Rijkswaterstaat
Ministerie van Infrastructuur en Milieu

RWS INFORMATIE - ONGECLASSIFICEERD

TOPAAS: Deel 1 Handleiding (Kader)

Datum	28 maart 2018
Status	Definitief

Colofon

Naam Standaard	Kader TOPAAS
Beschrijving:	Het kader TOPAAS beschrijft een structurele aanpak voor faalkansanalyse van software intensive systemen.
Status:	Definitief
Datum	28 maart 2018
Versienummer:	1.0
Soort:	Kader
Verantwoordelijke PE:	Jean-Luc Beguin
Gebruik in proces:	Aanleg en Onderhoud
Netwerk:	HVWN, HWS en HWN
Object:	Alle RWS-infrastructuur
Hoofdkennisveld:	Assetmanagement
Kennisveld:	Risicogestuurd Beheer en Onderhoud (RGBO)
Informatie:	Probo@rws.nl
Verantwoordelijke afdeling:	RWS GPO – Afdeling Advies Technisch Management (ATM)
WW RWS Nummer:	1319

Overzicht wijzigingen

Versie	Datum	Wijzigingen
0.7	10 jan 2013	Versienummer 0.7, definitief
1.0	28 mrt 2018	Tekstuele update, model is ongewijzigd, toelichting is uitgebreider, zie commentaaroverzicht.

Inhoud

1 Inleiding

1.1 Context

In 2010 is door Rijkswaterstaat besloten tot het beheerst invoeren van risico-gestuurd beheer en onderhoud (RGBO) binnen asset management (AM). Met RGBO worden alle risico's voor het functioneren van een object in kaart gebracht, waardoor deze op een transparante en weloverwogen manier beheerst kunnen worden. Dit in tegenstelling tot traditioneel onderhoud dat veelal conditie-gestuurd is, gericht op het handhaven van een bepaald technisch niveau.

Het doel van RGBO is om de risico's in het functioneren van de drie netwerken via beheer- en onderhoudsacties zodanig te beheersen, dat de afgesproken prestaties worden geleverd tegen minimale (levensduur)kosten. RGBO maakt de relatie tussen de netwerkprestatie en onderhoud expliciet. In 2013 besloot het bestuur RWS tot een verdere doorontwikkeling van RGBO om volledig in control te komen middels een vervolgtraject RGBO, gevolgd door een herijking ervan in 2016.

Binnen Rijkswaterstaat is daartoe in 2016 de handreiking Prestatiegestuurde Risicoanalyses (PRA) opgesteld om het risicogestuurd denken toepasbaar te maken voor alle infrastructurele assets, die Rijkswaterstaat in beheer heeft. Deze handreiking integreert en vervangt daarmee de Leidraad RAMS en de Leidraad risicogestuurd beheer en onderhoud.

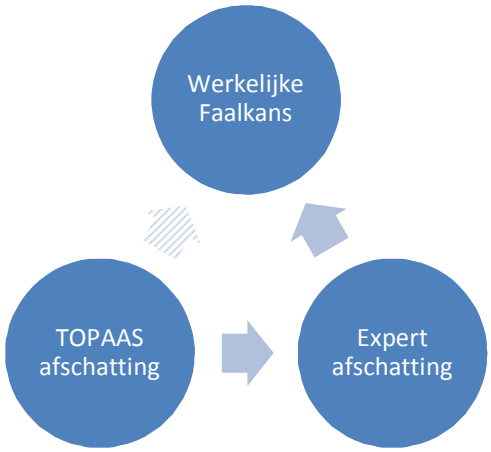
Prestatiegestuurde risicoanalyse (PRA) is een belangrijk instrument. De PRA brengt de balans in beeld tussen de prestaties van een object, de risico's die de prestaties beïnvloeden en de kosten van het in stand houden van de prestatie. Met hulp van PRA's kan Rijkswaterstaat onderbouwde beslissingen nemen bij aanleg, beheer en onderhoud.

In aanvulling op deze handreiking zijn verschillende methodes inhoudelijk verder uitgewerkt en vastgelegd in aparte handreikingen. Zo ook deze standaard die een structurele aanpak voor faalkansanalyse van software intensieve systemen beschrijft.

Software is in toenemende mate onderdeel van de kunstwerken die Rijkswaterstaat laat bouwen en beheren. Deze software vervult een cruciale rol in het bedienen, besturen en beveiligen van veel civiele en technische objecten: zonder goed functionerende software kan het object ongepland niet-beschikbaar zijn of zelfs onveilig zijn voor mens en omgeving. Deze software blijkt in praktijk ook de oorzaak te kunnen zijn van het falen van een object en moet dus meegenomen worden in de kwantitatieve onderbouwing van de RAMS-analyse, om zo een goed inzicht te geven in de toekomstige prestaties.

Software opnemen in de RAMS-analyse is echter problematisch. Op de vraag "wat is de faalkans van de geleverde software?" heeft de wetenschap op dit moment in veel gevallen geen praktisch antwoord. Dus in tegenstelling tot de civiele techniek en elektrotechniek, waar wel wetenschappelijk bewezen methoden zijn om op gestructureerde wijze een faalkans van een object af te geven, is dat bij software vaak niet het geval. Voor die specifieke situatie is TOPAAS bedoeld.

Een alternatief om een faalkans te berekenen, is hem af te laten schatten door een groep van experts. Dit is onder risicoanalisten een geaccepteerde aanpak, maar er kleven praktische bezwaren aan. Zo is het nogal omslachtig om voor elk project een groep bijeen te roepen en is er een aanzienlijke kans dat de experts het niet met elkaar eens worden op het project.
TOPAAS benadert de afschatting van een groep experts door middel van een gestructureerde vragenlijst. Hiermee is het mogelijk om op ordegrootte nauwkeurig een grove afschatting te maken van de faalkans van software.



Figuur 1: Relatie TOPAAS met de werkelijke faalkans

1.2 Doel

Deze handleiding is bedoeld voor RAMS-specialisten die betrokken zijn bij de beoordeling van software. Het hierin beschreven proces wordt door Rijkswaterstaat gezien als de geaccepteerde methode voor de uitvoering van software-betrouwbaarheidsanalyses. Naast de beschrijving van de noodzakelijke stappen in een analyse, worden ook de randvoorwaarden en de noodzakelijke vastlegging beschreven.

Deze handleiding is de tweede iteratie van zowel het model als van de handleiding, en verwerkt zo'n tien jaar praktijkervaring met het gebruik van TOPAAS in het veld. Deze handleiding vervangt alle voorgaande beschrijvingen van het TOPAAS-model.

Deze handleiding is een beschrijving van hoe de tweede iteratie van TOPAAS moet worden toegepast en laat het waarom achterwege. Uitleg over de inhoudelijke opbouw van het model is in de onderhoudsdocumentatie beschreven (zie Hoofdstuk 6 Referenties) en is omwille van de leesbaarheid uit deze handleiding gehouden. Naast het beschrijven van enkele modelmatige wijzigingen richt deze handleiding zich voornamelijk op een betere uitleg van de toepassing van het model, zodat de toepassing ervan eenvoudiger en eenduidiger wordt. Ook wordt er expliciet beschreven wat noodzakelijke verslaglegging is, zodat ook daarover helderheid naar alle partijen ontstaat.

Op sommige punten bieden we ook praktische handreikingen in zowel technische insteek als procesaanpak. Dit zijn bewust geen verplichte activiteiten, maar zijn bedoeld om de toepassing van het model op een verantwoorde manier te vereenvoudigen.

1.3 Wat is TOPAAS

De afkorting TOPAAS staat voor *Task Oriented Probability of Abnormalities Analysis for Software*. Hier zitten de volgende essentiële kenmerken van TOPAAS in besloten:

- TOPAAS kijkt gericht naar taakuitvoering door software. Software is in een RAMS-analyse geen amorf object waarbij elke fout gelijk gevolgen heeft voor betrouwbaarheid, beschikbaarheid of veiligheid. In een systeem vervult software een afgebakende taak (taakuitvoering) die mogelijk essentieel is voor één of meerdere RAMS-eisen.
- TOPAAS levert een faalkans (per vraag), een kwantitatieve maat voor betrouwbaarheid van de taakuitvoering. Toe te voegen als basisgebeurtenissen (Q) in een foutenboom.
- TOPAAS beschouwt alle vormen van afwijkend gedrag die een RAMS-doelstelling in gevaar kunnen brengen, voor zover ze uit de software komen. Dus niet alleen afwezig, te vroeg of te laat gedrag, maar ook ongewenst gedrag. Gedrag van de omgeving (systemen die buiten hun specificaties lopen) of falende hardware worden hier expliciet niet in meegenomen. Dit moet in de RAMS-analyse zelf worden opgepakt. Op het zogenaamde opvolgfalen wordt later dieper ingegaan.
- TOPAAS gaat uit van een context dat de relevante faalwijzen geïdentificeerd worden in de foutenboom. Het gebruik van foutenbomen impliceert ook dat de software op een gedegen wijze ontworpen, gebouwd en getest is.
- TOPAAS kan worden toegepast in alle fasen van systeemontwikkeling; ontwerp, realisatie, testen en productie.
- Deze faalkans is een a-priori faalkans die gebruikt kan worden om na praktijkervaringen verder Bayesiaans-te-updaten met daadwerkelijke prestaties.
- TOPAAS levert middels een parametermodel een TOPAAS-score op. De TOPAAS-score leidt tot een faalkans per vraag (Q).
- TOPAAS kan toegepast worden op alle soorten software systemen (inclusief COTS).

Kort gezegd is TOPAAS een afschattingsmethode om de kans te bepalen dat het resultaat, beslissing of aansturing van de software afwezig, verkeerd of niet tijdig is.

1.4 Wat is TOPAAS niet?

TOPAAS is een meetinstrument: het helpt om op een gestructureerde manier de faalkans van de taakuitvoering af te schatten. Het is zeker geen proces engineering tool om de ondergrens van de ontwikkelinspanning te bepalen. Daarvoor zijn *best practice* documenten als de IEC 61508 [1] veel beter geschikt: net als een kookboek beschrijft deze standaard op basis van een gewenst RAMS-niveau (faalkans) via het bijbehorende SIL-niveau de benodigde activiteiten. Daarmee zijn TOPAAS en IEC61508 complementair aan elkaar: de IEC61508 beschrijft de *common accepted practice* om een faalkans te realiseren, TOPAAS kijkt naar de daadwerkelijk gerealiseerde faalkans.

TOPAAS is ook geen ontwerptool. Alhoewel TOPAAS wel kan helpen in het vergelijken van meerdere ontwerpoplossingen in RAMS-termen is het zeker geen methode die garandeert dat een ontwerper de goede keuzes maakt.

TOPAAS is evenmin bedoeld om gericht onderliggende aspecten op te nemen in de vraagspecificatie bij aanbestedingen of opdrachten. Een te behalen RAMS-niveau

kan onderdeel zijn van de uitvraag: een aannemer kan middels aantoonbare expertise dit beantwoorden. Opnemen van afzonderlijke aspecten van TOPAAS waarmee dat niveau bepaald wordt, is dus ongewenst.

TOPAAS behandelt de betrouwbaarheid volgens de definitie van [8]. Opgemerkt wordt dat deze niet exact hetzelfde is als de in de IT-sector gangbare definitie van *"reliability"* volgens ISO/IEC 25010. TOPAAS levert geen bijdrage aan andere kwaliteitsaspecten van de ISO/IEC 25010, inschattingen voor zogenaamde Time-To-Failure of Time-To-Repair of andere onderhoudseigenschappen.

1.5 Toepassingsgebied

De faalkans van software kan op twee fundamentele manieren bepaald worden. De eerste manier is gebaseerd op de bevindingen van operationele software. Er wordt gekeken naar hoe de software zich gedraagt bij uitvoering in verschillende situaties. Dat kunnen zowel gecreëerde testsituaties zijn als situaties in productie. Voorbeelden hiervan zijn Reliability Growth Modelling (RGM) en Monte Carlo testen. De tweede manier is gebaseerd op de omstandigheden en externe kenmerken van de software zoals het totstandkomingsproces, de complexiteit en de omvang van de code. TOPAAS is gebaseerd op de tweede manier, zoals ook haar voorloper het TDT-model.

Elke methode voor faalkansanalyse heeft specifieke kenmerken waardoor zij in bepaalde situaties goed of juist minder goed toepasbaar is. Specifieke voorbeelden van faalkansanalysemethoden zijn:

- Faalkansanalyse tijdens ontwerp en realisatie fase (dus voordat draaiende software beschikbaar is) kan alleen met methoden als TOPAAS;
- Analyse op reeds draaiende veiligheidskritische functies waarbij voldoende defectgegevens beschikbaar zijn, kan bij voorkeur met Reliability Growth Modelling;
- Analyses op dergelijke functies zonder statistisch significante aantallen defects kan het beste met TOPAAS.

TOPAAS is gemaakt voor operationele industriële systemen die een merkbare invloed hebben op RAMS-prestaties van kunstwerken en industriële procesautomatisering, waar de betrouwbaarheid niet of niet eenvoudig via een andere weg is vast te stellen. Dit kunnen zowel nieuw ontwikkelde systemen zijn als COTS-producten.

Vaak zijn dit eenvoudige korte ketens van sensoren, systemen en actuatoren zonder dat functievervullers veel onderlinge interferentie kennen. Zodra taakuitvoeringen veel onderlinge afhankelijkheden kennen, doordat ze sterk met elkaar verweven zijn, is de in deze handleiding beschreven eenvoudige analyse niet toereikend en moeten er uitgebreidere analyses worden uitgevoerd. Uitgebreidere analyses zijn eveneens noodzakelijk om zeer hoge betrouwbaarheidseisen (faalkans 10^{-6} of lager) te verifiëren.

TOPAAS is niet gemaakt om alle IT-risico's te inventariseren. TOPAAS is gemaakt om de kans van intrinsiek falen van taakuitvoering van een goed ontworpen en goed getest systeem in te schatten. Kansen op falen ten gevolge van externe IT-dreigingen, bijvoorbeeld cybercriminaliteit en virussen, kunnen niet met TOPAAS worden afgeschat.

In de beslissingstabel hieronder zijn de verschillende methoden gekoppeld aan de onderstaande specifieke kenmerken:

- Ontwikkelfase waarin de software zich bevindt
- Frequentie van het gebruik van de software
- Aantal bevindingen uit test of productie
- Complexiteit van de software

De methoden zelf worden in bijlage A behandeld.

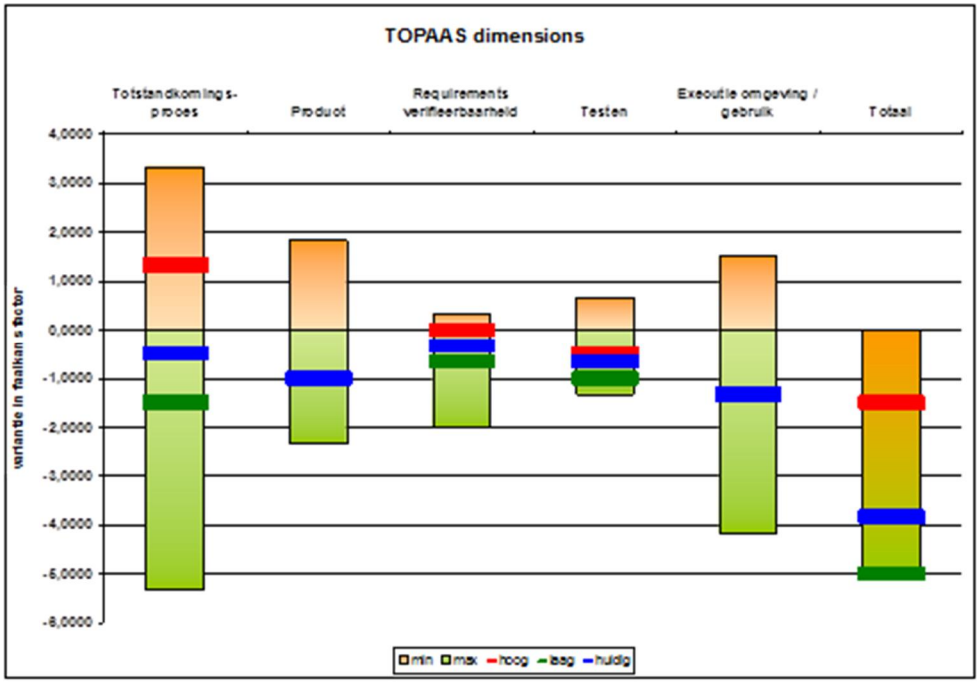
ontwikkelfase	ontwerp		realisatie		testen								productie							
frequentie van gebruik hoog	Y	N	Y	N	Y				N				Y				N			
aantal bevindingen significant	-	-	-	-	Y		N		Y		N		Y		N		Y		N	
complexiteit hoog	-	-	-	-	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
TOPAAS	X[1]	X	X[1]	X	X[1]	X[1]	X[1]	X[1]	X	X	X	X	X[1]	X[1]	X[1]	X[1]	X	X	X	X
RGM					X	X			X	X			X	X			X	X		
Monte Carlo						X		X		X		X		X		X		X		X
Duurtest						X		X		X		X		X		X		X		X

Figuur 2: Beslissingstabel, X1: Topaas toegepast op hoogfrequent gebruikte toepassingen levert een factor op die via memo 'Continue systemen en de faalkans van software'leidt tot een faalkans per periode (λ).

Het gebruik van de tabel is in volgend voorbeeld toegelicht:
Boven de dikke streep aan de linkerkant (eerste kwadrant) staan de verschillende kenmerken; bijvoorbeeld het *"aantal significante bevindingen"*. Boven de dikke streep aan de rechterkant (tweede kwadrant) staan de corresponderende condities; bijvoorbeeld het *"aantal bevindingen significant"* is niet van toepassing ("-"), waar ("Y") of niet waar ("N"). Bij de aangeduide conditie hoort de actie, cq. mogelijke wijze van betrouwbaarheidsbepaling onder de dikke streep in dezelfde kolom.
Als het project in de ontwikkelfase *"testen"* is, de frequentie van gebruik van het betreffende systeem is *"hoog"*, het aantal significante bevindingen is *"laag"* en de complexiteit is *"hoog"*, dan is TOPAAS de geschikte methode, met de kanttekening dat via de memo *'Continue systemen en de faalkans van software'* een faalkans per periode (*λ*) kan worden bepaald. Deze methode is echter nog in ontwikkeling en niet wetenschappelijk onderbouwd.

Het gebruik van TOPAAS in de ontwerp- of realisatiefase beperkt zich noodgedwongen tot het beoordelen van het voortbrengingsproces, de architectuur en eventueel plannen met betrekking tot traceerbaarheid en testdekking. Ook het al dan niet beschikbaar zijn van velddata kan worden meegewogen. Hiermee kan een eerste indicatie worden gegeven van de faalkans. De onzekerheid op de verschillende factoren levert daarbij een boven- en ondergrens waarbinnen de uiteindelijke faalkans zich waarschijnlijk zal bewegen. In onderstaande grafiek is een voorbeeld opgenomen waarin dit fenomeen per TOPAAS dimensie en voor de

totale faalkans is weergegeven. Een dergelijke aanpak levert dus geen betrouwbare kwantificatie, maar kan wel tekortkomingen of aandachtspunten identificeren.



Figuur 3: TOPAAS-dimensies

De kolommen geven de theoretische maximaal positieve en negatieve bijdrage aan de faalkans weer. Daarbinnen toont de blauwe balk de huidige TOPAAS-schatting. De rode en groene balken daaromheen, zijn een schatting van de mogelijk uiteindelijke waarden voor het specifieke project. Uiteraard is dit slechts één mogelijkheid om een en ander grafisch weer te geven.

Een faalkansanalyse met behulp van RGM kan overigens een goede aanvulling en verificatie zijn van eerder in het ontwikkeltraject uitgevoerde faalkansanalyse met behulp van TOPAAS. Wanneer de resultaten van een RGM-analyse op basis van draaiende software aanzienlijk afwijken van de TOPAAS-analyse, is nader onderzoek noodzakelijk.

Overigens moet vermeld worden dat IEC 61508 uitdrukkelijk géén onderdeel uitmaakt van de alternatieve methoden; IEC 61508 is geen methode voor faalkansanalyse! Het toepassen van IEC 61508 stelt een project- of ontwikkelteam in staat software te bouwen dat aan een bepaalde faalkansnorm voldoet door het volgen van de *"recommended"* en *"highly recommended practices"* bij het tot doel gestelde Safety Integrity Level. Een garantie dat de software deze faalkans haalt is het echter allerminst. Bepalingen, bijvoorbeeld met behulp van TOPAAS, moeten de uiteindelijk behaalde faalkans aantonen. Het toepassen van IEC 61508 is overigens wel een onderdeel van de beoordeling binnen TOPAAS.

1.6 Output en nauwkeurigheid van de resultaten

De TOPAAS-score wordt in de laatste stap afgerond tot een geheel getal en is daarmee tussen de 0 en -5. Een score boven de -2 levert geen bruikbare kwantificatie op. De interpretatie van de scores -2, -3, -4, -5 levert een faalkans op van (in foutenboom analyses ook wel de Q genoemd) 10^{-2}, 10^{-3}, 10^{-4} of 10^{-5} per

vraag. Voor de omgang met frequente gebruikte systemen wordt verwezen naar de memo *'Continue systemen en de faalkans van software'*.

We zien dat de nauwkeurigheid van de resulterende Q in ordegroottes is, dus de uitkomst is altijd 10^{-2}, 10^{-3}, 10^{-4} of 10^{-5} per vraag. De belangrijkste reden hiervoor is dat de expertschattingen niet nauwkeuriger zijn en schijnnauwkeurigheden vermeden dienen te worden. In de praktijk is er niet veel verschil op de topgebeurtenis tussen een taakuitvoering met een faalkans van 10^{-3} of een faalkans met $6 \cdot 10^{-4}$ en levert deze onnauwkeurigheid vaak geen problemen op. TOPAAS is nauwkeurig genoeg om te constateren dat software wel of geen belemmering voor de RAMS-prestaties is.

Uit wetenschappelijk onderzoek blijkt dat de afschatting van de software-faalkans door experts een conservatieve benadering kent met een maximale afwijking van één ordegrootte. De resultaten van TOPAAS zullen in de regel een redelijke tot conservatieve afschatting opleveren van de faalkans van de software. Het is altijd verstandig om de invloed van dit conservatisme op de analyse te onderzoeken met een gevoeligheidsanalyse.

Een faalkans van één weerspiegelt het totale gebrek aan vertrouwen in de taakuitvoering door experts en dient als bevinding te worden behandeld in de vorm van herstructureren van de oplossing.

1.7 Uitgangspunten

Deze handleiding gaat over het kwantificeren van de bijdrage van falen van software in een oplossing waar een gedegen architectuur aan ten grondslag ligt en die met professionaliteit is doorontwikkeld tot een werkzame oplossing. TOPAAS gaat dan ook minimaal uit van het doorlopen van een basaal ontwerp en testproces, wat ook een minimaal niveau van kwaliteitsborging inhoudt.

TOPAAS kan niet omgaan met systemen waar fundamentele fouten in het ontwerp aanwezig zijn. Het uitgangspunt is dat bij de validatie- en verificatieaanpak deze fouten als risico's worden meegenomen. Fouten op architectuurniveau (verandering van interne of externe zaken) kunnen gedurende de levensduur optreden. Uitgangspunt hierbij is dat het changeproces deze risico's adresseert.

Voordat de TOPAAS-analyse plaatsvindt, moet de verantwoordelijke analist dus vaststellen of het systeem vrij is van fundamentele fouten én dat het gevolgde voortbrengingsproces geschikt is voor softwareontwikkeling.

Eis: Een beschrijving of verwijzing van het gebruik van een passend voorbrengingsproces moet onderdeel zijn van de TOPAAS-rapportage.

1.8 Leeswijzer

In hoofdstuk 2 schetsen we de hoofdlijnen van het proces en onderscheiden we de twee processtappen. In hoofdstuk 3 de kwalitatieve analyse en in hoofdstuk 4 de kwantitatieve analyse.

TOPAAS is opgesteld in samenwerking met:
Dr. Wouter Geurts (CGI)
Ir. Jaap van Ekris (Delta-Pi)
Ed Brandt (Refis)

Dr. ir. Gerben Heslinga (Intermedion)
Drs. Gea Kolk (Movares)
Prof. dr. ir. Jan-Friso Groote (TU/e)
Prof. dr. ir. Mariëlle Stoelinga (UT/Radboud Universiteit)

1.9 Documentenoverzicht

TOPAAS bestaat uit de onderstaande documenten:

Deel	Naam	Type
1	Handleiding (voorliggend document)	Kader (bindend)
2	Vragenlijst	Informatief
3	Modelonderbouwing	Informatief
4	Referentie- en validatiedatabase	Informatief
5	Onderhoudsproces (in concept)	Informatief
6	Audit framework (in concept)	Informatief

Om een TOPAAS-analyse te maken zijn deel 1 Handleiding en eventueel deel 2 Vragenlijst benodigd. Indien men meer wil weten over de achtergrond van TOPAAS, dan wordt verwezen naar deel 3 Modelonderbouwing. Deel 4 Referentie en validatiedatabase is de achterliggende database met de referentieprojecten, die benodigd waren voor het opzetten van het TOPAAS-model. Dit deel is alleen voor Rijkswaterstaat bedoeld. In deel 5 Onderhoudsproces staat omschreven hoe de documenten van het kader TOPAAS beheerd en onderhouden moeten worden door Rijkswaterstaat. Dit deel is alleen voor Rijkswaterstaat bedoeld. In deel 6 is een audit framework omschreven. Hierin staat geschreven hoe een auditor een audit op een TOPAAS-analyse dient uit te voeren. Dit deel is nodig voor degene die een audit wil uitvoeren.

1.10 Algemene tips

Software heeft de neiging om een eerste orde object in de foutenboom te vormen. Hiermee krijgt een TOPAAS-afschatting vrij snel serieuze gevolgen op de inschatting van RAMS-prestaties van kunstwerken. Een gedegen aanpak tijdens de analyse, met behulp van risicoanalisten die ruime ervaring hebben met softwarebetrouwbaarheid, is dan ook essentieel om een structuur van de software te krijgen die goed te beoordelen is. Daarnaast is een gevoeligheidsanalyse nuttig als input voor bijvoorbeeld een meetprogramma, waarmee Bayesiaans updaten kan worden gevoed.

Informatie: Een goede verslaglegging van zowel beslissingen als de onderliggende "bewijsstukken" is noodzakelijk, voor zowel de modellering als de kwantificering. Ook het communiceren van de beslissingen en uitkomsten naar stakeholders is van belang om acceptatie van de software te krijgen.

In deze handleiding beschrijven we handvatten voor zowel beslissingen (modellering én kwantificering) en het minimum aan vastlegging.

Het is echter verstandig, ook voor het toekomstig software-onderhoud, alle onderzochte ontwerpoplossingen ook onderdeel te laten zijn van het ontwerpdossier. Dergelijke ontwerprationales blijken in praktijk zeer waardevol voor het toekomstig onderhoud van de software.

2 Werkwijze

2.1 Hoofdlijnen van het proces

Het uitgangspunt van TOPAAS is dat de bijdrage van software aan het falen van een object wordt meegenomen via de taakuitvoering van de software, waarbij falen van taakuitvoering in een ongewenste of onveilige situatie resulteert. Software doet in het algemeen meer dan de in de foutenboom opgenomen taakuitvoering. De betrouwbaarheidsanalyse van software richt zich op de delen van de software die specifieke taken in een object realiseert.

Het basisconcept voor een TOPAAS analyse is de TUB (TaakUitvoeringsBlok), dat gezien moet worden als de software analogie van een hardwarecomponent. Een hardwarecomponent (vlinderklep, pomp, relais) heeft een heel duidelijke functie en daarmee enkele faalmodi. Daarmee is het al redelijk helder wat de basisgebeurtenis *'vlinderklep faalt'* betekent. In analogie hiermee dienen ook basisgebeurtenissen voor software te worden gedefinieerd. Deze moeten worden benoemd als het niet (op tijd) of niet juist vervullen van een aan de software toegewezen taak (of taken). Een basisgebeurtenis *'software crasht'* ontbeert de duidelijke toelichting welke aan een stuk software toegewezen taak niet uitgevoerd wordt en daarmee de missie in gevaar brengt. Een betere basisgebeurtenis is *'software stuurt vlinderklep niet aan'*. Het begrip TUB wordt in paragraaf 3.3.2 gedefinieerd.

Om te komen tot een goede afschatting van software-falen, kent een TOPAAS-analyse twee stappen:

- Een kwalitatieve modelering van software-falen, die tot doel heeft taakuitvoering te herkennen en deze op te delen in concrete TUBs (zie hoofdstuk 3);
- Een kwantitatieve faalkansbepaling, die tot doel heeft voor elke TUB een faalkans af te schatten (zie hoofdstuk 4).

De kwalitatieve modellering is noodzakelijk om de uitkomsten te kunnen relateren aan een overkoepelende foutenboom en is cruciaal om te komen tot een goede technische afbakening van TUBs. Vaak leidt deze stap ook tot een verdere uitwerking van software-falen in de foutenboom. Door goed aandacht te besteden aan deze stap kan men ook makkelijker naar een scherp afgebakende definitie van de taakuitvoering van een TUB komen. Deze scherp afgebakende definitie is noodzakelijk in de kwantitatieve analyse om de vragen te kunnen beantwoorden.

De kwantificering van de faalkans van een TUB gebeurt in twee deelstappen. De eerste stap is om aan de hand van een vragenlijst de TOPAAS-score te bepalen. Het doel van de vragenlijst is om op een structurele manier naar een onderbouwde afschatting toe te werken. De tweede stap is om de TOPAAS-score te interpreteren aan de hand van de aanspreekfrequentie.

Informatie: Niet alleen de uiteindelijke antwoorden zijn van belang, maar ook de achterliggende redenering en bewijsvoering. Hiermee wordt het proces van scoring transparant naar de opdrachtgever toe.

2.2 Algemene eisen aan bemensing

Voor een software-betrouwbaarheidsanalyse is een RAMS-specialist met een significante ervaring op het gebied van software-architectuur nodig. De RAMS-specialist moet inhoudelijk de discussie kunnen voeren over de opdeling van de software in TUBs, met in het achterhoofd ook al een realistisch haalbare kwantificering van deze TUBs. Dit vereist enerzijds een goed begrip van architectuurconcepten en anderzijds een goed inzicht in de werking van TOPAAS. Het is uiterst wenselijk dat de TOPAAS-specialist ervaren is in software intensieve systemen en dat de specialist onafhankelijk is van het ontwikkelteam.

2.3 Algemene eisen aan de werkvorm

De modellering en de afschatting van software-falen zijn voor een deel subjectieve activiteiten. De keuzes die gemaakt worden beïnvloeden de uiteindelijke gemodelleerde faalkans. De algemene voorgestelde aanpak vanuit de TOPAAS-werkgroep in dit proces is gebaseerd op consensus en transparantie. Door consensus te bereiken over essentiële keuzes in modellering en aspecten die de kwantificering beïnvloeden ontstaat intersubjectiviteit: een resultaat dat door meerdere stakeholders gedragen wordt en individuele denkfouten minimaliseert. Ook mogen er over feitelijke vaststellingen geen discussies meer zijn. Transparantie is nodig om achteraf te kunnen verifiëren welke keuzes gemaakt zijn en hoe deze doorwerkten in het verdere analyseproces.

2.4 Algemene randvoorwaarden

Verslaglegging van de keuzes wat betreft de inzet van de analisten en werkvorm die gemaakt zijn, alsook de redenen waarom ze genomen zijn, is essentieel om het analyseproces transparant te krijgen. Vastlegging van de beperkingen in de aanpak (bijvoorbeeld beperkingen in de diepgang van de analyse) zijn ook onderdeel van deze verslaglegging.

2.5 Algemene eisen aan verslaglegging

Deze handleiding laat bewust in het midden hoe de verslaglegging ingericht wordt en hoe de kwaliteit hiervan gecontroleerd en geborgd wordt. Het is denkbaar dat een verantwoordingsdocument (RAMS-dossier) structureel wordt opgebouwd waar de TOPAAS-analyse een structureel onderdeel van is. Andere werkmodellen waarmee door middel van audits een onafhankelijke verslaglegging ontstaat kunnen ook een werkbaar model zijn. Dit is een keuze die besproken moet zijn met de opdrachtgever en de bouwer van de software.

De kern is dat alle richtinggevende beslissingen die noodzakelijk zijn voor onafhankelijke toetsing vastgelegd moeten worden.

Eis: De manier waarop het analyseproces is ingericht en de kwalificaties van de betrokken analisten moet worden vastgelegd.

3 Kwalitatieve analyse

3.1 Doel en overzicht

De kwalitatieve analyse heeft tot doel de topgebeurtenis te decomponeren zodat de bijdragen van hardware, software en menselijk handelen separaat opgenomen worden. Door invoering van het begrip taakuitvoering wordt deze verbinding gelegd: hardware *"aansturen van een motor"*, software *"nemen van een beslissing"*, menselijk handelen *"uitvoeren van een noodreparatie volgens werkinstructie"*.

Hiermee wordt bedoeld dat een basisgebeurtenis (falende taakuitvoering) gekoppeld wordt aan collecties broncode waarvan in een later stadium een faalkans te berekenen is. Deze collecties broncode noemen we TUBs (specifieker gedefinieerd in paragraaf 3.3.2 *"TUB"*). Opgemerkt moet worden dat de afbakening in TUBs mogelijk niet overeenkomt met de in software engineering gangbare technische onderverdeling van broncode in packages, modules of functies.

De kwalitatieve analyse leidt dus tot de identificatie van de TUB: een minimale software doorsnede die de taakuitvoering doet. De TUB dient hierbij scherp gedefinieerd te worden om zo het risico op overschatting van de faalkans te verkleinen. Meer regels code bevatten statistisch gezien meer fouten (defects), maar deze leiden niet allemaal tot falen van de taakuitvoering. Anderzijds is het vaak uit praktische overwegingen (meetbaarheid) handig om niet te scherp af te bakenen. Dit is een expliciet onderwerp in paragraaf 3.7 *"Kiezen van de TUBs"* en paragraaf 3.8 *"Praktische handigheden tijdens het ontwerp".*

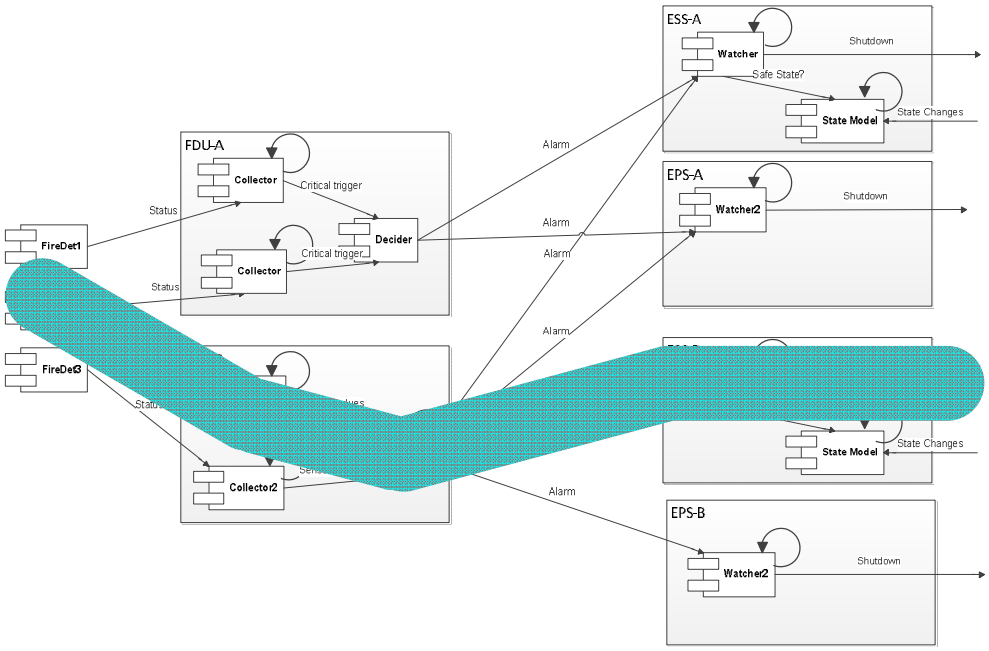
De kwalitatieve analyse kent twee inputs:

- De voorlopige foutenboom;
- De bouwtekeningen van de software: de architectuur.

Aan deze inputs worden inhoudelijk eisen gesteld, die in paragraaf 3.4 *"Eisen aan de inputdocumentatie"* gedetailleerder worden behandeld.
De stap van taakuitvoering naar TUBs is vaak niet eenvoudig te maken. Omdat in de architectuur de relatie tussen taakuitvoering en software modules vastgelegd is, beginnen we daarom met en ruwe schets: we identificeren de packages/modules/functies die betrokken zijn bij een taakuitvoering. Daarna worden de grenzen aangescherpt: de exacte grenzen van de TUBs worden bepaald door de modules te groeperen qua interne eigenschappen (procespersistentie, ontwikkelteam, etc.), en impact op een specifieke taakuitvoering en vervolgens te clusteren.

Op hoofdlijnen bestaat identificatie van de TUBs uit de volgende deelstappen:

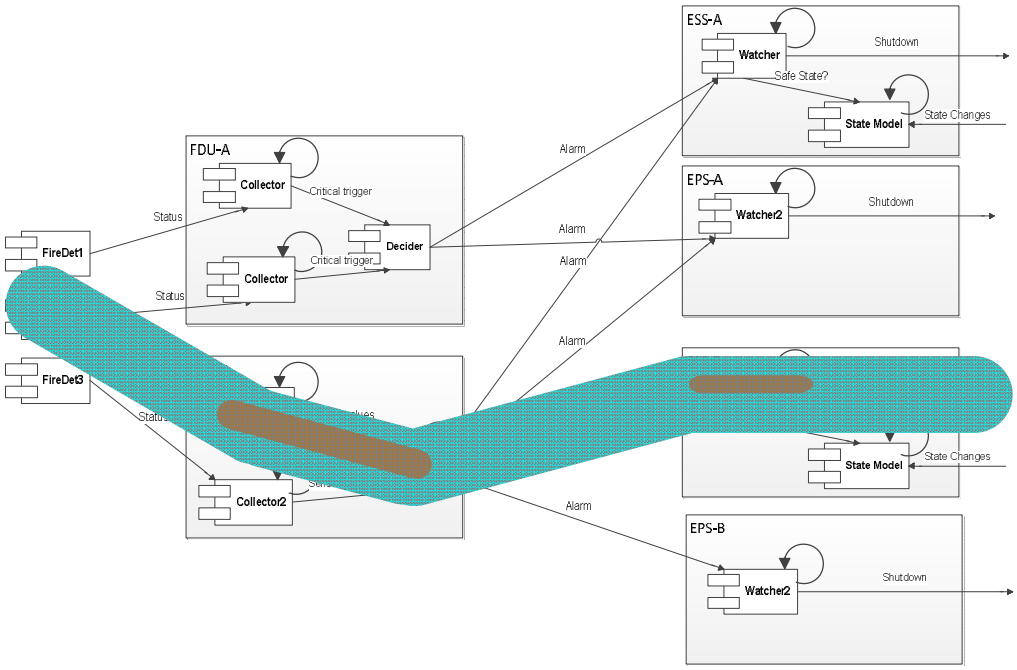
- Identificeren van software-gebeurtenissen in de foutenboom. Hierbij worden voor een specifieke RAMS-prestatie de relevante taakuitvoeringen van software geïdentificeerd (TUBs). Dit is uitgewerkt in paragraaf 3.5 *"Herkenning van falende taakuitvoering in de foutenboom".*
- Identificeren van de TUBs in de architectuur. Hierbij wordt vanuit een overzichtstekening geïdentificeerd welke modules betrokken zijn bij een taakuitvoering. Hierbij moet men aan de volgende selectie denken:



Figuur 4 Identificeren van de TUB

Hierin worden in architectuurschema's de modules als het ware gemarkeerd als ze betrokken zijn bij TUB. Dit is verder uitgewerkt in paragraaf 3.6 *"De identificatie van falende taakuitvoering in de architectuur"*.

- Het kiezen van de juiste afbakening van de TUBs op basis van de interne eigenschappen (zoals bijvoorbeeld ontwikkelorganisatie, procespersistentie, etc.) van de TUB en extern waarneembare effect van de TUB. In deze stap worden ook de delen van de modules gemarkeerd die beschouwd worden en die buiten beschouwing worden gelaten voor de specifieke taakuitvoering. Schematisch weergegeven (twee TUBs gemarkeerd in blauw) dus dit:



Figuur 5: Let op het blauw gearceerde gedeelte bij FDU-B en ESS-B

Dit is uitgewerkt in paragraaf 3.7 *"Kiezen van de TUBs".*

3.2 Proces en werkvorm

3.2.1 Bemensing

Voor kwalitatieve modellering van software zijn de architect van de software en een RAMS-specialist met een significante ervaring in software-architectuur nodig. De RAMS-specialist moet inhoudelijk de discussie kunnen voeren over de goede opdeling van de software in TUBs, met in het achterhoofd ook al een realistisch haalbare kwantificering van deze TUBs. Dit vereist enerzijds een goed begrip van architectuurconcepten en anderzijds een goed inzicht in de werking van TOPAAS.

3.2.2 Werkvorm

De kwalitatieve analyse is het eenvoudigst uit te voeren als een kleine workshop. Tijdens de workshop worden de volgende twee rollen onderscheiden: architect en RAMS-specialist. De rol van de architect is het goed vormgeven van de oplossing. De rol van de RAMS-specialist is het om de betrouwbaarheid van de beschreven architectuur te modelleren en de onderbouwing daarvan vast te leggen. Het identificeren van de taakuitvoeringen en TUBs moet dan ook een gezamenlijke actie zijn.

Er moet ook zeker ruimte zijn voor discussie: de afbakening van TUBs is geen deterministisch proces en er zijn meerdere valide opdelingen denkbaar. De discussie hierover, en wat dat betekent voor de uiteindelijke modellering, is een essentieel onderdeel van de onderbouwing en moet dus ook worden vastgelegd.

3.2.3 Randvoorwaarden

Er moet inzage zijn in de opzet van het totale systeem dat de RAMS-prestaties levert, de aanwezigheid van ontwerpkennis (in geschreven of menselijke vorm) en de achterliggende keuzes.

3.2.4 *Verslaglegging*

Informatie: De uitkomst van de kwalitatieve modellering (foutenboom model met taakuitvoeringen, opdeling TUBs) en alle keuzes die daarin zijn gemaakt moeten worden vastgelegd.

3.3 Definities en concepten

3.3.1 *Taakuitvoering*

Om gedrag van software in de foutenboom tastbaar te maken, wordt het concept taakuitvoering geïntroduceerd. Taakuitvoering is gedefinieerd als:

Eis: Taakuitvoering is het uitvoeren van taken door software waarbij die uitvoering extern waar te nemen is en waarvan men kan vaststellen dat deze goed of fout wordt uitgevoerd.

Voorbeelden van taakuitvoering zijn:

- het bepalen van waterstanden op basis van werkende sensoren;
- het besluiten tot het sluiten van een waterkering op basis van een waterstand;
- het activeren van de brandblusinstallatie op basis van een signaal van een brandmelder.

Taakuitvoering is het equivalent van menselijk handelen of elektrotechnisch schakelen. Voor taakuitvoering door mensen is een opgeleid persoon voldoende. Voor taakuitvoering door elektrotechniek is een werkende stroomkring nodig, die los gemodelleerd wordt. Zo dient taakuitvoering met software en computerresources ook gemodelleerd te worden.

TOPAAS gaat ervan uit dat software faalt als de RAMS-prestatie van de installatie in gevaar gebracht wordt (zoals beschreven in de topgebeurtenis) ofwel dat het resultaat, beslissing of aansturing van de software afwezig, verkeerd of niet tijdig is.

Hierbij wordt expliciet gerefereerd naar de topgebeurtenis in de foutenboom: een fout in de software hoeft zeker geen falen van de software te betekenen. Software kan ondanks fouten nog steeds gewenst gedrag vertonen. Dit is zeker het geval als er sprake is van fail-safe gedrag, waar softwarefouten altijd veilig gedrag opleveren. Deze fouten gaan dan mogelijk ten koste van de beschikbaarheid van de installatie. TOPAAS beschouwt een TUB met fouten niet als gefaald in zijn (veiligheids)taak als deze fail-to-safe gedrag vertoont in de specifieke situatie. Het is een belangrijk onderdeel van de faaldefinities in de foutenboom om onderscheid te maken tussen beschikbaarheids- en veiligheidsprestaties van de installatie.

3.3.2 *TUB*

In TOPAAS wordt een taakuitvoeringsblok (TUB) als volgt gedefinieerd:

Informatie: Een TUB is een afgebakende verzameling logische regels programmacode (of grafisch equivalent) die een taakuitvoering realiseert, waarbij het gedrag observeerbaar is en de interne eigenschappen gelijk zijn.

TUBs zijn specifiek testbare doorsnedes van de broncode die gerelateerd zijn aan taakuitvoering, maar komen mogelijk niet overeen met de indeling van de in software engineering gebruikelijke indeling in packages, modules of functies.

Hierbij geldt dat:

- Een duidelijke afbakening te onderkennen is ten opzichte van andere stukken code;
- Er een onderkende functionele doelstelling in het totaal van het systeem aanwezig is;
- Te verifiëren kwaliteitseigenschappen aanwezig, testbaar en waarneembaar zijn;
- De interne eigenschappen (ontwikkelteam, voortbrengingsproces, gebruikte ontwikkeltools, etc.) identiek zijn.

Een TUB kan dus uit een deel van een softwaremodule bestaan, maar ook uit een collectie van modules die over verschillende fysieke PLCs verspreid staan. De kern is dat een TUB onderscheidbaar is en enige mate van onafhankelijkheid (in zijn faalgedrag) kent. In praktijk is een TUB een doorsnede van één of meerdere software modules, waarbij de taakuitvoering van die TUB extern waarneembaar is. De verzameling logische regels wordt in zeer brede zin beschouwd:

- een TUB kan een deel van een specifieke module zijn (bijvoorbeeld een specifieke functie);
- een TUB kan alle software op een dedicated hardware component (PLC-node) zijn;
- een TUB kan alle broncode binnen een proces of executable op een mainframe zijn;
- een TUB kan een doorsnede van de broncode van een collectie samenwerkende processen in een systeem zijn.

Voorbeelden van TUBs zijn:

- Alle broncode die de bepaling van de waterstand doet op basis van een aantal sensoren;
- De broncode die op basis van een bepaalde waterstand besluit tot sluiting van een waterkering;
- Een stuk broncode die op basis van de gemeten luchtkwaliteit de ventilatoren aanstuurt;
- Een stuk broncode die een ventilator in noodbedrijf zet als de brandmelders actief worden.

3.4 Eisen aan de inputdocumentatie

3.4.1 Eisen aan de voorlopige foutenboom

De voorlopige foutenboom is van belang omdat het herkennen van taakuitvoering daar begint. In de foutenboom wordt op specifieke punten geconstateerd dat falen van software invloed heeft op de RAMS-prestatie van het kunstwerk. Aan de definitie van deze basisgebeurtenissen worden vanuit TOPAAS eisen gesteld om zeker te stellen dat het ook om falende taakuitvoering gaat.

Kern van deze eis is dat het expliciet om taakuitvoering moet gaan die faalt. Het grootste zorgpunt is dat er dubbelzinnige definities van falen worden gehanteerd, waardoor de faalkans niet afgebakend is. Zo is een enkele eerste orde basisgebeurtenis *"Software fout"* te algemeen voor een analyse. Het gaat er om dat

de foutenboom expliciet beschrijft wat de software geacht wordt te doen (taakuitvoering), maar het verkeerd afhandelen daarvan veroorzaakt de topgebeurtenis veroorzaakt. Idealiter wordt in de foutenboom de faalwijze op het niveau van *"PLC stuurt klep niet aan"* benoemd, waarna deze verder gedecomponeerd wordt in de PLC-hardware en de taakuitvoeringen van de PLC-software.

De foutenboom moet initieel software-falen specifiek benoemen, maar niet gedetailleerd willen uitwerken in de onderliggende oorzaken. Het is aan de kwalitatieve analyse om de foutenboom verder in te vullen met geïdentificeerde TUBs als onderliggende oorzaak. Meest logische abstractiepunt voor de foutenboom is als een sensor een signaal geeft en de software geacht wordt een actuator aan te sturen.

3.4.2 Eisen aan de bouwtekeningen van de software

Als men een analyse van de software wil maken, moet men een set "bouwtekeningen" hebben om als basis te dienen voor de doorontwikkeling van de foutenboom. Net als bij werktuigbouwkundige en elektrotechnische bouwtekeningen moet de architectuur van het systeem een consistent beeld scheppen van de samenhang van alle objecten die bij elke missie betrokken zijn. Belangrijke vragen zijn bijvoorbeeld of er single points of failure zijn en welke kritieke ketens specifiek gedrag van een kunstwerk bepalen.

Als dergelijke bouwtekeningen ontbreken dienen deze opgesteld te worden op basis van interviews of experimenten met het systeem. Uiteraard dienen dergelijke reverse-engineering activiteiten wel inhoudelijk kwalitatief geborgd en beschreven te worden.

De bouwtekeningen moeten dus inzage geven in hoe de software opgebouwd is. In de foutenboom wordt een specifieke basisgebeurtenis geïdentificeerd. Dit is de falende taakuitvoering die nader onderzocht wordt aan de hand van de architectuur. Om dit mogelijk te maken moeten de bouwtekeningen via drie overzichtentypes inzage geven in de volgende vraagstukken:

- Welke software componenten vormen het draaiend systeem: processen, handlers?
- Hoe is een taakuitvoering softwarematig uit processen opgebouwd? Met name:
    - Hoe werken de processen samen om het beoogde resultaat te bereiken?
    - Hoe communiceren processen onderling?
    - Zijn er parallelle of redundante processen?
    - Zijn er watchdogs?
    - Zijn er on-demand processen?
- Welke broncode draait in welke processen?
- Wat is het faalgedrag van componenten? Is dat functionaliteit of gerealiseerd op een 'hoger' niveau?
    - fail-safe gedrag is in een component aanwezig,
    - bij falen stopt een component met werken: redundantie of een watchdog geven aanleiding tot alternatieve successcenario's.
- Van welke hardware is de taakuitvoering afhankelijk?
- Waarvan is een softwaremodule in termen van resources (CPU, memory, disk) afhankelijk en met welke andere processen deelt hij deze?

Met deze informatie kan de taakuitvoering worden opgedeeld in TUBs.

In tegenstelling tot andere technische disciplines is er nog geen uniforme tekentechniek die al deze aspecten automatisch afdekt. Er groeit in de ICT-sector wel consensus dat architectuurbeschrijvingen uit meerdere views bestaan. Een view beschrijft een systeem op een specifieke manier met een bepaald doel voor ogen. Omdat er vaak meerdere partijen verschillende soorten informatie nodig hebben, betekent dit dat er ook verschillende views van hetzelfde systeem gemaakt moeten worden, waarbij de specifieke eigenschappen expliciet belicht moeten worden.

Informatie: In de ISO/IEC/IEEE 42010 [2] worden de generieke randvoorwaarden voor systeemontwerpen op basis van verschillende views vastgelegd, alsmede de rationales om ontwerpbeslissingen vast te leggen.

Om hier meer inhoud aan te geven kan bijvoorbeeld de 4+1 methode van Kruchten [3] gebruikt worden, die verder uitgewerkt is in Bijlage B: Gebruik van het 4+1 model (informatief).

3.5 Herkenning van falende taakuitvoering in de foutenboom

De identificatie van falende taakuitvoering in de foutenboom is eenvoudig, mits de hiervoor gestelde eisen aan falende taakuitvoering helder zijn beschreven door de basisgebeurtenissen. Als er één enkele eerste orde basisgebeurtenis is opgenomen met de naam "software fout", dan moet dat uiteraard verder uitgewerkt worden naar de specifieke faalvormen die van belang zijn voor de topgebeurtenis van de foutenboom.

Er moet goed onderscheid gemaakt worden tussen de verschillende taakuitvoeringen waarbij software betrokken is. Het aansturen van bijvoorbeeld een klep is een taakuitvoering (de happy flow), maar het omgaan met een falende klep wordt door TOPAAS expliciet als een andere taakuitvoering beschouwd (de unhappy flow). Zelfs als de klepaansturing één stuk software is. De reden hiervoor is dat de vaak complexe unhappy flow alleen van belang is als de klep faalt, terwijl de veel eenvoudigere happy flow veel frequenter wordt gebruikt en van nature dus vele malen betrouwbaarder is. Het omgaan met foutafhandeling door software wordt verder uitgewerkt in paragraaf 3.8.3.

Eis: De geïdentificeerde vormen van falende taakuitvoering moeten expliciet vastgelegd worden.

Deze falende taakuitvoeringen worden verder via TOPAAS uitgewerkt en het is voor externe reviewers (bijvoorbeeld Rijkswaterstaat) van belang om inzage te hebben in de beslissingen die daaraan ten grondslag liggen.

3.6 De identificatie van falende taakuitvoering in de architectuur

Om de falende taakuitvoering in de architectuur te kunnen identificeren, moet eerst gedefinieerd worden wat onder de architectuur verstaan wordt. Hierna gaan we in op hoe in een dergelijke beschrijving de taakuitvoering geïdentificeerd kan worden.

Hier wordt een methode uitgewerkt die het breedst toepasbaar lijkt. Specifieke inrichtingen van processen bij leveranciers kunnen aanleiding geven tot een andere vorm van architectuurbeschrijving die hier niet eenvoudig op te matchen is. Voor

TOPAAS maakt dit niet uit, zolang de opdeling van falende taakuitvoering naar TUBs maar mogelijk blijft, en dat de TUBs te koppelen zijn aan brokken software (modules) met interne en externe metrieken.

Door alle views van een architectuur te combineren kan een beeld worden opgebouwd hoe het systeem werkt, vergelijkbaar aan een hydraulische, elektrische of mechanische bouwtekening. Deze bouwtekeningen moeten dus óf door leveranciers opgeleverd worden bij oplevering óf door een software-analist in retrospect moeten worden vastgesteld op basis van alle andere aanwezige documentatie.
Als de architectuur beschreven is, dan is het koppelen van de falende taakuitvoering aan de architectuur vrij eenvoudig. Door de taakuitvoering te volgen door de architectuur en te identificeren welke delen van modules/broncode geraakt worden, kunnen de TUBs geïdentificeerd worden. Ook kunnen er verschillende scenario's geïdentificeerd worden die tot succes leiden.

Bijvoorbeeld: als de falende taakuitvoering aangeeft dat er sprake is van "geen sluitcommando gegeven op hoogwater", dan moet men in de scenario's op zoek naar alle manieren waarop het sluitcommando tot stand kan komen.

Er kunnen uiteraard meerderde scenario's zijn die tot succes leiden. Redundantie en fail-safe gedrag zijn typische voorbeelden van meerdere scenario's die tot succes kunnen leiden. In de uiteindelijke foutenboom zal dit leiden tot een AND-poort waarbij al de redundante scenario's moeten falen.

Doordat er TUBs geïdentificeerd zijn, is het ook mogelijk via de physical view te bepalen van welke hardware de specifieke scenario's afhankelijk zijn, zodat ook de hardware specifiek in de foutenboom kan worden opgenomen.

Een kanttekening moet worden geplaatst bij het opnemen van hardware componenten in de foutenboom. Veel hardware in de IT, zoals firewalls, routers, SANs en NAS-en, bevat firmware. Firmware kan in de theorie veel invloed hebben op het presteren van een systeem. TOPAAS gaat er impliciet vanuit dat voor dergelijke hardware de faalkans inclusief de software is, tenzij de taakuitvoering direct de dergelijke hardware bestuurt (bijvoorbeeld het openzetten of dichtzetten van een poort op de firewall). Als de taakuitvoeringen dergelijke hardware alleen gebruiken als onderdeel van de oplossing, dan wordt in de regel alleen het hardware+falen in de foutenboom gemodelleerd en de software op deze hardware hoeft dus niet in TOPAAS gemodelleerd te worden.

Eis: De volgende zaken moeten vastgelegd worden:
- de beschrijving van de systeemarchitectuur
- de koppeling van de falende taakuitvoeringen aan de scenario's, inclusief de onderlinge verhoudingen (of scenario's redundant zijn);
- de geïdentificeerde broncode per taakuitvoering;
- de afhankelijkheid van de specifieke scenario's van hardware (indien deze nog niet in de foutenboom zijn opgenomen).

3.7 Kiezen van de TUBs

Het is verstandig om op een goed abstractieniveau de TUB te definiëren door de taakuitvoering van software als basis te nemen.

Software modules en TUBs zijn niet één-op-één aan elkaar te relateren. De code moet immers te relateren zijn aan de taakuitvoering, waarbij die taakuitvoering onderdeel is van de foutenboom. In veel gevallen is de module wel intern coherent (vormt bijvoorbeeld een klepbesturing) maar zijn de verschillende delen van de module betrokken bij verschillende taakuitvoeringen. Zo kan een deel betrokken zijn bij het aansturen van een klep en een ander deel bij het afhandelen van foutmeldingen van dezelfde klep. Software die niet te relateren is aan de taakuitvoering en daar ook niet verstorend op kan werken is geen onderdeel van de TUB. In veel gevallen zal een TUB dus delen van één of meerdere software modules bevatten. In praktijk is een TUB dus een dwarsdoorsnede van één of meerdere softwaremodules die te relateren aan een specifieke taakuitvoering.

TOPAAS kijkt bewust vrij abstract tegen software aan: het gaat om de software en niet om de onderliggende hardware of firmware/besturingssysteem. De gebruikte TUBs abstraheren van het besturingssysteem, interne software structuur, libraries, drivers en netwerkverbindingen. Dit komt omdat een OS, driver of library in de regel niet spontaan faalt, maar eerder faalt omdat de aanroep vanuit de applicatie/module niet correct is. Met de huidige ontwikkelomgevingen is een Remote input/output of een Remote Procedure Call ook zo transparant dat communicatie over netwerken, behoudens het reeds gemodelleerde hardwarematige falen daarvan, eigenlijk geen reden meer is voor falen. Het falen zit dan eerder in de slecht ontworpen communicatie tussen modules, dan in het feit dat de communicatie via een netwerk verloopt.
Bij elke falende taakuitvoering is de betrokken broncode geïdentificeerd. In deze stap is het de bedoeling om een clustering en aanscherping van deze broncode te maken zodat daardoor ontstane TUBs ook gebruikt kunnen worden in de kwantitatieve analyse.

Informatie: Vuistregel is dat een TUB zoveel mogelijk code moet omvatten, tenzij de code onverenigbaar is.

Bij de clustering blijven de eisen voor TUBs gelden, waarbij de meest belangrijke is dat het gedrag extern waarneembaar moet zijn. Dit betekent dat het wel mogelijk moet zijn input en output aan elkaar te relateren en vast te stellen of dat correct gedrag is.

De broncode is onverenigbaar in een TUB als:

- de software gemaakt is door een ander ontwikkelteam. Hierdoor zijn veel (proces)parameters in de kwantitatieve analyse anders, zoals ervaring van de ontwikkelaars en cultuur in de organisatie;
- de natuur van het proces anders wordt, bijvoorbeeld doordat het ene proces een continue actief proces is en het andere een on-demand proces is dat alleen actief is in specifieke situaties.

Een punt van aandacht is dat de broncode soms maar gedeeltelijk onderdeel uitmaakt van een TUB en dus ook zo geadministreerd moeten worden in alle vervolgstappen. Dit heeft tot gevolg dat bijvoorbeeld een klepbesturing in meerdere TUBs gescheiden moet worden:

- één TUB (doorsnede broncode) die de klep onder normale omstandigheden aanstuurt;

- één TUB (doorsnede broncode) die falen van de klep afhandelt, die onderdeel is van een ander onderdeel van de foutenboom (te weten het incorrect afhandelen van een falende klep);
- één stuk code dat statusrapportages naar de user interface stuurt en niet meedoet in de faalkansanalyse.
- Het is verstandig om een clustering van modules als een aparte TUB te beschouwen indien deze gedeeld wordt met een andere taakuitvoering (zie paragraaf 3.8.2).

Eis: De volgende zaken moeten worden vastgelegd:
• De positionering van taakuitvoering over TUBs en de manier waarop de TUBs opgebouwd zijn uit broncode;
• De rationale achter deze opdeling.

3.8 Praktische handigheden tijdens het ontwerp

3.8.1 Slim opknippen van complexe taakuitvoering

Eén van de zaken waardoor snel ongunstige faalkansen ontstaan, is de combinatie van een complexe taakuitvoering die bovendien weinig gebruikt wordt. Vooral veiligheidssystemen die acteren op een complex berekende grenswaarde hebben hier last van. Als men deze in de modellering verdeelt in twee TUBs (één complexe TUB die een meting continue uitvoert en rapporteert en één eenvoudige TUB die slechts acteert bij overschrijding van de grenswaarde), dan wordt de faalkans aanzienlijk realistischer. Zeker als de complexe berekening onderworpen wordt aan goede monitoring, zijn hier zeer goede betrouwbaarheden te halen.

3.8.2 Goed omgaan met gedeelde modules

Sommige systemen kennen meerdere modi: reguliere modus (acties op basis van een grenswaarde), een noodstand en/of een onderhoudsbedrijf (waar handelingen niet volledig automatisch worden uitgevoerd) De goede modellering van deze modi is vrij essentieel: een taakuitvoering die weinig wordt aangesproken kent van nature een slechtere faalkans.

Denk bijvoorbeeld aan een ventilatiesysteem dat gebruikt wordt voor zowel de klimaatbeheersing in een tunnel als ook voor de noodventilatie in geval van brand. Nu kan men op een ventilatiesysteem de ingang "noodventilatie" aanbrengen, een input die waarschijnlijk nooit gebruikt en vrijwel niet getest wordt, of men kan de ventilatie ook in geval van brand gewoon een setpoint meegeven. Het laatste geval heeft als voordeel dat de kennis over het "waarom" van het setpoint beperkt blijft tot één enkele module. De kern is dat de kennis over een dergelijke toestand een minimale spreiding moet kennen over softwaremodules. Omdat de afhandeling door een ventilatiesysteem van een setpoint veel gemonitord en getest wordt, wordt deze TUB veel betrouwbaarder dan via een specialistische ingang met de verspreiding van deze inhoudelijke kennis van het "waarom" een setpoint gezet wordt. Door dit juist te ontwerpen én modelleren ontstaat er dus een aantoonbaar robuustere oplossing.

Hier moet wel heel expliciet gekeken worden naar de aansturing om vast te stellen of exact dezelfde delen van de softwaremodule worden gebruikt: als de aansturing anders is, moet men er eigenlijk vanuit gaan dat de TUB in stukken geknipt moet worden. Het maakt dus uit of een brandmeldinstallatie de ventilatie in noodbedrijf plaatst of gewoon op vol vermogen zet.

3.8.3 Omgaan met falende hardware (opvolgfalen)

Veel taakuitvoeringen gaan niet alleen over software, maar ook over de interactie tussen hardware en software. Hierin wordt het goed afhandelen van falende hardware ook van belang. In de regel is deze afhandeling groot en complex. Er ontstaat een situatie die onverwacht anders is en eerst door de software "begrepen" moet worden voordat deze correct afgehandeld kan worden.

TOPAAS gaat er vanuit dat het afhandelen van falende hardware tot andere taakuitvoering hoort dan het initieel aansturen van dezelfde hardware en dat dit ook in de foutenboom zo gemodelleerd is. Bijvoorbeeld het falen van een klep faalt op twee manieren :

- De aansturingssoftware stuurt een klep niet, te laat of verkeerd aan op basis van een geldige vraag van opening van de klep;
- De klep opent niet door mechanisch falen en de aansturingssoftware handelt dit te laat of verkeerd af en stuurt bijvoorbeeld de redundante klep niet goed aan. Dit wordt opvolgfalen genoemd.

In de foutenboom moet dit dus aanleiding geven tot twee aparte basisgebeurtenissen voor de software. Het modelleren in twee aparte basisgebeurtenissen geeft in praktijk een scherper beeld van de faalkans van de software:

- Het primaire aansturen zonder fouten (happy flow) is vaak diepgaand getest en kent in de praktijk vaak een zeer beperkte omvang en complexiteit. Hierdoor is deze software vaak betrouwbaar;
- Het afhandelen van fouten van de hardware (unhappy flow) is vaak minder diepgaand getest en is in de regel door de vele condities en uitzonderingen bovendien groot en complex. Hierdoor is deze software vaak merkbaar onbetrouwbaarder dan de primaire aansturing van dezelfde hardware. In de meeste foutenbomen is dit in praktijk geen probleem: hardware kent een hoge betrouwbaarheid (vaak in de ordegrootte van 10^{-4} tot 10^{-5} op aanvraag). Hierdoor heeft het slecht afhandelen van hardware falen geen merkbare invloed meer op de uiteindelijke betrouwbaarheid van de taakuitvoering.

4 Kwantificering falen

4.1 Kwantificatieproces

4.1.1 Bemensing

Voor de kwantificeringsstap op basis van TOPAAS zijn bij voorkeur aanwezig:

- Een, bij voorkeur, onafhankelijke RAMS-specialist met een lange ervaring in softwarearchitectuur;
- De softwarearchitect;
- De ontwikkelaars;
- De projectleider;
- De kwaliteitsmanager van het project.

4.1.2 Werkvorm

In paragraaf 4.2 staat de TOPAAS-vragenlijst. De werkvorm voor scoring van de TOPAAS-vragen is afhankelijk van de insteek:

- als het gaat om een intern gedreven TOPAAS analyse, dan kan men volstaan met een workshop waar gezamenlijk de vragen worden beantwoord en bediscussieerd. Consensus over de antwoorden en de gevolgen daarvan is dan wellicht belangrijker dan de uitkomst van de TOPAAS-afschatting. Notulering van de besproken zaken is dan nuttig, ook als de rationale achter beslissingen.
- als de analyse onderdeel uit maakt van de goedkeuring van een installatie, dan zal er eerder voor een auditachtige aanpak gekozen worden. De nadruk ligt dan sterk op bewijsvoering en schriftelijke onderbouwing van de keuzes.

Tot slot bevat paragraaf 4.3 bevat praktische tips.

4.1.3 Randvoorwaarden voor kwantificering

Randvoorwaarde voor goede resultaten is eerlijkheid en openheid over de feiten. TOPAAS is niet primair bedoeld als auditinstrument met *"checks and balances"*. TOPAAS dwingt een leverancier een behoorlijk diepe kijk in de keuken te geven. Dit proces valt of staat dan ook met het vertrouwen dat men in de onafhankelijke RAMS-specialist heeft. Onafhankelijkheid van de uiteindelijke leverancier en van de opdrachtgever is gewenst om te voorkomen dat de leverancier het gevoel heeft dat alles één-op-één doorgebriefd wordt naar de uiteindelijke opdrachtgever.

Een integraal auditframework, dat meet welke bewijsstukken aanwezig zijn, heeft waarde voor het proces van TOPAAS-analyses. De aanwezigheid van bewijsstukken, die ook onder reguliere *Quality Assurance of Product Assurance* vallen, geeft meer extern vertrouwen aan een rapport. Omdat TOPAAS zich richt op de inhoudelijke kwaliteit van een product, dienen de bewijsstukken inhoudelijk te worden beoordeeld, maar leidt het ontbreken van bewijsstukken niet tot een lagere score. Iedere score dient wel onderbouwd te worden door middel van documentatie. Dat kan documentatie zijn die in het kader van het TOPAAS-onderzoek wordt opgesteld.

4.2 TOPAAS-vragenlijst

De TOPAAS-score wordt bepaald door per aspect een vraag te beantwoorden. Het antwoord bepaalt de aspectscore. In totaal zijn er 15 aspecten, dus ook 15 vragen,

die per TUB[1] beantwoord moeten worden. In dit hoofdstuk worden de aspecten beschreven en wordt ook het minimale niveau van bewijsvoering en vastlegging beschreven.

TOPAAS is gemaakt om te kunnen omgaan met onzekerheid: als er niets bekend is en aanvullend onderzoek praktisch onmogelijk is, dan moet er voor de keuze *"Onbekend"* gekozen worden. Dit heeft in de regel wel tot gevolg dat de faalkans van de TUB ongunstiger wordt. Als men helemaal niets weet van een TUB, dan eindigt deze op een faalkans van één. Dit is uiterst conservatief, maar komt wel overeen met de perceptie van de expertgroep. Hoe meer men weet, hoe beter de faalkans van een TUB wordt.

Als de mogelijkheid er is om redelijkerwijs onderzoek te doen, moet dit ook gedaan worden. In de praktijk zou men eigenlijk alleen *"Onbekend"* kunnen scoren als de TUB bestaat uit COTS-softwarecomponenten waarvan een leverancier weigert inzage te geven, of bij oude legacy-systemen waarvan de projectdocumentatie niet meer te achterhalen is. Het scoren van *"Onbekend"* bij een vraag verplicht de invuller wel om hierover verantwoording af te leggen in de rapportage:

- waarom het redelijkerwijs niet mogelijk was om scoring uit te voeren;
- waarom het onwaarschijnlijk is dat er sprake kan zijn van een slechtere (numerieke) score dan *"Onbekend"*.

4.2.1 Ontwikkelproces

Het gebruik van de IEC 61508 heeft veel invloed op de kwaliteit van het eindproduct. Wel wordt bewust conservatiever geschat dan dat de IEC suggereert: een SIL-4 systeem zal op basis van deze meting een faalkans van 10^{-3} krijgen. Dit is omdat het causale verband tussen het SIL-level en de betrouwbaarheid van het systeem niet wetenschappelijk is aangetoond.

1 Het ontwikkelproces voldoet aan één van de SIL's van de IEC 61508		
1	Onbekend, het ontwikkelproces voldoet niet aantoonbaar aan een SIL-niveau.	0
2	Ontwikkelproces voldoet aantoonbaar niet aan een SIL-niveau door het gebruik van Not Recommended practices.	½
3	Ontwikkelproces voldoet aantoonbaar aan SIL-1 niveau.	-½
4	Ontwikkelproces voldoet aantoonbaar aan SIL-2 niveau.	-1
5	Ontwikkelproces voldoet aantoonbaar aan SIL-3 niveau.	-2
6	Ontwikkelproces voldoet aantoonbaar aan SIL-4 niveau.	-3

Voorbeelden van *Not Recommended Practices* onder de IEC 61508-3, tabel A.2 [1] zijn:

- Gebruik maken van kunstmatige intelligentie
- Dynamische herconfiguratie van de procesallocatie op basis van beschikbare resources. Let wel: dit gebeurt in praktijk ook bij oplossingen in gevirtualiseerde serveromgevingen en clusteromgevingen.

Als men niet aantoonbaar voldoet aan een SIL-niveau, dan valt men automatisch terug naar het niveau *"Onbekend, het ontwikkelproces voldoet niet aantoonbaar aan een SIL niveau"*. Het aantoonbaar voldoen aan de eisen van een specifiek SIL-level

[1] In sommige analyses kan blijken dat veel taken door dezelfde (niet nader te splitsen) software entiteit worden uitgevoerd. Is er sprake van meerdere TUBs, dan zal de score per TUB hetzelfde zijn.

voor een TUB moet onafhankelijk worden vastgesteld, aan de hand van de daarvoor gestelde eisen in de IEC 61508. Dit kan onderdeel zijn van een TOPAAS-beoordeling, maar het kan ook een resultaat zijn van een audit. Deze beoordeling moet door een onafhankelijke partij zijn uitgevoerd op het specifieke project dat de TUB heeft voortgebracht.

Vanuit de randvoorwaarden van TOPAAS geldt dat er minimaal een gedefinieerd kwaliteitsproces gehanteerd moet worden. Indien er geen basale kwaliteitsborging in het proces aanwezig is die passend is bij softwareontwikkeling, dan is TOPAAS niet toepasbaar en mag de gehele vragenlijst niet gehanteerd worden.

Opgemerkt moet worden dat de IEC 61508 een bepaalde status heeft en geen vrijblijvend karakter kent. Het is een door de industrie zelf opgestelde *common practice* beschrijving van hoe leveranciers geacht worden te werken gegeven. Daarmee is het in feite een lat geworden waar juristen leveranciers tegenaan houden om te bepalen of ze verwijtbaar nalatig zijn. Het niet volgen daarvan is dan ook uiterst onverstandig omdat men zich als leverancier onttrekt aan de vastgelegde *common practice* van de industrie. Als men veiligheidskritische systemen bouwt is het dus verstandig deze norm, of zijn specifieke implementaties, te volgen.

Bij dit aspect geldt zeer expliciet dat er niet "rekenkundig geoptimaliseerd mag worden": vul de SIL-levels in die gerealiseerd zijn. Het volgen van een SIL-3 proces maar het administratief invullen van een SIL-2 proces in de TOPAAS-vragenlijst om rekenkundig beter uit te komen wordt gezien als het niet naar waarheid invullen van de lijst.

Eis: De vastlegging moet bestaan uit een verwijzing naar de rapportages van een onafhankelijke partij (een andere organisatie), waarbij de specifieke TUBs in de rapportage expliciet worden benoemd en onderzocht. Voor COTS TUBs moet men het certificaat opvragen, waarbij expliciet gesteld moet worden dat het certificaat op hetzelfde versienummer afgegeven moet zijn en alle "installatievoorwaarden" expliciet geadresseerd zijn.

4.2.2 Gebruik van inspecties

Inspecties op documenten en code hebben een zeer sterke invloed op de kwaliteit van een TUB. In praktijk zijn structurele reviews vaak (kosten)effectiever dan testen. Omdat in de IEC 61508 deze op SIL3 en SIL4 niveau al verplicht zijn, en Fagan inspecties aangeraden worden, worden deze op die SIL-niveaus anders gescoord.

2 Gebruik van Inspecties			
		Normaal	**SIL3/ SIL4**
1	Onbekend	0	NVT
2	Geen inspecties uitgevoerd	⅓	NVT
3	Aantoonbare inspecties/reviews op ontwerpen en code uitgevoerd	0	⅓
4	Aantoonbaar Fagan-inspecties uitgevoerd op alle ontwerpen, code en testdocumenten	-½	0

Aantoonbaarheid van de (Fagan)-inspecties blijkt uit meerdere onderdelen:

- De intentie tot het uitvoeren van inspecties/reviews moet blijken uit het kwaliteitsplan, projectplan of Software Development Plan (SDP), nader onderbouwd met een procesbeschrijving van het inspectie/review proces.
- De goede uitvoering moet blijken uit één van de volgende bewijsstukken voor elk te reviewen object:
  - Reviewformulieren/notulen van reviewsessies;
  - Ondertekening van documenten als reviewer;
  - Accordering voor vrijgave door een reviewer door middel van elektronische vastlegging, zoals een versiebeheer systeem, document management systeem of Wiki.

Voor een beschrijving van Fagan-inspecties wordt verwezen naar Bijlage D.

Eis: Hierbij wordt verwacht dat (een verwijzing naar) het kwaliteitsplan, projectplan of SDP worden vastgelegd. Ook de procesbeschrijving van het inspectie/review proces zijn onderdeel van de vastlegging. Andere objecten zijn toegankelijk voor steekproeven.

4.2.3 *Hoeveelheid wijzigingen van de TUB*

Een ontwerp dat regelmatig aan grootschalige wijzigingen blootgesteld wordt, heeft een grotere kans om fouten te bevatten. Dit komt omdat wijzigingen niet geheel doordacht of onvolledig worden doorgevoerd. De kans neemt toe dat wijzigingen in ontwerpdocumenten wel worden doorgevoerd, maar in onderliggende documenten niet (geheel) correct worden doorgevoerd.

3 Hoeveelheid wijzigingen ten opzichte originele ontwerp/eisenpakket van de TUB		
1	Onbekend	0
2	Zeer frequente of enkele fundamentele wijzigingen	⅔
3	Weinig wijzigingen, met zeer geringe impact	0
4	Geen wijzigingen	-⅓

De kern van dit aspect is de gezamenlijke omvang van de wijzigingen en de impact die de wijzigingen hebben op alle daaruit voortvloeiende ontwerpdocumenten. Het doel van dit aspect is om de dynamiek van significante wijzigingen vast te stellen. Onder fundamentele wijzigingen worden verstaan:

- Architectuurwijzigingen, zoals het verschuiven van verantwoordelijkheden van de ene module naar de andere;
- Verandering van basisaannames in het ontwerp, zoals het inherent veilig zijn van deelsystemen, etc.;
- Verandering van basisconcepten in het ontwerp, zoals de veiligheidsfilosofie;
- Veranderingen van randvoorwaarden waaronder het systeem moet opereren.

De indicatie *'zeer frequente of enkele fundamentele wijzigingen'* betekent dat er meer wijzigingen worden doorgevoerd dan waar de organisatie op ingericht is (de wijzigingencadans). Hiervoor moeten de volgende drie onderwerpen beschouwd worden:

- De procesinrichting: de *"Agile"* aanpak kan veel beter gecontroleerd een grote hoeveelheid changes doorvoeren dan een *"Waterval"* procesinrichting;

- De wendbaarheid van de organisatie: een klein hecht team kan eenvoudiger omgaan met wijzigingen dan een groot team;
- De effectiviteit van het changemanagementproces en de daarmee samenhangende controles. In een organisatie waar zeer sterke controles staan op inhoudelijke werkpakketten is het makkelijker aanpassingen goed door te voeren dan een organisatie waar controles afwezig zijn.

Op basis van bovenstaande punten kan de normale wijzigingencadans bepaald worden. Deze varieert typisch van één keer per jaar tot één keer per twee weken.

Opgemerkt wordt dat bij het volledig opnieuw beginnen met het ontwerp en de daaruit volgende detailontwerpen, etc. deze wijzigingencadans opnieuw start. Als fundamentele uitgangspunten en concepten wijzigen en het ontwikkelteam besluit dat het ontwerp van het product volledig overgedaan moet worden, dan wordt dit door TOPAAS gezien als een nieuw ontwerp zonder wijzigingen.

De hoeveelheid wijzigingen kan aantoonbaar gemaakt worden op verschillende manieren:

- Door de impact assessments van de change requests te categoriseren;
- Door registraties uit een eventueel aanwezig change management proces;
- Door een verschil te maken tussen de eerste versie van een ontwerp/implementatie en de uiteindelijke versie.

Eis: In de rapportage moet de keuze onderbouwd worden door een telling van de wijzigingen. De wijzigingen moeten wel inzichtelijk zijn voor eventuele inspectie.

4.2.4 Cultuur en samenwerking

Cultuur is een niet te onderschatten onderdeel van samenwerken en de effectiviteit van die samenwerking. Met name de effectiviteit van de samenwerking heeft veel invloed op de faalkans van een TUB. Voor het scoren van dit aspect is de cultuur in het specifieke ontwikkelprojectorganisatie van de software van belang.

4 Cultuur en samenwerking		
1	Onbekend	0
2	Op regels gebaseerde organisatie	⅓
3	Doelgerichte organisatie	0
4	Zelflerende organisatie	-½

Deze typering is gebaseerd op de definitie van cultuur en samenwerking die in de IAEA-TECDOC-1329 [5] als volgt is gedefinieerd:

Type organisatie Kenmerken	**Op regels gebaseerde organisatie**	**Doelgerichte organisatie**	**Zelflerende organisatie**
Kijk op fouten	Verwijten aan personeel i.p.v. luisterend en lerend	Fouten leiden tot meer controles en training	Fouten ziet men als gelegenheden om te leren en te verbeteren

Type organisatie Kenmerken	Op regels gebaseerde organisatie	Doelgerichte organisatie	Zelflerende organisatie
Tijd focus	Korte termijn is het allerbelangrijkste	Mensen worden beloond voor het overtreffen van doelen, los van de gevolgen op langere termijn	Korte termijn performance wordt geanalyseerd om langere termijn te verbeteren
Rol van managers	Managers stellen regels vast en pressen werknemers de gestelde doelen te bereiken	Managers gebruiken technieken zoals "Management by Objectives"	Managers coachen mensen ter verbetering van hun performance en ondersteunen samenwerking
Omgaan met conflicten	Conflicten worden zelden opgelost en groepscompetitie blijft aanwezig	Conflicten worden ontmoedigd in naam van het teamwork	Conflicten worden opgelost door oplossingen die voor beide partijen acceptabel zijn
Kijk op mensen	Mensen zijn componenten in een systeem	Besef dat het gedrag van mensen invloed heeft op hun prestaties	Mensen worden gerespecteerd en gewaardeerd voor hun bijdrage

De harde aantoonbaarheid hiervan is een lastig onderwerp. Men moet hier vertrouwen op een self-assessment van de organisatie, eventueel aangevuld met anekdotisch bewijs.

Eis: De rapportage dient van bovenstaand framework gebruik te maken in de onderbouwing.

4.2.5 *Opleidingsniveau en ervaring ontwikkelteam*

Ervaring met het domein is een belangrijke voorwaarde voor het doorgronden van de eisen en het voorkomen van veelgemaakte fouten in een specifiek domein.

5 Opleidingsniveau en ervaring ontwikkelteam		
1	Onbekend	0
2	Geen kennis met systeemontwikkeling voor het specifieke domein (onbewust onbekwaam)	1
3	Te weinig kennis met systeemontwikkeling voor het specifieke domein (bewust onbekwaam)	½
4	Gewenste kennis met systeemontwikkeling voor het specifieke domein (bewust bekwaam)	0
5	Uitstekende kennis en veel ervaring met systeemontwikkeling voor het specifieke domein (onbewust bekwaam)	-½

De gewenste ervaring richt zich voornamelijk op kennis met het specifieke domein van mensen die betrokken zijn bij de eisenanalyse, de architectuur, het ontwerp en de testplannen. Met domein wordt expliciet een objectdomein bedoeld volgens [6] en [7] of een specifiek domein zoals een systeem voor een stormvloedkering, sluis, beweegbare brug of tunnel. De volgende eisen worden aan de kennis gesteld:

- De domeinkennis moet actueel zijn: als projecten langer dan vijf jaar oud zijn is de kennis over het domein wellicht te ver weggezakt of zelfs achterhaald;
- De domeinkennis moet op het juiste abstractieniveau zijn: iemand die programmeerwerk voor een sluis uitgevoerd heeft is niet direct geschikt voor een eisenanalyse van een sluis;
- De systeemontwikkelkennis moet passen bij de aard van het systeem: dit betreft benodigde kennis van formele methoden, testmethodieken, testcoverage en documentatiestandaarden;
- De kennis moet ook daadwerkelijk worden ingezet voor uitvoering (eisenanalyse, ontwerpen, testen, etc.) of voor kwaliteitsborging daarvan bijvoorbeeld door middel van reviews op die deliverables.

Bij uitstekende kennis en veel ervaring moet men denken aan eisenanalisten, ontwerpers of testers die jaren aantoonbaar aan verschillende projecten in het domein hebben gewerkt op een vergelijkbaar abstractieniveau.

De kennis en ervaring kan eenvoudig aantoonbaar gemaakt worden door de specialisten met domeinkennis te identificeren en het volgende vast te leggen:

- De CV van de medewerker(s) waaruit de domeinkennis blijkt;
- De documenten waar de medewerker(s) voor verantwoordelijk zijn geweest, zowel in een creërende rol als in een reviewende rol.

Eis: Voor een score van keuze 4 of 5 is een expliciete vastlegging van de benodigde kennis vereist.

4.2.6 Samenwerking met de opdrachtgever

Het doel van dit aspect is het voorkomen van suboptimale oplossingen doordat:

- een opdrachtgever niet snapt wat hij moet bestellen bij de softwareontwikkelaar;
- er slechte ontwerpbeslissingen genomen worden doordat de belangen van softwareontwikkeling als ondergeschikt wordt beschouwd aan andere disciplines.

De opdrachtgever is bij complexere projecten, zoals bouwcombinaties, ook vaak een complex geheel van actoren. Vanuit TOPAAS wordt er gedacht aan de directe opdrachtgever van het softwareontwikkelteam, wat in een combinatie dus de combinatie zelf is, of een hoofdaannemer als er alleen software ontwikkeld wordt.

6 Samenwerking met opdrachtgever		
1	Onbekend.	0
2	Niet nauw betrokken opdrachtgever met weinig IT kennis, sterk contractueel/financieel gedreven opdrachtgever.	½
3	Zijdelings betrokken opdrachtgever met matige IT kennis.	0
4	Sterk betrokken opdrachtgever met voldoende kennis en open dialoog waarbij de opdrachtgever bereid is om overall architectuur te wijzigen als dat de software betrouwbaarheid ten goede komt. Er is sprake van een systems engineering aanpak voor de gehele ontwikkeling van het	-½

	systeem.	

Er zijn noodzakelijke randvoorwaarden bij grotere objecten (zoals bruggen, sluizen en tunnels) die een echte systems engineering aanpak mogelijk maken:

- Tijdens globale ontwerpen van de installatie zijn ontwerpers met IT-kennis betrokken om te garanderen dat er geen onbeheersbaar fysiek ontwerp ontstaat;
- In de trade-off analyses voor de te besturen hardware zijn software-engineers betrokken;
- Ontwerpen/aanschaf van hardware wordt pas afgerond als de ontwerpen van software ook klaar zijn en aantonen dat de hardware ook bestuurbaar is;
- Changes worden multidisciplinair beschouwd en niet monodisciplinair afgehandeld;
- Problemen in producten/ontwerpen worden multidisciplinair beschouwd.
- De aantoonbaarheid van dit punt bestaat uit:
- CV's van medewerkers van de opdrachtgever, die betrokken zijn bij het globale ontwerp;
- De aanwezigheid van een expliciete systems engineering aanpak, bijvoorbeeld door de vastlegging van de trade-off analyses;
- De multidisciplinaire afhandeling van changes en problemen in het ontwerp, bijvoorbeeld conform de leidraad SE.

Eis: De vastlegging bestaat uit de rol van de ontwerpers en opdrachtgever met hun CV en enkele voorbeeld-changes/problems indien aanwezig.

4.2.7 Complexiteit beslissingslogica van de TUB

Complexe software is zeer moeilijk betrouwbaar te krijgen. Complexiteit heeft drie belangrijke effecten die de betrouwbaarheid van de TUB nadelig beïnvloeden:

- Complexe software bevat veel paden. Door gebrek aan overzicht en inzicht van de exacte werking van de TUB kunnen eenvoudig fouten gemaakt worden;
- Complexe software is moeilijk te overzien, waardoor modificatie en bug fixing veel vaker resulteert in nadelige neveneffecten;
- Complexe software is zeer moeilijk testbaar, waardoor de effectiviteit van testen significant afneemt.

7 Complexiteit beslislogica van de specifieke TUB		
1	Onbekend.	0
2	Beslislogica is zeer complex (bevat veel vertakkingen en uitzonderingen), McCabe index groter dan 60.	½
3	Beslislogica matig complex (bevat enkele uitzonderingssituaties), McCabe index tussen 30 en 60.	0
4	Beslislogica is redelijk eenvoudig (bevat enkele zeer geïsoleerde uitzonderingssituaties), McCabe index tussen 10 en 30.	-⅓
5	Beslislogica en foutherkenning zijn erg eenvoudig, McCabe Index kleiner dan 10.	-½

Formeel is de insteek dat het zuiver gaat om de TUB. Zoals gemeld bij de kwalitatieve analyse betekent dit dat er onderscheid gemaakt moet worden tussen de vaak eenvoudige happy flow en het vaak complexere afhandelen van fouten. Het zijn immers verschillende delen van de broncode. Hierbij ligt deze scheiding vaak

midden door de broncode van een module heen, waardoor het simpelweg laten berekenen door tools niet werkt. Voor eenvoudige modules maakt dit niet uit: als de McCabe index voor de module al kleiner is dan 10, dan is die dat ook voor de specifieke TUB.

Bij grotere modules, waar meerdere TUBs samenkomen, ligt dit lastiger. Met name het handmatig tellen van een grote hoeveelheid paden is zeer lastig. De meest pragmatische benadering is dan de McCabe index van de eenvoudig te bepalen TUBs aftrekken van de McCabe index van de complete module (mits deze volledig disjunct zijn). Ook kan men op deze wijze niet-kritische software uit de analyse halen. Deze benadering is redelijk eenvoudig en zelfs gedeeltelijk geautomatiseerd uit te voeren.

Opgemerkt wordt dat complexiteit niet altijd vermijdbaar is. Domeincomplexiteit (de complexiteit van het probleem dat opgelost moet worden) kan zich soms vrij direct vertalen in een complexe oplossing. In dergelijke situaties zou het opvallend zijn dat de softwarematige modelering van een zeer complex probleem een zeer eenvoudige oplossing tot gevolg heeft.

Een complexe oplossing voor een probleem met weinig domeincomplexiteit kan wel eens een teken zijn van een slecht ontwerp. Nu kan de implementatie van *defensive programming* en fail-safe gedrag wel wat negatieve effecten op de complexiteit hebben, maar het is wel een slecht teken als de oplossing veel complexer is dan het domein. In de regel komt dit voort uit een ontwerp wat niet voldoet aan het principe van *"maximale cohesie en minimale koppeling"*.

Nu kan het zijn dat de broncode van de software zelf niet beschikbaar is. Men kan dan overwegen om het aantal paden door de software te tellen in de beslissingslogica, als benadering. Dit kan door naar het extern observeerbaar gedrag van de TUB te kijken (of een beschikbare specificatie daarvan) en op basis daarvan een inschatting maken.

Alhoewel er verschillende formele wiskundige definities zijn van de McCabe index, is de meest eenvoudige definitie:

McCabe Index = 1 + D

Waarbij D = het aantal beslispunten (if statements, while statements, switch/case statements) is.

Deze definitie kan zowel toegepast worden op broncode, als op beschreven beslislogica van een TUB. Hierbij wordt opgemerkt dat structurele monitoring tijdens ontwerp en bouw van de complexiteit van software redelijk eenvoudig in te richten is door middel van tooling.

Eis: De vastlegging van de complexiteit bestaat uit drie delen:
- Een (door een tool gegenereerd) overzicht van de complexiteit van alle modules;
- Een overzicht van de complexiteit van elke onderkende TUB, onderverdeeld naar de (delen van) software modules waar hij uit bestaat;
- Een verklaring van de verdeling van de complexiteit over de TUBs, als een module onderdeel is van meerdere TUBs. Hierbij moeten ook onderdelen van de code, die volledig buiten beschouwing gelaten worden, verklaard worden.

4.2.8 Omvang TUB

De omvang van de TUB is bepalend voor de faalkans. Statistisch neemt de kans op fouten toe als de code groter wordt en de overzichtelijkheid en begrijpelijkheid neemt af waardoor sneller fouten ontstaan.

8 Omvang TUB (Lines of code)		
1	Onbekend	0
2	Meer dan 50.000	½
3	Tussen 10.000 en 50.000	⅓
4	Tussen 5.000 en 10.000	0
5	Tussen 1.000 en 5.000	-⅓
6	Minder dan 1000	-½

Als software speciaal voor een doel gemaakt wordt, is het redelijk eenvoudig om de regels te tellen die bij het realiseren van een TUB betrokken zijn. Bij ontwerp of bij COTS is het vaak onmogelijk om het aantal regels code vast te stellen. Dit is vaak wel mogelijk met behulp van functiepuntenanalyse gecombineerd met de kengetallen van de IFPUG (International functie punt Users Group).

Programmeertaal	**Gemiddeld aantal regels code per functiepunt**
Basic Assembly	320
Macro Assembly	213
C	128
C++	53
FORTRAN	107
Pascal	91
PL/I	80
Ada83	71
Ada95	49
Visual Basic	32

Indien met behulp van een grafische programmeeromgeving wordt gewerkt, wordt één grafisch element (een blokje of een verbindende lijn) als één line of code opgevat. Indien er vanuit een grafische programmeeromgeving code wordt gegenereerd door een betrouwbare generator (bij voorkeur gecertificeerd voor IEC 61508, ISO 26262 of DO-178B/C), dan is de grafische omgeving bepalend voor de scoring van dit item en niet de gegenereerde code. Voorbeelden van gecertificeerde grafische omgevingen die code kunnen genereren zijn Scade en SimuLink.

De vastlegging van de omvang van TUBs bestaat uit vier delen:

- Een (door een tool gegenereerd) overzicht van de omvang van alle modules in aantallen regels code;
- Een overzicht van de omvang van elke onderkende TUB in aantallen regels code, onderverdeeld naar de individuele (delen van) software modules waar hij uit bestaat;
- Een verklaring van de verdeling van de omvang over de TUB, als een module onderdeel is van meerdere TUBs. Hierbij moeten ook onderdelen van de code die volledig buiten beschouwing gelaten worden, verklaard worden;

- In het geval van code generatie: een onderbouwing van de betrouwbaarheid van de code generator, bij voorkeur door middel van certificaten.

4.2.9 Helderheid van gebruikte architectuurconcepten

Een goede architectuur van de totale oplossing is een belangrijke voorwaarde om software goed begrijpbaar, bouwbaar, testbaar en onderhoudbaar te krijgen. Het andere uiterste is slecht georganiseerde code, ook wel spaghetticode genaamd, waar geen heldere afbakening van verantwoordelijkheden benoemd zijn. Hierdoor is het resultaat moeilijk te begrijpen en zeer moeilijk te testen.

9 Helderheid gebruikte architectuurconcepten		
1	Onbekend.	0
2	Geen heldere afbakening taken en verantwoordelijkheden modulen in ontwerp benoemd.	½
3	Wel taken en verantwoordelijkheden op hoofdlijnen benoemd, maar geen navolging gegeven in ontwikkeling.	⅓
4	Er is een scheiding van taken en verantwoordelijkheden tussen modulen beschreven, welke het principe van "maximale cohesie en minimale koppeling respecteert", maar deze is passief bewaakt tijdens het ontwikkelproces.	0
5	Er is een scherpe scheiding van taken en verantwoordelijkheden tussen modulen beschreven op basis van geldende documenten, welke het principe van "maximale cohesie en minimale koppeling respecteert", en deze is aantoonbaar actief bewaakt tijdens het ontwikkelproces.	-½

Maximale cohesie en minimale koppeling

Een cruciaal onderdeel in de vraagstelling is het expliciet vastleggen van de architectuur en de inhoudelijke eis van *"maximale cohesie en minimale koppeling"*. *"Maximale cohesie en minimale koppeling"* valt onder *seperation of concerns*. Dit ontwerpprincipe vereist dat elke softwaremodule een intern sterk samenhangend geheel vormt met een scherp afgebakende functie ten opzichte van de rest. De modules communiceren onderling maar zeer spaarzaam en via een afgesproken interface.

Passief en actief bewaken

Een ander punt is de bewaking van de architectuur. Passief bewaken wil zeggen dat er een architectuur beschreven en gebruikt is als basis voor ontwerpen, maar dat er geen expliciete controles plaatsvinden of modules ook daadwerkelijk volgens die architectuur gebouwd zijn. Actieve bewaking wil zeggen dat ontwerp- en testdocumenten expliciet gerelateerd zijn aan de architectuur en dat er ook reviews en inspecties plaatsvinden op deze documenten of deze de architectuur ook daadwerkelijk waarmaken. Ook een aantoonbaar actieve begeleiding van het ontwikkelteam door de architect valt hieronder.

Aantoonbaar moet zijn:

- De beschrijving van de architectuur, inclusief de onderliggende beslissingen en overwegingen (rationale) waarom bepaalde architectuurkeuzes gemaakt zijn (noodzakelijk voor keuze 3, 4 en 5);
- De expliciete referentie van de architectuur in de onderliggende ontwerpdocumenten (noodzakelijk voor keuze 4 en 5);
- De controle van de correcte invulling van de architectuur in die onderliggende ontwerpdocumenten, door bijvoorbeeld dit expliciet

onderdeel te laten zijn van reviews en inspecties of door het een aparte activiteit te laten zijn (noodzakelijk voor keuze 5).

In de vastlegging moet normaal gesproken de architectuur al grotendeels opgenomen zijn (onderdeel van de identificatie van TUBs). Om aan te tonen dat de architectuur daadwerkelijk wordt toegepast, is het voldoende om naar de ontwerpdocumenten te refereren in de onderbouwing van de scoring. Het vastleggen van de bewaking is afhankelijk van de vorm van bewaking:

- Als de bewaking verankerd is in het reviewproces, dan is het voldoende om aan te tonen dat de architectuurfunctie structureel betrokken is bij de reviews;
- Als de bewaking separaat is, moeten de daaruit voortkomende registraties als bewijsvoering beschikbaar zijn;
- Als de architect een actief bewakende rol in het bouwproces vervult, dan moet dat blijken uit gespreksverslagen of ontwerpdocumentatie.

4.2.10 Gebruik van een certified compiler

Compilers hebben een belangrijke invloed op de kwaliteit van software. De meest voor de hand liggende eigenschap van een compiler is dat deze een betrouwbare manier moet zijn om programmacode om te zetten in executeerbare machinecode. Maar compilers hebben ook kwalitatieve eigenschappen, zoals expliciete beperkingen in de constructies gebruikt in programmacode.

10 Gebruik van een certified compiler			
		Normaal	**SIL3/ SIL4**
1	Onbekend.	0	NVT
2	Gebruik van een willekeurige compiler.	⅓	NVT
3	Gebruik van een compiler waar de ontwikkelaar langdurige ervaring mee heeft.	0	⅓
4	Gebruik van een certified compiler in combinatie met een gevalideerde safe subset.	-½	0
5	Gebruik van een certified compiler in combinatie met een gevalideerde safe subset met bijbehorende kalibratiesets en testprotocol om compiler te ijken/testen, wat voor elke nieuwe versie van de compiler structureel gebeurt.	-⅔	-⅓

De safe subset is een subset van de programmeertaal en wordt deels bepaald door:

- de programmeertaal, waar unsafe features van de taal worden uitgesloten;
- de specifieke (versie van de) compiler, doordat in de scopebepaling van de certificering van het specifieke merk, type én versie van de compiler is vastgelegd welke constructies van de broncode op gecertificeerde betrouwbare wijze omgezet worden in executeerbare programmacode.

De safe subset is dus zowel taal- als compilerafhankelijk en moet per geïnstalleerde (versie van de) compiler expliciet worden vastgesteld.

De kalibratietesten en de bijbehorende testprotocollen zijn mechanismes om te controleren of een compiler ook daadwerkelijk goed geconfigureerd is en goed wordt toegepast in zijn werkomgeving. In praktijk moeten bepaalde programmacode-optimalisaties uit staan. De kalibratieset van de compilerfabrikant controleert dit

door een stuk broncode te compileren wat een vooraf vastgestelde output moet genereren.

Er zijn maar weinig certified compilers. Dit wordt ook door de IEC 61508 erkend door het toestaan van compilers met *"increased confidence from use"* als optie voor SIL3/SIL4 systemen. Bij ontwikkeling op SIL 3/4 wordt, vanuit de IEC 61508, een certified compiler wel als optie beschreven. De IEC 61508 deel 7 schrijft specifiek over het gecertificeerd zijn:

"The certification of a tool will generally be carried out by an independent, often national, body, against independently set criteria, typically national or international standards. (…). To date, only compilers (translators) are regularly subject to certification procedures; these are laid down by national certification bodies and the exercise compilers (translators) against international standards such as those for Ada and Pascal".

Opgemerkt wordt dat met name TÜV wel actief certificering uitvoert op compilers en er ook C en C++ compilers gecertificeerd zijn.

Veel PLC-leveranciers kennen een SIL-3 of Safety-mode, waarin er alleen bepaalde constructs gehanteerd mogen worden. Vaak zijn deze ook door TÜV gecertificeerd. Dit wordt beschouwd als een gecertificeerde compiler.

Eis: Aangetoond moet worden:
- Aantoonbaar gebruik van de compiler in andere projecten (keuze 3);
- De certificeringsgegevens van de compiler samen met de bijbehorende subset, waarbij de safe subset expliciet vastgelegd moet zijn in *coding guidelines* voor het project (keuze 4 en 5);
- IJkrapporten van de compiler (keuze 5).

De referenties, certificeringsgegevens, subset, coding guidelines en ijkrapporten moeten als onderliggende onderbouwing worden opgenomen in de RAMS-rapportage.

4.2.11 *Traceerbaarheid van eisen*

De traceerbaarheid van eisen garandeert dat eisen ook expliciet geïmplementeerd en getest worden. Hiermee wordt aangetoond dat een TUB doet wat hij moet doen.

11 Traceerbaarheid van eisen door het proces heen			
		Normaal	**SIL3/ SIL4**
1	Onbekend	0	NVT
2	Geen traceerbaarheid	⅓	NVT
3	Aantoonbaar traceerbaar naar testscripts	0	NVT
4	Aantoonbaar traceerbaar naar architectuur en testen	-⅓	⅓
5	Aantoonbaar traceerbaar van veiligheidskritische eisen tot aan de code en individuele testen toe	-⅔	0
6	Aantoonbare volledige traceerbaarheid	-1	-⅓
7	Aantoonbaar wiskundig/logisch bewezen correcte traceerbaarheid	-2	-½

De eenvoudigste manier om traceerbaarheid te bewerkstelligen is door in het softwareontwikkelproces stapsgewijs de gebruikerseisen te vertalen naar testscripts en technische systeemeisen. Vervolgens worden softwaremodules aangewezen die

de systeemeisen waar gaan maken. Op deze manier worden de gebruikerseisen traceerbaar tot aan de code toe. In het testproces bestaan ook standaardtechnieken genaamd "Verification Cross Reference Index". Hierbij wordt elke gebruikerseis expliciet gekoppeld aan testscripts.

Wiskundig-correct-bewezen-traceerbaarheid (keuze 7) houdt in dat er model checking toegepast is of een gedragsequivalentie aangetoond is tussen de implementatie en een correcte (gevalideerde) specificatie. Dit is voorlopig alleen voor kleinschalige systemen te realiseren. Als de links tussen alle gebruikerseisen, tests, onderliggende systeemeisen en uitvoerende code duidelijk zijn dan is de traceerbaarheid in de softwaremodule volledig (keuze 6). Dit geldt misschien alleen voor veiligheidskritische eisen (keuze 5) of alleen gedeeltelijk (keuze 4) of tot testen (keuze 3).

Eis: De keuze met onderbouwing moet worden vastgelegd in de RAMS-analyse. Onderdeel van de onderbouwing moet zijn het beschrijven van de werkwijze van het traceerbaar houden van eisen.

4.2.12 Testtechnieken en dekkingsgraad

Testen hebben een grote invloed op het uiteindelijke vertrouwen in de TUB. Met name de diepgang en dekkingsgraad van de testen zijn van belang. Opgemerkt wordt dat het daarbij expliciet gaat om de dekkingsgraad in relatie met de taakuitvoering van de TUB.

12 Testtechnieken en dekkingsgraad van de TUB			
		Normaal	**SIL3/ SIL4**
1	Onbekend	0	NVT
2	Geen gedocumenteerde testen uitgevoerd	0	NVT
3	Wel testen gedocumenteerd, maar geen formele testtechnieken gehanteerd. Dekkingsgraad onbekend	-⅓	NVT
4	Formele testtechniek(en) gehanteerd met lage dekkingsgraad	-½	⅔
5	Formele testtechniek(en) gehanteerd met gemiddelde dekkingsgraad	-⅔	½
6	Formele testtechniek(en) gehanteerd met hoge dekkingsgraad	-1	0
7	Formele testtechniek(en) gehanteerd met aantoonbare (gemeten) hoge dekkingsgraad	-1⅓	-⅓

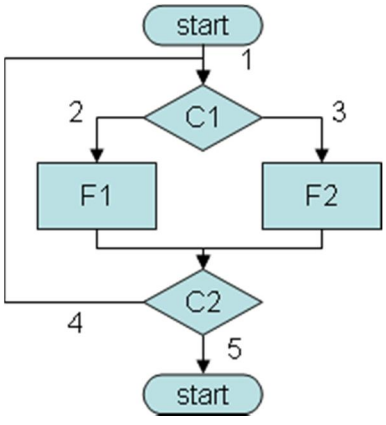
Bij het formeel testen worden er technieken toegepast uit de wiskunde en logica. Het startpunt is een geverifieerd model van het softwareontwerp of de TUB zelf als die geannoteerd is met formele specificaties. Daaruit worden automatisch testen gegenereerd die een gewenste eis testen met een gecontroleerde dekkingsgraad. Het formele testproces maakt het verband tussen eis en test case heel duidelijk en is daardoor van grote waarde voor het aantonen van softwarebetrouwbaarheid. Een paar tools die model-based testgeneratie ondersteunen zijn T-VEC, Conformiq, Reactis, Unitesk. Een academische tool dat intensief werd gebruikt in de telecommunicatiewereld is TGV met TorX als opvolger.

De dekkingsgraad van testgevallen kan op verschillende manieren worden uitgedrukt:

- afhankelijk van gebruikte specificatietechniek:
    - beslissingstabellen op basis van statement-, condition- of multiple condition coverage
    - procescycli (aan de hand van beslispunten) op basis van 'testmaat 1', '- 2' of '-n' (zie toelichting bij de testmaat)
    - combinatie van equivalentieklassen op basis van pairwise, triplewise, etc.;
- als percentage van doorlopen paden tijdens testtraject ten opzichte van theoretisch mogelijke paden met behulp van tools zoals McCabe. Dit vereist dus inzet van tools tijdens testtraject en beschikbaarheid van source code daarbij. De dekkingsgraad is in dit geval:
    - hoog: 90% van de mogelijk door te lopen paden
    - gemiddeld: 50% van de mogelijk door te lopen paden
    - laag: 10% de mogelijk door te lopen paden
- als percentage van het aantal doorlopen combinaties van inputvariabelen ten opzichte van theoretisch mogelijke combinaties van inputvariabelen. De dekkingsgraad is in dit geval:
    - hoog: 90% van de theoretisch mogelijke input
    - gemiddeld: 50% van de theoretisch mogelijke input
    - laag: 10% de theoretisch mogelijke input.

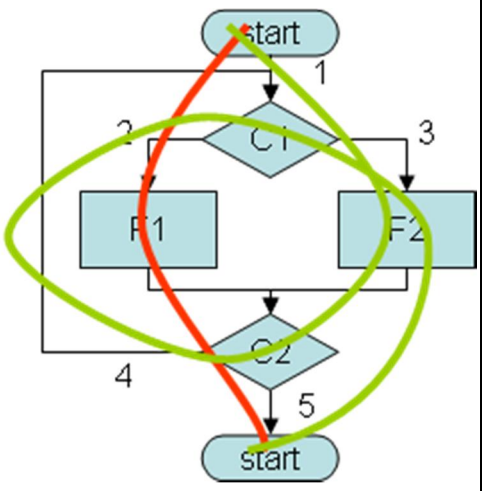
Toelichting bij de testmaat
De testmaat gaat uit van het principe dat men bij beslispunten in de beslissingslogica een aantal uitgangen heeft. De combinatie van verschillende in- en uitgangen bij opeenvolgende beslispunten beschrijven dus verschillende 'functionele paden' in een programma. Hoe hoger de testmaat, hoe groter het aantal combinaties en hoe hoger de dekkingsgraad. Bovendien is het toepassen van een testmaat een meetbare verantwoording over de onderkende testgevallen.

Om dit te illustreren staat hieronder een eenvoudige TUB met voorbeelden van onderkende testscenario's volgens testmaat 1 en testmaat 2. Uitgangspunt van het voorbeeld is:



Hierbij gelden de volgende voorwaarden:
- De uitgangen van de condities (C1 en C2) zijn onafhankelijk van elkaar.
- De uitgevoerde bewerkingen (F1 en F2) kunnen herhaald worden.
- Invoer voor de TUB bepaalt de afhandeling in C1 en C2;
- De uitvoer van bewerkingen in F1 en F2 is meetbaar. Hiermee wordt bedoeld dat het gedrag van de TUB observeerbaar is.

Testmaat 1:



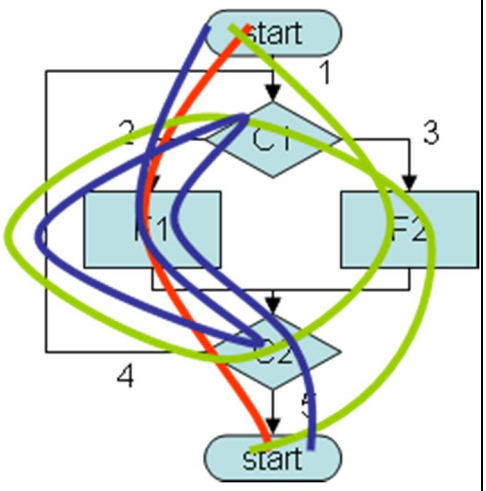
Elke combinatie van uitgangen van één (1) conditie doorlopen.
Combinaties:

1; 2; 3; 4; 5

Feitelijk zijn dit geen combinaties maar individuele uitgangen. Testmaat 1 is de meest simpele vorm. Twee scenario's volstaan, bijvoorbeeld:

A: 1, 2, 5
en
B: 1, 3, 4, 3, 5

Testmaat 2:



Elke combinatie van uitgangen van twee (2) condities doorlopen.
Combinaties:

1-2; 1-3; 4-2; 4-3; 2-4; 2-5; 3-4; 3-5

De scenario's uit het voorbeeld van Testmaat 1 alleen volstaan dan niet:

A: 1, 2, 5 (covers 1-2 en 2-5)
B: 1, 3, 4, 3, 5(covers 1-3, 3-4, 4-3 en 3-5)

Dan ontbreken nog combinaties 4-2 en 2-4

Additioneel benodigd scenario, bijvoorbeeld:

C: 1, 2, 4, 2, 5

Testmaat 2 leidt in het algemeen tot meer testgevallen dan testmaat 1. Daarbij wordt aangetekend dat er enerzijds dubbelingen kunnen voorkomen maar dat er anderzijds ook nog padcombinaties zijn die niet getest worden:

- 1, 3, 4, 2, 5
- 1, 2, 4, 3, 5 of
- 1, 2, 4, 3, 4, 2, 5.

Elke volgende testmaat (combinaties van 3, 4, 5, enz. opeenvolgende condities) geeft een nog hogere dekkingsgraad.

Op deze manier kan de dekkingsgraad geformaliseerd worden. Het vooraf stellen van een testmaat maakt het mogelijk achteraf vast te stellen in hoeverre aan de gestelde eisen is voldaan. Beoordeling van de testscenario's vindt plaats aan de

hand van het voldoen aan de uitvoervoorspelling, gekoppeld aan de TUB-invoer. Daarmee is het belang aangetoond van de relatie met de functionele specificaties bij het bepalen van dekkingsgraad. Specificaties zijn noodzakelijk om de benodigde paden te onderkennen en als basis voor de uitvoervoorspelling.

Met bovenstaande voorbeelden is aangetoond dat 'code coverage' geen inzicht geeft in de behaalde functionele dekkingsgraad.

Eis: Per TUB dient vastgelegd en aangetoond te worden:
- welke dekkingsgraad behaald is;
- hoe de dekkingsgraad aangetoond is.

Hierbij moeten ontwerpen, testplannen en testrapportages toegankelijk zijn voor steekproeven door de opdrachtgever.

4.2.13 *Multiprocesomgeving*

Multiprocesomgevingen leiden tot concurrerende processen voor dezelfde resources. Hierdoor kan met name tijdigheid, maar soms ook stabiliteit, veel minder sterk gegarandeerd worden.

13 Multiprocesomgeving		
1	Onbekend.	0
2	Meerdere TUBs in een gevirtualiseerde omgeving.	½
3	Meerdere TUBs die onafhankelijk van elkaar op één stuk hardware worden uitgevoerd.	⅓
4	Maximaal één TUB op een dedicated OS met een dedicated CPU.	0
5	Eén TUB op een dedicated CPU en memory op geen of een triviaal OS.	-⅓

Score 2 en 3 wordt verkregen als meerdere TUBs op één CPU actief zijn die niet op een natuurlijke manier in elkaars verlengde liggen. TUBs liggen in elkaars verlengde als ze:

- dezelfde hardware aansturen (dus de besturing van een slagboom is typisch onderdeel van 2 of meer TUBs: het sluiten van de slagboom en het openen daarvan);
- hardware aansturen en het falen van diezelfde hardware afhandelen (dus een klep aansturen en het reageren op het falen van diezelfde klep);

Bij score 4 is ook inbegrepen: Meerdere TUBS op een stuk hardware die welgedefinieerd sequentieel worden uitgevoerd met a priori beperkt resourcegebruik. Bij variërend resource gebruik van de TUB blijft score 3 gelden. Onder variërend resource gebruik valt al gezamenlijk gebruik van stack en heap ruimte.

Een triviaal OS is een besturingssysteem met een expliciete harde tijdsconstraint per module, bijvoorbeeld een op round robin scheduling gebaseerde real time operating systeem. Hiermee wordt het tijdsgedrag deterministisch, wat de betrouwbaarheid ten goede komt.

Eis: De specifieke eigenschappen, zoals de timeboxing eigenschappen van de OS scheduler, dienen in de architectuurbeschrijving aangetoond en vastgelegd te zijn. De architectuurbeschrijving is al onderdeel is van de kwalitatieve analyse.

4.2.14 Aanwezigheid van representatieve velddata van de taakuitvoering

Bij velddata gaat het om data van de toepassing van de TUB in andere oplossingen. Denk aan een ventilatiebesturing die wereldwijd honderden keren wordt toegepast in tunnels en nu wordt toegepast op een specifieke tunnel. Deze velddata laat zien dat de TUB een bewezen trackrecord heeft. Het kan zijn dat, bijvoorbeeld bij veiligheidssystemen, er ook in het veld weinig praktische ervaring is met het daadwerkelijk gebruik van de TUB. Maar wanneer er voldoende representatieve testen uitgevoerd zijn, bijvoorbeeld in Site Acceptance Testen van andere toepassingen, kunnen deze gegevens bijdragen aan dat inzicht. Het idee is dat een intensief toegepaste TUB ook diepgaand getest en gebruikt is, waardoor deze van nature betrouwbaarder zal zijn. Hier moet goed worden vastgesteld of de huidige toepassing ook overeenkomt met de toepassing die voor het verkrijgen van de velddata gebruikt is, omdat anders de betrouwbaarheid van het huidige gebruik niet aangetoond is.

14 Aanwezigheid representatieve velddata gedurende taakuitvoering			
		Normaal	**SIL3/ SIL4**
1	Onbekend.	0	NVT
2	Geen velddata of testdata aanwezig, zelfs niet uit (schaduw)draaien van de TUB.	⅓	⅓
3	Beperkte gegevens aanwezig en geanalyseerd tijdens uitvoeren van de taakuitvoering door de TUB.	0	0
4	Significante hoeveelheid gegevens aanwezig tijdens uitvoeren van de taakuitvoering door de TUB.	-1	-⅓
5	Veel representatieve velddata/testdata aanwezig van identieke of sterk vergelijkbare toepassingen met dezelfde TUB.	-2	-½

Test- en velddata zijn ervaringscijfers afkomstig van testtrajecten of productie van andere toepassingen van de TUB. Zij geven bijvoorbeeld inzicht in het aantal foutloze draaiuren of het aantal foutloos doorlopen testscenario's van een TUB. Om op grond van velddata uitspraken te kunnen doen over de betrouwbaarheid van de TUB, moeten er significante aantallen gegevens beschikbaar zijn (vanuit de leverancier). Daarnaast is het essentieel dat zij verkregen zijn in situaties die representatief zijn voor de manier waarop de TUB in de praktijk wordt gebruikt.

Als er gedurende een significante tijd in de reële productieomgeving is getest (schaduwdraaien) mag dit als velddata gezien worden. Van belang is hier de realistische productieomgeving. Het mag geen synthetische of gesimuleerde testdata bevatten, maar mag alleen in de daadwerkelijke eindsituatie beproefd zijn.

Significantie van de aantallen is weergegeven in de mogelijke antwoorden bij dit aspect:

- Bij keuze 2 wordt met *"geen data"* bedoeld dat de testdata geen realistische operationele scenario's afdekken en velddata niet onder keuze 3 vallen.

- Bij keuze 3 wordt met *"beperkte gegevens"* bedoeld dat de TUB erg weinig gebruikt wordt in vergelijkbare toepassingen, minder dan, ordegrootte, vijf keer per maand wereldwijd, of met betrekking tot testdata, waarbij het aantal bruikbare testresultaten gering is.
- Bij keuze 4 wordt met *"significante hoeveelheid gegevens"* bedoeld dat de TUB beperkt wordt gebruikt in vergelijkbare toepassingen, ten minste, ordegrootte, minimaal vijf keer per maand wereldwijd of met betrekking tot testdata, dat het aantal bruikbare testresultaten significant is.
- Bij keuze 5 wordt met *"veel representatieve velddata"* bedoeld dat de TUB veel wordt gebruikt in vergelijkbare toepassingen, ten minste, ordegrootte, vijf keer per dag wereldwijd of met betrekking tot testdata, dat het aantal bruikbare testresultaten vergelijkbaar groot is.

Het al of niet representatief zijn is afhankelijk van de volgende factoren:

- De relevante taakuitvoering moet daadwerkelijk regelmatig plaatsvinden om als velddata te worden aangemerkt. Een beveiligingssysteem dat op veel plaatsen is geïnstalleerd, maar nog nooit daadwerkelijk heeft ingegrepen kent dus geen representatieve velddata.
- Testresultaten moeten zijn verkregen met scenario's die vergelijkbaar zijn met productiesituaties en in aantal ook zodanig verdeeld zijn over de relevante taakuitvoering.
- De technische omgeving waarin de TUB operationeel of getest is, is representatief voor de omgeving van de beoordeelde TUB. Het gaat uitdrukkelijk om representatieve technische omgeving: de TUB moet sterk vergelijkbare objecten in de omgeving besturen om representatieve resultaten op te leveren.

Om dit aantoonbaar te maken is het volgende noodzakelijk:

- In het geval van testdata: er moet een gestructureerd testtraject hebben plaatsgevonden dat een adequate logging van testscenario's en resultaten heeft of,
- in het geval van velddata, moet de data verkregen zijn door een gestructureerd feedbackproces voor succesvolle en falende toepassing van de TUB.

Eis: De vastlegging dient aangeven:

- op welke onderliggende gegevens de keuze gebaseerd is;
- hoe de gegevenscollectie ingewonnen is door de leverancier van de TUB;
- de onderbouwing waarom de velddata/testdata representatief is voor de toepassingen van de TUB

4.2.15 Monitoring

Monitoring houdt de prestaties van de TUB in de gaten tijdens de taakuitvoering in de uiteindelijke productieomgeving. In een gemonitord systeem wordt goed gedrag vastgesteld worden fouten gedetecteerd en geregistreerd als ze optreden, terwijl in een niet gemonitord systeem fouten ongemerkt aanwezig kunnen blijven. Essentieel bij monitoring is dat de totale oplossing gemonitord wordt en dat de correcte specifieke taakuitvoering van de specifieke TUB gezien en geregistreerd wordt.

15 Monitoring van de TUB		
1	Onbekend	0

2	Geen aanwezig	⅓
3	Weinig/kort gemonitord gedurende taakuitvoering	0
4	Langdurige monitoring, maar niet frequente taakuitvoering	-⅓
5	Langdurige/frequente monitoring tijdens taakuitvoering	-½

De monitoring is specifiek op de betreffende TUB gericht. Het laat toe dat het vertrouwen dat men heeft in langer bestaande TUBs met een aantoonbaar trackrecord ook tot uiting komt in de TOPAAS-score. Men kan dan denken aan TUBs die al enkele jaren goed functioneren in hetzelfde onderhavige object (bijv. tunnel of brug). Met deze factor wordt ook zeer expliciet gedoeld op operationele toepassing van de TUB in de productieomgeving. Testgegevens en velddata vallen expliciet buiten de definitie van monitoring.

Een nadere toelichting bij de keuzes:

- Bij keuze 3 wordt met *"Weinig/kort gemonitord"* bedoeld dat de TUB beperkt wordt aangesproken, hooguit, ordegrootte, één keer per maand.
- Bij keuze 4 wordt met *"Langdurige monitoring, maar niet frequente taakuitvoering"* bedoeld dat de TUB ten minste, ordegrootte, één keer per maand maar hooguit één keer per jaar wordt aangesproken om zijn taakuitvoering uit te voeren.
- Bij keuze 5 wordt met *"Langdurige/frequente monitoring tijdens taakuitvoering"* bedoeld dat de TUB ten minste, ordegrootte, vijf keer per dag wordt gebruikt en ten minste, ordegrootte vijf keer per jaar, wordt aangesproken om zijn taakuitvoering uit te voeren.

Eis: Het volgende dient te worden vastgelegd:
- hoe de monitoring ingericht is;
- wat de aanspraakfrequentie van de TUB is;

4.3 Praktische tips

4.3.1 Omgang met grote groepen TUBs van dezelfde leverancier

Het is mogelijk om voor elke TUB een volledig nieuwe kwantitatieve analyse te doen. Het is echter praktisch om onderscheid te maken tussen vragen die over alle TUBs een identiek beeld zullen leveren en andere vragen die per TUB zullen verschillen.

De gegevens die waarschijnlijk voor alle TUBs gelijk zijn binnen hetzelfde project zijn:

- Aspect 1: Voldoen van het ontwikkelproces aan een van de SIL's van de IEC 61508.
- Aspect 2: Gebruik van inspecties.
- Aspect 4: Cultuur en samenwerking.
- Aspect 6: Samenwerking met opdrachtgever.
- Aspect 9: Helderheid gebruikte architectuurconcepten.
- Aspect 10: Gebruik van een certified compiler.
- Aspect 11: Traceerbaarheid van eisen door het proces heen.

De gegevens die waarschijnlijk per TUB verschillen zijn:

- Aspect 3: Hoeveelheid wijzigingen ten opzichte originele ontwerp/eisenpakket.

- Aspect 5: Opleidingsniveau en ervaring ontwikkelteam.
- Aspect 7: Complexiteit beslissingslogica.
- Aspect 8: Omvang TUB (Lines of code).
- Aspect 12: Testtechnieken en dekkingsgraad.
- Aspect 13: Multiprocesomgeving.
- Aspect 14: Aanwezigheid representatieve velddata gedurende taakuitvoering.
- Aspect 15: Monitoring.

4.3.2 Praktisch omgaan met veel verschillende TUBs

Enkele gegevens zijn specifiek voor een TUB. Maar TUBs zijn delen van één of meerdere modules. In de onderbouwing van de kwalitatieve analyse wordt de relatie gelegd tussen TUBs en broncode. Het is verstandig om de volgende informatie daar altijd in één groot overzicht te plaatsen (per module met een onderverdeling van de TUBs):

- Aspect 7: Complexiteit beslissingslogica
- Aspect 8: Omvang TUB (Lines of code)
- Aspect 12: Testtechnieken en dekkingsgraad

Dit zijn zeer aan elkaar gerelateerde gegevens die specifiek betrekking hebben op de broncode en de verdeling daarvan. Hierdoor wordt dit eenvoudiger te beheren en analyseren.

4.3.3 Omgaan met grote monolitische deelsystemen

In sommige ontwerpen ontstaan delen code waar het moeilijk tot onmogelijk is om de verschillende taakuitvoeringen te isoleren. Dit staat uiteraard op zeer gespannen voet met het architectuurprincipe "maximale cohesie en minimale koppeling", maar in sommige situaties is dit onvermijdelijk.

Vanuit de analyse levert zo'n monolitisch ontwerp een verzameling TUBs op die allemaal betrekking hebben op één stuk broncode. Vanuit de RAMS-analyse is het noodzakelijk om deze TUBs individueel te scoren, wat zich vertaalt in veel TUBs met identieke scoring en identieke onderbouwing.

Het is in die situatie wel verstandig om scherp te blijven op aspecten die per TUB nog steeds kunnen verschillen:

- Aspect 7: Complexiteit beslissingslogica (wellicht vast te stellen op basis van de vastgestelde beslissingsstructuur en niet op basis van de broncode).
- Aspect 12: Testtechnieken en dekkingsgraad.
- Aspect 14: Aanwezigheid representatieve velddata gedurende taakuitvoering.

Hierdoor kan de score per TUB nog steeds verschillen, ondanks dat dezelfde monolitische software deze realiseert.

4.3.4 Omgaan met kennis en kwaliteit

De onderstaande vier vragen zijn indirect aan elkaar gerelateerd:

- Aspect 2: Gebruik van inspecties.
- Aspect 5: Opleidingsniveau en ervaring ontwikkelteam.
- Aspect 9: Helderheid gebruikte architectuurconcepten.
- Aspect 11: Traceerbaarheid van requirements door het proces heen.

Aspect 2 gaat onder andere over de uitgevoerde peer reviews. Aspect 5 stelt mogelijk eisen aan de toepassing van domeinervaring in het toepassen van reviews. Aspect 9 stelt mogelijk eisen aan de bewaking van architectuurprincipes in het uitvoeren van reviews. En aspect 11 is mogelijke kwalitatief geborgd door middel van reviews.

Generiek onderliggend thema is dat als reviews/inspecties van documenten controleren op zowel de bouwbaarheid/plausibiliteit als op het voldoen aan de gestelde eisen uit bovenliggende documenten en het project dit laat uitvoeren door de juiste personen veel van de genoemde eisen relatief eenvoudig in te vullen zijn.

5 Rapportage

In de voorgaande hoofdstukken is veel gesproken over de vastlegging in een RAMS-dossier. Hierin worden de volgende zaken verwacht:

Eis: Een beschrijving van de architectuur van de oplossing (de output van hoofdstuk 3.6).

Eis: De identificatie van alle basisgebeurtenissen waar software een rol speelt (de output zoals beschreven in paragraaf 3.5), een verdeling van deze falende taakuitvoeringen in TUBs en software modules (de output van 3.7), samen met een rationele achter deze opdelingen.

Eis: De onderbouwing per TUB voor de faalkans (de output van hoofdstuk 4.1 en 4.2).

6 Referenties

[1] IEC61508 Functional safety of electrical/electronic/programmable electronic safety-related systems, CEI/IEC, 1998
[2] ISO/IEC/ IEEE 42010 Systems and software engineering — Architecture description, ISO / IEC / IEEE, 2011
[3] P. Kruchten, Architectural Blueprints—The "4+1" View Model of Software Architecture, IEEE Software 12 (6), November 1995, pp. 42-50
[4] E.T.H. Brandt en R. P. Henzen, Handboeken Betrouwbaarheidsanalyse, Refis Reliability Engineering, 2005
[5] IAEA, Safety Culture in Nuclear Installations: Guidance for Use in the Enhancement of Safety Culture, IAEA-TECDOC-1329, ISBN:92-0-119102-2, http://www-ub.iaea.org/MTCD/Publications/PDF/te_1329_web.pdf
[6] ANSI/ISA-88.*.*-2010 Batch Control
[7] ANSI/ISA-95.*.*-2010 (IEC 62264) Enterprise-Control System Integration)
[8] Handreiking prestatiegestuurde risicoanalyses (PRA), Rijkswaterstaat, Oktober 2016
[9] Handreiking Bayesiaanse update, Rijkswaterstaat, December 2016
[10] Rijkswaterstaat memo: DOC-0055 Continue systemen en de faalkans van software, 28 maart 2018, J. Horstman

7 Afkortingen en termen

Afkorting of term	**Betekenis**[2]
Beschikbaarheid	Deze heeft twee definities: • de verwachte fractie van de totale tijd dat een systeem, onder gegeven omstandigheden, functioneert; • de kans dat een systeem, onder gegeven omstandigheden, functioneert wanneer het op een willekeurig tijdstip wordt aangesproken.
Betrouwbaarheid	De kans dat een systeem zonder falen zijn functie vervult, gedurende een bepaalde periode en onder gegeven omstandigheden.
COTS	Commercial Of The Shelf. TUBs, zoals bijvoorbeeld turbinebeveiliging, die als compleet product worden ingekocht.
Falen	Een gebeurtenis of een verzameling gebeurtenissen, waardoor een systeem zijn functionaliteit of een deel van zijn functionaliteit verliest.
Foutenboom	Grafische weergave van de relatie tussen het falen vansysteemelementen en het falen het systeem, uitgedrukt in de *Ongewenste Topgebeurtenis* (OTG). Deze grafische weergave wordt veelal gefaciliteerd door programmatuur, die tevens de kans op of de niet-beschikbaarheid van de OTG berekent.
IFPUG	International functie punt Users Group
IT	Informatie Technologie
λ (lambda)	Faalfrequentie (dimensie 1/uur)
OS	Operating System
ProBO	Probabilistisch Beheer en Onderhoud
Q	Faalkans per vraag
RAMS	Reliability, Availability, Maintainability, Safety
Rationale	Beschrijving en onderbouwing van ontwerpbeslissingen, conform ISO/IEC/ IEEE 42010
SDP	Software Development Plan
SOBEK	Software voor het voorspellen van te verwachte lokale waterstanden
Systeem	Samenhangend geheel van fysieke onderdelen dat bedoeld is om een bepaalde functie te vervullen. Anders gezegd: een afhankelijk van het gestelde doel binnen de totale werkelijkheid te onderscheiden verzameling elementen, die onderlinge relaties hebben. De netwerken van Rijkswaterstaat zijn systemen, maar de onderdelen daarvan ook. Elk systeem is onderdeel van een groter geheel en is dus in feite een deelsysteem. Het hangt dus van de context af waar de grenzen van het systeem worden getrokken.
TOPAAS	Task Oriented Probability of Abnormalities Analysis for Software
(Ongewenste) topgebeurtenis	(partieel) functieverlies van een systeem. De ongewenste topgebeurtenis (OTG) is de gebeurtenis, waarvan de kans wordt berekend in een kwantitatieve risicoanalyse. Veelal is dit het falen van de hoofdfuncties van het systeem, zoals hoog water keren, laten passeren scheepvaart, laten passeren wegverkeer, afvoeren water, et cetera.
TUB	Taakuitvoeringsbouwblok
UML	Unified Modeling Language

[2] Terminologie komt primair uit de Handreiking Prestatiegestuurde Risicoanalyse (PRA) [8]

Bijlage A : Alternatieve faalkansanalysemethoden

A.1 Reliability Growth Modelling

Reliability Growth Modelling (RGM) is gebaseerd op het uitgangspunt dat de betrouwbaarheid van TUBs toeneemt met het herstellen van gevonden fouten. De betrouwbaarheidsgroei die zich daardoor manifesteert is namelijk niet willekeurig. De groei voltrekt zich volgens een bepaalde curve afhankelijk van de aard van het systeem, de manier waarop het gebruikt wordt, de soorten fouten die gevonden worden en de wijze waarop deze worden opgelost.

Wanneer voldoende informatie wordt verzameld over de TUB en het falen daarvan, is het mogelijk een eventuele betrouwbaarheidsgroeicurve af te leiden. Het 'eventuele' daarvan is gelegen in het feit dat de modellen die de groeicurven beschrijven niet per definitie uitputtend zijn. Zelfs wanneer een TUB aan alle voor het model gestelde voorwaarden voldoet, bestaat er de theoretische kans dat nog onbekende variabelen een rol spelen.

Bovendien moeten de modellen gevoed worden met statistisch significante aantallen fouten om enigszins betrouwbaar te zijn.

En tot slot vergt het toepassen van betrouwbaarheidsgroeimodellen de nodige expertise op het gebied van statistiek, ook bij het gebruik van speciaal daarvoor ontwikkelde tools.

A.2 Monte Carlo

Door middel van de Monte Carlo methode, waarbij de TUB met random inputwaarden wordt getest en waarbij de resultaten kunnen worden beoordeeld als zijnde goed of fout, kan de kans op falen van een taakuitvoering worden vastgesteld na een gegeven aantal testen. Het aantal uit te voeren testen is afhankelijk van de benodigde faalkans.

Gebaseerd op de binominaalverdeling is de volgende tabel te berekenen voor x foutloze testen:

	Aantal foutloze testen van typerend gebruik											
Faalkans	**1**	**10**	**30**	**100**	**300**	**1.000**	**3.000**	**10.000**	**30.000**	**100.000**	**300.000**	**1.000.000**
10^{0}	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00
10^{-1}	10,00	65,13	95,76	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00
10^{-2}	1,00	9,56	26,03	63,40	95,10	100,00	100,00	100,00	100,00	100,00	100,00	100,00
10^{-3}	0,10	1,00	2,96	9,52	25,93	63,23	95,03	100,00	100,00	100,00	100,00	100,00
10^{-4}	0,01	0,10	0,30	1,00	2,96	9,52	25,92	63,21	95,01	99,99	100,00	100,00
10^{-5}	0,00	0,01	0,03	0,10	0,30	1,00	2,96	9,52	25,91	63,21	95,01	99,99

Tabel 1: Confidence level (%) bij gegeven faalkans en aantal foutloze testen/gebruik

In deze tabel wordt de benodigde faalkans gekoppeld aan de omvang van de daarvoor noodzakelijke steekproef/test en de daaruit resulterende confidence in dit antwoord. In de regel wordt uitgegaan van een confidence level van 95% voordat men de benodigde faalkans in een foutenboom mag opnemen. Dus om bijvoorbeeld een faalkans van 10^{-2} te kunnen aantonen zijn minimaal 300 foutloze testen nodig. Bij dit voorbeeld moet worden opgemerkt dat het hier om zeer eenvoudige taakuitvoering door een TUB gaat. Het opzetten van Monte Carlo testen voor echte software is de bepaling van de steekproefgrootte en de testinrichting zeker niet triviaal. Bij de opzet hiervan is het dus van belang ervaren adviseurs te betrekken (bijvoorbeeld van gespecialiseerde consultancy bedrijven of universiteiten) om betrouwbare uitspraken te kunnen doen.

Wat dit voorbeeld van zeer eenvoudige software al aantoont is dat bij kleine faalkansen, betrouwbare software dus, het aantal uit te voeren testen al snel groot wordt. Als men een test volledig automatiseert, kan men met een testtijd van vijf seconden "maar" 15.000 gevallen per dag testen, wat betekent dat het testen enkele dagen tot weken in beslag neemt. De bijkomende moeilijkheid bij deze volumes ligt hem enerzijds in de doorlooptijd van deze testen en anderzijds in de controle of de taakuitvoering door de TUB tijdens de test foutloos was. Bij grote aantallen testen moet die controle geautomatiseerd worden. Dit vereist een stuk verificatiesoftware die onafhankelijk van de getest TUB ontwikkeld is en zelf ook gevalideerd moet worden. Alhoewel dit alles zeker niet onmogelijk is, kan het zijn dat de totale doorlooptijd voor deze testen en controles daardoor praktisch te groot wordt, zelfs als de taakuitvoering die de TUB vervuld zeer eenvoudig en snel af te testen is.

A.3 Duurtesten

Voor sommige eenvoudige bouwstenen, met name de zuiver reactieve zonder complexe beslissingslogica, is het een optie om dezelfde test zeer vaak te herhalen, met kleine variaties, om zo aan te tonen dat de faalkans ook daadwerkelijke gehaald wordt. Hier komt men onvermijdelijk op de vraagstelling hoe groot de steekproef (aantal herhalingen) moet zijn om met een geaccepteerde zekerheid een uitspraak te doen over de betrouwbaarheid van een TUB, waarvoor ook Tabel 2 *"Confidence level bij gegeven faalkans en aantal foutloze testen/gebruik"* geldt. Ook hier komt men al snel in de aantallen die gehanteerd worden bij Monte Carlo testen met de bijbehorende praktische problemen en moet men wederom gespecialiseerde kennis hebben voor de opzet van de testen.

Bijlage B : Gebruik van het 4+1 model (informatief)

Om meer diepgang aan de architectuurcomponent te geven wordt in deze bijlage de 4+1 methode van Kruchten [3] toegelicht. Deze methode is expliciet geen verplichting om te volgen voor TOPAAS, maar is wel één van de meest gebruikte implementaties van de ISO/IEC/IEEE 42010 en is gebruikt voor veiligheidskritische toepassingen. Voordeel van de 4+1 methode is ook dat deze in UML goed uit te voeren is, wat op een zeer brede tool-support kan rekenen. In het vervolg van dit hoofdstuk werken we als illustratie de views van de 4+1 methode uit met de aandachtspunten voor de koppeling tussen TOPAAS en het begrip foutenboom. Hiermee proberen we een beter begrip te kweken voor de relatie tussen een foutenboom, de architectuur en TOPAAS TUBs.

Een softwarearchitectuur volgens de 4+1 methode bevat de volgende views:

- **Logical view**, die een beschrijving geeft van de functionele onderdelen van een systeem en de functionaliteit voor de eindgebruiker. Ook geeft deze view de randvoorwaarden voor goed functioneren van de software en zijn onderdelen.
- **Development view**, die een beschrijving geeft van het systeem vanuit de ontwikkelaar. Deze view beschrijft de organisatie van de software gedurende de ontwikkeling in termen van pakketten broncode, gebruikte standaardbibliotheken van code, compilers, linkers en de onderlinge afhankelijkheden daartussen.
- **Process view**, die een beschrijving geeft van alle actieve processen gedurende executie, inclusief alle benodigde onderlinge communicatie, synchronisatie, redundantie en concurrency-control.
- **Physical view**, die een beschrijving geeft van de hardware-infrastructuur (PLCs, servers en netwerken) en de allocatie van de run-time onderdelen daarop, vanuit het perspectief van de system engineer.
- **Use-case view/Scenario's**, dat, door middel van een kleine set voorbeelden (use-cases) of scenario's, inzicht geeft in de samenhang tussen de verschillende modules. Hierbij wordt expliciet niet alleen het primaire scenario beschreven, maar ook de alternatieve scenario's die ontstaan ten gevolge van redundantie of monitoring met restart functie (watchdog resets). Ten minste moet deze view alle uit te voeren taken weergeven die voor de foutenboom van belang zijn. De scenario's zullen als basis dienen voor het testprotocol of het prototype.

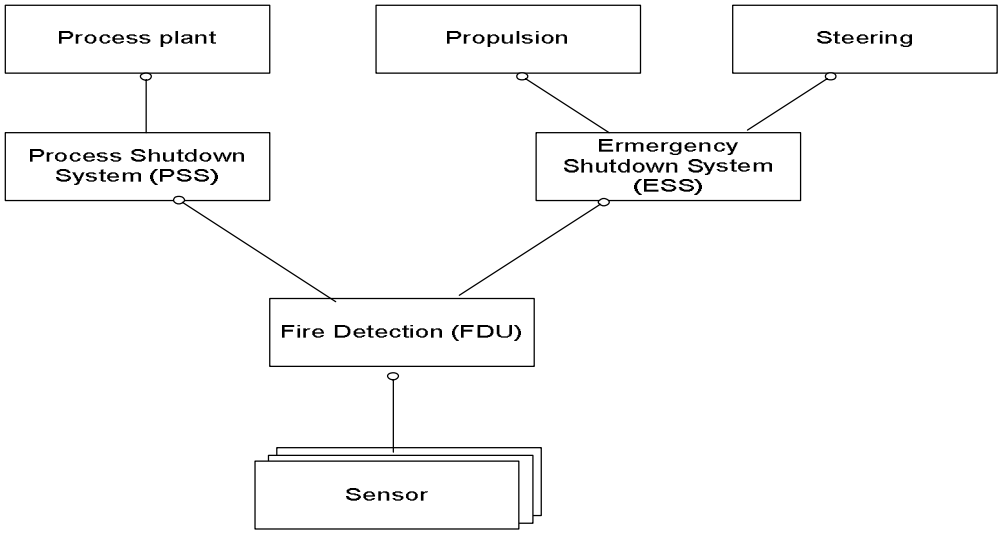
Tussen de views zit altijd een zeer grote overlap. Zo zal:

- de process view beschrijven hoe modules met elkaar samenhangen;
- de physical view beschrijven hoe diezelfde modules verdeeld worden over hardware en netwerkverbindingen;
- de scenario's view beschrijven hoe specifieke modules samenwerken om taakuitvoering te realiseren.

Met deze views tezamen kan een beeld worden opgebouwd hoe het systeem werkt, vergelijkbaar aan een hydraulische, elektrische of mechanische bouwtekening. Deze bouwtekeningen moeten dus óf door leveranciers opgeleverd worden bij oplevering óf door een software analist in retrospect moeten worden vastgesteld op basis van alle andere aanwezige documentatie.

B.1 Voorbeeld van een 4+1 architectuur
Om een 4+1 architectuur wat tastbaarder te maken, beschrijven we er één op een vrij abstract niveau.

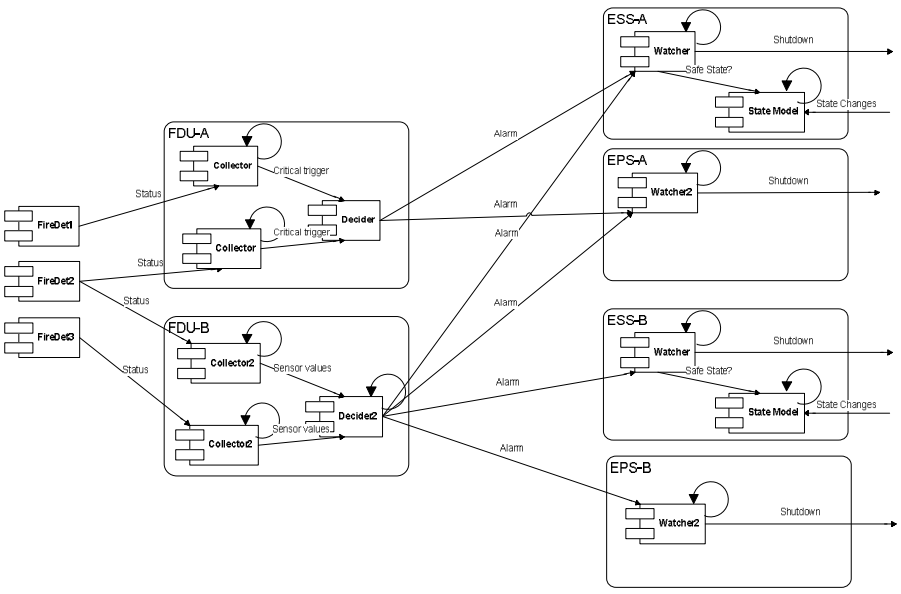
Logical View
De logical view ziet er als volgt uit:



Figuur 6: Logical view

Dit is een opbouw van de brandmeldinstallatie van een schip, waarbij bij detectie van brand alle mechanische processen (zowel primair als secundair) stilgelegd moeten worden. Het laat zien dat de brandmeldinstallatie (FDU) het noodstopsysteem van het schip (ESS) en het industriële proces via een noodstop (PSS) stil legt.

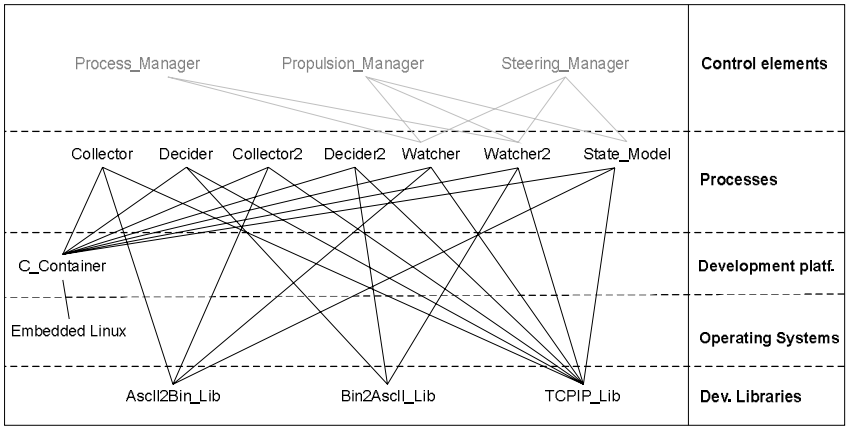
Process View
De Process view ziet er als volgt uit:



Figuur 7: Process View

In deze view gaat het over de samenhang van de onderkende processen/modules, ongeacht de fysieke belegging van deze processen over hardware, of hardware/software redundantie.

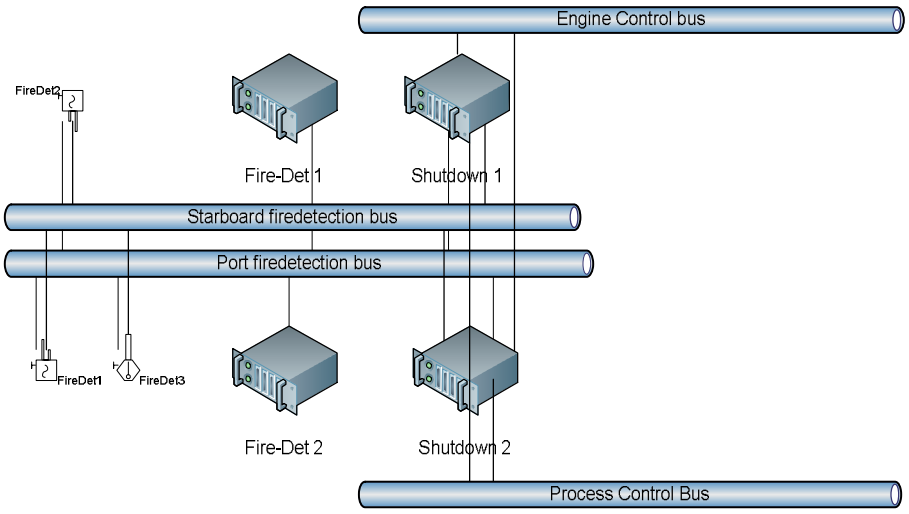
Development view
De development is een view voor ontwikkelaars voor software, en verteld welke libraries er gebruikt en gedeeld worden:



Figuur 8: Development view

Zoals het bovenstaande voorbeeld laat zien, worden de processen uit de process view hier gerelateerd aan onderliggende libraries. Een toepassing hiervan zou bijvoorbeeld kunnen zijn om common mode failures te kunnen ontdekken ten gevolge van library-fouten.

Physical view
De fysieke layout kan als volgt zijn:

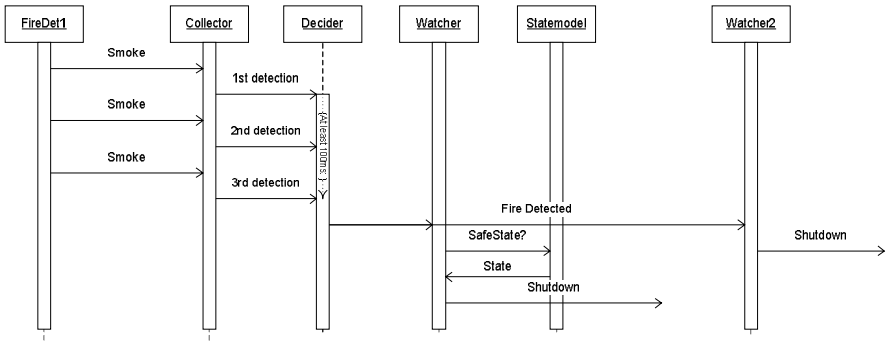


Figuur 9: Physical View

Waarbij de FDU-A en FDU-B processen redundant op de Fire-Det 1 en 2 servers zijn geplaatst, en de ESS en EPS processen redundant op de shutdown servers.

Use-case view/Scenario's

Nu blijft de vraag over hoe het scenario van branddetectie daadwerkelijk verloopt. Dit is als volgt:



Figuur 10: Use-case View

Hieruit blijkt dus hoe processen samenwerken om functionaliteit te realiseren.

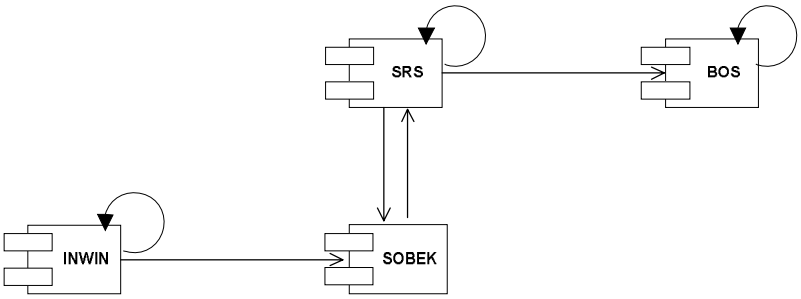
B.2 Aandachtspunten voor gebruik van een 4+1 model
Om tot een goed bruikbare architectuurbeschrijving te komen, zijn er wel wat praktische aandachtspunten, die zich met name concentreren op het minimale detailniveau wat voor de verschillende views gebruikt wordt:

- Physical view, omdat fysieke componenten uit deze view ook in de foutenboom terecht komen, is het verstandig om alle individuele hardware componenten op hetzelfde abstractieniveau beschreven worden als de rest van de foutenboom te beschrijven;
- De Process view dient op het detailniveau van de scenario's aan te sluiten. Dit impliceert dat elke genoemd process/module in de Scenario-view moet ook benoemd zijn in de process-view;
- Use-case view/Scenario's moeten alle taakuitvoeringen die voor de foutenboom van belang zijn herkenbaar zijn. Deze moeten dus aansluiten op het niveau van detaillering dat ook in de foutenboom gehanteerd wordt;

Bijlage C : Voorbeeld (informatief)

Het BOS is het Beslis Ondersteund Systeem dat bij de Maeslantkering bepaalt of de kering gesloten dient te worden. Het bepaalt tevens alle tijdstippen die in de sluitprocedure van belang zijn, bijvoorbeeld de waarschuwingen naar het Havenbedrijf Rotterdam, het openen van de dokdeuren, het uitvaren, etcetera. In de initiële foutenboom wordt als basisgebeurtenis en falende taakuitvoering het object "Onterecht geen overschrijding" opgenomen dat uitgewerkt en gekwantificeerd moet worden.

Het voert hier te ver om de architectuur in detail te behandelen, maar op (zeer vereenvoudigde) hoofdlijnen ziet het BOS er als volgt uit:

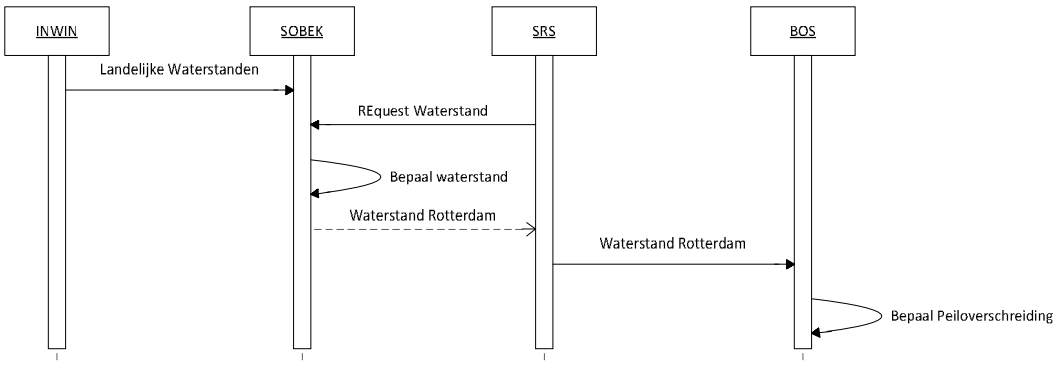


Figuur 11: Process view van INWIN, SRS, SOBEK en BOS

Hier zal INWIN ruwe waterstanden uit het meetnet aanleveren aan SOBEK. SOBEK wordt periodiek bevraagd over de verwachte waterstand bij Rotterdam, waarna BOS bepaalt of dat boven de maximum waterstand is.

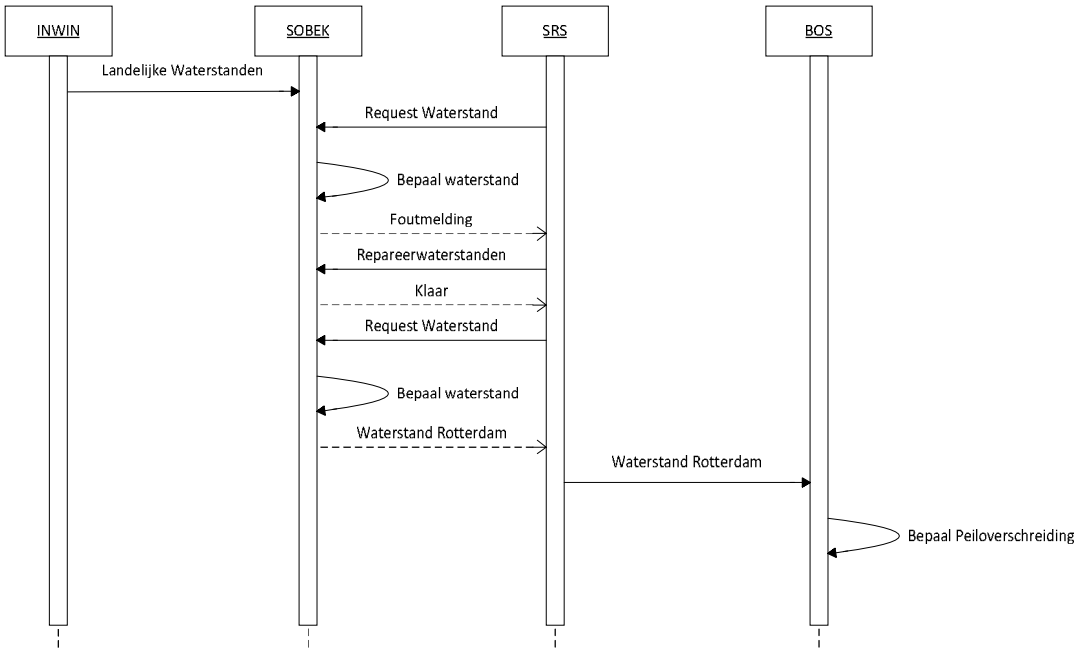
Bij de bepaling eerste kwalitatieve modellering worden INWIN, SOBEK, SRS en BOS als onderdeel van de analyse gezien.

Nu zijn er twee scenario's betrokken bij het bepalen van een peil-overschrijding:



Figuur 12: Regulier Scenario Peil-overschrijding

En het alternatieve scenario, bij “kapotte” waterstanden is als volgt:



Figuur 13: Alternatief Scenario Peil-overschrijding

Nu de grove contouren helder zijn, moeten de specifieke TUBs bepaald worden. De peiloverschrijding wordt bepaald op basis van een set beslisregels en een verwachte waterstand bij Rotterdam. De verwachte waterstand wordt geproduceerd in de SRS module door softwarefuncties *SRS:ValideerModelInput* en *SOBEK:BepaalWaterstandRotterdam*.
Afhankelijk van incompleetheid van inputdata voor SOBEK wordt de functie *SRS:RepareerInput* uitgevoerd. Condities voor *SRS:RepareerInput* zijn voor deze functie zijn:

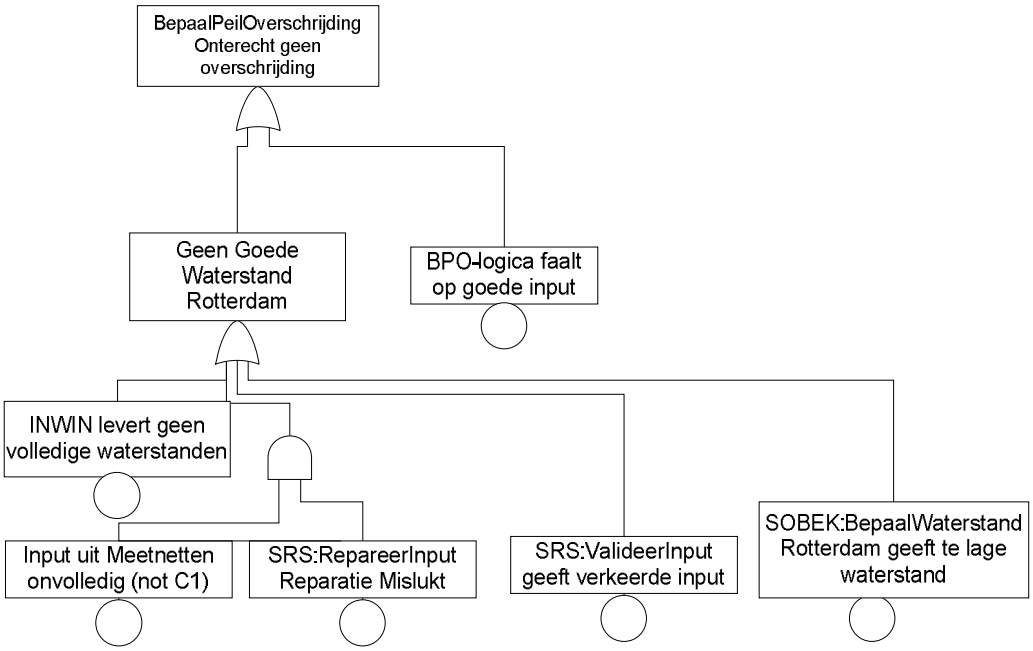
- C1: er is een recente (>-3 uur) verwachting van de waterstand bij Hoek van Holland of een redelijk recente verwachting (> - 12 uur) van de waterstand bij Hoek van Holland beschikbaar
- C2: het astronomisch getij bij Hoek van Holland en een gemeten waterstand bij Hoek van Holland is beschikbaar
- In geval van C1 wordt *SRS:RepareerInput* niet uitgevoerd
- In geval van C2 (en niet C1) wordt *SRS:RepareerInput* uitgevoerd
- In overige gevallen faalt *SRS:RepareerInput* geheel (stopt).

Kijkt men naar de opzet van de architectuur, dan zien we de volgende TUBs:

- De taken *SRS:ValideerModelInput*, *SOBEK:BepaalWaterstandRotterdam* en BPO-logica worden iedere 10 minuten uitgevoerd, waarbij alleen de uitvoering van BPO-logica afhankelijk is van de taakuitvoering.
- De taken *SRS:ValideerModelInput* en *SRS:RepareerInput* zijn geïmplementeerd via de module SRS, draaiend op de Stratus ftServer, gebruikmakend van BOS-database access. Geïmplementeerd in C/C++ met bijbehorende bibliotheken.

- De taak *SOBEK:BepaalWaterstandRotterdam* is geïmplementeerd via de module SOBEK, draaiend op de Stratus, geïmplementeerd in Fortran90 gebruikmakend van fortran bibliotheken.
- De softwarefunctie BPO-logica is geïmplementeerd in de BOS module PSI.

De uitgewerkte foutenboom zou er op dit onderdeel zo uit kunnen zien. Het niet aanwezig zijn van meetdata is in de huidige boom ook verder uitgewerkt. Dat is hier ook nodig. Ter illustratie is het als basisgebeurtenis opgenomen.



Figuur 14: Foutenboom BepaalPeilOverschrijding faalt

Om kwantificering te kunnen uitvoeren zijn detailgegevens van de software nodig. Een belangrijke analyse daarin is de hoeveelheid broncode die de taak in beslag neemt en wat de complexiteit van deze code is. De softwarefunctie *SRS:RepareerInput* wordt door de module *SRS* geïmplementeerd. De taak *SRS:ValideerInput* is slechts 10% van de module *SRS*.

De code voor *BPO-logica* is geïmplementeerd in de BOS module *PSI* in methoden:

- *psbepmodelevaluatie* met een cyclomatische complexiteit 14 en aantal code statements 116 LOC)
- *psbeppeiloverschr* (complexiteit=10 code 162).

De module *PSI* is ongeveer 40kLOC, waarvan specialistische taken (zoals *beppeiloverschr*) 10kLOC zijn. Voor de kwantificering wordt 30kLOC genomen met een complexiteit 15 (zijnde de maximale complexiteit in het algemene gedeelte).

De TUB INWIN, met als falende taakuitvoering "INWIN geeft geen waterstanden" wordt geschat met behulp van een statistische analyse, de frequentie van deze taakuitvoering (1/10min) maakt dat de afgelopen 10 jaar 144*365*10 = 50.000 runs vergeleken kunnen worden met de feitelijke waterstanden.

Kijkt men naar de *SOBEK: Bepaalwaterstanden* TUB, met als falende taakuitvoering *"SOBEK Bepaalwaterstanden geeft te lage waterstand"*, dan krijgt men de volgende scoring:

Totstandkomingsproces		
1	Het ontwikkelproces voldoet aan een van de SIL's van de IEC 2 Ontwikkelproces voldoet aantoonbaar aan SIL 1 niveau *Gebouwd door kleine club goede SW engineers met "matige" specifieke procesvastlegging binnen ISO9001 proces.*	-½
2	Gebruik van Inspecties 2 Inspecties op ontwerpen en code uitgevoerd *Reviews uitgevoerd, geen formele inspecties*	0
3	Hoeveelheid wijzigingen ten opzichte originele ontwerp/eisenpakket 3 Geen wijzigingen *Hier wisselt het gedrag voor SRS met betrekking tot gedrag uitzonderingen. De main stream is constant*	-⅓
4	Cultuur en samenwerking ontwikkelorganisatie 2 Doelgerichte organisatie *Organisatie met een degelijk trackrecord en een goed georganiseerd intern werkproces*	0
5	Opleidingsniveau en ervaring ontwikkelaars 4 Aantoonbaar uitstekende kennis en veel ervaring van het specifieke domein *Gebouwd door kleine club goede SW engineers. Goed in de materie, geen procesvastlegging.*	-½
6	Samenwerking met Opdrachtgever 3 Sterk betrokken opdrachtgever met voldoende kennis, open dialoog waarbij de opdrachtgever bereid is wijzigingen te implementeren op systeemniveau om suboptimaal gedrag te vermijden. Er is sprake van een systems engineering approach voor de overall ontwikkeling van het systeem. *Projectorganisatie en aansturing daarvan lieten ruimte voor redesign, indien noodzakelijk. Allen betrokken partijen waren capabel en betrokken.*	-½

Product		
7	Complexiteit beslissingslogica van de TUB 3 Beslislogica is redelijk eenvoudig (bevat enkele zeer geïsoleerde uitzonderingssituaties), McCabe index tussen 10 en 30.	-⅓
8	Omvang TUB (Lines of code) 2 Tussen 10.000 en 50.000	⅓
9	Helderheid gebruikte architectuurconcepten 3 Er is een scheiding van taken en verantwoordelijkheden voor modules beschreven, welke het principe van "maximale cohesie en minimale koppeling respecteert", maar deze is passief bewaakt tijdens het ontwikkelproces	-⅓
10	Gebruik van een certified compiler 2 Gebruik van een compiler waar men langdurige ervaring mee heeft	0

Traceerbaarheid van eisen en verifieerbaarheid		
11	Traceerbaarheid van eisen door het proces heen 2 Traceerbaar naar architectuur/testen	-⅓

Testen		
12	Testtechnieken en dekkingsgraad 4 Formele testtechniek(en) gehanteerd met gemiddelde dekkingsgraad	-⅔

Executieomgeving/gebruik		
13	Multiprocesomgeving 2 Meerdere processen op één stuk hardware	½
14	Aanwezigheid representatieve velddata gedurende taakuitvoering 4 Veel representatieve velddata aanwezig van identieke of sterk vergelijkbare toepassingen	-2
15	Monitoring 2 Weinig/kort gemonitord gedurende taakuitvoering	0
Totaal		-4⅔

Dit geeft aanleiding tot een geschatte faalkans per vraag van $10^{-4⅔}$, afgerond op 10^{-4} wat volledig overeenkomt met de inschatting door experts die nauw bij het project betrokken zijn.

Bijlage D : Het Fagan-inspectieproces (informatief)

De Fagan-inspectie (ooit door Michael Fagan voor IBM bedacht) is een softwarereviewmethode waarbij een (tussen)product, bijvoorbeeld een ontwerp of code, grondig wordt bekeken met het doel om fouten te corrigeren voordat het project naar een volgende ontwikkelfase doorgaat. De inspectie wordt na elke ontwikkelfase uitgevoerd door drie tot zes mensen met rollen als moderator, inspecteur, notulist of auteur. Het proces bestaat uit zes stappen:

- **Planning.** De moderator plant de inspectie. Hij stelt de inspectie team samen, plant de bijeenkomsten en verspreidt het benodigde materiaal.
- **Aftrap [optioneel].** Zo nodig wordt het te inspecteren product toegelicht door de auteur.
- **Voorbereiding.** De inspecteurs bereiden zich individueel voor op de inspectiebijeenkomst. Ze bestuderen het product vanuit de eigen expertise en zoeken naar fouten, met behulp van de 'hogere documenten' en de controlelijsten. Na afloop vullen ze de gevonden fouten en de benodigde tijd in op het individuele bevindingenformulier. De moderator verzamelt alle individuele bevindingenformulieren.
- **Inspectie.** De inspecteurs zoeken, bespreken en classificeren gezamenlijk de fouten in het (tussen)product. De moderator leidt de bijenkomst en zorgt dat er naar fouten gezocht moet worden - niet naar oplossingen - en dat de auteur niet daarop aangevallen wordt. De notulist noteert de gevonden fouten op een registratieformulier.
- **Herstel.** De auteur herstelt alle gevonden ernstige fouten en houdt bij welke fouten gecorrigeerd zijn en hoeveel tijd het heeft gekost.
- **Afronding.** De moderator controleert of de auteur het herstelwerk goed heeft gedaan en of het product aan de eindcriteria voldoet. Verder maakt de moderator een samenvatting van de inspectie waarin onder andere worden vermeld het aantal deelnemers, de bestede tijd per stap, het aantal fouten, de hersteltijd en of het product geaccepteerd is.

Typerend voor Fagan-inspecties is het gebruik van start- en eindcriteria die het begin en het afronden van een ontwikkelfase conditioneren. Die komen voort uit de randvoorwaarden die in de ‘hogere documenten’, namelijk de documentatie van het hogere abstractieniveau, beschreven staan. Een gedetailleerd ontwerp is bijvoorbeeld het 'hogere document' van code.
Uit de praktijk blijkt dat het inspectieresultaat sterk afhangt van de nauwkeurigheid waarmee het proces uitgevoerd wordt. Daarom is het van belang dat de mensen die ermee beginnen, getraind zijn in het uitvoeren van Fagan-inspecties en over een minimale infrastructuur beschikken: standaard inspectieformulieren, controlelijsten en een database om de gegevens in op te slaan.

Studies bij verschillende bedrijven (zoals AT&T, HP, NASA, IBM, ICL, BNR) hebben aangetoond dat Fagan-inspecties 5 tot 20 keer effectiever zijn dan testen, de productiviteit verhogen met 14 tot 23 procent en het fouten herstellen tien keer efficiënter maakt.

TOPAAS

Nummer:	1319
Versie:	2.0
Status:	In beheer
Type:	Kader
Inhoudelijk beheerder:	Jeroen Horstman
Verantwoordelijke afdeling:	Afd. Advies Technisch Management
Netwerken:	Hoofdvaarwegennet, Hoofdwatersysteem, Hoofdwegennet
Rollen:	Technisch Manager
Fase:	Realisatie
Proceseigenaar	Proceseigenaar Aanleg en Onderhoud