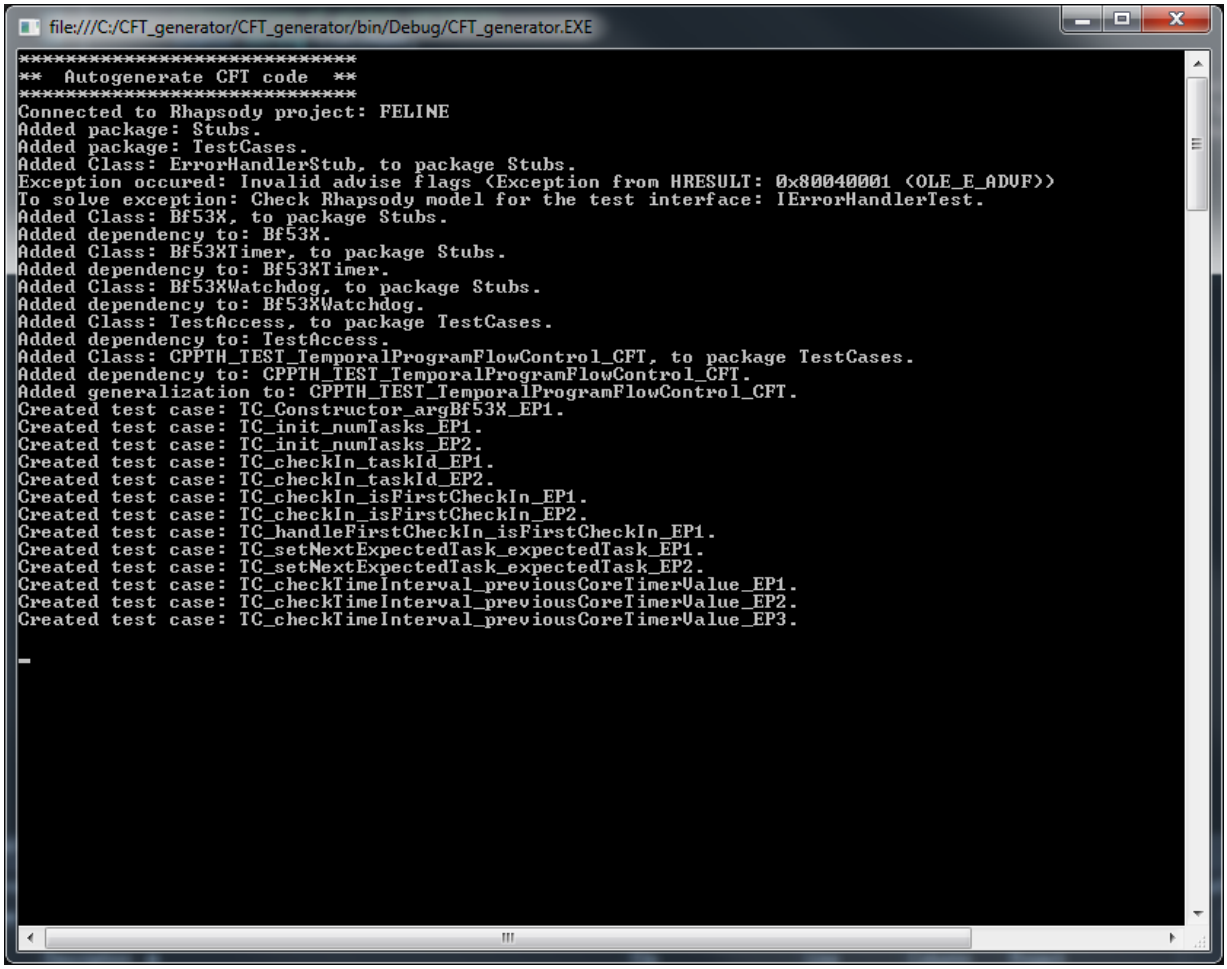
Til sidst i funktionen samles alle de strenge der er blevet oprettet og sammensat af celledata og implementeres i den aktuelle test case (se figur 18).

```
195     //String append and implement
196     outString = beginString + referenceString + preconString + actionString + verifyString + BVAString + endString;
197     currentOp.body = outString;
```

Figur 18 Streng sammensætning og implementering

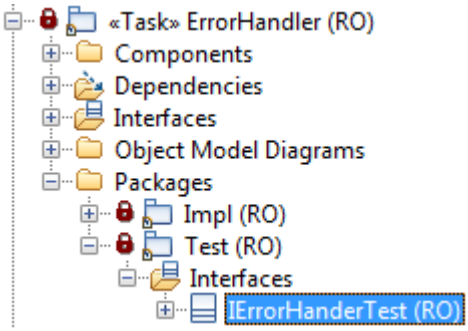
Test og resultater

Ved kørsel af applikationen uden introduktion af fejl i EP-BVA arket fremkommer en exception (se figur 19) i oprettelsen af en generalization. Dette forudsætter at target rodmappen (i dette tilfælde PRACTICE_TemporalProgramFlowControl_CFT) eksisterer.



Figur 19 Standard kørsel af program

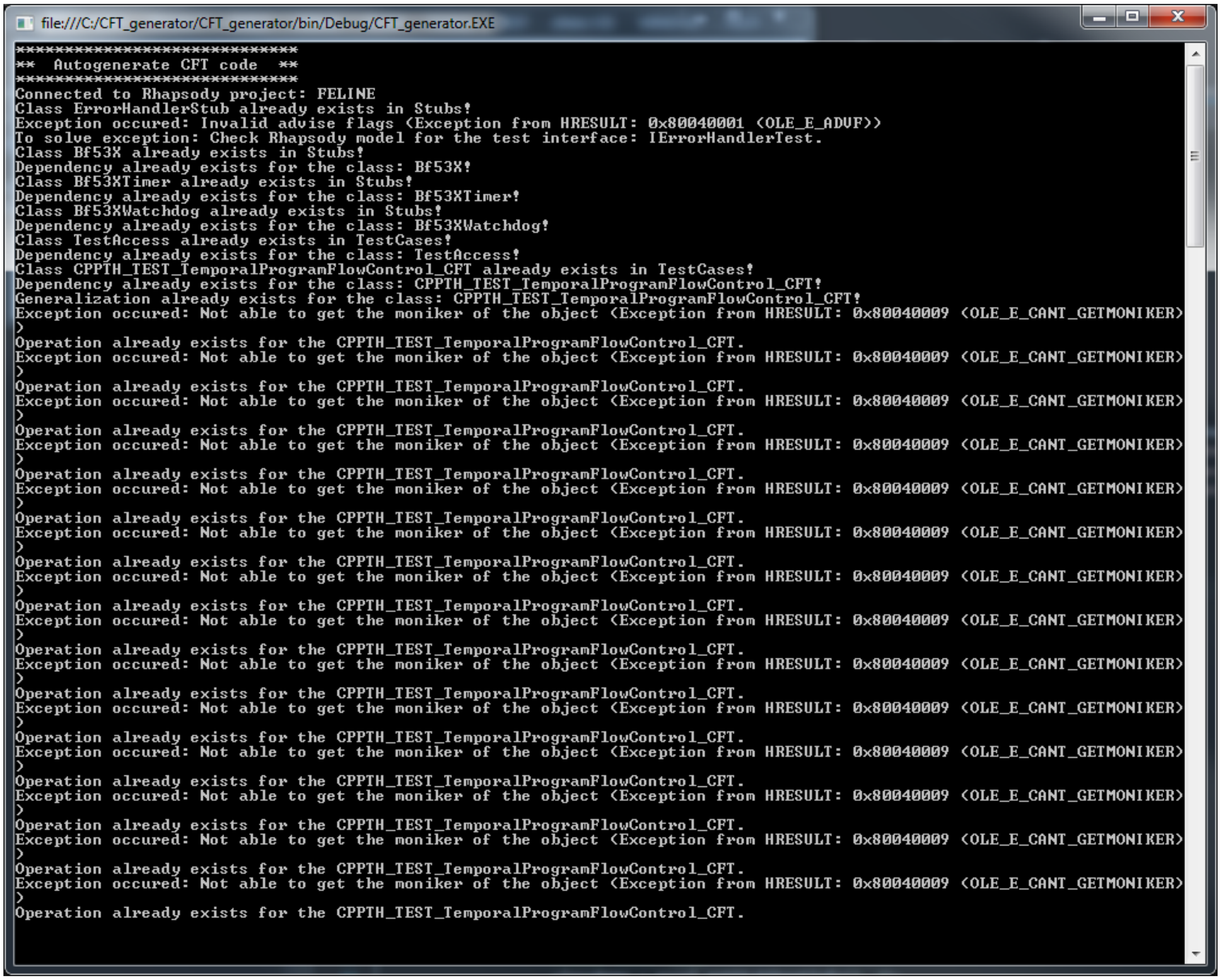
Det drejer sig om test interfacet IErrorHandlerTest. Ved at finde dette interface i Rhapsody modellen afsløres fejlen som værende en slåfejl i navnet. Det viser sig at det indtastede navn er IErrorHanderTest.



Figur 20 Slåfejl i interface navn

Ud over dette oprettes alle klasser inklusiv stubbe og test cases korrekt med dertilhørende dependencies og generalizations (Se bilag 4).

Ved kørsel af applikationen med de eksisterende genererede elementer fremkommer en længere liste af exceptions (se figur 21).



Figur 21 Exceptions ved forsøg på overskrivning af elementer

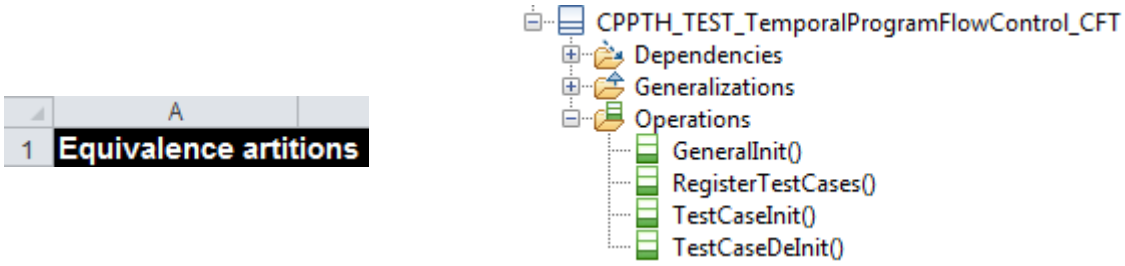
Det ses at det ikke er muligt at overskrive elementer. Dvs. for at rette i disse kræver det sletning og efterfølgende generering.

Brud på syntaksregler

Følgende er en demonstration af, hvad konsekvensen er ved brud af de beskrevne syntaksregler.

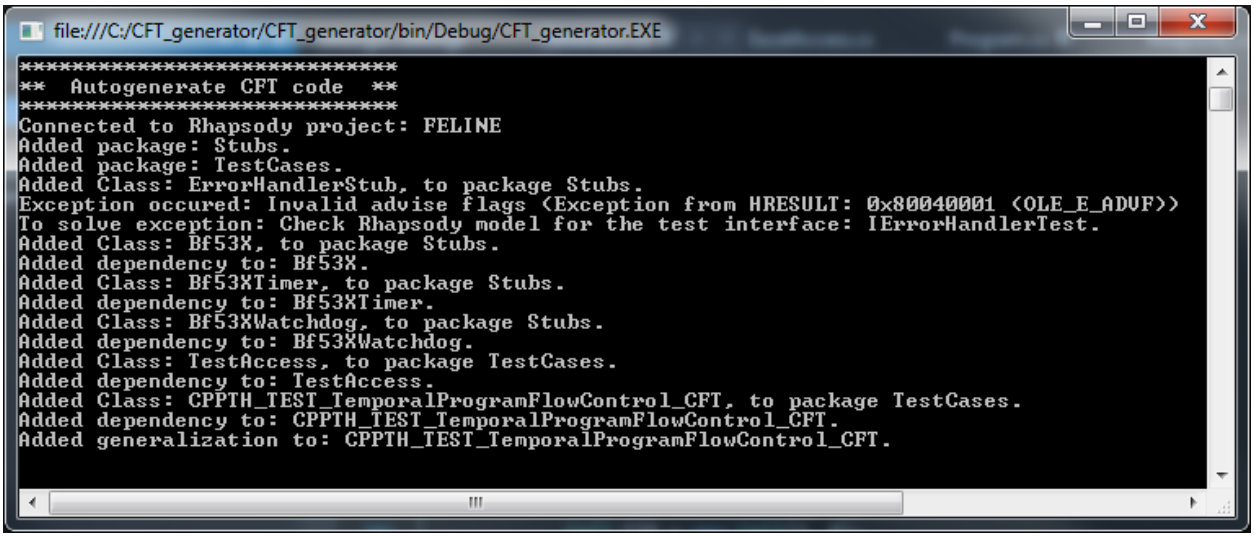
Regel 1

Ved at ændre top headeren i EP-BVA arket (se figur 22) for Constructoren, forventes den while loop der sørger for gennemløb af excel tabsne stoppe. Dette betyder at ingen test cases bliver dannet (se figur 22) da Constructoren er den første funktion.



Figur 22 Tv. Rettelse i EP-BVA ark. Th. resultat i Rhapsody

Derudover forventes konsollen ikke at komme med fejlbeskeder(se figur 23).



Figur 23 Screendump af konsol output

Regel 2/3

Ved at ændre funktions-/parameternavne (se figur 24) forventes det at de respektive test cases bliver succesfuldt oprettede, dog med den fejl at disse navne har fået overført fejlen i navnet

Function	setNextExpectedask
Parameter	**Type**
expectedTask	Indirect
Behaviour	
expectedask	Indirect
Behaviour	

Figur 24 Rettelse i EP-BVA ark

```
IF_START_TEST (__FUNCTION__,
          "Verifies the method setNextExpectedask")
{
 // Actions
 ACTION_INPUT("expectedask", "75");
```

Figur 25 Resultat i Rhapsody

```
Created test case: TC_setNextExpectedask_expectedTask_EP1.
Created test case: TC_setNextExpectedask_expectedask_EP2.
```

Figur 26 Screendump af konsol output

Regel 4/5

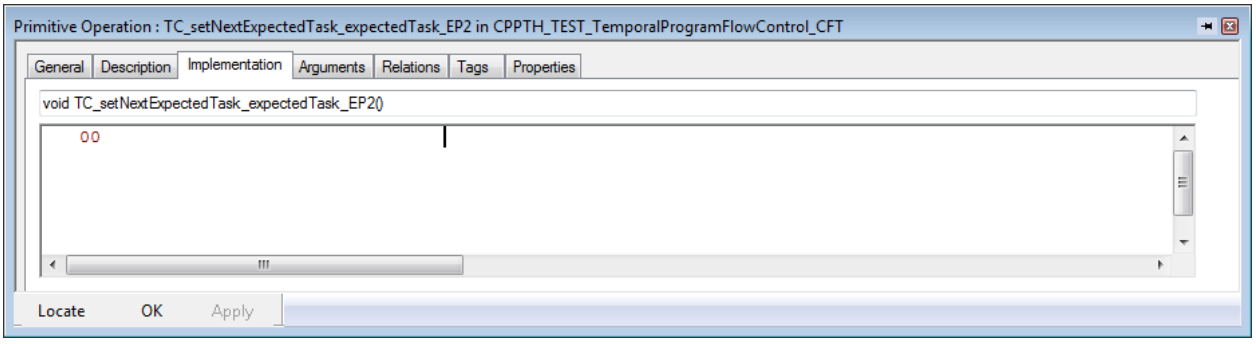
Ved at ændre '=' til "equals" forventes det at applikationen smider en array out of bounds exception. Dette gør at selve test case funktionen bliver oprettet i Rhapsody, men den string som skulle indsættes som implementering af test casen kommer til ikke at indeholde noget.

*	75	numberOfTasks equals 20
expectedTask equals 0		

Figur 27 Rettelse i EP-BVA ark

```
Exception occured: Index was outside the bounds of the array.
To solve: Check the "Dependency" cell in the EP-BVA sheet for errors(syntax requires complete statements).
```

Figur 28 Screendump af konsol output



Figur 29 Resultat i Rhapsody

Konklusion og videre udvikling

I det følgende konkluderes projektets resultater og der perspektiveres til videre udvikling med applikationen.

Konklusion

Formålet med dette projekt var at planlægge, designe og udvikle en applikation til at generere unittests, i form af CFTer, ud fra EP-BVA ark i samarbejde med SFI mhp. en optimering af en omfattende testprocess involveret i det høje niveau af sikkerhed der kræves i SIL 3 produkter. Denne applikation med dertilhørende EP-BVA layout forventes at ligge til grund for en videreudvikling der i sidste ende vil betyde en implementering i et automatisk byggemiljø i SFIs udviklingsprocess.

Det lykkedes at få applikationen til at trække data ud fra excel arkenes relevante celler. Derudover lykkedes det at få applikationen til at manipulere en åben Rhapsody model. Dette betyder at applikationen, med et EP-BVA i det rette format, er i stand til at generere en række operationer, klasser og afhængigheder mellem disse som ellers skulle oprettes manuelt. Det er muligt at få feedback i form af en konsol. For at kunne implementere dette i et automatisk byggemiljø kræver det i stedet en form for logning i .txt format eller andet.

Videreudvikling

Forbedringer

Åbenlyse forbedringer involverer implementering af generering af FT og SMT, da disse er udeladt. Derudover kan redundansen i de testcases med identiske BVAer skæres væk. Det sker ved at ændre kodestrukturen i indlæsning af informationer fra excel arkene. Ifm. BVA kunne Verify delen af test casen laves så den reference til EP behaviour der findes i BVA behaviour cellerne udnyttes. En anden forbedring kunne være at lave en excel macro til at køre applikationen inde fra EP-BVA arkene således at man ikke behøver åbne andet end disse for at generere koden.

100% kode generering

For at få en 100% code coverage i generering af CFTer ville det være oplagt at lave en grundig overhaul af EP-BVA arkets format for derved at kunne opstille nogle syntaktiske regler. Dette ville lægge fundamentet for et EP-BVA ark, der kan bruges til at generere al kode til disse CFTer. Derefter ville det være enkelt at implementere applikationen i et automatisk byggemiljø for derved at automatisere processen fuldt ud. Der vil ved denne løsning være et behov for omfattende review og test af den kode der bliver genereret for at kunne verificere at denne fortsat fungerer efter hensigten og overholder de strenge krav. Dog betyder det også at man vil kunne spare arbejdstimer på længere sigt ifm. implementering af tests og review af tests.

EP-BVA generering

Da dokumentationen er en vigtig del af udviklingen, især til SIL 3 produkter, kan man lave en generering af EP-BVA indholdet fra test koden. Ved at benytte applikationen til at generere skelettet af CFTerne kan man efterfølgende gå ind og udfylde disse tests med testkode (Se figur 26 for skitse). Ved at udvide applikationen med en EP-BVA update funktionalitet kan man udfylde EP-BVA arkene med den kritiske

information om EP og BVA forholdene fra Rhapsody. Dette vil være en let og sikker måde at kunne lave og argumentere for at dokumentationen genereret overholder de strenge kvalitetskrav. En ulempe ved dette ville være at der stadigt ligger en mængde arbejde i manuelt at udfylde test cases med kode.

Litteraturliste

[1]IEC 61058-1: Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General Requirements, Edition 2.0, 2010, International Electrotechnical Comission

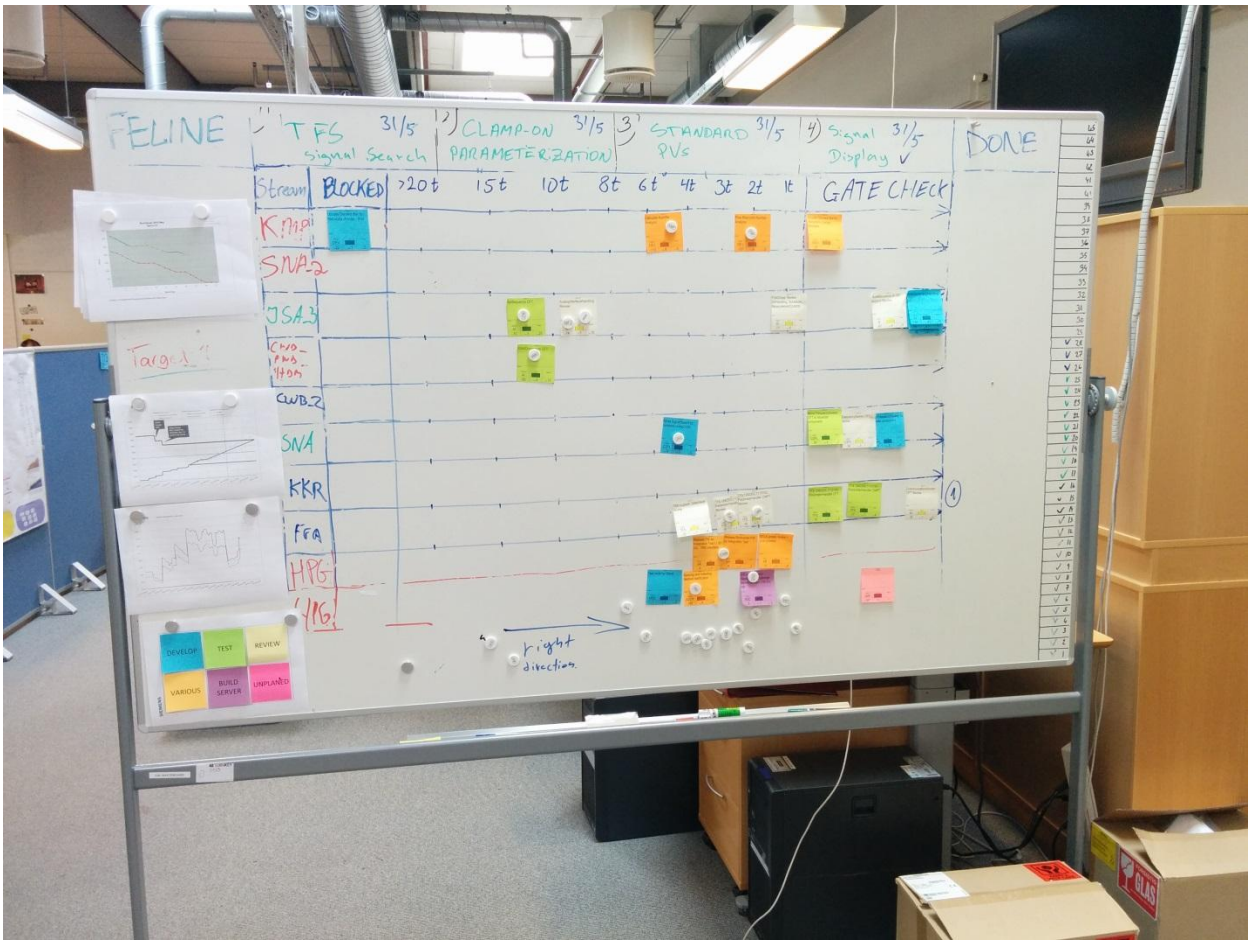
[2]Rational Rhapsody API Reference Manual, 1997, IBM
https://www.ibm.com/developerworks/community/blogs/a8b06f94-c701-42e5-a15f-e86cf8a8f62e/resource/Documents/rhapsody_api_reference_manual.pdf?lang=en

[3]C# 3.0 Cookbook, Third Edition, December 2007, Jay Hilyard & Stephen Teilhet

[4] Siemens internt fortroligt dokument- Feline SEN SW Test Level Analysis, version 003, 14-10-2014, Carsten Wiwe Beck, A5E34063280A.

Bilag

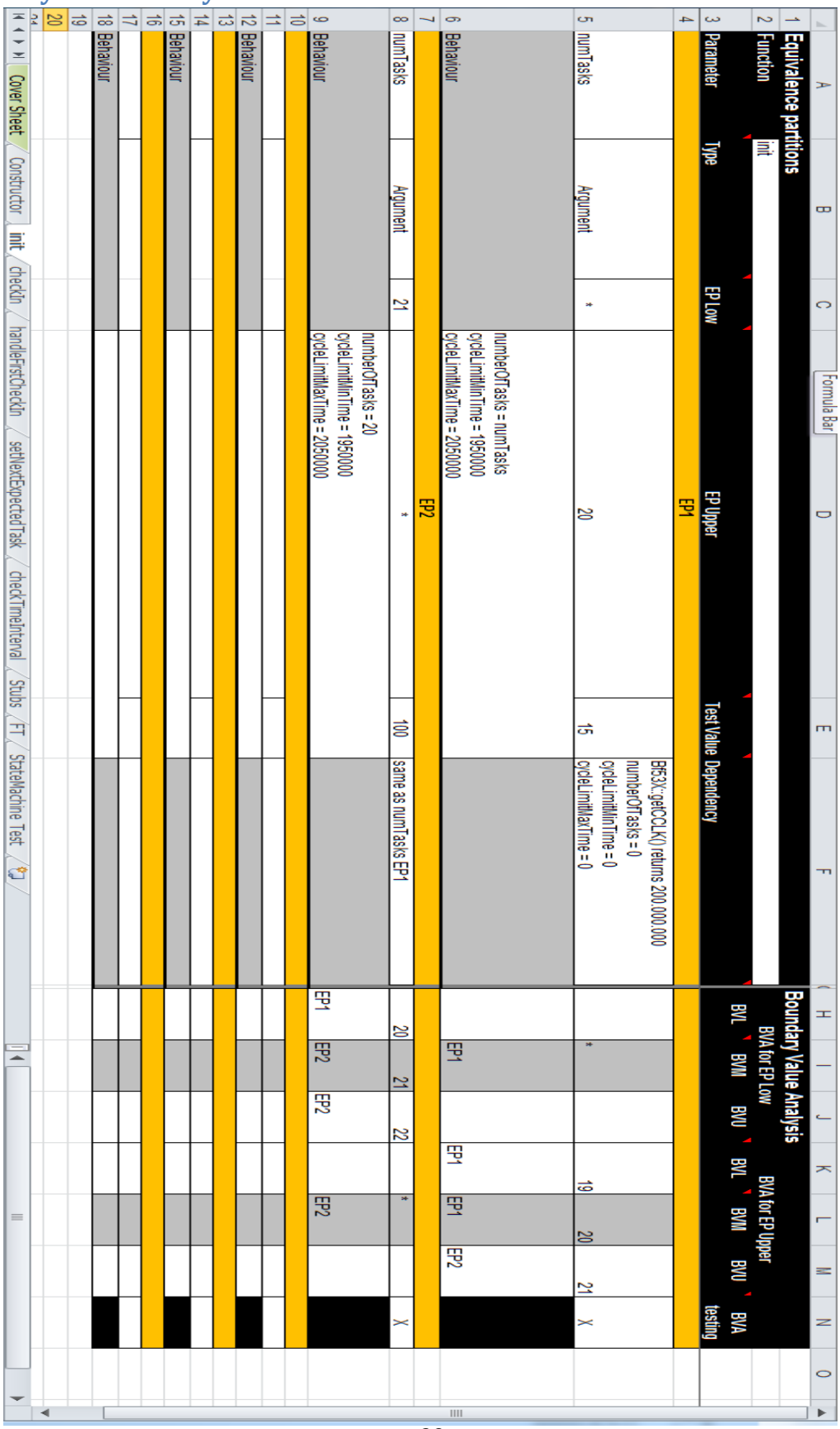
Bilag 1: Kanban board



Bilag 2: TemporalProgramFlowControl checkIn EP-BVA

	A	B	C	D	E	F	G	H	J	K	L	M	N	O
1	**Equivalence partitions**								**Boundary Value Analysis**					
2	**Class:**	TemporalProgramFlowControl							BVA for EP Low			BVA for EP Upper		
3	**Function**	**Parameter**	**Type**	**EP**	**EP Low**	**EP Upper**	**Test Value**	**Dependency**	**BVL**	**BVM**	**BVU**	**BVL**	**BVM**	**BVU**
27	**checkIn**													
28	checkIn	taskId	Argument	EP1	taskId == ExpectedTaskOrder[expectedTask]		TASK_6	expectedTask = 5 numberOfTasks = 10 ExpectedTaskOrder[5] = TASK_6		taskId == ExpectedTaskOrder[exp				
29	Behaviour				No FATAL_ERROR expectedTask = 6					EP1			EP1	
30	checkIn	taskId	Argument	EP2	taskId != ExpectedTaskOrder[expectedTask]		TASK_2	same as taskId EP1		taskId != ExpectedTaskOrder[exp				
31	Behaviour				FATAL_ERROR					EP2			EP2	
32	checkIn	isFirstCheckIn	Indirect	EP1	TRUE		TRUE	expectedTask = 0 numberOfTasks = 10 taskId = TASK_1 Bf53XTimer::getCoreTimerValue() returns 0xFFFFF830 (-2000) cycleLimitMinTime = 3000 cycleLimitMaxTime = 5000 previousCoreTimerValue = 0		TRUE				
33	Behaviour				isFirstCheckIn = false previousCoreTimerValue = 0xFFFFF830 call Bf53XWatchdog::kickWD expectedTask = 1 No FATAL_ERROR					EP1			EP1	
34	checkIn	isFirstCheckIn	Indirect	EP2	FALSE		FALSE	same as isFirstCheckIn EP1		FALSE				
35	Behaviour				isFirstCheckIn = false previousCoreTimerValue = 0xFFFFF830 call Bf53XWatchdog::kickWD expectedTask = 1 ErrorHandler is called with ERROR_TPFCSchedulingCycleTimeViolation No FATAL_ERROR					EP2			EP2	

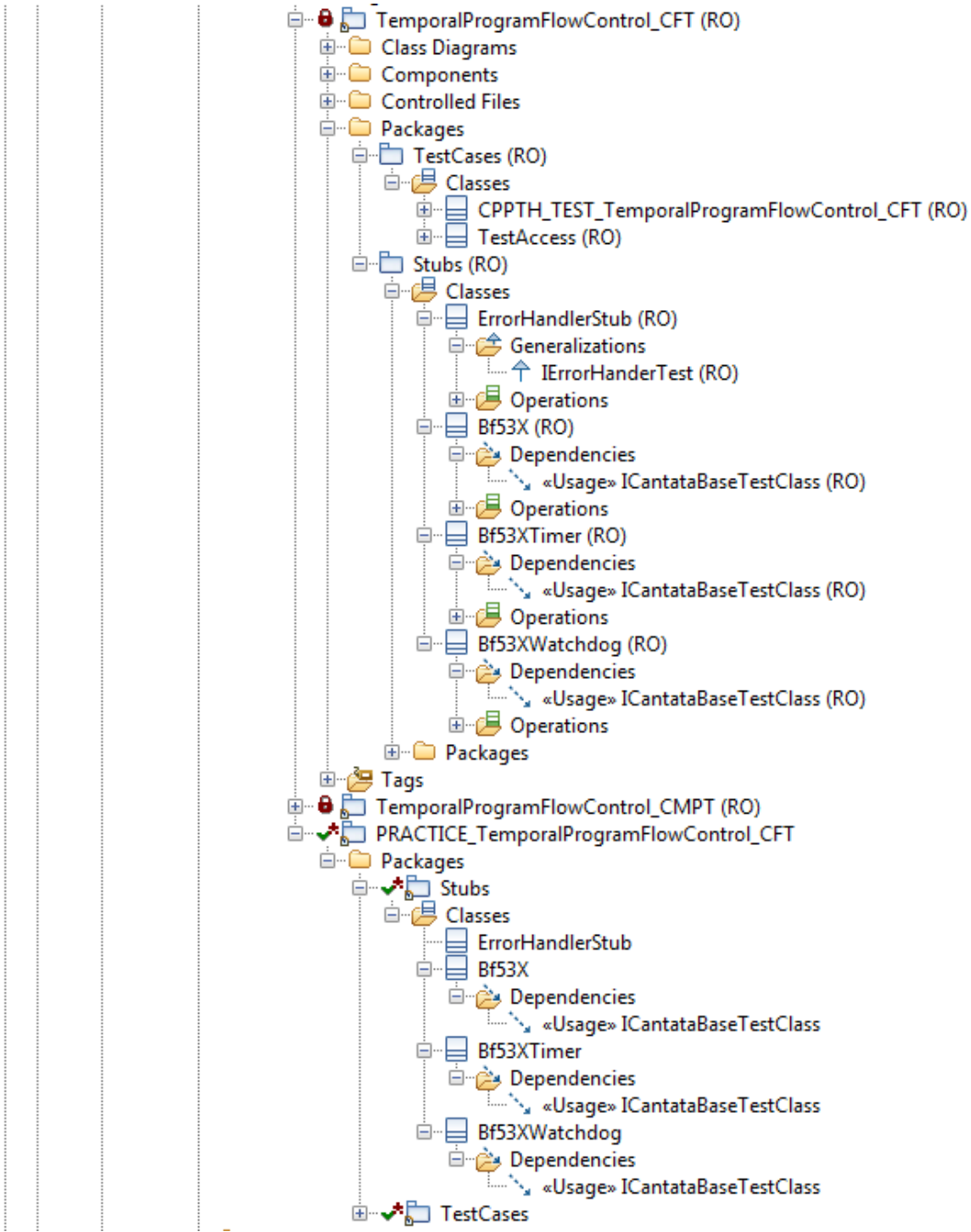
Bilag 3: Nyt EP-BVA layout



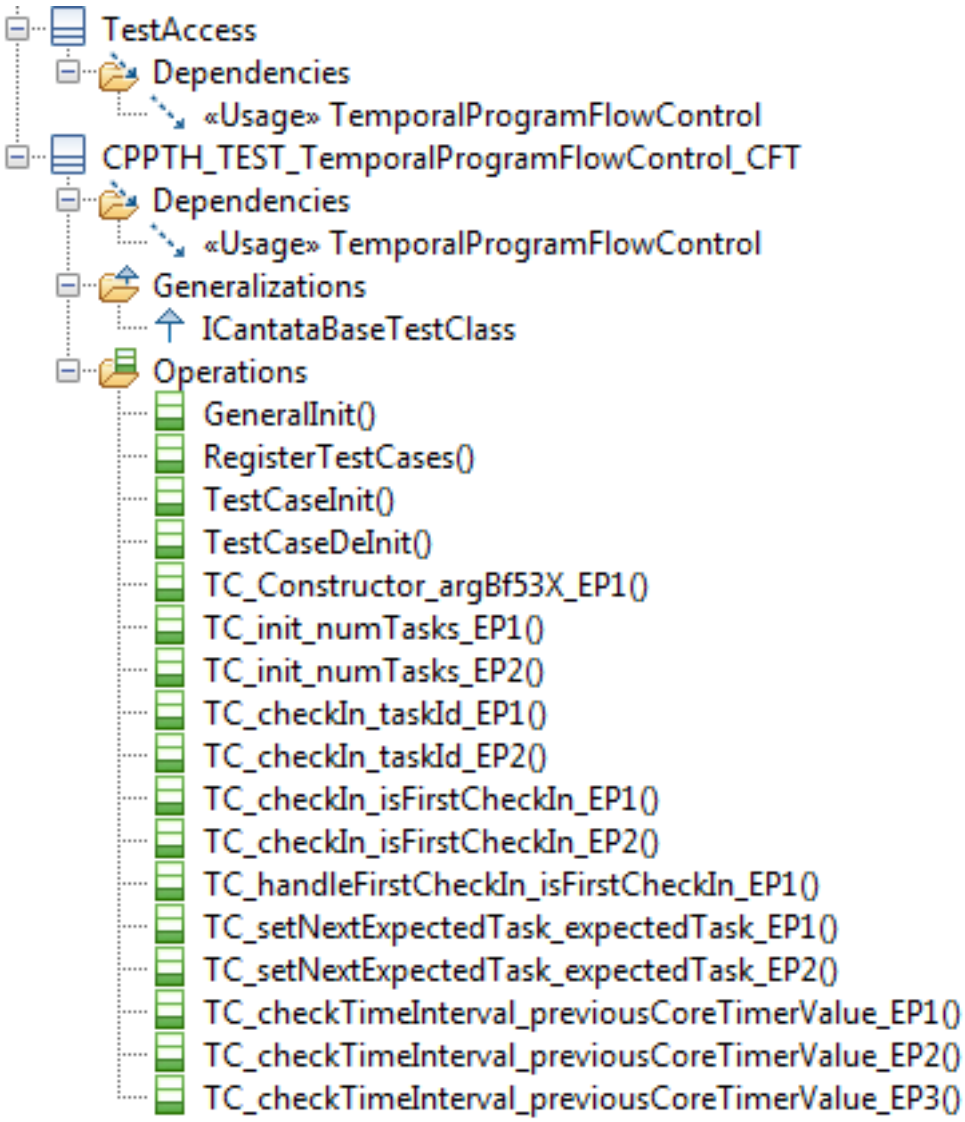
Bilag 4: Konsol output

4.a Normalkørsel stubbe

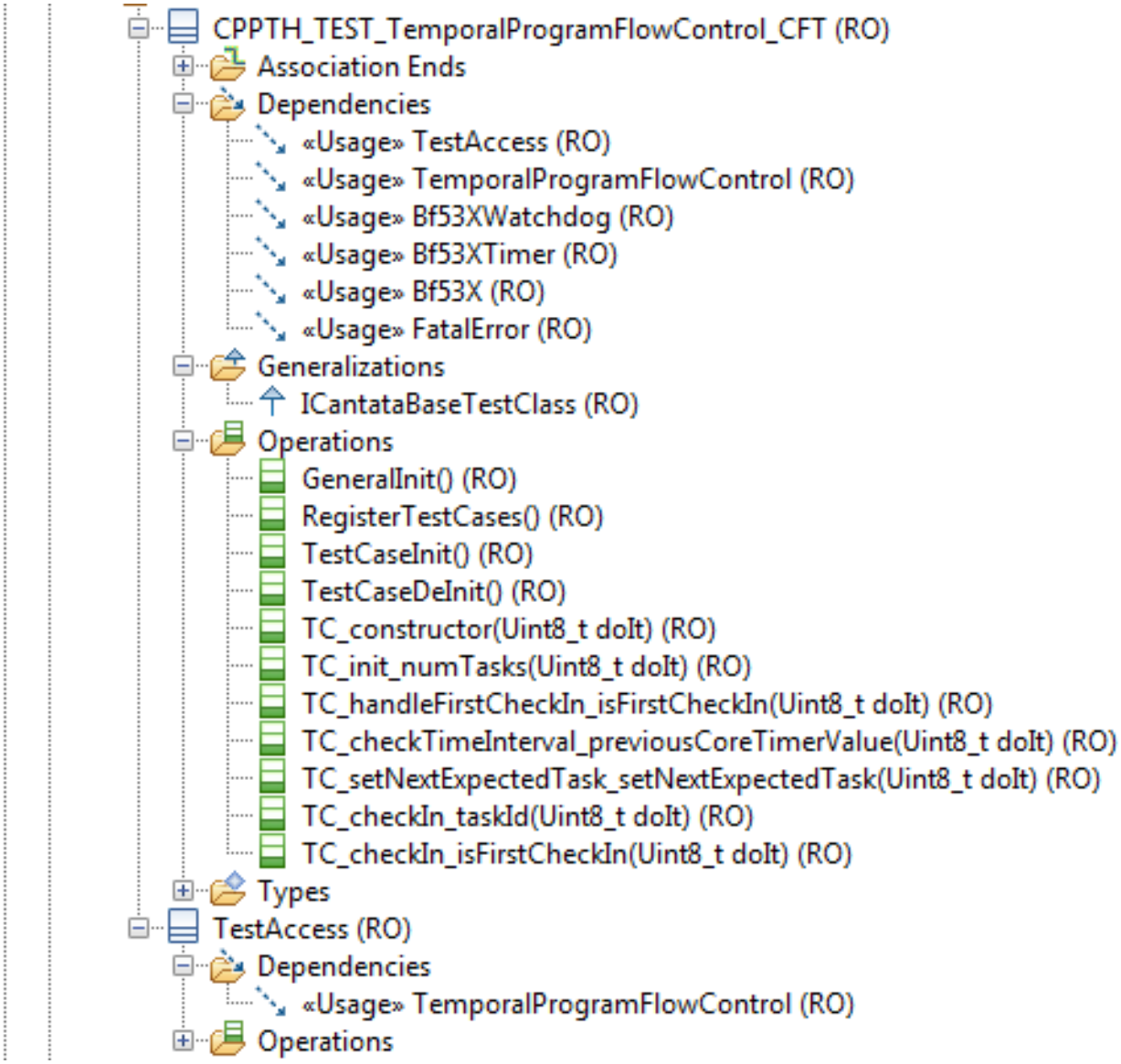
De generede elementer i PRACTICE_TemporalProgramFlowControl_CFT sammenholdes med dem fra TemporalProgramFlowControl_CFT



4.b Normalkørsel test cases genereret



4.c Normalkørsel test cases manuelt



Bilag 5: Kildekode

5a: Generator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using rhapsody;


namespace CFT_generator
{
    class Generator
    {
        //Initiation of global variables
        static String className;
        //Instantiation of excel cell name variables
        static String param = "Parameter", type = "Type", EPL = "EP Low", EPU = "EP
Upper", testVal = "Test Value", depend = "Dependency", LBVL = "Lower BVL";
        static String LBVM = "Lower BVM", LBVU = "Lower BVU", UBVL = "Upper BVL", UBVM =
"Upper BVM", UBVU = "Upper BVU", behave = "Behaviour";
        static String BLBVL = "Behaviour Lower BVL", BLBVM = "Behaviour Lower BVM", BLBVU
= "Behaviour Lower BVU", BUBVL = "Behaviour Upper BVL";
        static String BUBVM = "Behaviour Upper BVM", BUBVU = "Behaviour Upper BVU",
BVATest = "BVA testing";
        static int stepCount = 1;

        static IRPClass currentClass;
        static Dictionary<String, Cell> lookupTable = new Dictionary<String, Cell>();
        static ExcelAccess excel = new ExcelAccess();
        static Rhapsody rhp = new Rhapsody();
        const int START_ROW = 4;
        const int START_COL = 1;
        const int START_SHEET = 2;

        public void generateCFTs()
        {
            Console.WriteLine("*****************************");
            Console.WriteLine("**  Autogenerate CFT code  **");
            Console.WriteLine("*****************************");


            //Open excel workbook
            if (excel.openExcelWorkbook(CFT_generator.Settings1.Default.FilePath))
            {
                //Open Rhapsody model using COM API
                if (rhp.openProject())
                {
                    //Read excel file0
                    className = excel.getStubSheetInfo(2, 2);

                    if (rhp.confirmMakeCFTPackages(className))
                    {
                        buildDictionary();
                        generateStubs(START_ROW);
                        generateTestCases();
```

```
                    }
                }

                excel.closeExcel();
                Console.WriteLine("");
            }
        }

        //Method for stub generation
        static void generateStubs(int row)
        {
            IRPPackage stubPackage;

            String stubClassName, stubName = "";

            stubPackage = rhp.getStubsOrTCPackage(className, "Stubs");
            stubClassName = "#BEGIN";
            //Test setup: Stub generation
            while (!stubClassName.Equals("#END"))
            {
                stubClassName = excel.getStubSheetInfo(row, 1);
                stubName = excel.getStubSheetInfo(row, 2);
                if (!stubClassName.Equals("#END"))
                {
                    currentClass = rhp.makeClass(stubPackage, stubName);
                    if (stubClassName != stubName)
                    {
                        rhp.makeGeneralization(currentClass, "I" + stubClassName +
"Test");
                    }
                    else
                    {
                        rhp.makeDependency(currentClass, "ICantataBaseTestClass");
                    }
                }
                row++;
            }
        }

        //Method for generation of test cases including required top classes and init
classes
        static void generateTestCases()
        {
            IRPPackage testCasePackage;
            //Test case class generation
            testCasePackage = rhp.getStubsOrTCPackage(className, "TestCases");
            currentClass = rhp.makeClass(testCasePackage, "TestAccess");
            rhp.makeDependency(currentClass, className);
            currentClass = rhp.makeClass(testCasePackage, "CPPTH_TEST_" + className +
"_CFT");
            rhp.makeDependency(currentClass, className);
            rhp.makeGeneralization(currentClass, "ICantataBaseTestClass");

            makeTCOperations(START_ROW, START_COL);
        }

        //Method for parsing cells and creating testcase operations
        static void makeTCOperations(int row, int col)
```

```
        {
            bool endLoop = false;
            int sheetNumber = START_SHEET;
            String sheetHeader = "", EPHeader = "", currentFunction = "", currentParam =
"", fullOpName = "", referenceString = "", actionString = "";
            String verifyString = "", outString = "", registerString = "", BVAString =
"", inputString, beginString;
            String preconString = "\n\t// Preconditions\n\tPRECONDITION(\"-\");\n\t\n\t",
endString = "\n\n}\nEND_TEST();", ss = "(\"\",\"\",\"\",\"\");\n";
            RPOperation currentOp = null, registerTCOp = null;



            try
            {
                //Create init operations
                currentOp = currentClass.addOperation("GeneralInit");
                currentOp.body = "OPEN_LOG(\"testlog.ctr\", false);\nSTART_SCRIPT(\"" +
className + "\", true);\n\nREPORT_INFO" + ss + "REPORT_HISTORY" + ss + "\nSPEC_INFO" + ss
+ "SPEC_HISTORY" + ss + "\nSPEC_TARGET(\"The target of this test suite is to perform a
Class Function Test (CFT) on the " + className + "class.\");";
                registerTCOp = currentClass.addOperation("RegisterTestCases");
                currentClass.addOperation("TestCaseInit");
                currentClass.addOperation("TestCaseDeInit");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception occured: {0}", e.Message.ToString());
                if (e.Message.Contains("moniker of the object"))
                {
                    Console.WriteLine("Operation already exists for the " +
currentClass.name + ".");
                }
            }
            //While loop for looping through the excel sheets
            while (endLoop == false)
            {
                EPHeader = excel.getSheetCellData(row, col, sheetNumber);
                sheetHeader = excel.getSheetCellData(1, 1, sheetNumber);
                //Check to see if the current sheet is a valid EP sheet
                if (sheetHeader == "Equivalence partitions")
                {
                    currentFunction = excel.getSheetCellData(2, 2, sheetNumber);
                    beginString = buildBeginString(currentFunction);

                    //Check to see if there is another EP section in the current sheet
                    if (EPHeader.Contains("EP"))
                    {
                        actionString = "// Actions\n\t";
                        verifyString = "// Verification\n\t";
                        BVAString = "";
                        try
                        {
                            //Testcase setup of operation
                            currentParam = getCellContent(param, row, col, sheetNumber);
                            fullOpName = "TC_" + currentFunction + "_" + currentParam +
"_" + EPHeader;
                            currentOp = currentClass.addOperation(fullOpName);
```

```
                            //String generation of implementation for the current
Rhapsody test case operation
                            referenceString = "// Reference\n\tREFERENCE(\"" +
currentFunction + " " + EPHeader + "\")\n\t";

                            actionString = "//-------------------------- STEP " +
stepCount + " --------------------------\n\t// " + currentFunction + " " + EPHeader +
"\n\t\n\t// Actions\n\tACTION_INPUT(\"" + getCellContent(param, row, col, sheetNumber) +
"\", \"" + getCellContent(testVal, row, col, sheetNumber) + "\");\n\t";
                            stepCount++;
                            inputString = getCellContent(depend, row, col, sheetNumber);
                            actionString = buildOutStringFromMultipleLines(inputString,
actionString, false);

                            inputString = getCellContent(behave, row, col, sheetNumber);
                            verifyString = buildOutStringFromMultipleLines(inputString,
verifyString, true);

                            //Generate BVA test steps if needed
                            if (getCellContent(BVATest, row, col,
sheetNumber).Contains('X'))
                            {
                                if (getCellContent(LBVL, row, col, sheetNumber) !=
getCellContent(LBVU, row, col, sheetNumber))
                                {
                                    referenceString = referenceString + "REFERENCE(\"" +
currentFunction + " " + LBVL + "\")\n\t";
                                    referenceString = referenceString + "REFERENCE(\"" +
currentFunction + " " + LBVM + "\")\n\t";
                                    referenceString = referenceString + "REFERENCE(\"" +
currentFunction + " " + LBVU + "\")\n\t";

                                    BVAString = buildBVAString(BVAString,
currentFunction, EPHeader, row, col, sheetNumber, LBVL, BLBVL);
                                    BVAString = buildBVAString(BVAString,
currentFunction, EPHeader, row, col, sheetNumber, LBVM, BLBVM);
                                    BVAString = buildBVAString(BVAString,
currentFunction, EPHeader, row, col, sheetNumber, LBVU, BLBVU);

                                }
                                if (getCellContent(UBVL, row, col, sheetNumber) !=
getCellContent(UBVU, row, col, sheetNumber))
                                {
                                    referenceString = referenceString + "REFERENCE(\"" +
currentFunction + " " + UBVL + "\")\n\t";
                                    referenceString = referenceString + "REFERENCE(\"" +
currentFunction + " " + UBVM + "\")\n\t";
                                    referenceString = referenceString + "REFERENCE(\"" +
currentFunction + " " + UBVU + "\")\n\t";

                                    BVAString = buildBVAString(BVAString,
currentFunction, EPHeader, row, col, sheetNumber, UBVL, BUBVL);
                                    BVAString = buildBVAString(BVAString,
currentFunction, EPHeader, row, col, sheetNumber, UBVM, BUBVM);
                                    BVAString = buildBVAString(BVAString,
currentFunction, EPHeader, row, col, sheetNumber, UBVU, BUBVU);
```

```
                                }
                            }
                            //String append and implement
                            outString = beginString + referenceString + preconString +
actionString + verifyString + BVAString + endString;
                            currentOp.body = outString;
                            Console.WriteLine("Created test case: TC_" + currentFunction
+ "_" + currentParam + "_" + EPHeader + ".");
                        }
                        catch (Exception e)
                        {
                            Console.WriteLine("Exception occured: {0}",
e.Message.ToString());
                            if (e.Message.Contains("moniker of the object"))
                            {
                                Console.WriteLine("Operation already exists for the " +
currentClass.name + ".");
                            }
                            if (e.Message.Contains("Index was outside the bounds of the
array"))
                            {
                                Console.WriteLine("To solve: Check the \"Dependency\"
cell in the EP-BVA sheet for errors(syntax requires complete statements).");
                            }
                        }
                        registerString = registerString + "RegisterTestCase(&" +
currentClass.name + "::" + fullOpName + ", 1);\n";
                        row += 3;
                        stepCount = 1;
                    }
                    else
                    {
                        registerString = registerString + "\n";
                        row = START_ROW;
                        sheetNumber++;
                    }
                }
                else
                {
                    endLoop = true;
                }
            }
            if (registerTCOp != null)
            {
                registerTCOp.body = registerString;
            }
        }

        //Method for building verify or action string from an excel sheet cell with
multiple lines
        static String buildOutStringFromMultipleLines(String inStr, String outStr, bool
trueIfVerifyFalseIfAction)
        {
            String lineString, tempString;
            String[] lines = inStr.Split('\n');
            if (trueIfVerifyFalseIfAction)
            {
```

```
                foreach (String s in lines)
                {
                    lineString = s.Replace("=", "is");

                    outStr = outStr + "CHECK_NAMED(\"verify that " + lineString + "\", 1,
0);\n\t";
                }
            }
            else
            {
                foreach (String s in lines)
                {
                    tempString = s.Replace("returns", "=");
                    String[] halves = tempString.Split('=');
                    outStr = outStr + "ACTION_INPUT(\"" + halves[0].Replace(" ", "") +
"\", \"" + halves[1].Replace(" ", "") + "\");\n\t";
                }
                outStr = outStr + "\n\t//Action implementations\n\tEXPECTED_CALLS(
//\n\t\t\"\"\n\t\t\t\t\t\t);\n\tACTION(\"\");\n\t\n\tEND_CALLS();\n\t\n\t\n\t";
            }

            return outStr;
        }

        //Method for building begin string
        static String buildBeginString(String inStr)
        {
            String outStr;
            if (inStr != "Constructor")
            {
                outStr = "IF_START_TEST (__FUNCTION__,\n\t\t\t\t\t\"Verifies the method "
+ inStr + "\")\n{\n\t";
            }
            else
            {
                outStr = "IF_START_TEST (__FUNCTION__,\n\t\t\t\"Verifies that variables
are initialized correct when " + currentClass.name + " is constructed\")\n{\n\t";
            }
            return outStr;
        }

        static String buildBVAString(String outStr, String cFunc, String EPHead, int row,
int col, int sNum, String currentBVA, String currentBBVA)
        {
            String tempString;
            outStr = outStr + "\n\t//-------------------------- STEP " + stepCount + " --
-----------------------\n\t// " + cFunc + " " + currentBVA + "\n\t\n\t//
Actions\n\tACTION_INPUT(\"" + getCellContent(param, row, col, sNum) + "\", \"" +
getCellContent(currentBVA, row, col, sNum) + "\");\n\t";
            stepCount++;
            tempString = getCellContent(depend, row, col, sNum);
            outStr = buildOutStringFromMultipleLines(tempString, outStr, false);

            outStr = outStr + "// Verification\n\tCHECK_NAMED(\"verify that behaviour is
same as " + getCellContent(currentBBVA, row, col, sNum) + "\", 1, 0);\n\t";
            return outStr;
        }
```

```
        //Method preparing for navigation of excel sheet cells by storing cell objects in
a dictionary
        static void buildDictionary()
        {
            Cell C1 = new Cell(1, 0);
            Cell C2 = new Cell(1, 1);
            Cell C3 = new Cell(1, 2);
            Cell C4 = new Cell(1, 3);
            Cell C5 = new Cell(1, 4);
            Cell C6 = new Cell(1, 5);
            Cell C7 = new Cell(1, 7);
            Cell C8 = new Cell(1, 8);
            Cell C9 = new Cell(1, 9);
            Cell C10 = new Cell(1, 10);
            Cell C11 = new Cell(1, 11);
            Cell C12 = new Cell(1, 12);
            Cell C13 = new Cell(2, 3);
            Cell C14 = new Cell(2, 7);
            Cell C15 = new Cell(2, 8);
            Cell C16 = new Cell(2, 9);
            Cell C17 = new Cell(2, 10);
            Cell C18 = new Cell(2, 11);
            Cell C19 = new Cell(2, 12);
            Cell C20 = new Cell(1, 13);

            lookupTable.Add(param, C1);
            lookupTable.Add(type, C2);
            lookupTable.Add(EPL, C3);
            lookupTable.Add(EPU, C4);
            lookupTable.Add(testVal, C5);
            lookupTable.Add(depend, C6);
            lookupTable.Add(LBVL, C7);
            lookupTable.Add(LBVM, C8);
            lookupTable.Add(LBVU, C9);
            lookupTable.Add(UBVL, C10);
            lookupTable.Add(UBVM, C11);
            lookupTable.Add(UBVU, C12);
            lookupTable.Add(behave, C13);
            lookupTable.Add(BLBVL, C14);
            lookupTable.Add(BLBVM, C15);
            lookupTable.Add(BLBVU, C16);
            lookupTable.Add(BUBVL, C17);
            lookupTable.Add(BUBVM, C18);
            lookupTable.Add(BUBVU, C19);
            lookupTable.Add(BVATest, C20);
        }

        //Simple cell class used to parse excel sheet content
        private class Cell
        {
            private int rowOffset, colOffset;

            public Cell(int rw, int cl)
            {
                rowOffset = rw;
                colOffset = cl;
            }
```

```
            public int getRowOffset()
            {
                return rowOffset;
            }

            public int getColOffset()
            {
                return colOffset;
            }
        }

        //Method to improve code overview
        static String getCellContent(String cellKey, int r, int c, int sheetNum)
        {
            return excel.getSheetCellData(lookupTable[cellKey].getRowOffset() + r,
lookupTable[cellKey].getColOffset() + c, sheetNum);
        }
    }
}
```

5b: ExcelAccess.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Excel = Microsoft.Office.Interop.Excel;

namespace CFT_generator
{

    class ExcelAccess
    {
        //Excel
        Microsoft.Office.Interop.Excel.Application excelApp;
        Excel.Workbook excelWorkbook;
        Excel.Sheets excelSheets;
        Excel.Worksheet activeWorksheet;

        public Boolean openExcelWorkbook(String path)
        {
            Boolean ret = false;

            //Open first sheet in workbook
            try
            {
                excelApp = new Excel.Application();
                excelWorkbook = excelApp.Workbooks.Open(path, 0, false, 5, "", "", false,
Excel.XlPlatform.xlWindows, "", true, false, 0, true, false, false);
                excelSheets = excelWorkbook.Worksheets;
                activeWorksheet = excelWorkbook.Sheets[2];
                ret = true;
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception occured while parsing input file:");
                Console.WriteLine("{0}", e.Message.ToString());
            }

            return ret;
        }

        public Boolean closeExcel()
        {
            Boolean ret = false;

            //Clean up
            excelWorkbook.Close();

            return ret;
        }

        public String getSheetCellData(int row, int col, int sheetIndex)
        {
            if (sheetIndex == 0)
            {
                activeWorksheet = excelWorkbook.Sheets[excelWorkbook.Sheets.Count - 2];
```

```
            }
            else
            {
                activeWorksheet = excelWorkbook.Sheets[sheetIndex];
            }
            return activeWorksheet.Cells[row, col].Text;
        }
    }
}
```

5c: Rhapsody

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using rhapsody;

namespace CFT_generator
{
    class Rhapsody
    {
        RPProject activeProject;

        public Rhapsody()
        {
        }

        public Boolean openProject()
        {
            Boolean retval = false;
            try
            {
                IRPApplication obj = null;

                //Get open Rhp application and store project
                obj =
(IRPApplication)System.Runtime.InteropServices.Marshal.GetActiveObject("Rhapsody.Applicat
ion");
                activeProject = obj.activeProject();
                Console.WriteLine("Connected to Rhapsody project: {0}",
activeProject.name);
                retval = true;
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception occured: {0}", e.Message.ToString());
                if (e.Message.Contains("Operation unavailable"))
                {
                    Console.WriteLine("Most likely cause: Rhapsody instance not
started");
                }
            }
            return retval;
        }

        public IRPClass makeClass(IRPPackage classPackage, String className)
        {
            IRPClass newClass = null;
            try
            {
                if (classPackage.findNestedElement(className, "Class") == null)
                {
                    newClass = classPackage.addClass(className);
                    Console.WriteLine("Added Class: " + className + ", to package " +
classPackage.name + ".");
                }
```

```
                else
                {
                    Console.WriteLine("Class " + className + " already exists in " +
classPackage.name + "!");
                    newClass = (IRPClass)classPackage.findNestedElement(className,
"Class");
                }

            }
            catch (Exception e)
            {
                Console.WriteLine("Exception occured: {0}", e.Message.ToString());
            }
            return newClass;
        }

        public void makeDependency(IRPClass derivedClass, String baseClassName)
        {
            IRPDependency newDepend = null;
            if (derivedClass.dependencies.Count == 0)
            {
                try
                {
                    newDepend = derivedClass.addDependency(baseClassName, "Class");
                    newDepend.addStereotype("Usage", null);
                    Console.WriteLine("Added dependency to: " + derivedClass.name + ".");
                }
                catch (Exception e)
                {
                    Console.WriteLine("Exception occured: {0}", e.Message.ToString());
                }
            }
            else
            {
                Console.WriteLine("Dependency already exists for the class: " +
derivedClass.name + "!");
            }
        }

        public void makeGeneralization(IRPClass derivedClass, String testInterfaceName)
        {
            if (derivedClass.generalizations.Count == 0)
            {
                try
                {
                    IRPClass baseClass =
(RPClass)activeProject.findNestedElementRecursive(testInterfaceName, "Class");
                    derivedClass.addGeneralization((RPClassifier)baseClass);
                    Console.WriteLine("Added generalization to: " + derivedClass.name +
".");
                }
                catch (Exception e)
                {
                    Console.WriteLine("Exception occured: {0}", e.Message.ToString());
                    if (e.Message.Contains("advise flags"))
                    {
                        Console.WriteLine("To solve exception: Check Rhapsody model for
the test interface: " + testInterfaceName + ".");
```

```
                    }
                }
            }
            else
            {
                Console.WriteLine("Generalization already exists for the class: " +
derivedClass.name + "!");
            }
        }

        public Boolean confirmMakeCFTPackages(String className)
        {
            IRPPackage CFTPackage, currentPackage;
            String fullPackageName = "PRACTICE_" + className + "_CFT";
            try
            {
                CFTPackage =
(IRPPackage)activeProject.findNestedElementRecursive(fullPackageName, "Package");
                if (CFTPackage == null)
                {
                    Console.WriteLine("Package: " + fullPackageName + " not found!");
                    return false;
                }
                else
                {
                    currentPackage = (IRPPackage)CFTPackage.findNestedElement("Stubs",
"Package");
                    if (currentPackage == null)
                    {
                        CFTPackage.addNestedPackage("Stubs");
                        Console.WriteLine("Added package: Stubs.");
                    }
                    currentPackage =
(IRPPackage)CFTPackage.findNestedElement("TestCases", "Package");
                    if (currentPackage == null)
                    {
                        CFTPackage.addNestedPackage("TestCases");
                        Console.WriteLine("Added package: TestCases.");
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception occured: {0}", e.Message.ToString());
                if (e.Message.Contains("Operation unavailable"))
                {
                    Console.WriteLine("To solve exception: Check if package already
exists.");
                }
            }
            return true;
        }

        public IRPPackage getStubsOrTCPackage(String className, String packageName)
        {
            IRPPackage CFTPackage, currentPackage;
            currentPackage = null;
            String fullPackageName = "PRACTICE_" + className + "_CFT";
```

```
            try
            {
                CFTPackage =
(IRPPackage)activeProject.findNestedElementRecursive("PRACTICE_" + className + "_CFT",
"Package");
                currentPackage = (IRPPackage)CFTPackage.findNestedElement(packageName,
"Package");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception occured: {0}", e.Message.ToString());
            }
            return currentPackage;
        }
    }
}
```

DTU Space
Institut for Rumforskning og -teknologi
Danmarks Tekniske Universitet

Elektrovej, bygning 327
2800 Kgs. Lyngby
Tlf 4525 9500
Fax 4525 9575

www.space.dtu.dk