Rijkswaterstaat
Ministerie van Verkeer en Waterstaat

TOPAAS

Een structurele aanpak voor faalkansanalyse van software intensieve systemen

Datum 1 april 2011
Status Definitief

TOPAAS

Een structurele aanpak voor faalkansanalyse van software intensieve systemen

Datum 1 april 2011
Status Definitief

Colofon

Uitgegeven door	RWS Dienst Infrastructuur
Informatie	S.E. van Manen
Telefoon	06 – 55 74 52 95
Fax	088 – 797 4006
Uitgevoerd door	Ed Brandt (Refis)
	Dr. Alessandro Di Bucchianico (TU/e)
	Ir. Jaap van Ekris (DNV)
	Prof. Dr. Ir. Jan-Friso Groote (TU/e)
	Dr. Wouter Geurts (Logica)
	Dr. Ir. Gerben Heslinga (Intermedion)
	Drs. Gea Kolk (Movares)
Gereviewd door	Ir. Peter van Gestel (Delta Pi)
	Ing. Piet de Groot (NRG)
	Ir. Harry van der Graaf (RWS)
	Ir. Sipke van Manen (RWS)
Datum	1 april 2011
Status	Definitief
Versienummer	2.0

Inhoud

1 Inleiding

Rijkswaterstaat is bezig om op alle primaire waterkeringen en andere kunstwerken probabilistisch beheer te introduceren. Centraal in de aanpak van probabilistisch beheer is de risicoanalyse, die sturend is in de testintervallen, gegarandeerde reparatietijd en modificaties. Ook het falen van de gebruikte software is gemodelleerd. Voor de initiële inschatting van de faalkans van de software is de TDT-methode ontwikkeld. In praktijk blijkt deze onbetrouwbare resultaten te leveren.

In opdracht van Rijkswaterstaat heeft een consortium van Det Norske Veritas, Movares, Technische Universiteit Eindhoven, Logica, Refis en Intermedion een verbeterde methode ontwikkeld die zowel richtlijnen geeft voor het modelleren van softwarefalen in foutenbomen als het schatten van de faalkans van een taakuitvoering door een softwaremodule.

Deze methode is gerapporteerd in [8] en TOPAAS genoemd. Aan de hand hiervan zijn een aantal experimenten (pilots) uitgevoerd. De resultaten van die pilots zijn beschreven in een evaluatie [16] en deze evaluatie doet een aantal aanbevelingen voor verbetering. In deze tweede versie van [8] zijn de aanbevelingen verwerkt. Ook is de tekst hier en daar redactioneel aangepast, met name ter verduidelijking voor de niet-ICT'er. Tevens wordt aanbevolen een korte handleiding voor het toepassen van TOPAAS te maken.

Kern van TOPAAS is dat software in modulen kan worden opgedeeld en dat het (mogelijk) falen van deze modulen in een foutenboom als basisgebeurtenissen kunnen worden opgenomen. Falen van een softwaremodule kan vervolgens opgedeeld worden in falen ten gevolge van onverwachtheid van input en het falen van de beslislogica van de softwaremodule zelf.

Schatten van de faalkans van een softwarecomponent (module) is moeilijk: er zijn wel methoden, maar die vereisen zonder uitzondering input die vaak niet (voldoende) voorhanden is. Om toch te komen tot een faalkansschatting van een softwaremodule wordt op basis van expert opinion een schatting gemaakt, waarbij het Bayesiaanse gedachtegoed wordt gevolgd. Deze schatting is vervolgens vervat in een parametermodel, waarbij de factoren die in ogenschouw worden genomen voortkomen uit de expertgroep en internationaal onderzoek. De invloed van de factoren is ingeschat door experts en vervolgens gekalibreerd met een twintigtal referentieprojecten. Conclusie is dat de uitkomsten van het parametermodel een zeer sterke correlatie vertoont met de inschatting van de experts.

Concluderend mag men stellen dat deze methode, bij afwezigheid van betere wijzen van schatting, een redelijk betrouwbare Bayesiaanse schatting van de faalkans levert.

2 Aanleiding en vraagstelling

2.1 Aanleiding

Om tot een eerste risico-inschatting te komen van de faalkans van software is er door NRG een methode ontwikkeld, genaamd de TDT-methode, Otterloo [18]. Deze methode kent twee delen:

- De analyse van de systemen om tot modellering van het softwarefalen te komen in de foutenboom (FTA);
- De analyse van een softwaremodule om daarmee tot een schatting van de faalkans van die component te komen.

De analyse die gebruikt wordt voor de modellering van softwarefalen is gebaseerd op de aanpak voor risicoanalyse van elektromechanische installaties: men kijkt naar de faalkans van componenten.

Voor de schatting van de kans op softwarefalen in primaire waterkeringen maakt de methode gebruik van drie parameters:

- Toepassingsdichtheid: een maat voor de breedte van de toepassing van de component;
- Dynamisch gedrag: Een inschatting van de dynamiek van de applicatie aan de hand van de I/O parameters en de interne complexiteit;
- Totstandkoming: Een inschatting van de professionaliteit van de wijze waarop de software ontwikkeld is.

Voor elk van deze parameters wordt een score tussen 1 en 4 bepaald, waarbij de faalkans toeneemt bij een toename van het product van deze factoren. De score varieert dus van 1 (1x1x1) tot 64 (4x4x4).

De methode van modellering kent zwaktes in het aantoonbaar maken van de faalkans van software. De gebruikte werkwijze om tot modellering van de foutenboom te komen heeft, analoog aan elektrotechnisch falen, per softwaremodule maar één enkele faalmodus ('functioneert niet'). Dit doet geen recht aan de verschillende taken die softwaremodulen moeten uitvoeren voor het realiseren van een missie. Daarnaast kan software volgens bepaalde ontwikkelstrategieën ook extra functionaliteit hebben om foutgedrag te ondervangen, wat op dit moment niet meegenomen wordt in de modellering. Ook zijn vormen van opvolgfalen onderbelicht: faalvormen als deadlock worden niet systematisch in ogenschouw genomen.

Er zijn sterke kanttekeningen te plaatsen bij de werkwijze die de TDT methode gebruikt voor het schatten van de daadwerkelijke faalkansen in de foutenboom. De methode is initieel ontwikkeld om gevoeligheidsanalyses uit te kunnen voeren voor softwaremodulen in de Maeslantkering, Hartelkering en Hartelsluis. Het gaat hierbij dus om ruwe inschattingen met het doel om te bepalen hoeveel invloed een softwaremodule heeft op de totale faalkans, niet om de bepaling van een te verwachte faalkans. Zoals NRG zelf al aangeeft in de inleiding “mag er geen waarde aan de kansen gehecht worden”.

De TDT methode leidt, mede door deze tekortkomingen, tot aantoonbaar foute resultaten. Een voorbeeld van een dergelijk foutief resultaat is het volgende: één systeem dat door twee groepen ontwikkeld wordt:

- Als het systeem door onervaren mensen op eenvoudige wijze snel in elkaar gezet wordt krijgt het een TDT score van 1x1x4 = 4
- Als ditzelfde systeem door hoog opgeleide ervaren professionals ontwikkeld wordt, onder de SIL-4 eisen van de IEC [11] dan krijgt het een TDT score van 1x4x1 = 4. Dit komt voornamelijk doordat de door de IEC vereiste defensieve programmering en diagnostische controles in de TDT methode leidt tot extra complexiteit en dus een grotere faalkans. Dit terwijl de aanname van de IEC is dat deze aanpak een sterk positief effect op de betrouwbaarheid zal hebben.

Deze gelijke score is in directe tegenspraak met wat de internationale gemeenschap verwacht van deze systemen, hetgeen problematisch is voor de acceptatie van de getallen door het grote publiek.

2.2 Vraagstelling

Rijkswaterstaat heeft behoefte aan een verbeterde methode die de voordelen van de huidige methode kent, maar significant beter de toets der kritiek van ICT experts en wetenschappers kan doorstaan. De criteria voor een goed raamwerk zijn samen te vatten als 'vertrouwenwekkend' en 'praktisch bruikbaar'.

Vertrouwen ontstaat door:

- Betrouwbare uitkomsten, die stroken met het buikgevoel van experts en internationale standaarden;
- Onderbouwde resultaten waar mogelijk;
- Goede dekking qua factoren die een rol spelen bij software betrouwbaarheid;
- Interne checks mogelijk (parameters onderling te correleren);
- Externe checks (kalibratie met specialistische methoden).

Een bruikbare methode ontstaat door:

- Gebruik van gegevens die voorhanden zijn of eenvoudig zijn te herleiden;
- Het bieden van alternatieven voor moeilijk beschikbare parameters;
- bruikbaar voor bijsturing van een ontwerp door de opdrachtgever.

2.3 SWOT analyse

Van de TDT-methode is een Strength/Weakness-analyse gedaan (zie Bijlage A "SWOT Analyse"). Deze methode heeft als zeer sterk punt dat deze op vlotte wijze, met minimale data, een inschatting kan maken van componenten in een primaire kering. Echter, op zowel de modellering in de foutenboom als de faalkansschatting is zeer gefundeerde kritiek te leveren die de weerbaarheid van Rijkswaterstaat tegen kritiek van buitenaf aantast. Met name het feit dat door experts benoemde factoren niet zijn meegenomen is een zwakte.

Op de nieuwe methode, TOPAAS (Task Oriented Probability of Abnormalities Analysis for Software) genaamd, is een Threats/Opportunity analyse toegepast (zie Bijlage A "SWOT Analyse"). Voordeel van de methode is dat dit een breed toepasbare methode zou kunnen opleveren die breder geaccepteerd is als zijnde het best haalbare. De verwachting is dat het toepassen van de methode op de waterkeringen al enkele mogelijkheden voor kalibratie zal opleveren.

2.4 Bekende methoden

2.4.1 Formele methoden

Formele methoden kunnen aantonen dat logische softwarefouten afwezig zijn, of kunnen aanwijzen waar de fouten zitten. Gevolg is dus dat als deze methoden strikt worden toegepast én er rigoureuze reviews op worden uitgevoerd, de faalkans ten gevolge van logische softwarefouten de absolute nul benadert. Echter, deze methode moet in het design-proces worden toegepast en is in praktijk niet af te dwingen voor COTS (Commercial Off-The-Shelf) componenten. Ook verplaatst dit gedeeltelijk het probleem naar de juistheid van de specificaties.

2.4.2 Reliability Growth Modelling

Reliability Growth Modelling is gebaseerd op het uitgangspunt dat de betrouwbaarheid van software 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 software en de fouten, 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 softwaresysteem 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, Brandt et al. [6]. Potentieel probleem is dat bij zeer veiligheidskritische componenten het ontwikkelproces zo rigoureus wordt ingezet dat de software systematisch te weinig fouten introduceert om deze modellen betrouwbaar toe te passen. Erger nog, in veel processen wordt een component afgeschreven als het een significant aantal fouten bevat. Het kan dus zijn dat het systematisch onmogelijk is om voldoende informatie te verzamelen voor het toepassen van deze modellen.

Uit een in 2005 uitgevoerde proef, Brandt [5] op de besturingssoftware van de Ramspolkering, is de conclusie getrokken dat met de huidige gegevens en werkwijze een betrouwbaarheidsanalyse weliswaar kan worden uitgevoerd, doch dat er belangrijke risico's ontstaan wanneer niet alle vereiste gegevens beschikbaar zijn. In die gevallen dient een andere methode toegepast te worden voor het bepalen van de faalkans. Juist deze beschikbaarheid van gegevens is achteraf een probleem. Het vergt namelijk een vroegtijdig ingezet meetprogramma.

2.4.3 Monte Carlo simulatie

Door middel van de Monte Carlo methode, zie bijvoorbeeld Metropolis and Ulam [17] waarbij de software met random inputwaarden wordt getest en waarbij de resultaten kunnen worden beoordeeld als zijnde goed of fout, kan de kans op falen worden vastgesteld na een gegeven aantal tests. Het

aantal uit te voeren tests is afhankelijk van de gevonden faalkans. Bij kleine faalkansen, betrouwbare software dus, wordt het aantal uit te voeren tests al snel groot. De grote moeilijkheid ligt hem dan in de controle of de uitvoer juist is, gegeven de invoer, of, anders gezegd, of de taakuitvoering 100% foutloos is. Bij grote aantallen tests moet die controle worden geautomatiseerd en dat vergt dan dus een ander stuk software, onafhankelijk van de te testen software. Daarnaast kan het zijn dat de totale rekentijd voor deze tests en controles praktisch te groot wordt.

2.4.4 Conclusie

Concluderend kan men stellen dat er wel methoden beschikbaar zijn die valide resultaten leveren. Resultaten die betrouwbaarder zijn dan de kwantitatieve schatting door een model als TOPAAS. Echter, deze methoden vereisen informatie die niet voorhanden is of uitermate bewerkelijke werkzaamheden voor systemen van enige omvang en complexiteit. TOPAAS kan worden ingezet als de hoeveelheid informatie of de benodigde inspanning het niet toelaat dat betrouwbaardere modellen worden toegepast.

2.5 Leeswijzer

Deze rapportage is als volgt opgedeeld in de volgende onderdelen:

- Hoofdstuk 3 richt zich op de definitie van falen en faalkansen. Het legt het verschil uit tussen het traditionele technische falen van electro/mechanische componenten, menselijk falen en software falen;
- Hoofdstuk 4 richt zich op het modelleren van softwarefalen in foutbomen;
- Hoofdstuk 5 richt zich op het schatten van de intrinsieke faalkans van een softwaremodule;
- Hoofdstuk 6 beschrijft twee voorbeeldtoepassingen van de methode;
- Hoofdstuk 7 beschrijft de conclusies en de richting voor toekomstig onderzoek.

In dit document worden (programmatuur)componenten aangegeven met een *cursief* lettertype.

3 Vormen van falen

De faalkans van software is een moeilijk definieerbaar begrip. In tegenstelling tot elektronische en mechanische componenten zijn er geen standaard modellen om sterkteberekeningen uit te kunnen voeren op basis van een software ontwerp (vergelijkbaar aan MIL-HDBK-217). We proberen tot een definitie te komen door 'falen van software' te vergelijken met bekendere vormen van falen, namelijk hardware falen en menselijk falen. In onderstaande tabel worden deze drie vormen van falen vergeleken.

Technisch (hardware) Falen	Menselijk Falen	Software Falen
• Falen is veelal random	• Falen kan random en systematisch zijn	• Falen is veelal systematisch[1]
• Wijziging van omgeving kan soms falen veroorzaken • Ontwerpfouten zijn mogelijk waar te nemen door de (versnelde) veroudering te monitoren	• Wijzigen van omgeving kan falen veroorzaken • Ontwerpfouten, zoals in procedures en werkinstructies, afleesinstrumenten, etc. zijn mogelijk waar te nemen	• Wijzigen van omgeving kan een belangrijke aanleiding tot falen zijn • Ontwerpfouten zijn moeilijk te detecteren buiten taakuitvoering: men kan niet door regelmatige inspectie zien of de component (binnenkort) gaat falen.
• Verwacht gedrag buiten ontwerp specificaties is te voorspellen • Falen is voor een relatief groot deel merkbaar of merkbaar te maken door inspecties en testen	• Verwacht gedrag buiten ontwerp specificaties is enigszins te voorspellen • Falen is voor een relatief groot deel onmerkbaar of lastig merkbaar te maken vanwege herstelgedrag	• Verwacht gedrag buiten ontwerp specificaties is lastig voorspelbaar • Softwarefalen is vrijwel altijd onmerkbaar falen: de fout is altijd aanwezig, maar wordt vrijwel alleen gedurende de taakuitvoering ontdekt
• Door monitoren van gedrag kun je falen voorspellen • Monitoring verkleint faalkans	• Door monitoren van gedrag kun je falen voorspellen • Monitoring/training verkleint faalkans	• Falen van software kent geen voorspellende indicatoren • Monitoring kan de faalkans vergroten (door toename complexiteit)
• Kent geen herstelgedrag	• Kent herstelgedrag	• Kent redundantie en diagnose, maar slechts in zeer beperkte vorm herstelgedrag

[1] Met systematisch falen wordt bedoeld dat indien de module faalt onder bepaalde condities (bepaalde invoer, bepaalde settings van omgevingsparameters en bepaalde volgorde- en tijdscondities), de module consequent altijd zal falen onder (precies) dezelfde condities.

Technisch (hardware) Falen	Menselijk Falen	Software Falen
• Bij identificatie van falen wordt component vervangen	• Bij herkenning van falen wordt gedrag aangepast	• Bij identificatie falen wordt de component gerepareerd.

Uit bovenstaande vergelijking blijkt dat software meer overeenkomsten vertoont met menselijk falen dan met hardware falen. De voorgestelde aanpak voor de bijdrage van software aan het falen van een systeem zal dus logischerwijs erg lijken op die voor menselijk falen. Dus een focus op de taken en factoren equivalent aan trainingsniveau (kwaliteit van ontwerp en implementatie), uitvoeringsattitude (mate van interne monitoring en bewezen track record) en wijzigingen in de omgeving lijkt met een benadering die lijkt op menselijk falen (kijken naar het existentiële falen van handelingen in plaats van zoeken naar kwalitatieve reden van falen) dan ook sterk op zijn plaats.

Om te komen tot een betrouwbare schatting is het noodzakelijk om een complex (software) systeem te decomponeren tot een verzameling software modules en vervolgens de kans per module af te schatten. De notatie van decompositie is, zoals gebruikelijk in veel faalkansberekeningen, de foutenboom. Deze foutenboom is gebaseerd op het ontwerp van het software systeem. Op de wijze waarop falen van software moet worden gemodelleerd in de foutenboom wordt later teruggekomen in Hoofdstuk 4.

Gegeven is dat een faalkans van software niet berekend kan worden op basis van bekende faalgegevens uit het veld of ervaring met de toepassing van de component op de specifieke installatie. Falen van een software module in het veld leidt tot systematische verandering (met het doel van verbetering) van het product, wat een andere faalkans tot gevolg heeft. Ook heeft de onderliggende complexiteit van de beslislogica het effect dat veelal het routinematige gedrag de ervaringsgegevens domineren, terwijl de uitzonderingssituaties de daadwerkelijke faalkans in praktijk bepalen.

De kans die voor software falen wordt gehanteerd is meer een uiting van waarschijnlijkheid in Bayesiaanse zin dan dat er sprake is van falen in statistische kanstermen. Om tot een goede schatting van een faalkans te komen wordt er, in navolging van het Bayesiaans gedachtegoed, gebruik gemaakt van een gedeeld expert oordeel van een groep experts. Om tot een herhaalbaar en praktisch bruikbaar proces te komen wordt deze inschatting gemodelleerd in een parameter model. Het doel is dan om het parametermodel toe te passen als simulatie van de expert opinions om zo de kans op softwarefalen te bepalen. De gebruikte parametrisering in het model is een reflectie van de expert opinions. Hiermee wordt de afhankelijkheid van een specifieke groep experts verminderd en wordt de controleerbaarheid vergroot. Met andere woorden: het parametermodel levert een benadering van een expert-group op, om zodoende de faalkans Bayesiaans te benaderen. De specifieke totstandkoming en invulling van dit model zijn beschreven in hoofdstuk 5.

4 Kwalitatieve modellering van software falen

Modellering van software falen als onderdeel van een risicoanalyse gebeurt bij de meeste kunstwerken van Rijkswaterstaat met een foutenboom. Hierbij wordt het falen van de taak die de software vervult als basisgebeurtenis in de foutenboom opgenomen. In de geest van de reeds bestaande foutenboom modellering wordt een parallel voorgesteld die in praktijk een verdere detaillering van een foutenboom betekent. Deze detaillering behelst de faalmodi van software te concretiseren in lijn met het falen van hardwarecomponenten en menselijk handelen:

- Een software module wordt gezien als de softwarematige analogie van een hardware component c.q. een menselijke taak
- Het falen van een software module lijkt in die zin op een hardwarematige component faalmode c.q. geen of onjuiste taakuitvoering
- Net als bij hardware componenten en menselijke taken, moet de software module een welgedefinieerde interactie hebben met andere hardware componenten, menselijke taken of software modules.

In dit hoofdstuk wordt allereerst de definitie van een softwaremodule behandeld. Vervolgens komen we tot de kern van dit hoofdstuk, dat is de beschrijving hoe men komt tot een kwalitatieve risicoanalyse van een complex systeem.

4.1 Definitie van een basisgebeurtenis bij softwarefalen

Één van de kernproblemen met faalkans modellering in foutenbomen is de vraag wat de basisgebeurtenissen zijn. Ook in de fysieke wereld is dit een vraagstuk: men kan modelleren dat een pomp niet werkt, maar men kan ook specifieker zijn door te modelleren dat één van de deelcomponenten van een pomp niet werkt. Een basisgebeurtenis in een foutenboom behelst grofweg drie typen:

1. falen van een hardware component
2. het niet, niet op het juiste moment of onjuist uitvoeren van een taak door de mens en
3. het niet, niet op het juiste moment of onjuist uitvoeren van een taak door een softwaremodule.

Zoals al eerder aangegeven zijn de benaderingswijzen van het bepalen/schatten van de faalkans anders (zie Hoofdstuk 2 voor een uitleg over het verschil in faalgedrag en de benaderingswijzen). Echter, de modelering in de foutenboom, mits goed uitgewerkt tot een afdoend detailniveau, is vrijwel identiek. In dit hoofdstuk beschrijven we wat de cruciale definities zijn in de analyse van de software.

4.1.1 Definitie softwaremodule in software intensieve systemen

Uiteindelijk doel van de systemen is de realisatie van een missie. De missie is als volgt gedefinieerd:

> De **Missie** van een complex (software intensief) systeem is de uiteindelijke handeling die gerealiseerd moet worden door de samenwerkende systemen en de aangestuurde hardware.

De definitie van de missie is het begin van elke foutenboom. Voorbeelden van missies zijn:

- De veilige sluiting van een waterkering bij een stormconditie
- Het veilig routeren van een trein over een specifiek stuk spoor
- Het veilig routeren van een vliegtuig
- Het veilig besturen van een vliegtuig
- Het veilig besturen van een schip

Veilig is een domeinspecifiek iets wat in elke context bepaald moet worden. In treinverkeersleiding is het bijvoorbeeld geaccepteerd dat een trein stilgezet wordt, in luchtverkeersleiding is het stil moeten zetten van een vliegtuig in de lucht de meest onveilige situatie.

In theorie kan de huidige stand der techniek een foutenboom creëren tot op het niveau van de individuele beslispunten in de logica van de software (i.e. tot het niveau van if-statements en lussen toe). Dit levert echter een zeer uitgebreide boomstructuur op waarbij men al snel het overzicht verliest en waar de grote hoeveelheid van moeilijk af te schatten elementen de kwantificering van het softwarefalen ten aanzien van een specifieke missie bepaalt: de faalkans van een individueel if-statement is immers zeer moeilijk te bepalen.

Men kan ook veel abstracter te werk gaan. Gaat men op een te abstract niveau zitten dan komt men uit op een betekenisloos object wat teveel omvat om een zinnige schatting te maken. Men zal dus, net als in de foutenboom van fysieke objecten, een zinnig abstractieniveau moeten kiezen dat zinnig is ten aanzien van het doel van de foutenboomanalyse (voor meer informatie zie §4.1.2 "Abstractieniveau van softwaremodule").

De expertgroep schat dan ook in dat het verstandig is om op een gebalanceerd abstractieniveau de software module te definiëren door de taken die software moet uitvoeren als basis te nemen. In dit document wordt een software module als volgt gedefinieerd:

> Een **Softwaremodule** is een taak of een consistente verzameling taken die door een afgebakende verzameling logische regels programmacode (of grafisch equivalent) wordt gerealiseerd, waarbij geldt dat:
> - Een duidelijke afbakening te onderkennen is ten opzichte van andere stukken code én er een onderkende functionele doelstelling in het totaal van het systeem aanwezig is;
> - Te verifiëren kwaliteitseigenschappen en taakuitvoering aanwezig zijn.

De verzameling logische regels wordt in zeer brede zin beschouwd:

- een softwaremodule kan een specifieke applicatie zijn

- een softwaremodule kan alle software op een dedicated hardware component (PLC) zijn,
- een softwaremodule kan een proces of executable op een mainframe zijn
- een softwaremodule kan een collectie samenwerkende processen in een systeem zijn.

Kern is dat een softwaremodule onderscheidbaar is en enige mate van onafhankelijkheid (in zijn faalgedrag) kent.

Binnen de softwaremissie, die wordt vastgesteld op het geheel van alle hardware en softwaremodulen, heeft elke softwaremodule zijn eigen taak. In de definitie van softwaremodule speelt het begrip taak, als onderdeel van een missie, dan ook een belangrijke rol.

> **Taken** zijn uitvoeringshandelingen van software die waar te nemen zijn en waarvan men vast kan stellen dat deze goed of fout zijn ten opzichte van het bereiken van bepaalde hogere orde doelen.

Voorbeelden taken van de Maeslantkering zijn:
- INW: het inwinnen van waterstanden
- PSI: het nemen van de beslissing tot sluiting
- BES: het geven van stuurcommando's naar de werktuigen

> **Falen** is de (te lange) afwezigheid van gewenst gedrag, of het uitvoeren van verkeerd gedrag, van een softwaremodule dat noodzakelijk is om zijn taak binnen de missie te volbrengen.

Expliciet valt hieronder:
1. Geen gedrag vertonen
2. Het verkeerde gedrag vertonen
3. Gedrag niet op het juiste tijdstip vertonen

De definitie van falen vermijdt expliciet de vraag waar het falen (in technische zin) door wordt veroorzaakt en waar de "schuldvraag" ligt. De "schuldvraag" betreft de vraag of de component binnen of buiten zijn gespecificeerde gebruiksdomein is gebruikt. In praktijk wordt deze vraag overigens wel geadresseerd door verdere modellering in de foutenboom, zie §4.1.3 "Faalmodi van software".

Kern is dat voor het realiseren van de missie een component ongewenst (en onverwacht) gedrag vertoont; het maakt wat dat betreft niet uit of dit gedrag wordt veroorzaakt door een interne logische fout of dat het komt door een deadlock. Pas als het falen van een softwaremodule volledig toe te wijzen is aan het uitblijven van triggers uit andere systemen, mag dat falen worden aangerekend aan die andere systemen. De betreffende, laatst sturende component, faalt dan niet. In die zin wordt er sterk de nadruk gelegd op fouttolerantie en een defensieve programmeerstijl. Voor elke component moet gelden dat ook het uitblijven van signalen van aansturende modules tot veilig gedrag moet leiden ten aanzien van de overall missie.

Nu het falen van een softwaremodule is benoemd, kan men ook de faalkans van een software module benoemen:

> De **Faalkans** is kans dat een software module faalt op basis van een vraag tot een gewenste taakuitvoering

Nogmaals wordt hier expliciet benadrukt dat de faalkans wordt bepaald op basis van "een vraag tot gewenste taakuitvoering", hetgeen impliceert dat bij het uitblijven van de vraag het falen toegerekend moet worden aan het falen van de vraaggenererende module en als de vraag wel gesteld wordt eigenlijk per definitie al het daaropvolgende falen gerekend kan worden tot de bestudeerde softwaremodule.

4.1.2 Abstractieniveau van softwaremodule

De in §4.1.1 "Definitie softwaremodule in software intensieve systemen" beschreven definitie van een softwaremodule kan nog steeds op meerdere abstractieniveaus worden beschouwd. Tot op welk detailniveau moet een complex software-intensief systeem worden gedecomponeerd om zinnige uitspraken over de faalkans te kunnen doen?

Praktisch handvat is dat men het liefst zo abstract mogelijk de software-module definieert, waardoor men zo min mogelijk kleine softwaremodulen hoeft te beschouwen. Er is echter een bovengrens aan het niveau van abstractie: op een specifiek moment zijn parameters die de faalkans bepalen niet meer zinnig berekenbaar doordat de code ontstaan is vanuit meerdere ontwerpen of dat er meerdere teams/bedrijven verantwoordelijk zijn geweest voor de realisatie.

Belangrijk gevolg van de definitie van het begrip softwaremodule is dat deze impliceert dat een module weinig verwevenheid mag kennen met andere modulen, en dus voldoet aan het software architectuurprincipe van "Maximale cohesie en minimale koppeling". Een module die sterk verweven is met een andere module is immers moeilijk daarvan te onderscheiden. Als softwaremodulen sterke verwevenheid of wederzijdse afhankelijkheid kennen met andere softwaremodulen, dan is het te adviseren het als één enkele module te beschouwen, om zo te voorkomen dat men veel wederzijdse afhankelijkheden moet modelleren.

Keten-afhankelijkheid moet hierbij in ogenschouw worden genomen: een softwaremodule levert input voor de volgende softwaremodule. Keten-afhankelijkheid van meerdere modulen is een acceptabel onderdeel van een foutenboom, mits het niet tot sterk verweven modulen leidt: modulen die veel interactie kennen om tot één enkele beslissing te kunnen komen. Kern is dat in elke module in de keten zijn eigen duidelijk afgebakende taken en verantwoordelijkheden kent die in zijn definitie van zijn taak niet direct en volledig afhankelijk zijn van een andere module.

Een voorbeeld van de problematiek rondom abstractie zijn *SOBEK* en *de Schil Rond SOBEK (SRS)* uit de software van de Maeslantkering. Deze modulen kennen een sterke wederzijdse afhankelijkheid, waarbij in praktijk falen van *De Schil* direct consequenties heeft voor *SOBEK*. De functionaliteit

van *De Schil* is ook in sterke mate afhankelijk van *SOBEK*: de functie van de schil is om de inputvalidatie van *SOBEK* voor zijn rekening te nemen. Indien beide componenten door hetzelfde ontwikkelteam onder hetzelfde ontwikkelproces ontwikkeld zouden zijn, zouden ze als één software component beschouwd kunnen worden: het kan voor de eenvoud en overzichtelijkheid verstandiger zijn om deze modulen als één geïntegreerde module te modelleren. Let wel dat een tegenargument voor deze benadering in veel gevallen is dat parameters in de faalkansschatting sterk afwijkend kunnen zijn omdat andere ontwikkelteams of ontwikkelmethoden gebruikt zijn voor de ontwikkeling.

Een tweede voorbeeld zijn de redundante *INWIN*-eenheden van de Oosterscheldekering. Deze eenheden zijn identiek en krijgen dezelfde input. Deze modulen kennen een zeer intensieve relatie met elkaar: ze communiceren voortdurend met elkaar om te voorkomen dat ze aansturingsfouten introduceren of door race-conditions in verschillende toestanden eindigen. Het is vanuit de faalkansanalyse het eenvoudigst om deze softwaremodulen als één enkel systeem te beschouwen: de intensiteit van de communicatie zou het waarschijnlijk te complex maken om het degelijk te kunnen analyseren.

4.1.3 Faalmodi van software

De bepaling van de faalmodi is een cruciaal onderdeel van de analyse om te komen tot een faalkans. Na een globale analyse is alleen bekend welke softwaremodulen een bedreiging van de missie kunnen opleveren. Maar de grote vraag is welke vorm van falen (faalmodus) precies het vertonen van het gewenste gedrag op het juiste moment bedreigt.

Bij de modellering van faalmodi van software kan men verschillende benaderingswijzen kiezen waarbij, net als in de mechanische wereld, vaak de compleetheid van de te onderscheiden faalmodi en de mogelijkheid tot berekening leidend is voor de keuze van de taxonomie. Bekende taxonomieën van de wijzen waarop software kan falen, zijn:

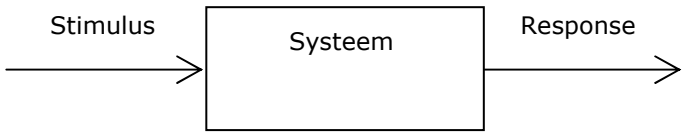
- Bug taxonomy van Beizer [2];
- Common bugs door Vinideep [24];
- Dependability Threats door Avizienis [1].

Deze taxonomieën werken van binnen naar buiten: ze gaan in op de vraag wáárom de software faalt (zoeken naar de interne bron van falen). Dit is interessant voor post-mortem analyse van software maar ongeschikt voor onze doelstelling omdat de kwantitatieve aanwezigheid van deze problemen bij bestaande software niet te achterhalen zijn. Ook leveren deze taxonomieën te weinig informatie op om de te verwachten impact op de betrouwbaarheid af te schatten.

In deze aanpak beschouwen we falen als een gevolg van de oorzaken die in de taxonomieën genoemd zijn, maar we bekommeren ons niet om de nog dieper liggende oorzaken hiervan. Deze benadering is gelijk aan de benadering voor menselijk falen: hoewel er veel onderzoek gedaan wordt naar neurologische grondoorzaken van menselijk falen, zijn deze zeer moeilijk voorspelbaar en meetbaar. Kijkt men echter naar psychologische

randvoorwaarden die de verwerking van het stimulus-response-systeem zouden kunnen beïnvloeden, dan wordt de schatting van de faalkans een stuk eenvoudiger. Hierdoor beschouwen we een softwaremodule als een stimulus-response-systeem waarbij falen voort kan komen uit onjuiste inputs of een onjuiste verwerking van correcte inputs. De niet goed te observeren dieper liggende oorzaken van deze foutieve verwerking doen er dan niet meer toe.

Als achtergrond bij het doorgronden van faalmodi, bekijken we het transformatie-probleem van input naar gewenste taakuitvoering (het waarnemen van signalen, het analyseren van die informatie, het vervolgens nemen van een beslissing en het geven van specifieke signalen). Een concept dat men ook ziet in de studie naar menselijk falen, waar men de mens modelleert als een organisch stimulus-response-systeem:

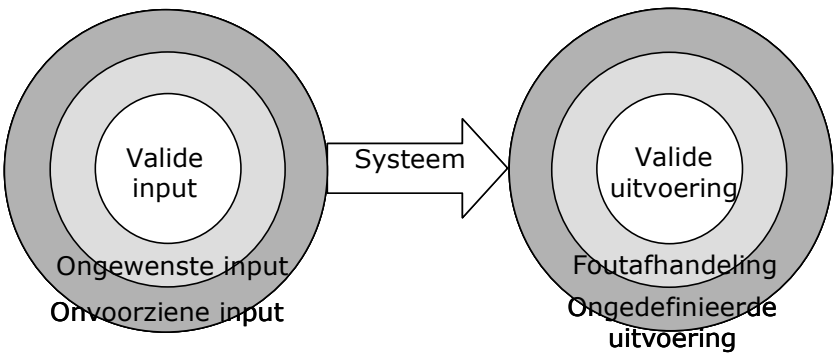


Figuur 1: Stimulus response systeem

Als men de totale verzamelingen van inputs beschouwt, is het dus de taak van de specifieke softwaremodule deze te transformeren naar taakuitvoering, waarbij de volgende zaken cruciaal zijn:

- **valide invoer**, welke bij correcte implementatie afgebeeld wordt op valide besturingsgedrag, hetgeen tijdens testen van de softwaremodule component dient te worden geverifieerd
- **voorziene ongewenste invoer**, welke als het goed is, afgebeeld wordt op foutafhandeling en/of gewenst faalgedrag. Iets dat bij kritieke systemen ook expliciet wordt geverifieerd tijdens testen
- **Onvoorziene vormen van input**, welke zullen leiden tot ongedefinieerd gedrag van het systeem.

Systematisch levert dit het volgende overzicht, waarbij het systeem wordt gemodelleerd als transformator van verzameling inputs op outputs:



Figuur 2: Systeem als transformatie input naar output

Er wordt hier expliciet opgemerkt dat specifieke ontwikkelstrategieën verschuivingen in de omvang van deze verzamelingen teweeg kunnen brengen: door betere omgevingsanalyses of het gebruik van defensief programmeren, Dijkstra [7] kan men de fractie van "onvoorziene input" verkleinen door de fractie "ongewenste input" te vergroten: het systeem kent een grotere ontwerptolerantie ten aanzien van ongewenste input waardoor deze robuuster wordt tegen foutieve data uit zijn omgeving. Consensus in de expertgroep is dat aan een defensieve strategie wel nadelen kleven: als men defensief programmeren tè ver doorzet dan kan de applicatie door zijn toegenomen omvang en complexiteit als geheel wel eens onbetrouwbaarder worden.

Het gecombineerde domein van "valide input" en "voorziene ongewenste input" vormt het geaccepteerde inputbereik voor een softwaremodule vanuit zijn specificatie: dit is zijn ontwerptolerantie. Dit bereik is normalerwijs onderdeel van de specificaties. Het domein van "onvoorziene vormen van input" is per definitie geen onderdeel van de specificaties (die is het complement daarvan ten opzichte van het complete inputdomein).

Bij de faalkans modellering van een softwaremodule dient onderscheid gemaakt te worden in twee faalvormen:

1. **Falen ten gevolge van onvoorziene invoer**
   Het gaat om het gebruik van de softwaremodule buiten zijn ontwerptoleranties, wat per definitie ongedefinieerde taakuitvoering van de softwaremodule tot gevolg heeft. Het gebruik van een component buiten zijn specificaties kan komen door een onjuist/onvolledig vaststellen van de requirements en/of het onjuist/onvolledig uitvoeren van een omgevingsanalyse. Ook kan het voortkomen uit verstoringen in omgevingssystemen, wat tot onverwachte invoer leidt. Opgemerkt wordt dat in praktijk het lastig is om de omgeving volledig te modelleren bij een softwaremodule met veel inputs: de analyse is in dat geval dus altijd minder volledig en een robuustheidsstrategie is hier bepalend.
2. **Falen ten gevolge van een fout in de beslislogica (het intrinsieke falen van de component)**
   Het gaat hier om alle foute handelingen die gevolg zijn van verwachte (valide en invalide) invoer die óf het verkeerde gedrag tot gevolg heeft óf het juiste gedrag op het verkeerde moment oplevert. Het gaat hier dus om fouten binnen de ontwerptoleranties van de softwaremodule, waarbij een transformatie van de invoer zich dus binnen het domein van de valide/invalide invoer bevindt, maar wat niet resulteert in het gewenste gedrag om de missie te volbrengen. Zoals al is gemeld in de definitie kan het hier zowel gaan om het te lang uitblijven van gedrag of om het vertonen van het verkeerde gedrag.

In de hierop volgende alinea's worden deze punten verder uitgewerkt.

Falen ten gevolge van een fout in de beslislogica
Het falen ten gevolge van een fout in de beslislogica houdt in dat de software niet conform de specificatie is gemaakt. Dit kan worden veroorzaakt door mensen die, bij het maken van software, ontwerp en/of programmeerfouten maken welke vervolgens tijdens testen en gebruik verborgen zijn gebleven. Het kwantificeren van falen ten gevolge van een fout in de beslislogica zal in hoofdstuk 5 verder uitgewerkt worden.

Falen ten gevolge van onvoorziene input
Het falen ten gevolge van onvoorziene input is het gevolg van een mismatch tussen de invoer waar de softwaremodule voor ontworpen is en de werkelijke invoer die de omgeving genereert. Het gaat dan om een onvoorziene input waar in de specificatie van software (bewust of onbewust) geen rekening mee is gehouden. Deze faalconditie heeft meerdere potentiële oorzaken:

- Het kan komen omdat de omgeving van de module letterlijk uit zijn specificatie loopt. Een voorbeeld is doordat voorziene stormcondities te conservatief zijn ingeschat of doordat omliggende softwaremodules falen en dus ongespecificeerde output gaan genereren.
- Dit kan een ontwerpfout zijn van de component of zijn omgeving die ertoe leidt dat delen van het werkdomein van de applicatie niet in de specificatie van de applicatie zijn vastgelegd.
- De fout kan ook komen doordat de implementatie van de specificaties complexe afhankelijkheden kent. Een voorbeeld hiervan is dat de specificatie afhankelijk is van aannames over de hardware zoals timing die afhankelijk is van scheduling en responstijden van drivers.

Het kan een bewuste ontwerpkeuze zijn om specifiek “omgevingsfalen” niet te ondervangen in een softwaremodule. Één van de belangrijkste overwegingen kan zijn dat het de complexiteit van de component (en dus de kans van foutief handelen) vergroot. Het kan ook een bewust onderdeel van de overall faalstrategie van een volledig systeem zijn: sommige componenten mogen in kritieke missies geen faalgedrag gaan vertonen, ongeacht of ze hun deeltaak nog kunnen uitvoeren, omdat bovenliggende systemen dat op andere wijze monitoren. Praktisch voorbeeld hiervan is de aansturing van de straalmotor van een vliegtuig: de aansturing zal altijd de motor door laten gaan, ongeacht de potentiële fysieke schade, totdat een piloot besluit de motor handmatig uit te zetten.

Consensus binnen de expertgroep is dat men eigenlijk alleen bij softwaremodulen met een externe interface echt kan spreken van onvoorziene input. Bij systemen met louter interfaces binnen het systeem zou onvoorziene input alleen het gevolg kunnen zijn van falen van een voorliggende component in de keten, waardoor deze component ongespecificeerde output/gedrag gaat vertonen, waardoor dus eigenlijk het falen op het conto van de voorliggende component moet worden gerekend.

4.2 Bouwtekeningen van systemen bestaande uit software

Als men een analyse van software wil maken, moet men een set “bouwtekeningen” hebben om als basis te dienen voor de ontwikkeling 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 softwaremodulen die bij elke missie betrokken zijn. Belangrijke vragen zijn bijvoorbeeld of er single points of failure zijn en welke kritieke ketens van softwaremodulen het gedrag van de kering bepalen.

De bouwtekeningen moeten dus inzage geven in hoe een missie is opgebouwd uit taken die door componenten worden uitgevoerd. Door dit samen te nemen kan men de stap naar een foutenboom maken. In deze foutenboom is de specifieke falende taakuitvoering van de component erg belangrijk: dat wordt de basisgebeurtenis.

Maar ook moet deze bouwtekeningen inzage geven relaties tussen faalmechanismen. Falen kan, bijvoorbeeld, gerelateerd zijn door gebruik van libraries en andere gemeenschappelijke bouwstenen, maar er kan ook redundantie of een watchdog geïmplementeerd zijn welke aanleiding geeft tot alternatieve successcenario's. Deze zaken moeten herkenbaar zijn in de bouwtekeningen.

Een analyse van de architectuur behoort dus antwoord te kunnen geven op de volgende vragen die in vervolgstappen gesteld gaan worden:

1. Welke softwaremodulen onderscheiden/identificeren we?
2. Hoe is een missie softwarematig uit deze softwaremodulen opgebouwd, hoe werken de componenten samen om het beoogde resultaat te bereiken (hoe communiceren ze onderling, zijn er parallelle/redundante processen, zijn er watchdogs, zijn er on-demand processen)?
3. Van welke hardware is een softwaremodule afhankelijk?
4. Waarvan is een softwaremodule in termen van hardware (CPU/memory/disk) afhankelijk, en met welke andere processen deelt hij deze?
5. Welke libraries gebruikt een module?

In de ICT-sector groeit consensus over de beste manier van systeemontwerpen. In de IEEE 1471 [12] zijn de randvoorwaarden voor systeemontwerpen vastgelegd. De meest gebruikte werkwijze van ontwerpen die aan de IEEE 1471 eisen invulling geeft is de 4+1 methode van Kruchten [13]. Deze methode is ook in gebruik in het analyseren van luchtverkeersleidingsystemen ten behoeve van FAA certificatie.

Een software architectuur volgens 4+1 bevat de volgende views:

- **Logische 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 en houdt zich bezig met 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 communicatie, synchronisatie, redundantie en concurrency-control
- **Physical view**, die een beschrijving geeft van de hardware-infrastructuur (PLC's, servers en netwerken) en de allocatie van de run-time onderdelen daarop, vanuit het perspectief van de system engineer. Hier moeten dus alle individuele hardware componenten genoemd zijn die ook in de foutenboom worden onderscheiden
- **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 componenten. 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.

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

4.3 Van bouwtekening naar de structuur van de foutenboom

Als de "bouwtekening" van de software, die de samenhang tussen alle softwaremodulen beschrijft, bekend is, dient men gaan onderzoeken welke systemen betrokken zijn bij de missie. Kernvraag van dit onderdeel is dan ook welke sofware op welk moment van belang is in verband met het volbrengen van de te onderzoeken functie of set van functies. Op basis van een Failure Mode en Effect Analyse van het totale systeem wordt nagegaan welke modulen mogelijk bijdragen aan de faalkans van de ongewenste topgebeurtenis.

4.3.1 Primaire structuur foutenboom

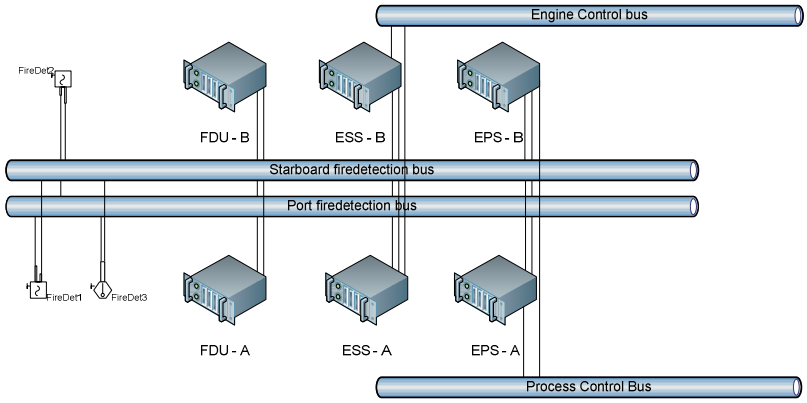
De expertgroep staat voor dat de foutenboom wordt opgebouwd vanuit het functionele perspectief (wie beslist wat wanneer in welke volgorde) en niet vanuit de componentgedreven (welke componenten zijn er) perspectief. Dit impliceert dat men modelleert vanuit de gedachte dat de taakuitvoering van software in de foutenboom wordt beschreven. Daarbij wordt zowel het falen van de beslistaak van de software als het falen van de uitvoering van de stuurcommando's naar specifieke hardware (kleppen, pompen, etc.) volgend op die beslissing in de foutenboom beschouwd.

Dit impliceert dat men bij het modelleren begint bij het modelleren van het falen van het nemen van het besluit (bij de Maeslantkering bijvoorbeeld het falen van het *Beslis Ondersteunend Systeem* om het sluitcommando te genereren). Het falen van de uitvoering van de stuurcommando's (bij de Maeslantkering bijvoorbeeld het falen van hardware als kleppen) zou men pas in een latere instantie moeten beschouwen. Zo benadrukt men sterker

dat software het proces drijft in een specifieke volgorde, en dat hardware ondergeschikt is aan de besluitvorming.

De primaire structuur van de foutenboom heeft zijn basis in de beschreven relevante scenario's van de software die de missie beschrijven. Consensus binnen de expertgroep is dat elk potentieel uitblijven van een taak die een softwaremodule uitvoert binnen een missie (elke beslissing/stuursignaal van een softwaremodule in een scenario) gezien moet worden als een gebeurtenis in de faalkansanalyse (zie ook Wang et al. [25]). Deze aanpak wordt verder gedetailleerd toegelicht in de hierop volgende alinea's.

voorbeeld: brandmeldinstallatie
Initieel kan men kijken naar de hardwarestructuur (physical view) van het ontwerp, zoals deze hardwarestructuur van een brandmeldinstallatie:



Figuur 3: Physical view van een brandmeldinstallatie

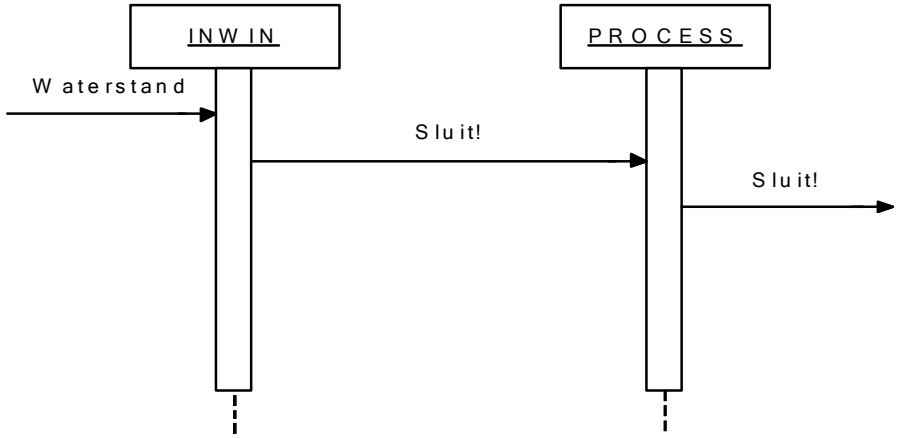
Deze hardwarestructuur laat alle connecties zien tussen hardwarecomponenten (waar softwaremodules op aanwezig zijn) en de netwerken die beschikbaar zijn.

Echter, zoals deze hardwarestructuur al laat zien, is het onduidelijk hoe al deze componenten van een brandmeldinstallatie samenwerken om tot een beslissing te komen. Om hiertoe te komen moet men inzage hebben in hoe de beslissing tot stand komt. Om tot een verantwoorde modelering te komen, is het noodzakelijk gedetailleerd te beschrijven hoe de modules samenwerken. Deze modules moeten uiteraard voldoen aan de in §4.1.1 "Definitie softwaremodule in software intensieve systemen" gestelde definitie van een softwaremodule.

Als men het systeem modelleert in een scenario, met de benodigde interacties tussen softwaremodulen om tot taakuitvoering te komen, dan is de modelering in de foutenboom een relatief kleine stap. Het genereren van het gewenste gedrag is te beschouwen als een taak van de softwaremodule en het uitblijven van het gedrag is dan als basisgebeurtenis te beschouwen in de foutenboom. Elk uitblijven van gewenste taakuitvoering kan, zoals in §4.1.3 "Faalmodi van software" beschreven is, opgedeeld worden in een

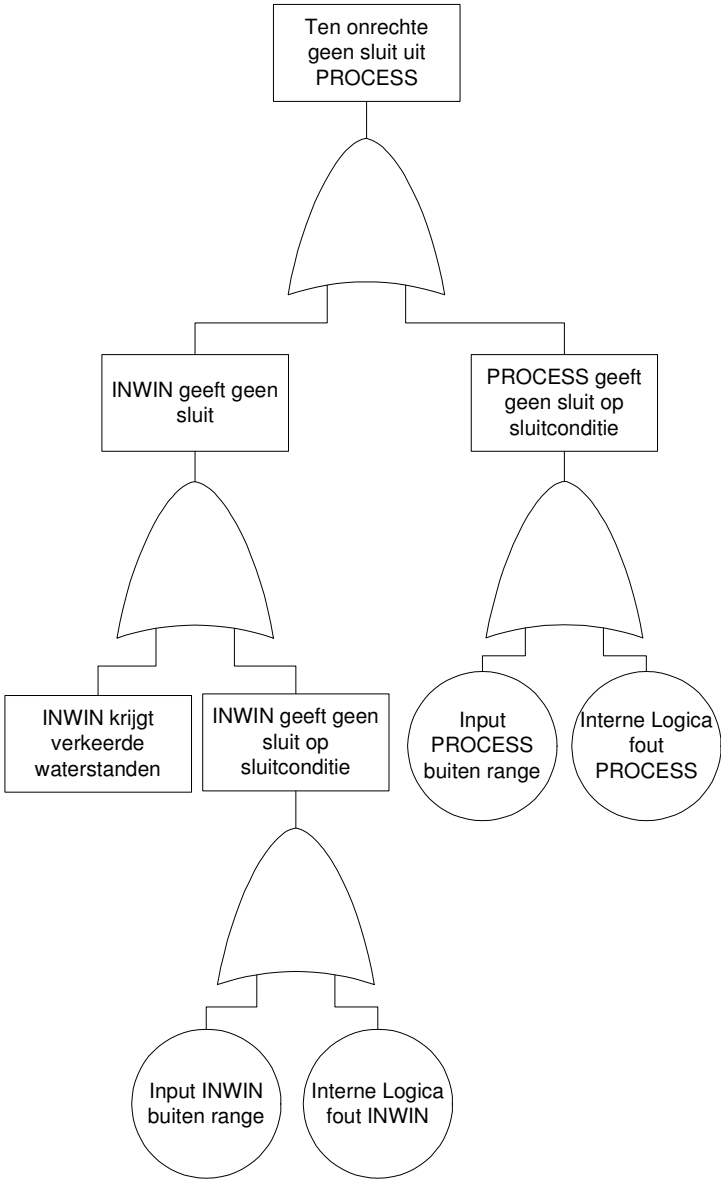
basisgebeurtenis “onverwachte input” en een basisgebeurtenis “falen van beslislogica”.

Zoals gezegd wordt in de praktijk elke gewenste actie van een softwaremodule uit het scenario een basisgebeurtenis in de foutenboom, waarbij deze basisgebeurtenis vervolgens wordt uitgesplitst naar een basisgebeurtenis “onverwachte invoer” en een basisgebeurtenis “fout beslislogica”. Bijvoorbeeld het onderstaande voorbeeld (abstracte versie van het noodsluitsysteem van de Oosterscheldekering):



Figuur 4: Noodsluitsysteem Oosterscheldekering

In dit voorbeeld zijn drie acties gedefinieerd, waarbij de laatste actie (*Sluit!*, gegeven door *Process*) eigenlijk het doel van de missie is. Als men dit modelleert in een foutenboom dan krijgt men:



Figuur 5: Foutenboom noodsluitsysteem Oosterscheldekering

Hier ziet men dat het uitblijven van elke handeling leidt tot een basisgebeurtenis in de foutenboom. Zoals ook blijkt uit dit eenvoudige voorbeeld, ontstaan er 2 basisgebeurtenissen: één voor het buiten de specificatie gebruiken van de softwaremodule en één voor het falen van de beslislogica.

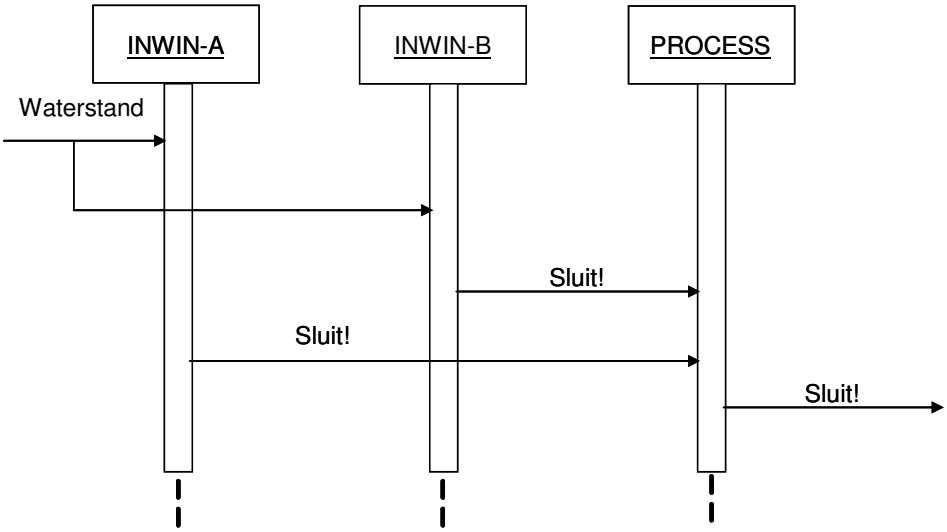
4.3.2 Alternatieve scenario's

In veel gevallen zijn er alternatieve scenario's om hetzelfde te bereiken: denk aan redundantie of watchdogs die falen ondervangen en alternatieve successcenario's introduceren. Deze alternatieve scenario's om de missie te laten slagen moeten expliciet worden beschreven en worden verwerkt in de foutenboom.

Meest praktische oplossing is het laten aangrijpen van dergelijke redundantie op de plek waar de beslissing wordt genomen. Er is immers

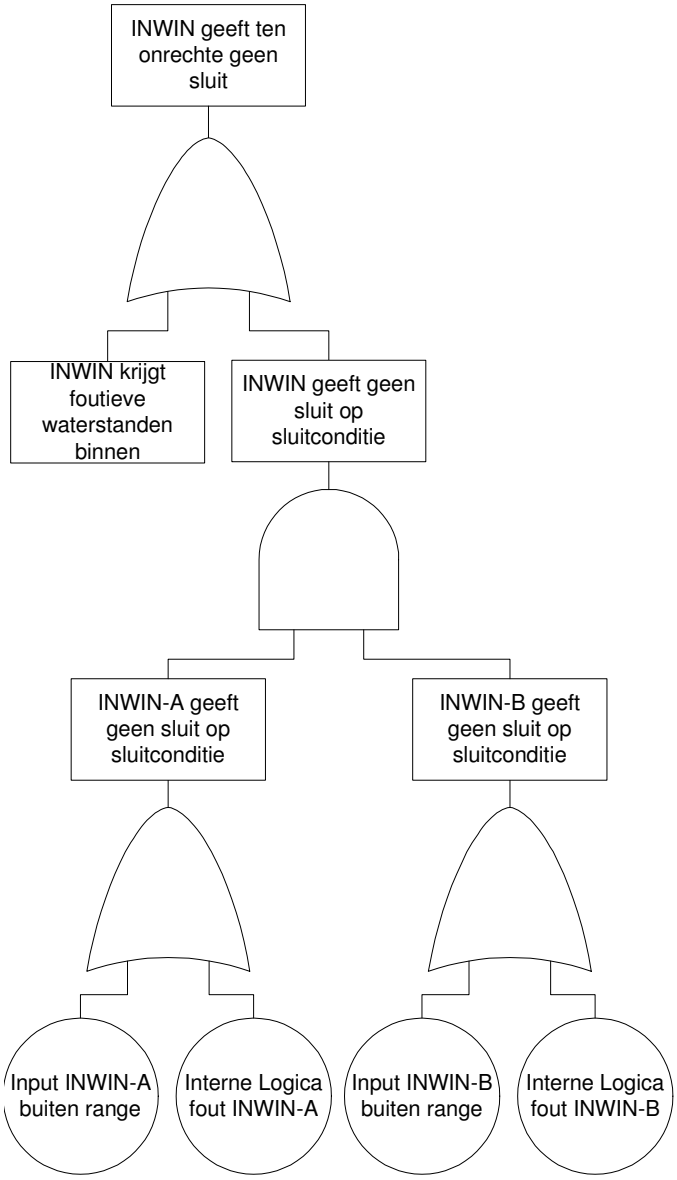
altijd een punt waar het redundante (alternatieve) scenario en het reguliere scenario bij elkaar komen in de besluitvorming. Dat wil zeggen waar in de niet-redundante situatie een gebeurtenis "geen signaal van softwaremodule X" heeft, krijgt deze nu een AND-gate met "geen signaal van softwaremodule X1" en "geen signaal van softwaremodule X2". In dergelijke gevallen zal er dus een AND-gate ontstaan die modelleert dat beide aanleverende systemen moeten falen voordat de input stokt.

Als men kijkt naar de redundante oplossing van het hiervoor gebruikte voorbeeld, dan krijgt men dit alternatieve scenario voor het genereren van het sluit-commando op basis van een 2e (redundante) *INWIN*-eenheid.



Figuur 6: Redundante uitvoering noodsluitsysteem Oosterscheldekering

Het redundante systeem introduceert dus een alternatieve manier voor *Process* om het *Sluit!* commando te ontvangen. Als men dan de basisgebeurtenis "*INWIN* geeft geen sluit" beschouwt, ontstaan er meerdere wegen om *Process* te triggeren om een besluit te nemen. Hierdoor verandert dit deel in de foutenboom als volgt.

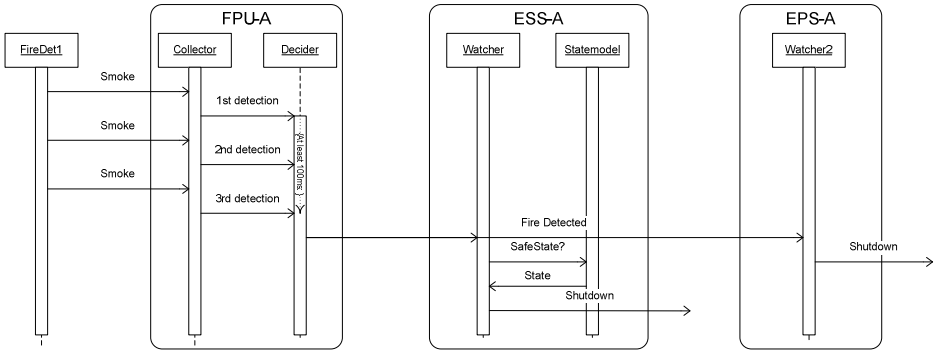


Figuur 7 Foutenboom redundante uitvoering noodsluitsysteem Oosterscheldekering

Hier ziet men dan ook dat er nu 2 systemen geen trigger moeten afgeven voordat de faalconditie "*INWIN* geeft ten onrechte geen sluit" realiteit wordt. Hierbij komt *INWIN-A* uit het primaire scenario, en *INWIN-B* uit het alternatieve scenario.

Uiteraard kennen *INWIN-A* en *INWIN-B* een common cause failure component. Voor de modellering daarvan wordt verwezen naar §4.3.1 "Primaire structuur foutenboom" onder het kopje "Common cause failures".

Het systeem komt tot verder leven als ook de scenario's helder worden in combinatie met andere opdelingen. Bijvoorbeeld zijn in het onderstaande voorbeeld van de brandblusinstallatie de kaders om processen getekend om aan te geven op welke hardware deze aanwezig is.



Figuur 8: Processen brandblusinstallatie, afgebeeld op de hardware

In dit scenario zijn alle acties tussen softwaremodulen gemodelleerd, alsmede alle taken die ze vervullen binnen een missie. Hierdoor ziet men expliciet de koppeling naar de eerdere fysieke bouwtekeningen: de softwaremodulen zijn "gevangen" in een box die de groeperingen van de processen naar fysieke hardware modelleert.

Dit specifieke voorbeeld heeft overigens last van deadlock: *Watcher* in *ESS-A* kent geen time-out voor uitblijven status. Het feit dat er deadlock ontstaat is in dit geval voor de faalkans eigenlijk niet van belang. Kern is dat in dit geval een signaal uitblijft en er geen repair-gedrag (verlopen van een timer op de *Watcher*) is.

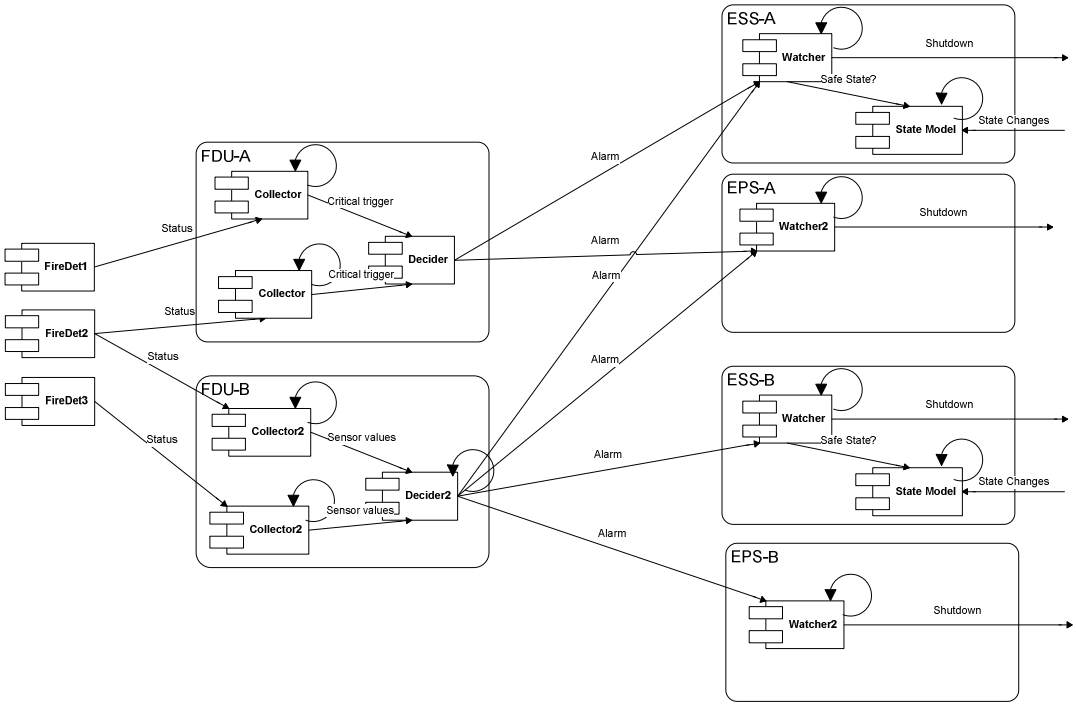
4.3.3 Onmerkbaar falen

Onmerkbaar falen is een potentieel probleem met software. In veel gevallen gaat het om het afsterven of niet kunnen opstarten van processen waarbij gedurende niet-kritische perioden niet gecontroleerd wordt of de processen nog (kunnen) lopen. Men kan hierbij denken aan on-demand processen of applicaties op een mainframe, waarbij niet gemonitord wordt of het proces nog leeft en ontvankelijk is voor input.

De precieze modellering hangt sterk af van details in het ontwerp. Het gaat dan steeds om on-demand processen waarbij bijvoorbeeld geen tussentijdse statusevaluaties plaatsvinden of anderszins geen interne checks worden uitgevoerd.

Een andere vorm van ongemerkt falen is een module die continu of met grote frequentie zijn taak uitvoert, maar waar de taak afhankelijk is van eerdere runs. Denk hierbij aan het doorrekenen van meteorologische systemen (globale drukvelden). Zonder kalibratie en bijsturing verlopen de resultaten zodanig dat na een aantal iteraties de resultaten onbruikbaar worden.

voorbeeld: brandmeldinstallatie
Hieronder wordt een voorbeeld getoond van een process view met enkele on-demand processen.



Figuur 9: Process view brandmeldinstallatie

In het bovenstaande blijkt dat de *Decider* in *FDU-A* een on-demand process (ontbreken van het lusje op het proces) is, wat dus gevoelig is voor onmerkbaar falen ten gevolge van bijvoorbeeld memory-shortages. Andere processen zijn continue processen die regelmatig een status rapporteren, en daarmee zou dus het afsterven van een proces gedetecteerd kunnen worden door opvolgende processen.

Het ontwerp van *Decider2* in *FDU-B* is ontworpen als een continu lopend proces dat een eventueel on-demand vraag kan afhandelen. De kans op onmerkbaar falen wordt verkleind omdat het tussentijds de status rapporteert.

4.3.4 Common cause failures

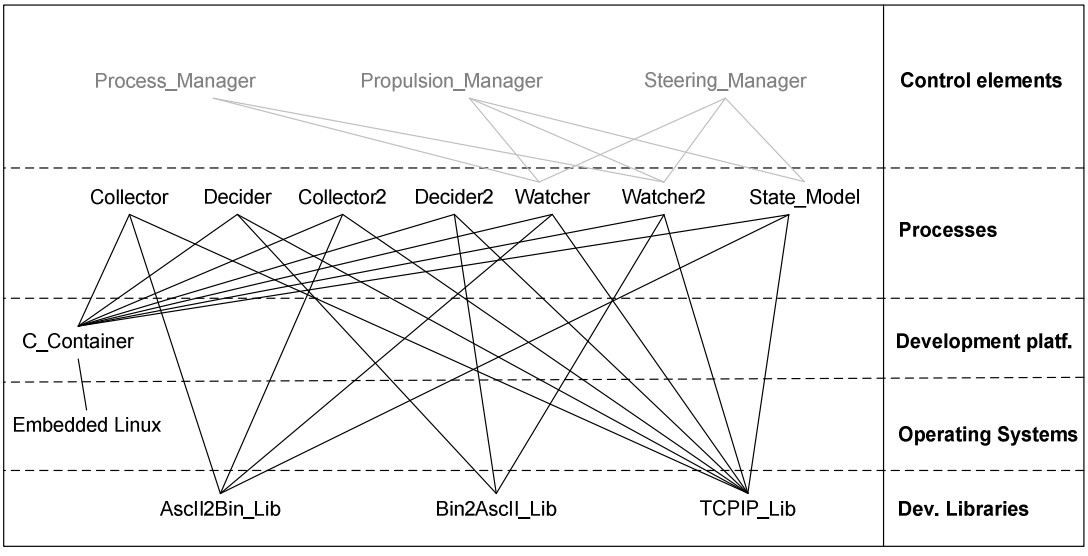
Er zijn in software common cause failures te herkennen. Dit houdt in dat twee of meer softwaremodules falen door een gemeenschappelijke faalmodus.

In tegenstelling tot de fysieke wereld is de common cause factor bij redundant uitgevoerde software uitzonderlijk groot. Door het strikt deterministisch gedrag van software is er immers vooral sprake van systematisch falen: modules met dezelfde beslislogica zullen gelijktijdig en op identieke wijze falen als ze dezelfde input in dezelfde volgorde

aangeboden krijgen. Dubbel uitvoeren van identieke componenten is in praktijk een veel geïmplementeerde vorm van redundantie maar deze vorm van eenvoudige redundantie beschermt in de praktijk alleen tegen de random fouten in de onderliggende hardware (denk hierbij aan een corruptie van een bit in het geheugen of de processor) en niet tegen fouten in de beslislogica. Hardware-fouten worden in het algemeen als hardwarefalen expliciet gemodelleerd in de foutenboom. Dit is een aanname die algemeen geldt in foutenbomen.

De enige wijze om te beschermen tegen common cause failure van redundant uitgevoerde modules is het toepassen van "Multiprogramming": deze aanpak gaat uit van het idee dat software voor verschillende softwaremodules volledig onafhankelijk worden ontwikkeld door verschillende teams/bedrijven en draaien op eigen processoren. Uitgaande van de gescheiden aanpak zal de logica achter de beslissing niet of nauwelijks common cause failures kennen (uitgezonderd specificatiefouten). Echter, er kunnen afhankelijkheden op een lager technisch niveau liggen, bijvoorbeeld door het ongemerkt gebruiken van dezelfde libraries door de verschillende ontwikkelteams. Inzage in de gebruikte libraries, om zo tot een analyse te komen van de gedeelde afhankelijkheden, wordt verkregen door de development view.

Een voorbeeld is hieronder beschreven:



Figuur 10: Afhankelijkheden in de software door libraries

In dit voorbeeld delen redundante processen allen dezelfde libraries. Niet alleen de *C-Container* van *Linux* en de *TCPIP-lib* die door iedereen gebruikt wordt, maar ook *Ascii2Bin*, *Bin2Ascii* wordt door iedereen gebruikt. Dit zijn dus onderliggende common cause failures: als bijvoorbeeld een *TCPIP-library* faalt op een specifiek bericht, zal alle communicatie falen voor alle redundante componenten.

5 Schatting van faalkans taakuitvoering

Uit het vorige hoofdstuk is gebleken dat de vraagstelling zich toespitst op bepaling van de kans dat, gegeven een bepaald scenario met valide inputranges, de softwaremodule de verkeerde taakuitvoering heeft, of dat deze taakuitvoering uitblijft. Het gaat hier dus niet alleen om het hangen of crashen van de applicatie, maar zeker ook het nemen van een verkeerde beslissing of dat deze te lang op zich laat wachten.

Doel voor dit hoofdstuk is het beschrijven van de methode om te komen tot een faalkans schatting van de beslislogica van de softwaremodule bij een specifieke taakuitvoering als deze verwachte invoer krijgt. Het gaat bij software falen om situaties waarin de softwaremodule tijdens de missie op een bepaalde input anders reageert dan beoogt. Dit zal niet met toeval te maken hebben maar eerder met een input binnen de ontwerptolerantie waarbij de software deze input systematisch transformeert tot een verkeerde of te late handeling.

We moeten benadrukken dat de faalkans betrekking heeft op een specifieke taakuitvoering van een component, en niet op een component als geheel. Dit betekent dus dat als een component meerdere taken heeft deze allen gescheiden geanalyseerd moeten worden. Door een andere opbouw (meer omvang en complexiteit) en een ander gebruiksinterval moet men aannemen dat de software ook een andere betrouwbaarheid heeft bij specifieke taakuitvoeringen. Dit is het meest evident bij primaire besturingen die intern ook failover gedrag implementeren: de primaire operatie wordt vaak zeer frequent gebruikt en de herstelfunctie zeer zelden. Dit zal ook in de faalkans gereflecteerd moeten worden.

De toepassing van dit deel van de TOPAAS methode is, zoals al gemeld in de conclusie van §2.4 "Bekende methoden" van belang bij gebrek aan betrouwbaardere methoden van meting. Kenmerkend is dat er niet alle informatie aanwezig is, en dat bestaande methoden niet betrouwbaar kunnen worden toegepast door de afwezigheid van informatie of de praktische haalbaarheid van het uitvoeren van simulaties.

Het schatten van de faalkans is, zoals eerder benoemd in §4.1.3 "Faalmodi van software" kopje "Falen ten gevolge van een fout in de beslislogica", gebaseerd op een Bayesiaanse benadering van de faalkans van de specifieke taakuitvoering van de softwaremodule.

De schatting van de faalkans is gebaseerd op expert-opinion die vervat is in een parametermodel. Dit model voor de kwantificering van de faalkans van software is te vergelijken met het OPSCHEP model voor menselijke fouten: gebaseerd op specifieke parameters wordt een correctie op de basisfaalkans gegeven.

In dit hoofdstuk beschrijven we allereerst de scope van de methode, gevolgd door een beschrijving van de werking van de methode op

hoofdlijnen. Vervolgens wordt de specifieke invulling besproken in termen van factoren en dimensies.

5.1 Scope

De bepaling van de faalkans van software is een veld waarin veel wetenschappelijke discussie plaatsvindt over de geldigheid van methoden: er is niet één enkel algemeen geaccepteerd model dat niet fundamenteel ter discussie staat. Dit komt niet alleen omdat er weinig consensus is in de wereld over hoe deze op betrouwbare wijze gemeten wordt. Ook zijn er verschillende soorten faalkansbepaling bekend (zie bijvoorbeeld Smidts [22]), met elk hun eigen doelgebieden, waar nog geen harmonisatie heeft plaatsgevonden. Er zijn drie orthogonale dimensies te onderscheiden in de methoden van faalkansbepaling:

1. **Faalkansbepaling gebaseerd op product prestatie**, waarbij het faalverleden van de component en de gebruiks/testcoverage gebruikt wordt om tot een faalkans te komen al of niet in combinatie met andere metrieken en Monte Carlo simulatie (Reliability Growth Models). Zie Brandt [4] en [5];
2. **Faalkansbepaling gebaseerd op intrinsieke interne eigenschappen van het product**, waarbij de omvang, de kwaliteit en complexiteit van de code als basis wordt gebruikt om een schatting van de faalkans te maken;
3. **Faalkansbepaling gebaseerd op de correlatie met specifieke proces maatregelen (bijvoorbeeld CENELEC en IEC [11])**, hierbij wordt uitgegaan van de aangenomen correlatie tussen de kwaliteit van het voortbrengingsproces (zoals het gebruik van formele methoden) en het eindproduct.

Het TOPAAS model is bedoeld om te gebruiken als andere, wetenschappelijk meer geaccepteerde, methoden niet toepasbaar zijn, bijvoorbeeld doordat reliability growth modellen onbruikbaar blijken omdat noodzakelijke randvoorwaarden niet zijn ingevuld.

5.2 Beschrijving parametermodel op hoofdlijnen

Basis van het TOPAAS model is dat het een parametermodel betreft, gelijkend op het OPSCHEP-model [10] voor menselijk falen. Het parametermodel gaat uit van een basiskans die voor alle software gelijk is.

Op basis van beschikbare informatie over de softwaremodule wordt de basiskans bijgesteld. Hierbij wordt expliciet de nadruk gelegd op het beschikbaar zijn van informatie: informatie die niet beschikbaar is hoeft niet tot scoring te leiden omdat het TOPAAS model van nature voldoende conservatief is ingesteld (zie §5.3.1 "Basisfaalkans"). Daarom wordt alleen op basis van daadwerkelijk beschikbare informatie tot scoring overgegaan. Als men een parameter niet kent, kan men dus gewoon de scoring van dat onderwerp overslaan.

Voor elk stuk relevante informatie is een schaal bepaald waarop de softwaremodule ingedeeld moet worden, resulterend in een vermenigvuldigingsfactor. De vermenigvuldigingsfactor kan groter of kleiner dan één zijn. Deze vermenigvuldigingsfactor is bepaald op basis van een

veronderstelde correlatie met de faalkans. De uiteindelijke faalkans is dan de basisfaalkans vermenigvuldigd met alle mogelijke vermenigvuldigingsfactoren die van toepassing zijn.

Dus de basisfaalkans, vermenigvuldigd met een aantal verschillende factoren, bepaalt de faalkans van een softwaremodule in een specifieke missie. Een simpele vorm van een parameter model is

$$P = P_B * F_i * F_2 * \ldots * F_n$$

Ofwel de kans (P) is bepaald uit een basisfaalkans P_B vermenigvuldigd met een reeks factoren. In het TOPAAS model wordt de basisfaalkans (P_B) op 1 gezet (zie §5.3.1 "Basisfaalkans"). Alle F_i's zijn dan vermenigvuldigingsfactoren die een veronderstelde invloed hebben op de faalkans (zie §5.4 "Invulling methode" voor een overzicht van de geïdentificeerde factoren). In §5.3.4 "Werkwijze TOPAAS schatting" wordt specifiek invulling gegeven aan de gebruikte rekenmethode voor dit parametermodel.

5.3 Specifieke invulling parametermodel

Om het parametermodel in te vullen moet er een specifieke invulling voor het parametermodel worden beschreven. In dit hoofdstuk beschrijven we eerst de keuze voor de basisfaalkans, gevolgd door de identificatie van de relevante factoren. Daarna bespreken we de betrouwbaarheid en de daaruit volgende scoringsschalen. Deze paragraaf wordt afgesloten door de beschrijving van de kalibratie van de factoren.

5.3.1 Basisfaalkans

Als basisfaalkans is voor 1 gekozen, omdat dit de meest conservatieve waarde is die men kan kiezen. Bij vergelijkbare modellen, bijvoorbeeld het OPSCHEP model, wordt gekozen voor de gemiddelde faalkans. Zo'n gemiddelde is bij software echter onbepaald, daarom is voor een zeer conservatieve benadering gekozen. De keuze voor een basisfaalkans van 1 komt de onderhoudbaarheid van de methode ten goede, als op een later moment blijkt dat extra factoren toegevoegd moeten worden, zal de herijking van bestaande analyses minder vaak nodig blijken.

Gevolg is dat, als men absoluut niets weet van de softwaremodule, deze met een faalkans 1 in de foutenboom wordt opgenomen. Gezien de afwezigheid van enige informatie en het niet uitgevoerd hebben van enige simulaties (een parameter die een afnemer zelf kan beïnvloeden) lijkt dit geen onterechte aanname.

5.3.2 Identificatie relevante factoren

De methode is grotendeels afhankelijk van het in acht nemen van de juiste factoren. Om te komen tot de juiste factoren zijn er initieel twee brainstorm-sessies geweest met de expertgroep. Deze brainstormsessies hebben een lijst van factoren opgeleverd welke als basis hebben gediend voor het parametermodel.

Als verificatiestap is er in wetenschappelijke publicaties gezocht naar onderbouwde studies over extern waarneembare factoren die de betrouwbaarheid van software beïnvloeden. Er zijn geen wetenschappelijk onderbouwde studies aangetroffen over dit onderwerp.

Wel zijn er wetenschappelijke publicaties gevonden aangaande vergelijkbare brainstormsessies door expertgroepen. Dit zijn Smidts et al.[21][22], en Li [14]. Deze aanpakken zijn gedetailleerder (kennen meerdere factoren voor bij elkaar horende factoren met een redelijk sterke correlatie), maar de auteurs onderkennen geen categorieën die niet in het huidige TOPAAS model onderkend worden.

5.3.3 Betrouwbaarheid methode

Doel van deze methode is tot een schatting van de ordegrootte van de softwaremodule te komen, niet tot een heel nauwkeurig getal. Door het complexe gedrag van software is het in praktijk onmogelijk om met deze methode überhaupt beter te schatten dan een ordegrootte: elk nauwkeuriger getal is in werkelijkheid een schijnnauwkeurigheid.

Door alleen ordegroottes als schatting te geven reflecteert de geschatte faalkans zowel de door experts gehanteerde schattingswijze alsmede ook de benaderingswijze van de industrie. Zowel de expert-groep als de industrie is van mening dat een schatting op ordegrootte het meest nauwkeurige is dat nu mogelijk is:

- Expert-schattingen geven alleen een schatting in ordegrootte van faalkans
- Industrie-standaarden zoals de IEC61508 [11] geven SIL's (Safety Integrity Levels) af met een bandbreedte van één enkele ordegrootte (i.e. SIL2 is heeft als doel een faalkans die ligt tussen de 10^{-2} en 10^{-3} per vraag te produceren).

Hieruit blijkt dus dat zowel de expert-groep als de industrie het onmogelijk vinden nauwkeuriger te schatten dan in ordegroottes van faalkansen.

De beïnvloeding van een factor op de faalkans is beperkt tot de volgende vermenigvuldigingsfactoren:

- $10^{⅓}$, een toevoeging van ⅓ ordegrootte van de faalkans (factor 2)
- $10^{½}$, een toevoeging van ½ ordegrootte van de faalkans (factor 3)
- $10^{⅔}$, een toevoeging van ⅔ ordegrootte van de faalkans (factor 5)
- 10, een toevoeging van een hele ordegrootte van de faalkans (factor 10).

Andere waarden dan de hiervoor genoemde factoren zijn voor schattingen te fijnmazig en leiden tot een schijnnauwkeurigheid. Ook hebben dergelijke waarden in praktijk geen noemenswaardige invloed op de faalkansschatting. Een uitzondering zou kunnen worden gevormd door waarden die bepaald zijn op basis van wetenschappelijk onderzoek. Dit is echter, zoals al eerder gemeld, op dit moment nog onmogelijk door het gebrek aan gedegen breed wetenschappelijk onderzoek naar specifieke effecten van maatregelen op productkwaliteit.

Doel van deze methode is niet alleen om statisch en eenmalig de faalkans van een softwaremodule te bepalen. Het is ook een instrument dat kan helpen de schatting te verbeteren en kennis over softwaremodules te verwerken. Met name voor softwaremodules waar initieel weinig van bekend is kan men als afnemer op basis van maatregelen (meetprogramma's en eigen testen) de intrinsieke faalkans van de module verminderen.

5.3.4 Werkwijze TOPAAS schatting

De in §5.2 "Beschrijving parametermodel op hoofdlijnen" beschreven generieke formule voor een parametermodel is de volgende:

$$P = P_B * F_i * F_2 * ... * F_n$$

Alhoewel dit de normale benaderingswijze van parametermodellen is, wordt in praktijk in TOPAAS alleen de ordegrootte van de faalkans van software geschat (zoals bepleit in §5.3.3 "Betrouwbaarheid methode") in een logaritmische schaal. Ook zijn, omwille van het gebrek aan nauwkeurigheid, alle beïnvloedingsfactoren ook in ordegrootten gedefinieerd. Hierdoor is van P eigenlijk alleen de $^{10}\log P$ van belang: dit is de exponent die de ordegrootte van de faalkans aangeeft en gedetailleerdere informatie is niet noodzakelijk om tot een schatting te komen.

Eigenlijk zijn we dus alleen maar geïnteresseerd in de exponent van de faalkans. Schrijven we de ordegrootte van P als n (dus $n = {}^{10}Log\ P$) dan wordt de vermenigvuldiging een redelijk eenvoudige optelling:

$$n = {}^{10}log\ P_B + \sum_i {}^{10}log\ F_i$$

Aangezien er alleen schattingen worden gemaakt op basis van ordegroottes, en de eenvoud van optellen boven een vermenigvuldiging, is er dan ook voor gekozen om de deze formule te hanteren voor de schatting van de faalkans. Dit betekent dat de gehanteerde ordegrootte van de uiteindelijk geschatte faalkans de optelling is van de $^{10}\log$ van alle van toepassing zijnde factoren.

In het TOPAAS-model wordt dus uitgegaan van een optelling van de ($^{10}\log$) exponent van de beïnvloedingsfactoren F_i. De $^{10}\log$ van F_i noemen we B_i. Dit impliceert dus dat uiteindelijk de ordegrootte van de faalkans is (uitgaande van een basisfaalkans P_B van 1, zoals is beschreven in §5.3.1 "Basisfaalkans"):

$$P = 10^n \text{ met } n = \sum_i B_i$$

In de specifieke invulling van het parametermodel (zie §5.4 "Invulling methode") wordt er verder gebruik gemaakt van B_i-waarden.

5.3.5 Bepaling vermenigvuldigingsfactoren

Voor elke categorie moeten de schalen nu ingevuld worden. Deze invulling van de schalen is, waar mogelijk, gebaseerd op wetenschappelijk onderzoek. Complicerende factor is hierbij dat er weinig tot geen grootschalig wetenschappelijk onderzoek plaatsvindt naar de geïsoleerde

invloed van specifieke factoren op software betrouwbaarheid, zeker niet als het betrekking heeft op het domein van veiligheidskritische toepassingen.

Daar waar de wetenschap ons geen antwoord geeft, is de factor geschat door de expertgroep. In praktijk geldt dit voor alle schalen die bij de categorieën horen, uitgezonderd de schalen voor het SIL-level. Voor de schalen van het SIL-level is de invloed één ordegrootte conservatiever geschat dan dat de wereldwijde industrieconsensus aangeeft.

Kerngedachte bij de inschatting van factoren is dat de common practice in de industrie/toepassing het neutrale element oplevert (vermenigvuldigingsfactor 1, optelfactor 0). Dit is wederom een conservatieve inschatting: een partij die software ontwikkelt en zich dus conformeert aan de common practice (niet zijnde de best practice in de industrie), zal dus eindigen met een faalkans van 1 ($=10^0$). Dit symboliseert dat voor missie-/veiligheids-kritische systemen men beduidend meer effort moet doen dan de common practice in de IT-industrie te hanteren. Ook symboliseert dit dat men "gestraft" wordt door dingen te doen die tegen de common practice van de industrie ingaan. Dit heeft ook als bijkomend voordeel dat, als men geen inzage heeft in het proces, de praktische scoring (geen toevoeging waardering) overeenkomt met de common practice van de industrie.

Om toch tot een gedegen scoring te komen is er door de expertgroep besloten de factoren te kalibreren aan inschattingen van experts, om zo tot een empirische onderbouwing van het model te komen. Dit is uitgewerkt in §5.3.6 "Kalibratie model"

5.3.6 Kalibratie model

Als corrigerende maatregel op de individuele schatting van beïnvloedingsfactoren is het model beproefd op een aantal archetypen (stereotype project welke staat voor een klasse van systemen) en een aantal werkelijke projecten.

In alle gevallen hebben de experts eerst een eigen inschatting van de faalkans gemaakt: dit is dus de expert-opinion waarvan verwacht wordt dat het parametermodel deze benadert. Vervolgens is het parametermodel ingevuld en is de uitkomst vergeleken met de expert-uitkomst. Waar nodig is gekeken naar de factoren die afwijking veroorzaken. Meest majeure wijziging is dat voor meerdere categorieën onderscheid noodzakelijk bleek tussen de "niet-SIL3/SIL4" omgeving en "SIL3/SIL4" omgeving.
Enkele belangrijke uitkomsten van archetypen:

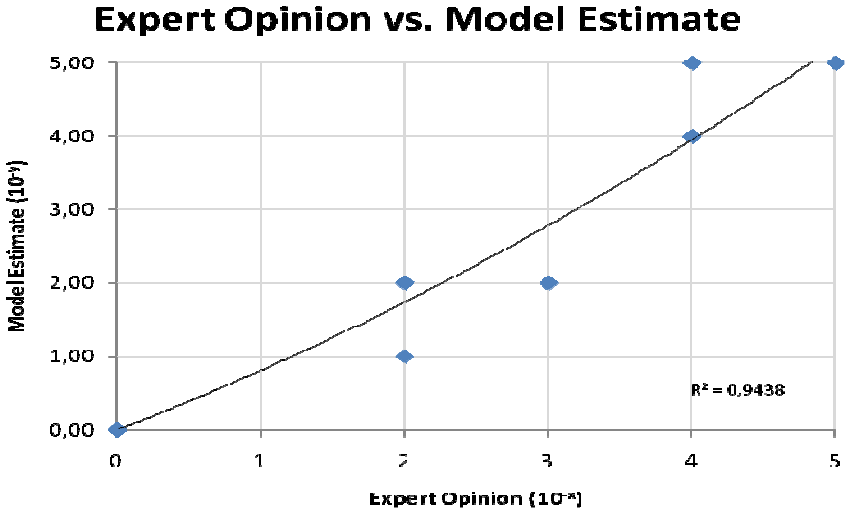
- Een Black-Box COTS component haalt maximaal een faalkans van 5.10^{-5}, gebaseerd op reputatie van de leverancier en ervaringsgegevens uit het veld, maar zonder inhoudelijke bekendheid met kwaliteit van de toegepaste ontwikkelprocessen. Als men alleen bekend is met velddata, zonder enige kennis van (de reputatie van) een leverancier, dan is een geschatte faalkans 10^{-2} haalbaar.
- Een optimaal ontwikkeld custom made component heeft een haalbare faalkans van 2.10^{-5}, bij volledige inzage in het proces van wat bij de ontwikkelaar is uitgevoerd/certificering.

- Zaken die op volledig duistere wijze in elkaar geknutseld worden door hobbyisten krijgen een faalkans van 1.

In totaal zijn er, naast de 7 archetypen, ook 13 echte projecten ingevoerd. Allen uit een veiligheidskritische/missioncritical omgeving met een sterke verbinding met de besturing van elektromechanische componenten. In de benodigde faalkans voor elke component zat een sterke variatie, alsmede ook de inschatting van de experts van de daadwerkelijke faalkans. Dit had de volgende resultaten:

Projectnaam	Expert	Model
ARCHITYPE - Ultimate White box	5	5
ARCHITYPE - Worst White Box	0	0
ARCHITYPE - Ultimate Black Box	4	4
ARCHITYPE - Worst Black Box	0	0
ARCHITYPE - Eigen knutselwerk	0	0
ARCHITYPE - Intensief gebruikte COTS component	3	2
ARCHITYPE - Zeer complex SIL-4 systeem	4	5
Rijkswaterstaat - INWin Oosterscheldekering	5	5
Rijkswaterstaat - Proces Oosterscheldekering	5	5
DNV – BLGS	2	2
DNV – MDP	4	4
DNV – SCADA	0	0
DNV – BEEG	5	5
DNV – VILM	0	0
Rijkswaterstaat - Sbk-BOS Interface (happy flow)	4	4
Rijkswaterstaat - Sbk-BOS interface (uitzonderingen)	2	1
Rijkswaterstaat – SOBEK	3	2
Rijkswaterstaat - MDL50	4	5
Rijkswaterstaat - Ramspol 2	2	2
Rijkswaterstaat - Ramspol 1	2	2

Hieruit blijkt dus al dat de uitkomsten van het model en de initiële inschatting door experts een sterke relatie met elkaar hebben. Zet men de uitkomst van het model uit tegen de inschatting door de experts voor deze 20 projecten, dan komt men tot de volgende grafiek:



Figuur 11: Relatie inschatting door experts en model (N=20)

Uit bovenstaande figuur blijkt dus dat er een behoorlijk sterke relatie is tussen de Expert Opinion en de schatting van het model. De correlatie is 0,9347.

Wel blijkt door de trendlijn dat, ondanks dat het model zeer conservatief schat, het model iets "optimistischer" is dan de experts. Uit de proeven blijkt ook dat het TOPAAS model optimistischer is dan het TDT model (in die gevallen waar vergelijkingsinformatie aanwezig was).

In Bijlage C "Kalibratie model" is een overzicht opgenomen van alle vragen, met de spreiding in antwoorden en hun bijdrage in de faalkans ten opzichte van de schatting van een expert. We gaan in deze paragraaf hier nog iets dieper op in.

Bij elke vraag zijn onder het kopje "Spreiding" staafdiagrammen bijgevoegd die de keuzen weergeven zoals die gemaakt zijn door de experts voor de 20 cases. Op de horizontale as staan dus de mogelijkheden zoals die volgen uit de tabellen in §5.4, aangevuld met een staaf (de meest linkse) die het lege antwoord (geen scoring) visualiseert. De spreiding van de antwoorden bij alle projecten is cruciaal: als de antwoorden bij een specifieke vraag absoluut geen spreiding kennen (altijd hetzelfde ingevuld), dan kan men geen conclusies verbinden aan de uitkomsten van die vraag voor situaties die anders scoren. Als er enige vorm van spreiding is, met name op de uitersten van de scoring, dan kan men in ieder geval nog uitgaan van interpolatie. Maar in alle gevallen is voor de validiteit van scoring ten opzichte van de kalibratie voor alle vragen een significante spreiding noodzakelijk. Zoals de bijlage aangeeft is er door de projecten echt verschillend gescoord, hetgeen aangeeft dat er niet één enkel type project gebruikt is voor de kalibratie, waardoor de scoring redelijk gegeneraliseerd mag worden naar projecten binnen hetzelfde domein.

Ander zorgpunt binnen parametermodellen is dominantie: hoewel er een zeer grote groep van factoren in ogenschouw wordt genomen, zou het uiteindelijk kunnen zijn dat één kleine groep specifieke factoren altijd de faalkans in zeer grote mate bepaalt. Een potentiële kandidaat hiervoor is het SIL-Level, met zijn zeer grote invloed op de faalkans (met een bereik van 4 ordegrootten). Daarom is bij "Correlatie met de uitkomst" voor elke vraag de bijdrage van het specifieke antwoord (indien aanwezig) van een project afgezet tegen de faalkans die bepaald is door de expert-opinion. Hiermee is het mogelijk om te verifiëren of er sterk dominante factoren in het parametermodel aanwezig zijn. Kijkt men naar de correlatie van de individuele factoren met het beoogde eindresultaat, dan ziet men dat bij elke vraag:

- de individuele bijdrage van veel factoren niet potentieel dominant te noemen zijn. Dat wil zeggen dat de invloed beperkt is tot maximaal één enkele ordegrootte van de uiteindelijke faalkans, waarbij bij veel factoren zelfs een afvlakking plaatsvindt in de range van 10^{-4} tot 10^{-5}, hetgeen betekent dat het verschil niet specifiek gemaakt wordt door die factor
- de individuele bijdrage van enkele grote factoren (te weten "SIL-Level", "opleidingsniveau en ervaring ontwikkelaars" en "aanwezigheid representatieve velddata") van behoorlijke invloed is, maar dat ze niet sterk lineair correleren met de verwachte uitkomst. Zelfs als men deze groepen op een willekeurige wijze combineert dan komt men niet tot een sterk correlerende set.

Dit impliceert dat er niet een dominante (set van) factoren aanwezig is die de faalkans bepaalt en andere factoren dus overbodig zou maken.

Op basis van deze kalibratie kan men dus concluderen dat:

1. De uitkomsten van het model een sterke overeenkomst vertonen met de schatting van de faalkans door ervaren experts;
2. De factoren allen een redelijke distributie in hun scoring gekend hebben tijdens kalibratie, waardoor de toepasbaarheid in het domein niet beperkt is tot specifieke soorten (veiligheidskritische) systemen in specifieke ontwikkelomgevingen;
3. De uitkomsten niet sterk gedomineerd worden door een subset van de factoren, hetgeen in praktijk alle andere factoren buiten die subset overbodig zou maken.

5.4 Invulling methode

De hieronder genoemde factoren hebben allen een redelijk eenvoudige schaal, waarbij elke factor nauwelijks rekening houdt met andere factoren. In Bijlage B "Raamwerk voor invoegen afhankelijkheden" is aangegeven hoe in principe kan worden omgegaan met afhankelijkheden tussen de factoren. Binnen de huidige aanpak wordt alleen een beperkte afhankelijkheid tussen het SIL niveau een aantal andere factoren gesteld. Dit is een bewuste aanpak om zo het model eenvoudig te houden.

Men moet zich realiseren dat er sprake zou moeten zijn van een verminderde meeropbrengst als alle factoren zeer positief zijn. In een IEC SIL4 omgeving [11] is bijvoorbeeld de extra toegevoegde waarde van zeer uitgebreide testperioden vrijwel verwaarloosbaar. Toch is er besloten alleen

maatregelen uit te sluiten op SIL3/SIL4 niveau als de maatregel op dat moment expliciet wordt afgedwongen in de IEC. Doordat alle waarden redelijk conservatief worden geschat en er uiteindelijk alleen gekeken wordt naar de ordegrootte van de faalkans, lijkt dit een goed offer ten gunste van de eenvoud.

Expliciet wordt opgemerkt dat er onderscheid gemaakt wordt tussen het hebben van informatie en het niet hebben van informatie. Als men geen informatie heeft over een factor, bijvoorbeeld het voortbrengingsproces, dan wordt men geacht gewoon de vraag over te slaan en geen scoring toe te kennen. Hierdoor ontstaat er een betere scheiding tussen informatie die aanwezig is en waar men vertrouwen in heeft en dingen die en niet weet.

Het schatten van de faalkans van de softwaremodule gebeurt via een score op de volgende dimensies
a) Totstandkomingproces
b) Product
c) Requirements traceability/verifieerbaarheid
d) Testen
e) Executieomgeving/gebruik
De aspecten die in deze dimensies een rol spelen zijn veelal begrippen die voornamelijk bekend zijn in de ICT-wereld. Het schatten waar de grenzen liggen tussen de genoemde niveaus per aspect is niet altijd SMART. Het bleek bij de genoemde pilot [16] dat de kennis van een ICT-expert (software expert) noodzakelijk was bij het goed inschatten van het juiste niveau. Ook een onafhankelijke blik op de aspecten is van belang. De softwaremaker zelf zal vaak een te rooskeurige blik hebben van zijn eigen proces of product. Daarom wordt ten zeerste aanbevolen om bij de beoordeling van de software een onafhankelijke ICT-expert te betrekken.

5.4.1 a) Totstandkomingproces

1 Het ontwikkelproces voldoet aan een van de SIL's van de IEC		
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, zoals Fuzzy Logic	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

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

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

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

5 Opleidingsniveau en ervaring ontwikkelaars		
1	Onbekend, of enige kennis en ervaring van het specifieke domein	0
2	Geen kennis en ervaring van het specifieke domein	1
3	Weinig kennis en ervaring van het specifieke domein	½
4	Gewenste kennis met systeemontwikkeling voor het specifieke domein (onbewust bekwaam)	0
5	Aantoonbaar uitstekende kennis en veel ervaring van het specifieke domein	-½

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

5.4.2 b) Producteigenschappen

7 Complexiteit beslislogica		
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

8 Omvang softwaremodule (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

9 Helderheid gebruikte architectuurconcepten		
1	Onbekend	0
2	Geen heldere afbakening taakuitvoering modulen in ontwerp benoemd	½
3	Wel taakuitvoering op hoofdlijnen benoemd, maar geen navolging gegeven in ontwikkeling	⅓
4	Er is een scheiding van taakuitvoering 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 taakuitvoering 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

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

5.4.3 c) Requirements traceability/verifieerbaarheid

11 Traceerbaarheid van requirements 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

5.4.4 d) Testen

12 Testtechnieken en dekkingsgraad			
		Normaal	**SIL3/SIL4**
1	Onbekend	0	NVT
2	Geen gedocumenteerde tests uitgevoerd	0	NVT
3	Wel testen gedocumenteerd, 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

5.4.5 e) Executieomgeving/gebruik

13 Multiprocesomgeving		
1	Onbekend	0
2	Meerdere applicaties draaien parallel op één OS in een netwerkomgeving	½
3	Meerdere applicaties op één stuk hardware	⅓
4	Één proces op een dedicated OS met een dedicated CPU	0
5	Dedicated CPU en memory op geen of een triviaal OS	-⅓

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

15 Monitoring		
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

5.5 Toelichting bij de parameters

5.5.1 *1 Het ontwikkelproces voldoet aan een van de SIL levels van de IEC*
Hier wordt bewust conservatiever geschat dan dat de IEC [11] 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.

5.5.2 *2 Gebruik van Inspecties*
De eerste 2 opties zijn niet van toepassing voor SIL 3 of 4, omdat inspecties de IEC inspecties verplicht stelt vanuit level 3, en dus tot de common practice behoren als men een SIL3/4 systeem realiseert. Onder SIL3/4 worden Fagan inspecties aanbevolen, zie bijlage Bijlage D, "Het Fagan-inspectieproces".

5.5.3 *3 Hoeveelheid wijzigingen ten opzichte originele ontwerp/eisenpakket*
Onder zeer frequente wijzigingen wordt een meer dan normaal verwacht aantal wijzigingen verstaan. Bij fundamentele wijzigingen moet o.a. gedacht worden aan bijv. het verschuiven van de modulegrenzen door opknippen of samenvoegen van modulen. Indien voor de optie 3 (Weinig wijzigingen, met zeer geringe impact) of 4 (Geen wijzigingen) wordt gekozen dient een versienummering en dienen logboeken aanwezig te zijn.

5.5.4 *4 Cultuur en samenwerking*
De typering van cultuur en samenwerking wordt volgens IAEA-TECDOC-1329 als volgt 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
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

5.5.5 *5 Opleidingsniveau en ervaring ontwikkelaars*
Er wordt uitgegaan van uitstekende kennis en ervaring van systeemontwikkeling. Met "Weinig kennis en ervaring" wordt gedacht aan (een of meer) projecten, langer dan drie jaar geleden. Met "uitstekende kennis en ervaring" wordt bedoeld dat er een of meer projecten van meer recente datum aangetoond kunnen worden.
Met "domein" wordt expliciet een objectdomein bedoeld: een systeem voor een stormvloedkering, voor een sluis, voor een beweegbare brug, voor een tunnel, etc.

5.5.6 *6 Samenwerking met Opdrachtgever*
Geen nadere toelichting.

5.5.7 *7 Complexiteit beslislogica*
Beslislogica is cruciaal: men kan kijken naar de McCabe complexiteit van de broncode, maar ook naar het extern observeerbaar gedrag van de component en op basis daarvan een inschatting maken. Meest geaccepteerde vorm van het meten van complexiteit van software is de McCabe index. Alhoewel er formele wiskundige definities zijn, is de meest eenvoudige definitie:

$$McCabe\ Index = 1 + D$$

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

Deze definitie kan zowel toegepast worden op broncode (tellen van de if/case, while en switch/case statements), als op beschreven beslislogica van de applicatie.

5.5.8 *8 Omvang softwaremodule (Lines of code)*
Bij ontwerp of bij COTS is het vaak onmogelijk om het aantal regels code vast te stellen. Wel is dit vaak mogelijk met behulp van Functiepuntenanalyse of op basis van kengetallen van Capers Jones.
Indien met behulp van een grafische programmeeromgeving wordt gewerkt, wordt één grafisch element als één Line of Code opgevat.

5.5.9 *9 Helderheid gebruikte architectuurconcepten*
Geen nadere toelichting.

5.5.10 10 Gebruik van een certified compiler

Er zijn maar weinig certified compilers. Zo zijn, bijvoorbeeld, ook de compilers die Microsoft levert bij Windows geen certified compilers. Bij ontwikkeling op SIL 3/4 wordt, vanuit de IEC [11], een certified compiler verplicht gesteld.
De IEC, 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".

5.5.11 11 Traceerbaarheid van requirements door het proces heen

Dat een software module "doet wat hij moet doen" betekent dat de opgestelde gebruikerseisen allemaal gerealiseerd zijn en het liefst aantoonbaar. De beste manier om dat te bewerkstelligen is door in het softwareontwikkelproces, stapsgewijs de gebruikerseisen te vertalen naar testscripts en technische systeemeisen. Vervolgens worden software modules aangewezen die de systeemeisen waar gaan maken. Op deze manier worden gebruikerseisen (de requirements) traceerbaar tot aan de code toe.

Wiskundig correct bewezen traceerbaarheid (optie 7) houdt in dat er model checking toegepast is of een gedragsequivalentie aangetoond is tussen de implementatie en een correcte specificatie. Dit is vooralsnog 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 software module volledig (optie 6). Dit geldt misschien alleen voor veiligheidskritische eisen (optie 5) of alleen gedeeltelijk (optie 4) of tot tests (optie 3).

5.5.12 12 Testtechnieken en dekkingsgraad

Bij het formeel testen worden er technieken toegepast uit wiskunde en logica. Het startpunt is een geverifieerd model van het softwareontwerp, of de software zelf als die geannoteerd is met formele specificaties. Daaruit worden automatisch tests gegenereerd die een gewenste requirement testen met een gecontroleerde dekkingsgraad. Het formeel testen proces maakt het verband tussen requirement en test case heel duidelijk en is daardoor van belangrijke waarde voor het aantonen van software betrouwbaarheid. Een paar tools die model-based testgeneratie ondersteunen zijn T-VEC, Conformiq, Reactis, Unitesk. Een academische tool die 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'
  - combinatie van equivalentieklassen op basis van pairwise of triplewise; etc..
- als percentage van doorlopen paden tijdens testtraject t.o.v. theoretisch mogelijke paden m.b.v. tools zoals McCabe (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 doorlopen combinaties van input variabelen ten opzichte van theoretisch mogelijke combinaties van input variabelen
  De dekkinggraad is in dit geval:
  - hoog: 90% van de theoretisch mogelijke input
  - gemiddeld: 50% van de theoretisch mogelijke input
  - laag: 10% de theoretisch mogelijke input.

5.5.13 13 Multiprocesomgeving

Geen nadere toelichting.

5.5.14 14 Aanwezigheid representatieve velddata gedurende taakuitvoering

Bij keuze 3 wordt met "beperkte gegevens" bedoeld dat de module erg weinig gebruikt wordt, minder dan, ordegrootte, 5 keer per maand.
Bij keuze 4 wordt met "significante hoeveelheid gegevens" bedoeld dat de module beperkt wordt gebruikt, ten minste, ordegrootte, 5 keer per maand.
Bij keuze 5 wordt met "veel representatieve velddata" bedoeld dat de module veel wordt gebruikt, ten minste, ordegrootte, 5 keer per dag.

5.5.15 15 Monitoring

De monitoring is specifiek op de eigen taakuitvoering van de module gericht. Daarbij kunnen ook andere omgevingen waarin de module draait betrokken worden.

Bij keuze 3 wordt met "Weinig/kort gemonitord" bedoeld dat de module beperkt wordt gebruikt, hooguit, ordegrootte, 1 keer per maand.
Bij keuze 4 wordt met "Langdurige monitoring, maar infrequente taakuitvoering" bedoeld dat de module ten minste, ordegrootte, 1 keer per maand wordt gebruikt en maar hooguit 1 keer per jaar wordt aangesproken op de bijzondere situatie waarvoor de software is geschreven.
Bij keuze 5 wordt met "Langdurige/frequente monitoring tijdens taakuitvoering" bedoeld dat de module ten minste, ordegrootte, 5 keer per dag wordt gebruikt en ten minste, ordegrootte 5 keer per jaar, wordt aangesproken op de bijzondere situatie waarvoor de software is geschreven.

6 Voorbeelden

6.1 BepaalPeiloverschrijding

6.1.1 Systeemanalyse

Het BOS is het BeslisOndersteund 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, etc.

In de analyse van het BOS (inclusief script) kan de systeemanalyse functiegedreven zijn door het procedurescript te beschouwen, waar in het geval van een stormconditie een *ProcedureScriptOpdracht* wordt uitgevoerd '*BepaalPeiloverschrijding'*. Op dit moment zijn er drie faalmodi 'verzaken in de bepaling van de Peiloverschrijding' (returnwaarde ONBEPAALD) 'onterecht niet melden van een Peiloverschrijding' en 'onterecht melden van een Peiloverschrijding'.

De peiloverschrijding wordt bepaald op basis van een set beslisregels en een verwachte waterstand bij Rotterdam. De verwachte waterstand wordt geproduceerd door de taken *SRS:ValideerModelInput* en *SOBEK:BepaalWaterstandRotterdam*.

Afhankelijk van incompleetheid van inputdata voor SOBEK wordt de taak *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).

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 missie.

De taken *SRS:ValideerModelInput* en *SRS:RepareerInput* zijn geïmplementeerd via de component *SRS*, draaiend op de Stratus ftServer, gebruikmakend van BOS-database access. Geïmplementeerd in C/C++ met bijbehorende bibliotheken.

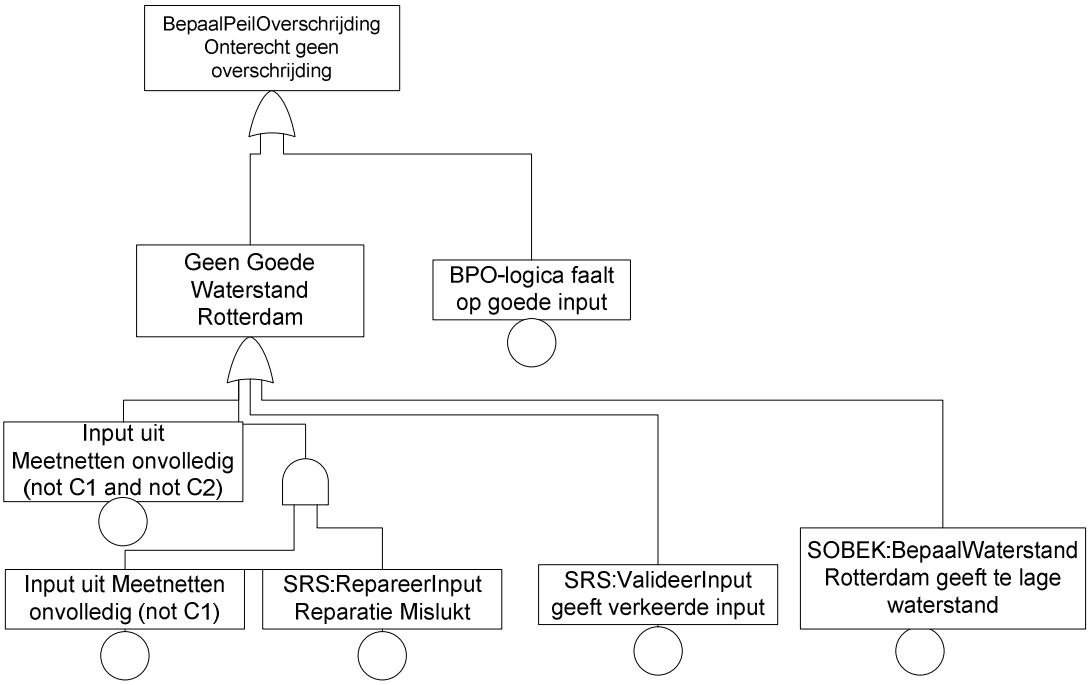
De taak *SOBEK:BepaalWaterstandRotterdam* is geïmplementeerd via de component *SOBEK*, draaiend op de Stratus, geïmplementeerd in Fortran90 gebruikmakend van fortran bibliotheken.

De taak *BPO-logica* is geïmplementeerd in de BOS component *PSI*.

Ondersteunend aan deze taken is het BOS-platform (database, ftServer, OS).

6.1.2 *Foutenboom analyse*

De foutenboom zou er op dit onderdeel zo uit kunnen zien (er vanuit gaande dat falen van het BOS-platform op een hoger niveau meegenomen wordt). 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 12: Foutenboom *BepaalPeilOverschrijding* faalt

6.1.3 *Kwantificering*

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

De code voor *BPO-logica* is geïmplementeerd in de BOS component *PSI* in functies *psbepmodelevaluatie* (cyclomatische complexiteit 14 en aantal code statements 116 LOC) en *psbeppeiloverschr* (complexiteit=10 code 162). De component *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 Taak "SOBEK: BepaalWaterstandRotterdam geeft te lage waterstand" wordt geschat met behulp van een statistische analyse, de frequentie van deze functie (1/10min) maakt dat de afgelopen 10 jaar 144*365*10 = 50.000 runs vergeleken kunnen worden met de feitelijke waterstanden.

Deze situatie is helaas niet mogelijk. De resultaten van de modelruns zijn niet allemaal bewaard gebleven. De lokatie 'Rotterdam' die door *SOBEK* wordt bepaald wordt niet gemeten.

Kijkt men naar de *SOBEK* naar BOS interface, dan krijgt men de volgende scoring:

Totstandkomingproces		
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 beslislogica 3 Beslislogica is redelijk eenvoudig (bevat enkele zeer geïsoleerde uitzonderingssituaties), McCabe index tussen 10 en 30.	-⅓
8 Omvang software module (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 componenten 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

Requirements traceability/verifieerbaarheid	
11 Traceerbaarheid van requirements 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 missie 4 Veel representatieve velddata aanwezig van identieke of sterk vergelijkbare toepassingen	-2
15 Monitoring 2 Weinig/kort gemonitord gedurende missie	0
Totaal	-4⅔

Dit geeft aanleiding tot een geschatte faalkans van $10^{-4⅔}$ = $2{,}2.10^{-5}$, wat volledig overeenkomt met de inschatting door experts die nauw bij het project betrokken zijn.

6.2 Component "off-the-shelf"

Kijkt men naar een COTS model van een bekende leverancier, denk aan bijvoorbeeld klepbesturingen van ABB die op een dedicated PLC wordt geleverd, dan is het nog steeds mogelijk een berekening uit te voeren. Men heeft alleen minder informatie, hetgeen aanleiding geeft tot het wegvallen van veel factoren. Op basis van de wel beschikbare (of geschatte) informatie krijgt men het volgende beeld.

Totstandkomingproces		
4	Cultuur en samenwerking ontwikkelorganisatie 3 Zelflerende organisatie *Gebaseerd op ervaren werkwijze tijdens uitrol product*	-½
5	Opleidingsniveau en ervaring ontwikkelaars 4 Aantoonbaar uitstekende kennis en veel ervaring van het specifieke domein *Gebaseerd op reputatie ontwikkelorganisatie (naam in het veld van bedrijf), in combinatie met ervaringen tijdens bouw systeem en uitrol product*	-½
Product		
7	Complexiteit beslislogica 4 Beslislogica en foutherkenning zijn erg eenvoudig, McCabe Index kleiner dan 10. *Gebaseerd op een inschatting aan de hand van de te automatiseren beslislogica: de uit te voeren handeling is zeer recht-toe-recht-aan.*	-½
8	Omvang software module (Lines of code) 5 Minder dan 1000 *Gebaseerd op een inschatting aan de hand van een Functiepunten-analyse.*	-½
Executieomgeving/gebruik		
13	Multiprocesomgeving 4 Dedicated CPU en memory op geen of een triviaal OS *Gebaseerd op het feit dat de applicatie op een dedicated PLC geleverd wordt.*	-⅓
14	Aanwezigheid representatieve velddata gedurende missie 4 Veel representatieve velddata aanwezig van identieke of sterk vergelijkbare toepassingen *Gebaseerd op het grootschalige gebruik van de klepbesturing bij veel installaties in de olie-industrie, in vergelijkbare fysieke omgevingen en risicoklasse.*	-2
Totaal		-4⅓

Dit geeft aanleiding tot een geschatte faalkans van $10^{-4⅓}$ = $4{,}6.10^{-5}$, wat volledig overeenkomt met de inschatting door experts.

7 Conclusie en aanbevelingen voor de toekomst

7.1 Conclusie

Een van de grootste zwaktes in de TDT-methode is de zeer beperkte hoeveelheid aan factoren die in ogenschouw worden genomen, en de inflexibiliteit om meer factoren te beschouwen (dit doordat schalen sterk verankerd zijn met de hoeveelheid factoren). Het TOPAAS-model is gebaseerd op de expert-opinion van Nederlandse experts, aangevuld met in de literatuur gevonden factoren van andere expert-groepen (zie §5.3.2 "Identificatie relevante factoren"). Het TOPAAS model is ook open in de zin dat het nieuwe factoren eenvoudig accepteert. Daarmee is het in dat opzicht eerder een raamwerk dan een methode.

De TDT-methode berekent een "betekenisloze" score, welke vervolgens omgezet wordt in een faalkans, iets wat in de literatuur sterk wordt afgeraden (zie Bijlage A "SWOT Analyse" onder kopje "Weaknesses TDT methodiek"). In tegenstelling tot de TDT-methode wordt er wel met betekenisvolle factoren gerekend tot een faalkans: er wordt immers altijd met potentiële invloed op de faalkans gerekend.

Een punt van aandacht is de (empirische) onderbouwing van de uitkomsten. Het TDT-model is daar zeer zwak in, het TOPAAS-model is daar beter in, in de zin dat het factoren zijn die door een expertgroep zijn vastgesteld en getest zijn met een twintigtal projecten, maar een verbetering van de onderbouwing door meer onderzoek wordt aanbevolen. Het TOPAAS-model leent zich beter voor de kalibratie en de verwerking van wetenschappelijke resultaten.

7.2 Onderzoek op korte/middellange termijn

Er is onderzoek noodzakelijk voor de korte/middellange termijn. Dit onderzoek zal zich moeten richten op het toetsen van de kwantitatieve toepasbaarheid van de methode in het veld. Hoewel er twee voorbeelden zijn uitgewerkt, zijn meer praktische toetsingen noodzakelijk om de praktische toepasbaarheid te toetsen en verbeteren. Een dergelijke praktijktoets zal zich moeten richten op:

- Het versterken van de meetbaarheid van factoren. Sommige factoren kunnen nu alleen worden gescoord door experts. Om grote verschillen in resultaten, als gevolg van verschillende interpretaties van de factoren, te voorkomen zijn voor veel factoren duidelijker richtlijnen nodig om de methode over te kunnen dragen naar reguliere gebruikers. Deze meetvoorschriften zullen in de praktijk ontwikkeld moeten worden op basis van observatie van de scoring door experts.
- De beschikbaarheid van informatie over COTS componenten die met enige moeite te achterhalen is, en de effecten op de uitkomsten van het TOPAAS model.
- Het discriminerend vermogen van het TOPAAS model: men verwacht een redelijke spreiding in de faalkans van de componenten, waarbij de onderlinge verhoudingen van componenten ook overeen moeten komen met de verwachting van experts.

- Onderzoek naar modellering met behulp van gebeurtenissenbomen in aanvulling op het gebruik van foutenbomen. Indien er sprake is van volgordelijk falen is de modellering met een foutenboom lastig. Dit is al eerder gebleken in onderzoek naar menselijk falen waar volgordelijk falen een rol speelt en de gebeurtenissenboom een handiger hulpmiddel bleek dan de foutenboom.

Naar aanleiding van deze aanbevelingen zijn vier praktisch toetsingen uitgevoerd. De bevindingen van die toetsen zijn gerapporteerd in [16] en de voorstellen tot wijzigingen zijn integraal verwerkt in deze tweede versie van dit rapport. Daaruit bleek dat het schatten van de scores en de mate waarin wordt aangetoond dat deze inschatting juist is, een proces is dat moet worden begeleid, en zo nodig geleid, door een onafhankelijke expert. Ook werd geconstateerd dat het nodig is een aparte handleiding te maken voor het toepassen van TOPAAS, zonder de wetenschappelijke onderbouwing.

7.3 Onderzoek op lange termijn

Op lange termijn zal men moeten streven naar het verder "onderbouwen" en "kalibreren" van de factoren onder de TOPAAS methode. Het TOPAAS model moet eigenlijk als een raamwerk gezien worden waar men verder op kan bouwen. Kern van het model is dan ook dat de factoren en hun bijdrage aangepast kunnen worden aan praktische ervaring én het voortschrijdende inzicht uit de wetenschap.

Inregelen van de methode kan alleen door meetmethoden te gebruiken die niet in het model besloten liggen. Dit impliceert per definitie dat men vervalt naar Reliability Growth Modelling of Monte Carlo simulatie. Doordat componenten een initiële inschatting hebben gekregen kan men ook kijken naar de lange-termijn betrouwbaarheid. Op lange termijn zou men daardoor voldoende data kunnen verzamelen om langzaam maar zeker de empirisch onderbouwde factoren te kunnen bepalen.

Belangrijk onderdeel van de kalibratie is dat Rijkswaterstaat tijdens het toepassen van deze methode alle factoren vastlegt en opslaat in één centrale database en dat men bij een bepaalbare faalkans van een component deze ook registreert. Zo kan men langzaam op empirische wijze de factoren bepalen en onderbouwen. Voor de wetenschap is deze informatie ook van zeer grote waarde.

8 Referenties

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, *Dependability and its threats - A taxonomy*. IFIP Congress Topical Sessions 2004: 91-120

[2] B. Beizer, *Software Testing Techniques*, International Thomson Computer Press; 2nd edition, 1990

[3] M. Ben-Daya en A. Raouf. *A revised failure mode and effects analysis model*. International Journal of Quality Reliability Management, 13:43-47, 1996

[4] E.T.H. Brandt en R. P. Henzen, *Rapport Betrouwbaarheidsanalyse*, Refis Reliability Engineering, 1 maart 2005

[5] E.T.H. Brandt en R. P. Henzen, *Handboeken Betrouwbaarheidsanalyse*, Refis Reliability Engineering, 2005

[6] E.T.H. Brandt, R. P. Henzen, I. Corro Raamans en A. Di Bucchianico, *Two case studies in applying statistical models in software development*, Journal for Quality Engineering, 2007

[7] E.W. Dijksta, *A Discipline of Programming*, Prentice-Hall, 1976

[8] Ekris, J. van et al., *TOPAAS Een structurele aanpak voor faalkans analyse van software intensieve systemen*, versie 1.01, 10 december 2008

[9] W. Gilchrist. *Modelling failure modes and effects analysis*. Int. J. Quality Reliability Management, 10(5):16-23, 1993

[10] Heslinga, G., Faalkansmodel voor menselijke fouten, toegespitst op menselijk handelen bij Haringvliet- en Volkeraksluizencomplex, Dok. nr. 8140-06-062, versie : 1.1, 7 september 2006

[11] IEC61508 *Functional safety of electrical/electronic/programmable electronic safety-related systems*, CEI/IEC, 1998

[12] IEEE1471 *Recommended Practice for Architecture Description of Software-Intensive Systems*, IEEE, 2000

[13] P. Kruchten, *Architectural Blueprints—The "4+1" View Model of Software Architecture*, IEEE Software 12 (6), November 1995, pp. 42-50

[14] M. Li, Y. Wei, D. Desovski, H. Nejad, S. Ghose, B. Cukic and C. Smidts, *Validation of a Methodology for Assessing Software Reliability*, Proceedings of the 15th International Symposium on Software Reliability Engineering, 2004

[15] J.-H. Lo, C.-Y. Huang, I.-Y. Chen, S.-Y. Kuo, and M.R. Lyu. *Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure*. J. Systems Software, 76:3-13, 2005

[16] Manen, S.E. van en H.J. van der Graaf, *Evaluatie TOPAAS – Aanbevelingen voor aanpassing*, versie 3, 25 februari 2011

[17] N. Metropolis and S. Ulam, *The Monte Carlo Method*, Journal of the American Statistical Association, volume 44, number 247, pp. 335–341 (1949)

[18] Ir. R.W. van Otterloo, *Methodiek ter bepaling van de betrouwbaarheid van software modules t.b.v. OKE*, NRG rapport 20813/02.54186/C, Arnhem, 24 maart 2003

[19] A. Pillay and J. Wang. *Modified failure mode and effects analysis using approximate reasoning*. Reliability Engineering & System Safety, 79:69-85, 2003
[20] C. Smidts and D. Sova. *An architectural model for software reliability quantification: sources of data*. Rel. Eng. Sys. Safety, 64, 1999
[21] C. Smidts, M. Li, R.W. Brill, *Ranking Software Engineering Measures Related to Reliability Using Expert Opinion*, Proceedings of the 11th International Symposium on Software Reliability Engineering, 2000
[22] C. Smidts and M. Li, *A Ranking of Software Engineering Measures Based on Expert Opinion*, IEEE Transactions on Software Engineering, Vol. 29, No. 9, September 2003
[23] C. Smidts (ed.) *Preliminary Validation of a Methodology for Assessing Software Quality*, NUREG, CR6848, 2004
[24] V. Vinideep and J. Pankaj, *List of Common Bugs and Programming Practices to avoid them*, http://www.cse.iitd.ernet.in/~jalote/papers/CommonBugs.pdf, 2005
[25] W.-L. Wang, Y. Wu en M-H Chen, *An Architecture-Based Software Reliability Model*, 1999 Pacific Rim International Symposium on Dependable Computing, December 16 - 17, 1999, Hong Kong, China
[26] W.-.L. Wang, D. Pan, and M.-H. Chen. *Architecture-based software reliability modeling.* J. Syst. Software, 79:132-146, 2006.

Bijlage A SWOT Analyse

A.1 Strengths TDT methodiek

- Breed raamwerk dat alle relevante hoofdonderwerpen (te weten proces, product en track record) lijkt af te dekken. Het lijkt het 'lamp post syndrome' te vermijden door een brede aanpak van parameters te onderzoeken, hoewel diepgang hierin ontbreekt (zie de zwaktes)
- Door het gebruik van een parametermodel met relatief eenvoudige inputparameters en eenvoudige rekenwijze kan men snel tot resultaten komen
- Inputparameters voor de berekening van componenten zijn relatief eenvoudig te verkrijgen
- Werkt op Commercial Off-The-Shelf (COTS) componenten waar leveranciers geen antwoorden kunnen/willen geven.

A.2 Weaknesses TDT methodiek

- De werking van de TDT methodiek is niet geverifieerd met bestaande velddata
- De TDT-methodiek ontbreekt het aan connectie met geaccepteerde methoden
- Er wordt eigenlijk gerekend met betekenisloze factoren, welke vervolgens in een schatting worden omgezet, waarbij de relatie tussen de beïnvloedende factoren en de uiteindelijke faalkans niet expliciet gelegd wordt. Er is dan ook significante kritiek op een dergelijke aanpak (zie Ben-Daya [3], Gilchrist [9] en Pillay and Wang [19])
- In de TDT-methodiek ontbreekt het aan diepgang. Cruciale eigenschappen die zijn overgeslagen, zoals:
  - Complexiteit van beslislogica
  - Uitvoeren van reviews gedurende ontwikkeling
  - Gebruik defensive programming
  - Etc.
- De methode geeft geen indruk hoe betrouwbaar de faalkans is (is het een wilde gok op basis van een zeer beperkt aantal input parameters, of is er zeer veel bewijs dat allemaal hetzelfde antwoord geeft). Dit blijkt ook niet uit de nauwkeurigheid van het antwoord (aantal cijfers achter de komma)
- De TDT methodiek gebruikt 10^{-6} per vraag als ondergrens van de betrouwbaarheid van software, welke nogal arbitrair is qua keuze. Elke vorm van onderbouwing van deze keuze ontbreekt en deze is nogal controversieel omdat de IEC de ondergrens bij 10^{-5} legt
- Er vindt geen weging van de invloed van de factoren ten opzichte van elkaar plaats. De methodiek behandelt alle methoden van meting als gelijkwaardige bijdragen, waar wellicht de ene methode van meting/parameter een belangrijkere schatting van de faalkans is dan anderen
- De methodiek doet geen uitspraken over de wijze van inschatten van de te gebruiken beta-factoren. Er is niet beschreven hoe je tot een verantwoorde inschatting van een beta-factor komt

- Het TDT-model beschrijft niet wat een softwaremodule is, ook is het noodzakelijke detailniveau voor de uitvoering van de foutenboom niet beschreven
- De wijze van foutenboom modellering is niet beschreven in de TDT methodiek. De onderliggende software architectuur (inclusief redundantie) wordt niet beschouwd, terwijl dit van wezenlijk belang is (Lo [15], Smidts and Sova [20] en Wang et al. [26]). Het praktische gevolg hiervan is dat de methode van modeleren van de foutenboom component-georiënteerd is en niet functie-georiënteerd, waardoor de modellering niet overeenkomt met de functies die software in de totale keten vervuld. Bijvoorbeeld het "script" in de Maeslantkering is onderaan de foutenboom als object gemodelleerd, terwijl de functionele eigenschappen eigenlijk meer basisgebeurtenissen zijn (bijvoorbeeld het niet afgeven van een "sluit" commando. Hierdoor is de modellering van de besturingslogica minder intuïtief in de foutenboom verwerkt
- Variabelen benoemd in de TDT methodiek zijn benoemd, maar moeilijk te kwantificeren en/of te verifiëren.
- De foutenboom modellering lijkt te eindigen op het uitblijven van een beslissing van een enkele besturende component, niet de achterliggende systemen die aan een beslissing ten grondslag liggen.
- Het gebruik leidt tot kwetsbaarheid voor externe kritiek: de methode is te beperkt opgezet en kan eenvoudig "aangevallen" worden. Onder vuur nemen van een methodiek is eenvoudig en kan zeer veel imagoschade geven aan de Nederlandse overheid.

A.3 Opportunities nieuwe methodiek

- Er is awareness bij Rijkswaterstaat dat de TDT methodiek niet voldoet
- Er is een redelijk aantal vergelijkbare projecten in hetzelfde domein aanwezig, hetgeen gedegen vergelijkingsmateriaal oplevert voor testen van de methodiek
- De methodiek kan breder toepasbaar zijn dan alleen bij natte infrastructuur
- Het opzetten van een nieuw parametermodel draagt bij aan een verdieping van het begrip softwarebetrouwbaarheid en een consensus over een aanpak. Het concept bij software betrouwbaarheid is fundamenteel anders dan bij technische systemen waar random falen een zeer grote rol speelt:
  - Het gaat bij software betrouwbaarheid, naast het intrinsiek falen van de softwaremodule, ook in grote mate om de mogelijkheid dat zich een situatie voordoet waar de software niet voor is geschreven en het bepalen van de kans dat die situaties zich voordoen
  - De fouten ten gevolge van niet conform specificaties ontwikkelen van de software worden door middel van testen onvoldoende waargenomen om de benodigde betrouwbaarheid te borgen
- Door een lange termijn visie te combineren met korte termijn verbeteringen, wordt kritiek dat de nieuwe methode niet perfect is, ontkracht. Door het koppelen aan een lange termijn programma geeft het vertrouwen aan de buitenwacht dat er aan perfectionering van de methodiek gewerkt wordt.

A.4 Threats nieuwe methodiek

- De methodiek kan té snel breed ingezet worden in de vorm van een in te vullen spreadsheet. Bij gebrek aan getraind inzicht en terugkoppeling van gebruikers kan dit tot valse zekerheid leiden. Dit terwijl safety critical systemen vragen om een continu alerte houding
- Potentiële complexiteit van de methodiek door de vele te beschouwen factoren die software kwaliteit beïnvloeden
- Software testen is in het algemeen moeilijker dan hardwaretesten, en dus is de voorspelbaarheid van falen minder
- Kan andere modelleringstechniek afdwingen waardoor de gehele foutenboom over moet.

Bijlage B Raamwerk voor invoegen afhankelijkheden

Het beschreven framework (zie §5.4 "Invulling methode") doet geen recht aan de verminderde meeropbrengst. Daar niet alle factoren onafhankelijk zijn, worden sommige scores dubbel geteld. Daarom moet een generalisatie doorgevoerd worden.

We introduceren 'score vectoren' v^j_a . Analoog als bij de F_i is de *i* index het aspect (bijvoorbeeld, gebruik van inspecties, of het gebruikte SIL). De *a* index geeft de score aan. De $\boldsymbol{v}^j$ is een vector bestaande uit 0, behalve op index *a*. Als de SIL score loopt van 0 t/m 4 is komt SIL3 overeen met $\boldsymbol{v}^{SIL}$ = (0, 0, 0, 1, 0). Als inspecties worden weergegeven met $\boldsymbol{v}^{INSP}$ op schaal 0 (geen) 1 (informeel), 2 (formele inspecties met moderator, kick-off, logging etc.). Wordt 'informele inspecties' aangegeven met $\boldsymbol{v}^{INSP}$= (0, 1, 0).

De correlatie tussen *SIL* en *INSP* is een 3 bij 5 matrix, waarvan een voorbeeld invulling is gegeven in onderstaande tabel.

		Inspectie		
		0	1	2
SIL-level	0	0	-½	-1
	1	0	-½	-1
	2	½	0	-½
	3	NVT	1	0
	4	NVT	NVT	0

Deze matrix $R^{INSP,SIL}_{ab}$ geeft aan dat geen inspecties (vector (1,0,0)) geen toegevoegde waarde hebben voor SIL0 en SIL1 (vectoren (1,0,0,0,0) en (0,1,0,0,0)). Geen inspecties op SIL4 is niet van toepassing omdat die combinatie onmogelijk is (SIL4 sluit deze uit). Fagan inspections zijn Highly Recommended voor SIL3 en SIL4 en hebben daar dus geen toegevoegde waarde. Een toelichting op Fagan inspecties wordt gegeven in Bijlage D "Het Fagan-inspectieproces".
Generaliserend naar alle afhankelijkheden tussen alle schalen wordt de totale faalkans daarmee

$$n = \log P = \Sigma_{i,j,a,b}\, v^i_a\, R^{ij}_{ab}\, v^j_b$$

met

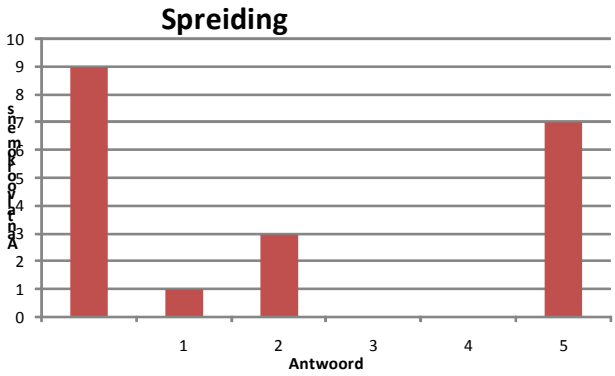
i,j = {SIL, INSP, etc.}
a,b = {0,1,...} afhankelijk van het aantal opties

Merk op dat R^{ij}_{ab} met *i=j* een vierkante matrix is met alleen maar 0 waarden.

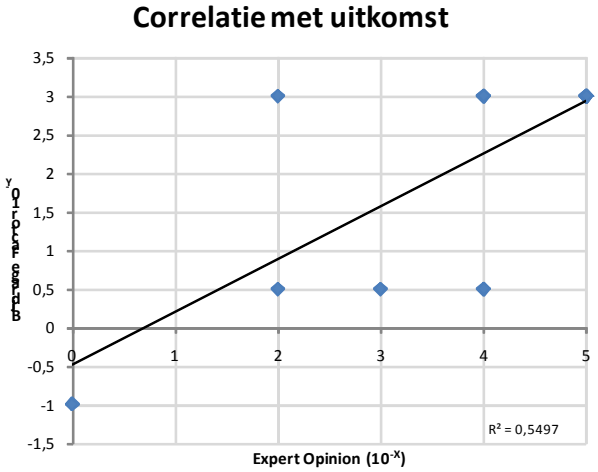
In dit document is alleen beperkte afhankelijkheid tussen SIL en enkele andere schalen meegenomen. Dit is omwille van de leesbaarheid genoteerd in overzichtelijkere tabellen. Het raamwerk staat complexere relaties toe.

Bijlage C Kalibratie model

C.1 1. Het ontwikkelproces voldoet aan een van de SIL levels van de IEC

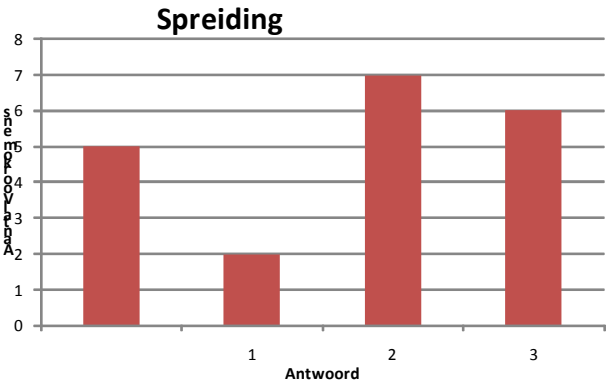


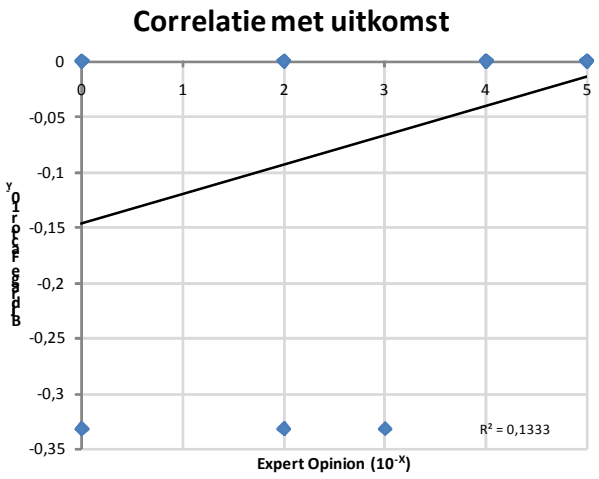
Uit deze grafiek blijkt dat er een redelijke spreiding in de antwoorden aanwezig is: 9 projecten zijn niet onderhevig geweest aan een specifiek SIL (geen antwoord gegeven), en 7 projecten zijn uitgevoerd onder een SIL-4 (antwoord 5).



N=11

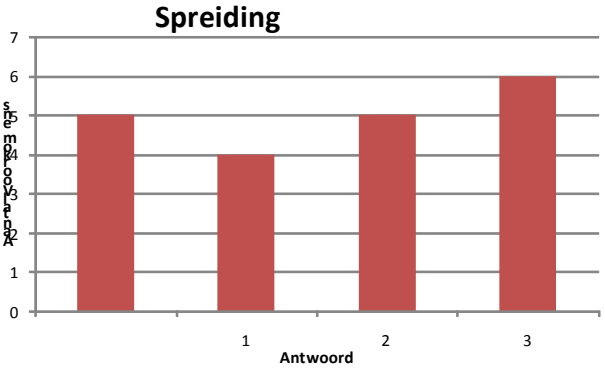
C.2 2. Gebruik van Inspecties

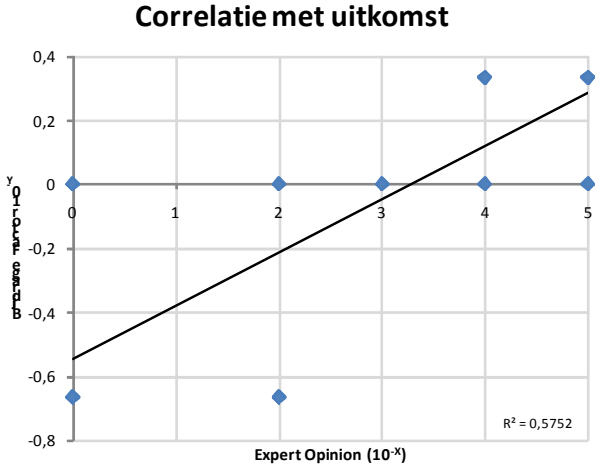




N=15

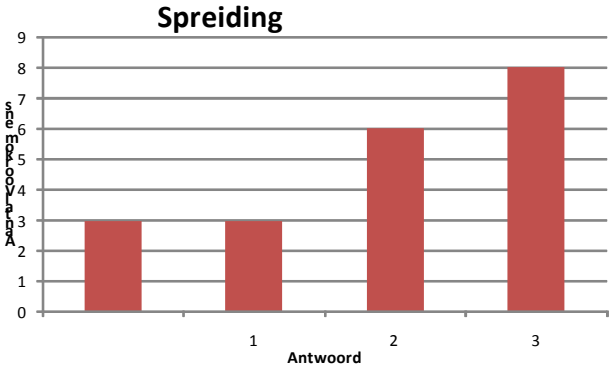
C.3 3. Hoeveelheid wijzigingen ten opzichte originele ontwerp/eisenpakket

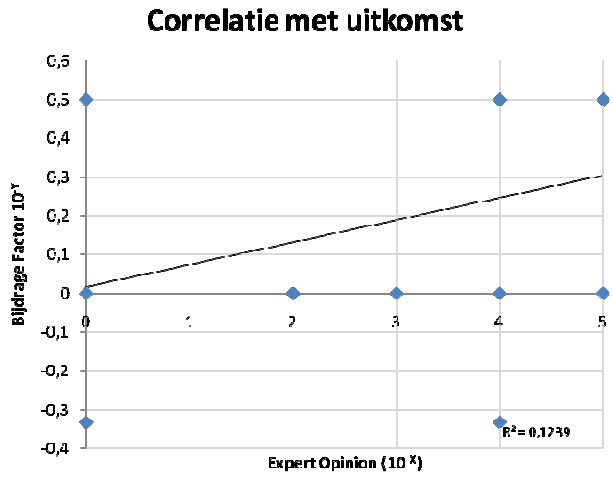




N=15

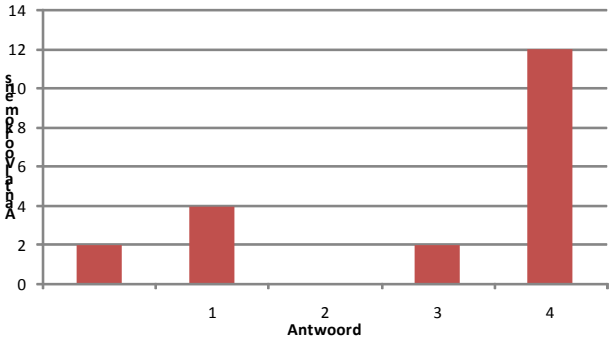
C.4 4. Cultuur en samenwerking

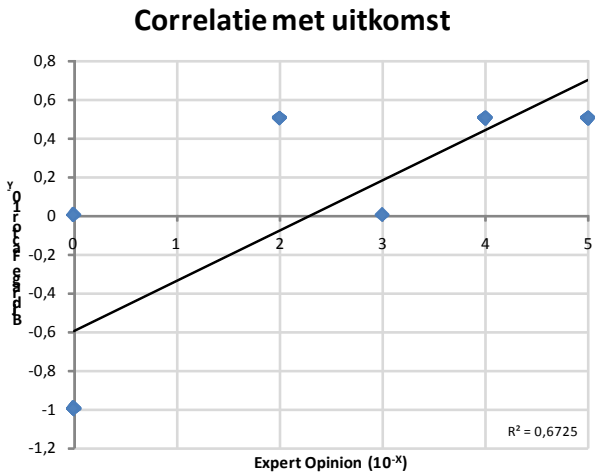




N=17

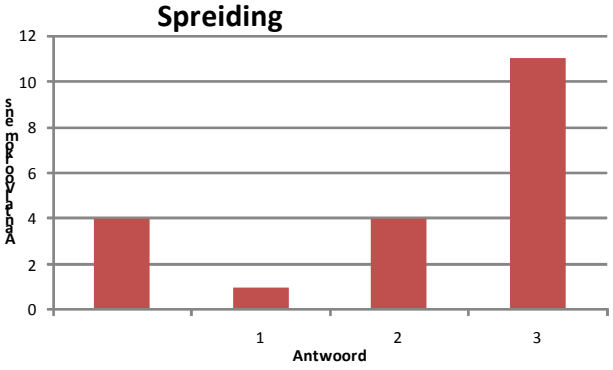
C.5 5. Opleidingsniveau en ervaring ontwikkelaars

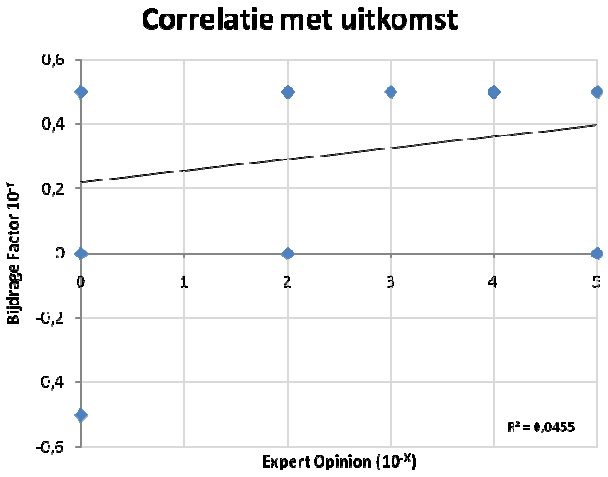




N=18

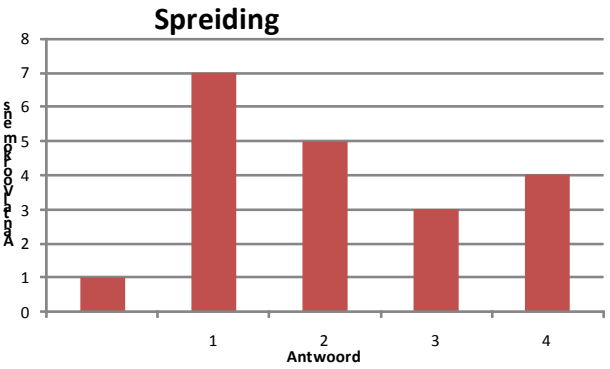
C.6 6. Samenwerking met Opdrachtgever

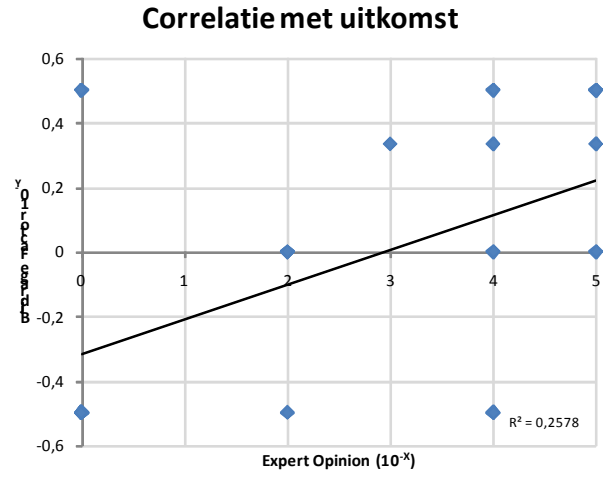




N=16

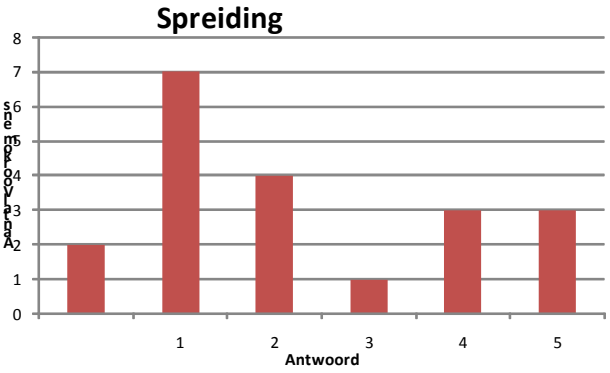
C.7 7. Complexiteit beslislogica

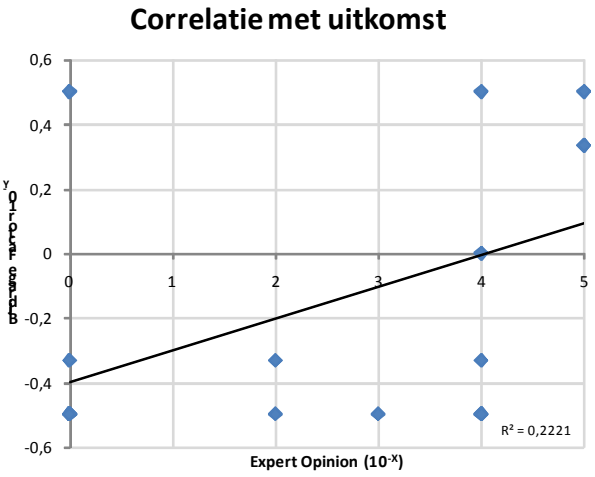




N=19

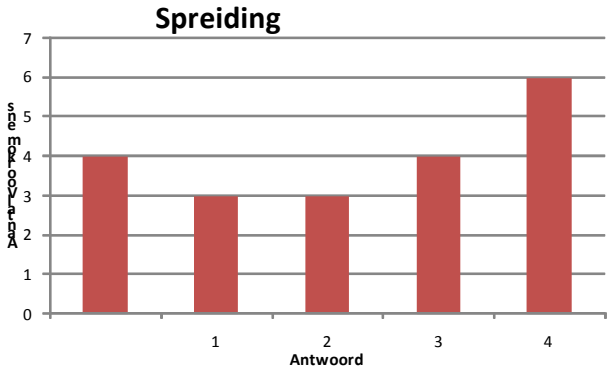
C.8 8. Omvang softwaremodule (Lines of code)

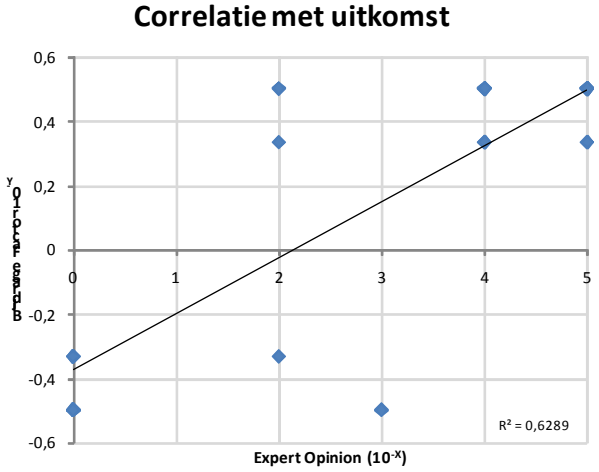




N=18

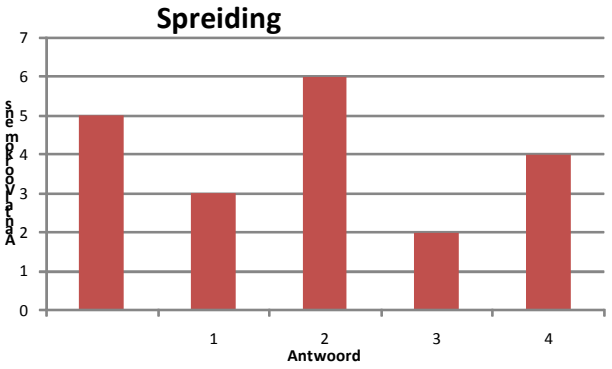
C.9 9. Helderheid gebruikte architectuurconcepten

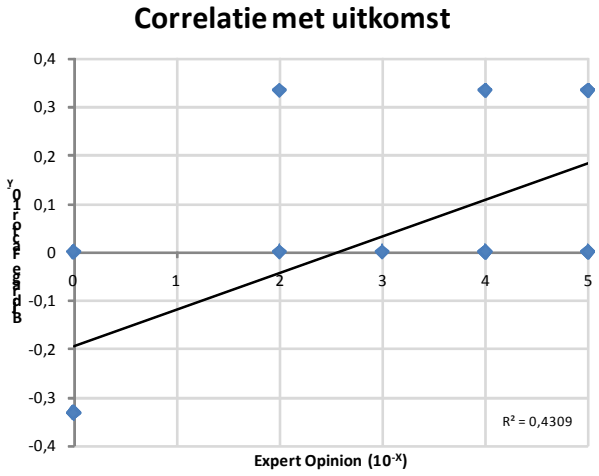




N=16

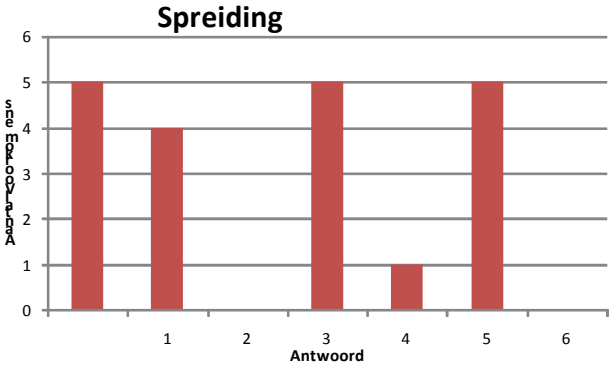
C.1010. Gebruik van een certificied compiler

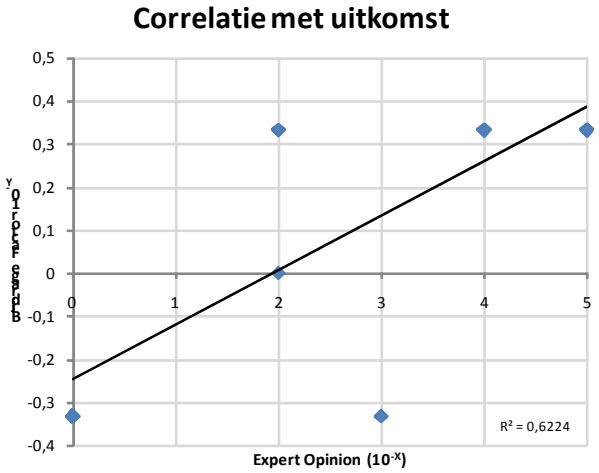




N=15

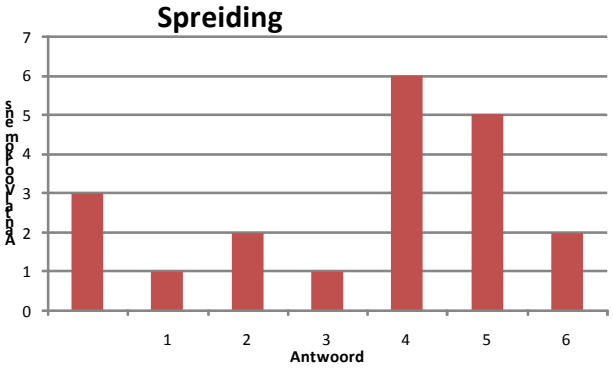
C.1111. Traceerbaarheid van requirements door het proces heen

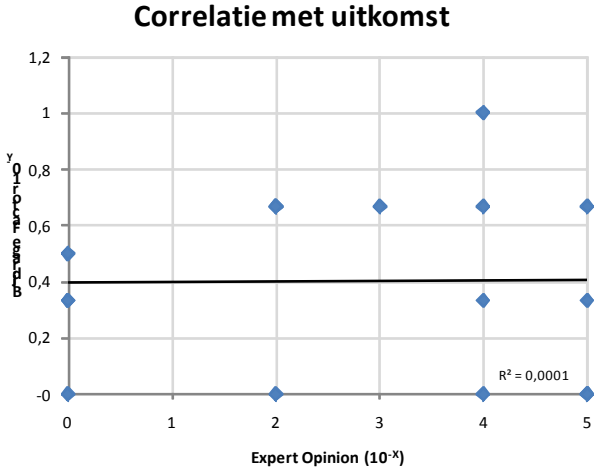




N=15

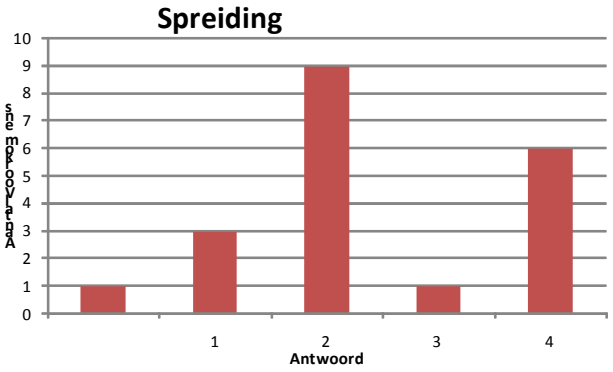
C.1212. Testtechnieken en dekkingsgraad

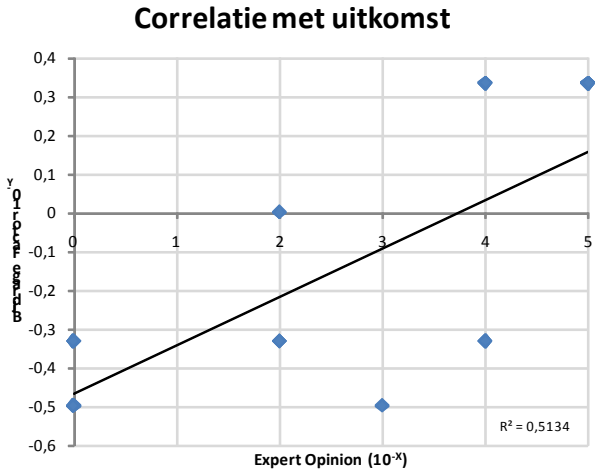




N=17

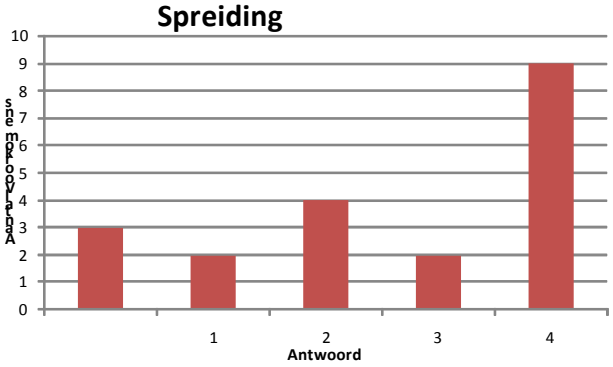
C.1313. Mulitprocesomgeving

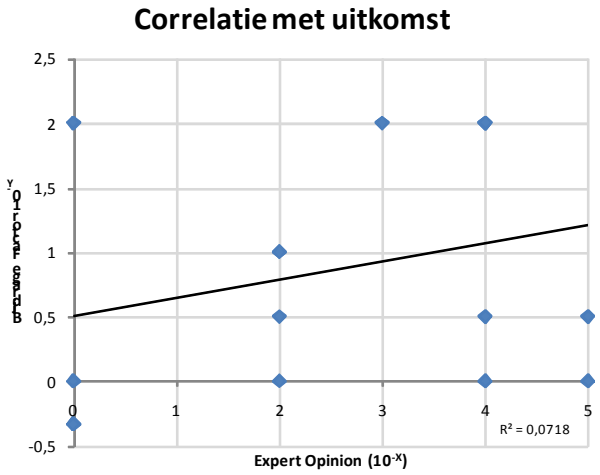




N=19

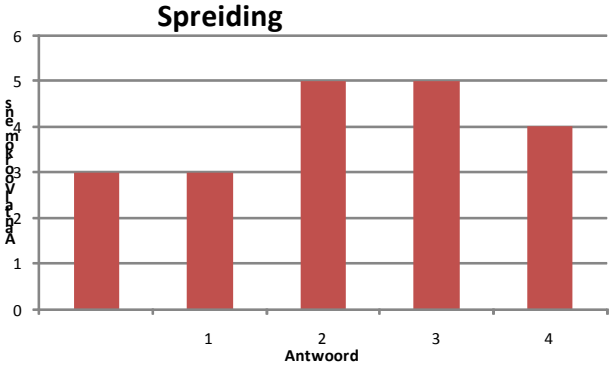
C.1414. Aanwezigheid representatieve velddata gedurende taakuitvoering

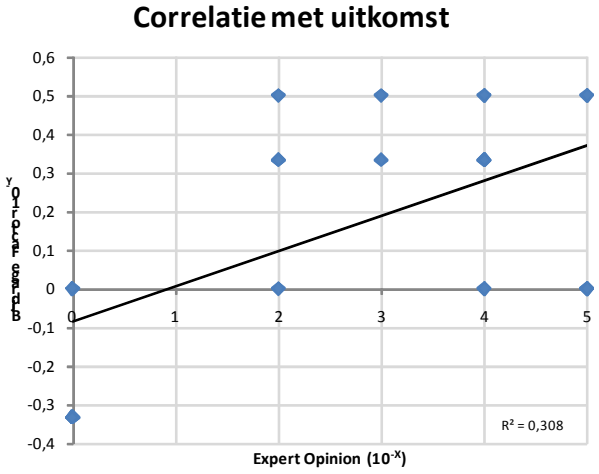




N=17

C.1515. Monitoring




Correlatie met uitkomst
0,6
0,5
0,4
0,3
0,2
0,1
0
-0,1
-0,2
-0,3
-0,4
0
1
2
3
4
5
R² = 0,308
Expert Opinion (10-X)

N=11

Bijlage D Het Fagan-inspectieproces

De Fagan-inspectie (ooit door Michael Fagan voor IBM bedacht) is een software review methodiek 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:

1. **Planning.** De moderator plant de inspectie. Hij stelt de inspectie team samen, plant de bijeenkomsten en verspreidt het benodigde materiaal.
2. **Aftrap**. [optioneel] Zo nodig wordt het te inspecteren product toegelicht door de auteur.
3. **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.
4. **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.
5. **Herstel**. De auteur herstelt alle gevonden ernstige fouten en houdt bij welke fouten gecorrigeerd zijn en hoeveel tijd het heeft gekost.
6. **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 belangrijk 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 10 keer efficiënter maakt.