

RWS INFORMATIE

[Richtlijn softwarebetrouwbaarheid]

Versienummer: 0.8

Concept

Colofon

Auteurs: Johan van den Bogaard, Ed Brandt, Jaap van Ekris, Alexander Hermanns

Uitgave: Rijkswaterstaat

Status: Concept

Versie: 0.8

Datum: 26-03-2021

Onze dank gaat uit naar Wouter Geurts, Jeroen Horstman, Yigal Levin en Bernhard Thieme voor hun inhoudelijke bijdragen en reviewcommentaren.

Inhoudsopgave

COLOFON.....	2
1 INLEIDEND HOOFDSTUK.....	6
1.1 INLEIDING	6
1.2 DOEL RICHTLIJN SOFTWAREBETROUWBAARHEID	7
1.3 RELATIE MET LEVENSCYCLUS VAN KUNSTWERKEN	8
1.4 WAT IS TOPAAS?	8
1.5 WAT BIEDT DE RICHTLIJN SOFTWAREBETROUWBAARHEID WEL EN WAT NIET?	9
1.6 UITGANGSPUNTEN	10
1.6.1 INZET MENSEN	10
1.6.2 GOVERNANCE EN PROJECTMANAGEMENT	10
1.6.3 RELATIE OPDRACHTGEVER/OPDRACHTNEMER (OG/ON).....	10
1.6.4 INTERN KWALITEITSSYSTEEM.....	10
1.7 LEESWIJZER	11
2 INLEIDING SOFTWAREBETROUWBAARHEIDSASPECTEN	13
2.1 INLEIDING	13
2.2 SOFTWARE IN EEN OBJECT	13
2.3 TAAKUITVOERING VAN SOFTWARE	13
2.4 KWALITEIT VAN SOFTWARE.....	14
2.5 FOUTEN, FALEN, VEILIGHEIDSKRITISCH FALEN EN BETROUWBAARHEID	14
2.5.1 BETROUWBAARHEID VAN SOFTWARE	15
2.5.2 VEILIGHEIDSKRITISCH FALEN EN MISSIEKRITISCH FALEN	15
2.6 FAALGEDRAG VAN SOFTWARE IN RAMS-TERMEN	16
2.6.1 SYSTEMATISCH OF RANDOM FALEN	16
2.6.2 EFFECTIVITEIT VAN TESTEN	17
2.6.3 MERKBAARHEID FALEN IN GEBRUIK	17
2.6.4 FAALGEDRAG VAN SOFTWARE IN DE TIJD	18
2.6.5 SOFTWAREHERSTEL.....	19
2.7 AANVAARDBAARHEID VAN RISICO	19
2.8 DE ROLLEN VAN SOFTWARE IN DE INSTALLATIE	20
2.9 BRONNEN VAN SOFTWAREFALEN	21
2.10 GEACCEPTEEERDE METHODES VOOR HET REALISEREN EN BEHOUDEN VAN DE BETROUWBAARHEID.....	22
2.11 DE LEVENSCYCLUS VAN SOFTWARE	23
3 FASE 0: AANTOONBAARHEID VAN SOFTWAREBETROUWBAARHEID.....	25
3.1 INLEIDING	25
3.2 DOSSIERVORMING	25

4	FASE 1: EISEN EN SPECIFICATIES	27
4.1	EISEN AAN BETROUWBAARHEID EN BESCHIKBAARHEID	27
4.1.1	KWALITEIT VAN DE BESCHREVEN EISEN	27
4.1.2	GREENFIELD-BEPALING GEAMBIËERDE FAALKANS	29
4.1.3	BROWNFIELD-BEPALING FAALKANS	29
4.1.4	SAFETY INTEGRITY LEVELS	29
4.2	OMGANG MET EEN-OP-EENRENOVATIE VAN SYSTEMEN	30
4.3	ACTIVITEITEN	31
4.4	OP TE LEVEREN PRODUCTEN/DOCUMENTEN	31
5	FASE 2: ONTWERP VAN DE OPLOSSING	32
5.1	ORIËNTATIE	32
5.1.1	ACCEPTATIE VAN DE EISEN	32
5.2	SYSTEEMARCHITECTUUR	32
5.2.1	OVERWEGINGEN IN DE SYSTEEMARCHITECTUUR EN BETROUWBAARHEID/BESCHIKBAARHEID 33	
5.2.2	BESCHRIJVING VAN DE ARCHITECTUUR	34
5.2.3	TRACEERBAARHEID VAN EISEN	35
5.2.4	COMPETENTIES VAN DE ONTWERPERS	35
5.2.5	ONTWERPRICHTLIJNEN	36
5.2.6	HERGEBRUIK VAN COMPONENTEN	36
5.2.7	ONTWERPREVIEWS	36
5.2.8	OMGANG MET MODEL DRIVEN DEVELOPMENT	37
5.2.9	INDEPENDENT VERIFICATION AND VALIDATION (IV&V)	37
5.3	BORGING VAN PROCESSEN EN PRODUCTEN	37
5.4	ACTIVITEITEN	38
5.5	OP TE LEVEREN PRODUCTEN/DOCUMENTEN	38
6	FASE 3: REALISATIE	39
6.1	ONDERSTEUNENDE TOOLS EN SOFTWAREBIBLIOTHEKEN	39
6.1.1	ZEKERHEID VAN DE KWALITEIT VAN DE COMPILER	39
6.1.2	ZELFGEMAAKTE GENERATIETOOLS	39
6.1.3	SOFTWAREBIBLIOTHEKEN	40
6.2	EXTERNE BORGING CODEKWALITEIT DOOR SIG/EXPLEO/ETC.	40
6.3	ACTIVITEITEN	41
6.4	OP TE LEVEREN PRODUCTEN/DOCUMENTEN	41
7	FASE 4: TESTEN	42
7.1	ONAFHANKELIJKHEID VAN DE TESTORGANISATIE	42
7.2	TESTSTRATEGIE	42
7.3	BEPALING DEKKINGSGRAAD TESTEN	43
7.4	TRACERBAARHEID VAN EISEN NAAR TESTEN	44

7.5	TESTOMGEVINGEN	44
7.6	REGISTRATIE VAN TESTRESULTATEN	45
7.7	AFHANDELING VAN AFWIJINGEN	45
7.8	RELEASEMANAGEMENT EN VRIJGAVE NAAR DE PRODUCTIE	46
7.9	TESTEN VAN MODIFICATIES EN REGRESSIETESTEN	46
7.10	ACTIVITEITEN	46
7.11	OP TE LEVEREN PRODUCTEN/DOCUMENTEN.....	47

8 FASE 5: GEBRUIK, BEHEER & ONDERHOUD 48

8.1	INLEIDING	48
8.2	VORMEN VAN ONDERHOUD EN CONSEQUENTIES VOOR SOFTWARE.....	48
8.3	WIJZIGINGSBEHEER	50
8.4	PROCES VAN WIJZIGING	52
8.4.1	CORRECTIEF ONDERHOUD	52
8.4.2	PREVENTIEF EN PERFECTIEF ONDERHOUD	52
8.4.3	ADAPTIEF EN ADDITIEF ONDERHOUD	53
8.4.4	ANDERE AANLEIDINGEN VOOR ONDERHOUD.....	53
8.4.5	SPECIFIEKE STAPPEN IN HET V-MODEL	54
8.5	LOGGING EN BEWAKING IN RELATIE TOT SOFTWAREBETROUWBAARHEID	55
8.5.1	INHOUD LOGS	55
8.5.2	BEWAARTERMIJN LOGS.....	56
8.5.3	PERIODIEKE REVIEW	56
8.6	BAYESIAANS UPDATEN	56
8.7	OVERIG GEBRUIK, BEHEER EN ONDERHOUDSASPECTEN	58
8.7.1	ROOT CAUSE ANALYSIS (RCA)	58
8.7.2	ACCEPTATIE-OMGEVING	58
8.7.3	REGRESSIETESTEN	59
8.7.4	RELIABILITY GROWTH MODELLING (RGM)	59
8.7.5	SERVICE LEVEL AGREEMENT (SLA)	60
8.8	ACTIVITEITEN	61
8.9	OP TE LEVEREN PRODUCTEN/DOCUMENTEN	62

9 OPSTELLEN EN GEBRUIK VAN AANBESTEDINGSEISEN..... 63

9.1	INLEIDING	63
9.2	PROBLEEMSTELLING.....	63
9.3	EISEN IN HET VOORTRAJECT	64
9.4	ALGEMENE AANDACHTSPUNTEN	64
9.5	SPECIFIEKE EISEN AAN PROCESSEN	65
9.6	SPECIFIEKE EISEN AAN DE ORGANISATIE	66
9.7	TE TOETSEN PRODUCTEN.....	67

1 Inleidend hoofdstuk

1.1 Inleiding

Rijkswaterstaat heeft de richtlijn Softwarebetrouwbaarheid opgesteld om te zorgen voor een betere betrouwbaarheid van de software in de levenscycli van de infrastructuur van Rijkswaterstaat. De richtlijn richt zich op softwarebetrouwbaarheid voor alle fasen van de levenscyclus van kunstwerken.

De richtlijn moet in samenhang worden gezien met de Leidraad Systems Engineering [#1481], de Handreiking Verificatiemethode Betrouwbaarheid en Beschikbaarheid [#1567] en de Handreiking Prestatiegestuurde Risicoanalyses [#5333].

Rijkswaterstaat maakt een transitie door van een civieltechnische organisatie naar een meer hybride organisatie waarin industriële automatisering (IA) een steeds grotere rol speelt. Deze transitie gaat niet vanzelf. Het is de bedoeling dat deze richtlijn Rijkswaterstaat met deze transitie gaat helpen via bekende Rijkswaterstaatkaders en duidelijke definities en procesbeschrijvingen binnen het IA-domein.

De projectbudgetten worden voor een relatief klein deel aan software besteed, en voor het grootste deel aan de civieltechnische delen. Daardoor lijkt software van weinig belang te zijn voor het succesvol afronden van een project. In de praktijk blijkt software echter steeds vaker een kritieke factor te zijn voor het slagen of falen van projecten en van de functievervulling van kunstwerken. Het gevolg daarvan is dat Rijkswaterstaat de betrouwbaarheid van software steeds meer centraal gaat stellen. Deze richtlijn geeft een aanzet voor de volgende stap op het gebied van betrouwbaarheid van software en de voor de koppeling daarvan met het prestatieniveau van een kunstwerk als geheel.

Software speelt een cruciale rol in het bedienen, besturen en beveiligen van veel civiele en technische objecten. Het zorgt ervoor dat het gespecificeerde dynamische gedrag/de functionaliteit van een systeem wordt uitgevoerd: wanneer de software niet goed functioneert, is het mogelijk dat het object zijn gespecificeerde functie niet kan uitvoeren, ongepland niet-beschikbaar is, of zelfs onveilig is voor mens en omgeving. Dit faalgedrag is vaak onvoorspelbaar doordat software enorm complex is. Ook leert de praktijk dat twijfel over de kwaliteit van de gebruikte software regelmatig leidt tot uitstel van ingebruikname van infrastructurele kunstwerken. Kortom, software is een essentieel onderdeel geworden van de veiligheid en beschikbaarheid van een kunstwerk.

Aangezien software tot de factoren behoort die ervoor verantwoordelijk kunnen zijn dat een kunstwerk haar functie niet kan uitvoeren, is het vanzelfsprekend dat software wordt meegenomen in de kwantitatieve onderbouwing van de risicoanalyses om zo een goed inzicht te geven in de prestatie van een kunstwerk als geheel.

Om de rol van software te kunnen opnemen in veiligheids- en betrouwbaarheidsanalyses is TOPAAS ontwikkeld. TOPAAS heeft tot doel de faalkans van software kwantitatief te beoordelen zodat die meegenomen kan worden in de risicoanalyse van een kunstwerk als geheel. TOPAAS is in 2008 ontwikkeld in samenwerking met toonaangevende kennisinstanties, softwareontwikkelaars en

wetenschappers. In 2018 heeft dezelfde groep een verbeterde versie van TOPAAS ontwikkeld, die nu binnen Rijkswaterstaat geldt.

TOPAAS is een instrument om de softwarebetrouwbaarheid van een kunstwerk goed te kunnen meten binnen Prestatiegestuurde Risicoanalyses. Het biedt echter onvoldoende handvatten bij de uitvraag en aanbesteding van nieuwe, betrouwbare softwaresystemen. Ook blijkt het lastig om tijdens de ontwikkel- en realisatiefase van softwaresystemen de kwaliteit van software te volgen en tijdig bij te sturen om tot het gewenste resultaat te komen.

Deze Richtlijn Softwarebetrouwbaarheid biedt de benodigde handvatten om ervoor te zorgen dat de kwaliteit van software tijdens de hele levenscyclus van kunstwerken voldoet aan de prestatie-eisen. TOPAAS maakt integraal onderdeel uit van de werkwijze die in deze richtlijn staat beschreven.

1.2 Doel richtlijn Softwarebetrouwbaarheid

Doel van de richtlijn Softwarebetrouwbaarheid is opdrachtgevers (zoals Rijkswaterstaat) en de markt (zoals aannemers, adviesbureaus, leveranciers) een werkwijze aan te bieden waarmee de betrouwbaarheid van software over de hele levenscyclus van de systemen goed kan worden beschreven, gevolgd, bijgestuurd en vastgesteld.

De richtlijn geeft een elementaire uitleg van softwarebetrouwbaarheid en biedt een standaard in taal en afspraken voor Rijkswaterstaat en de markt. Dit stelt medewerkers in staat om beter met elkaar te communiceren over dit onderwerp. Naar analogie van het welbekende V-model worden de verschillende fasen in de levenscyclus van software beschreven. Voor elke fase wordt een overzicht van de belangrijkste software-elementen opgesteld en wordt een samenvatting gegeven van de activiteiten en producten die nodig zijn om tot betrouwbare software te komen. De richtlijn sluit zoveel mogelijk aan op bestaande internationale standaarden voor softwarebetrouwbaarheid, in het bijzonder de IEC 61508 'Functional safety of electronic safety-related systems', ISO 12207 'Software life cycle processes' en de IEC 62061 'Safety of machinery'.

Deze richtlijn is geschreven voor de volgende doelgroepen, zowel aan opdrachtgeverszijde als aan de marktzijde:

- **RAMS-specialisten**: zij kwantificeren de softwarebetrouwbaarheid en integreren softwarebetrouwbaarheid in de risicoanalyses van een compleet kunstwerk. De richtlijn geeft RAMS-specialisten inzicht in de belangrijkste elementen van softwarebetrouwbaarheid, maakt hen bekend met het proces om tot betrouwbare software te komen en plaatst de kwantificeringsmethode TOPAAS in de context van softwarebetrouwbaarheid in de levenscyclus van software.
- **Software-engineers**: zij zijn betrokken bij de aanbesteding, ontwikkeling, realisatie, het beheer en onderhoud en de beoordeling van software. De richtlijn biedt hen duidelijkheid over de manier waarop Rijkswaterstaat omgaat met softwarebetrouwbaarheid binnen projecten.
- **Projectmanagers**: zij zijn verantwoordelijk voor de uitvoering van een project en faciliteren het project zodanig dat een goede afronding mogelijk is. Zij zijn tevens verantwoordelijk voor de kwaliteitsbeheersing van de software binnen het project. De richtlijn volgt het binnen Rijkswaterstaat welbekende V-model, dat alle hoofdstappen van een project beschrijft.

- Inkopers: zij zijn verantwoordelijk voor het aanbesteden van projecten. Zij moeten er bij de aanbesteding voor zorgen dat de gespecificeerde softwarebetrouwbaarheid goed contractueel wordt uitgevraagd en dat een gekwalificeerde opdrachtnemer wordt geselecteerd. De richtlijn reikt een aantal eisen aan om softwarebetrouwbaarheid goed uit te vragen.

Het in deze richtlijn beschreven proces beschouwt Rijkswaterstaat als de geaccepteerde werkwijze voor de beheersing van softwarebetrouwbaarheid.

1.3 Relatie met levenscyclus van kunstwerken

De minister van Infrastructuur en Waterstaat stelt beleidsdoelen op voor de netwerken die Rijkswaterstaat beheert. Rijkswaterstaat gebruikt een prestatiegestuurde werkwijze voor het realiseren van de beleidswensen. Hierbij wordt het beoogde prestatieniveau bepaald in termen van RAMS, waarbij RAMS een acroniem is voor betrouwbaarheid (R), beschikbaarheid (A), onderhoudbaarheid (M) en veiligheid (S).

Systems Engineering en RAMS zijn de standaard prestatiegerichte werkwijzen voor de ontwerp-, ontwikkel- en realisatiefase en voor de gebruiks-, beheers- en onderhoudsfase. Deze standaarden maken het mogelijk de prestatiebehoeften goed te analyseren en helpen een vertaling te maken naar passende prestatie-eisen voor de software. Zij de eisen eenmaal vastgesteld, dan gaat het ontwerp- en ontwikkelproces van start. Tijdens de realisatiefase komt het ontwerp tot stand en wordt het uiteindelijke product op vastgestelde momenten gevalideerd en geverifieerd aan de hand van de gestelde prestatie-eisen. Na de realisatiefase wordt het object voor de bruikbare levensduur beheerd en onderhouden. Hierbij wordt ervoor gezorgd dat het prestatieniveau gedurende de hele levensduur van het object gewaarborgd blijft. De hierboven gegeven beschrijving van het V-model is lineair en gangbaar bij civiele en technische systemen. Binnen softwareontwikkeling is een trend ingezet van lineaire ontwikkeling naar meer adaptieve werkwijzen, zoals Agile, DevOps en Continuous Delivery / Integration methoden.

Merk op dat civiele constructies vaak heel lang meegaan, technische installaties vaak zo'n 15-30 jaar en dat software vaak een veel kortere levenscyclus van 10-15 jaar kent.

1.4 Wat is TOPAAS?

TOPAAS staat voor Task Oriented Probability of Abnormalities Analysis for Software. TOPAAS is de methode die binnen Rijkswaterstaat gebruikt wordt om softwarebetrouwbaarheid te kwantificeren. In deze Richtlijn Softwarebetrouwbaarheid wordt ervan uitgegaan dat gewerkt wordt met de laatste vigerende versie van TOPAAS. TOPAAS is een belangrijk instrument binnen deze richtlijn.

De belangrijkste kenmerken van TOPAAS zijn:

- TOPAAS is gemaakt voor industriële systemen die een merkbare invloed hebben op RAMS-prestaties van kunstwerken en industriële procesautomatisering, waarvan de betrouwbaarheid niet of niet eenvoudig via een andere weg is vast te stellen. Dit kunnen zowel nieuw ontwikkelde systemen zijn als COTS-producten (Commercial-Of-The-Shelf systemen).
- TOPAAS kijkt gericht naar taakuitvoering door software. Software is in een RAMS-analyse geen amorf object en elke fout heeft direct gevolgen voor betrouwbaarheid, beschikbaarheid of veiligheid. In een systeem vervult

software een afgebakende taak (taakuitvoering) die mogelijk essentieel is voor één of meerdere RAMS-eisen.

- TOPAAS is een Bayesiaanse benadering van de faalkans, die middels een parametermodel een TOPAAS-score oplevert voor betrouwbaarheid van de taakuitvoering.
- De TOPAAS-score leidt tot een faalkans per vraag (Q), die als basisgebeurtenissen te gebruiken is in een kwantitatieve risicoanalyse, zoals een foutenboom. Deze faalkans is op ordegroottes nauwkeurig omdat het slechts een afschatting is.
- Deze faalkans is een a-priorische faalkans die gebruikt kan worden om na praktijkervaringen verder Bayesiaans te updaten met daadwerkelijke prestaties. (Zie voor Bayesiaans updaten [paragraaf 8.5.4]).
- TOPAAS neemt alle vormen van afwijkend gedrag in ogenschouw die een RAMS-doelstelling in gevaar kunnen brengen, voor zover ze uit de software komen. Dus niet alleen afwezig, te vroeg of te laat gedrag, maar ook ongewenst gedrag. Gedrag van de omgeving (systemen die buiten hun specificaties lopen) of falende hardware worden expliciet niet meegenomen in TOPAAS. Dit moet in de RAMS-analyse zelf worden opgepakt.
- TOPAAS gaat uit van een context waarin de relevante faalwijzen geïdentificeerd worden in de risicoanalyse. Het gebruik van risicoanalyses impliceert ook dat de software op een gedegen wijze ontworpen, gebouwd en getest is.
- TOPAAS is ontwikkeld als meetinstrument, maar het onderliggende gedachtegoed wordt binnen deze richtlijn Softwarebetrouwbaarheid toegepast in alle fasen van systeemontwikkeling: ontwerp, realisatie, testen en productie.
- Bij de ontwikkeling van TOPAAS is de standaard IEC 61508 leidend geweest.

Kort gezegd: TOPAAS is een methode om de faalkans van software te beoordelen en dient niet als handleiding om betrouwbare software te maken.

1.5 Wat biedt de richtlijn Softwarebetrouwbaarheid wel en wat niet?

De richtlijn Softwarebetrouwbaarheid omvat een gestructureerde werkwijze die erop gericht is om de software van een kunstwerk over de hele levenscyclus aan de gestelde betrouwbaarheidseisen te laten voldoen. Het is echter geen volledige process engineering tool om de ondergrens van de ontwikkelinspanning te bepalen (zoals de IEC 61508 is opgezet). De richtlijn Softwarebetrouwbaarheid vormt op basis van beste praktijken en standaarden, zoals de IEC 61508, een praktisch 'kookboek' om tot een gewenst prestatieniveau te komen. Deze richtlijn en de IEC 61508 en ISO 12207 zijn complementair aan elkaar.

De richtlijn vervangt niet de deskundigheid die mensen opdoen in opleidingen en door jarenlange ervaring. Ze is ook geen cursusboek om deze deskundigheid op te doen. Deze richtlijn laat aan de hand van terminologie en een procesbeschrijving zien hoe Rijkswaterstaat met softwarebetrouwbaarheid in de levenscyclus van kunstwerken werkt. Een belangrijk onderdeel daarvan is hoe de faalkans van software in de verschillende fasen van de levenscyclus bepaald kan worden, wat nodig is voor de overkoepelende risicoanalyse van een kunstwerk.

Een belangrijk verschil tussen deze richtlijn en bijvoorbeeld de IEC 61508 is dat de IEC 61508 zich hoofdzakelijk richt op de betrouwbaarheid van de veiligheidsfuncties van software, terwijl deze richtlijn zich met name richt op de betrouwbaarheid en beschikbaarheid van de missie kritische functies van een kunstwerk. Het

uitgangspunt daarbij is dat voor veiligheidsfuncties de IEC 61508 gevolgd en gerespecteerd wordt.

1.6 Uitgangspunten

Bij het gebruik van deze richtlijn gelden de volgende generieke uitgangspunten.

1.6.1 *Inzet mensen*

Een belangrijke factor om tot goede software te komen is de inzet van gekwalificeerde, getrainde en ervaren mensen. Mensen die veel ervaring hebben met het ontwerpen en bouwen van software. Maar ook ontwerpers die het infrastructurele domein goed kennen, en de specifieke problemen die bij de toepassing van technologie in dat domein komen kijken.

1.6.2 *Governance en projectmanagement*

Een project wordt meestal uitgevoerd door een opdrachtnemer die in opdracht werkt van de opdrachtgever, Rijkswaterstaat. Belangrijke factoren in het uiteindelijke resultaat van het project zijn tijd, druk en de kwaliteit van de samenwerking. Deze richtlijn hanteert als uitgangspunt dat projectmanagement leiding geeft aan het project en stuurt op een kwalitatief goed resultaat. Deze richtlijn stelt voorwaarden aan het proces binnen het project, maar ook aan het projectmanagement en de mensen die aan het project werken. Een professionele opdrachtgever moet zorgen dat de opdrachtnemer zijn werk naar behoren kan uitvoeren. Een opdrachtnemer moet zijn werk op een professionele manier inrichten en gekwalificeerde mensen gebruiken.

1.6.3 *Relatie opdrachtgever/opdrachtnemer (OG/ON)*

Los van alle processen en technische maatregelen zijn rust, reinheid, regelmaat en richting belangrijke onderdelen van een project: goed opgeleide professionals moeten de ruimte en tijd krijgen om hun werk goed te doen. De communicatie naar alle stakeholders moet transparant zijn, zodat alle betrokkenen weten waar ze aan toe zijn en hoe het project verdergaat. Er moet een basis van vertrouwen zijn waarop binnen het project voortgebouwd kan worden. Dit betekent ook dat wederzijdse projectbelangen gerespecteerd en besproken worden. De juiste projectopzet en samenwerking heeft ook een positief effect op de softwarebetrouwbaarheid, wat zich vertaalt in een betere TOPAAS-score.

1.6.4 *Intern kwaliteitssysteem*

Bij aanbestedingen van RWS is een ISO 9001 kwaliteitssysteem vereist. Binnen het softwareontwikkelproject verwacht men dan ook dat de ontwikkelorganisatie via audits toetst of het project-specifieke kwaliteitsplan gevolgd wordt. Bij veiligheidskritische systemen moet dit kwaliteitsplan daarnaast nog voldoen aan de voorschriften van IEC 61508, waarmee het dus ambitieuzer is dan een gemiddeld kwaliteitsplan voor de ontwikkeling van software.

Het kan dus gebeuren dat er bij het toetsen van een kwaliteitssysteem afwijkingen geconstateerd worden. Dit is acceptabel mits:

- het kwaliteitssysteem zelf (indien nodig) aangepast wordt om herhaling te voorkomen. In de praktijk is dit het dichtdraaien van de kraan;
- auditbevindingen worden behandeld als steekproefresultaten;
- geconstateerde fouten gecorrigeerd worden met een gerichte en beheerste actie. Hierbij moet men retrospectief te werk gaan; mogelijk constateert men dat een reviewproces onvoldoende grondig is uitgevoerd. In dat geval moeten alle ontwerpen die daar gevolgen van

zouden kunnen ondervinden, opnieuw gereviewd worden (en niet alleen die waarbij het in eerste instantie geconstateerd is). Dit is dweilen met de kraan open.

Het kan zijn dat hierdoor een restrisico aanwezig blijft. Bijvoorbeeld doordat dweilacties een lange doorlooptijd hebben, waardoor vrij laat in het project nog ontwerpfouten worden geconstateerd. Dit moet worden opgenomen in het risicodossier van het project.

Het is belangrijk om afwijkingen niet te snel de contractuele sfeer in te trekken. Een opdrachtnemer moet de ruimte en tijd krijgen om het proces te verbeteren. Een te gespannen omgang met afwijkingen kan ertoe leiden dat de opdrachtnemer mooi weer speelt om de audits maar ongeschonden te doorstaan. Het zal duidelijk zijn dat een kwaliteitssysteem dat goed functioneert en afwijkingen planmatig oplost, waardevoller is dan vlekkeloze audits.

1.7 Leeswijzer

Hoofdstuk 2 is bestemd voor alle doelgroepen. In dit hoofdstuk worden basisbegrippen en softwarebetrouwbaarheidsaspecten gedefinieerd die als basis dienen voor het vervolg van deze richtlijn. Daarnaast reikt het een gemeenschappelijke taal aan, zodat binnen projecten en naar buiten toe eenduidig en duidelijk kan worden gecommuniceerd. Ook wordt in dit hoofdstuk de levenscyclus van software beschreven, die afwijkt van de gangbare levenscyclus van technische en civiele systemen.

Hoofdstuk 3 gaat specifiek over het begrip aantoonbaarheid en is relevant voor alle lezers. Aantoonbaarheid en aantoonbaar werken zijn belangrijke begrippen om tot een goede kwaliteit van systemen te komen. Aantoonbaarheid van softwarebetrouwbaarheid is een begrip dat van belang is voor alle hoofdstukken in de richtlijn.

In de overige hoofdstukken van de richtlijn worden de hoofdstappen uit het V-model gevolgd.

Hoofdstuk 4 gaat over eisen en specificaties. Het beschrijft alle aspecten die van belang zijn voor het opstellen van eisen en specificaties voor softwarebetrouwbaarheid.

Hoofdstuk 5 behandelt het ontwerp van de oplossing, wat in dit geval neerkomt op het ontwerp- en validatieproces van de software. Onderwerpen die aan bod komen zijn onder andere de oriëntatiefase, systeemarchitectuur en beschrijving van de architectuur, redundantie en multiprogramming in relatie tot betrouwbaarheid en beschikbaarheid, het kwaliteitssysteem en de verificatie en validatie van de eisen.

Hoofdstuk 6 beschrijft de realisatiefase van software. Er wordt ingegaan op ondersteunende tools en softwarebibliotheken, compilers en generatietools en externe borging van de codekwaliteit.

Hoofdstuk 7 gaat over de testfase. Het gaat in op de teststrategie en de onafhankelijkheid van de testorganisatie, de traceerbaarheid van eisen, de te gebruiken testomgevingen, de registratie van testresultaten, de afhandeling van afwijkingen en het uiteindelijke vrijgeven van de producten.

Hoofdstuk 8 behandelt de fase die komt nadat alle testen succesvol zijn afgerond en de software is opgeleverd, namelijk die waarin de software in gebruik wordt

genomen, wordt beheerd en onderhouden. Dit hoofdstuk beschrijft onder meer alle onderhoudsvormen, het wijzigingsbeheer, logging en bewaking en het Bayesiaanse updaten van faalkansen.

Hoofdstuk 9 reikt tot slot een set met eisen aan die gebruikt kunnen worden bij het uitvragen en opstellen van contracten voor softwareproducten.

2 Inleiding softwarebetrouwbaarheidsaspecten

2.1 Inleiding

Softwarebetrouwbaarheid is een complex onderwerp dat veel spraakverwarring met zich meebrengt. In dit hoofdstuk brengen we de belangrijkste aspecten van softwarebetrouwbaarheid met elkaar in verband. We doen dit om de basis te leggen voor de volgende hoofdstukken en om de terminologie te harmoniseren.

2.2 Software in een object

Software vormt steeds vaker een belangrijk onderdeel van de functie van objecten van RWS. Alhoewel de grootste financiële kosten worden gemaakt in het civiele en elektromechanische deel van een object, worden de veiligheid en betrouwbaarheid vaak bepaald door het dynamische gedrag dat de software tot stand brengt.

Met software bedoelen we alle logica die de bediening, besturing en beveiliging van een object mogelijk maakt, met uitzondering van zuiver hardwarematige (relaisgeschakelde) besturingen. Dit loopt uiteen van Bedienconsole en sensoren tot de fysieke aandrijvingen.

Met hardware bedoelen we de fysieke omgeving die de bediening faciliteert (middels een gebruikersinterface) en die de besturing en beveiliging van een object feitelijk uitvoert, en waarvan het gedrag wordt bepaald door software.

Software en hardware kunnen een vergelijkbare betrouwbaarheid bereiken, maar de aard van de problematiek is anders. In de praktijk blijkt het behalen van een aantoonbaar hoge betrouwbaarheid in software vaak moeilijker. Waar in civiele en elektromechanische constructies namelijk relatief eenvoudig betrouwbaarheid gerealiseerd kan worden met ontwerpmarges, overdimensionering en sterkteberekeningen, ontbreken dergelijke strategieën voor software: daar liggen vaak (verborgen) ontwerpfouten ten grondslag aan verkeerd of ongewenst gedrag.

De betrouwbaarheid van software is in hoge mate afhankelijk van het vermogen van ontwerpers om de functionaliteit goed te overzien en van het vermogen van opdrachtgevers om eisen op de SMART-manier te formuleren, inclusief alle interacties met de omgeving van het systeem. Complexiteit en onduidelijkheid in de te realiseren functionaliteit en/of de omgeving zijn daarmee ook de grootste bedreigingen voor de betrouwbaarheid van software.

Het bewust eenvoudig houden van de (deel-)functionaliteit is dan ook een noodzakelijk uitgangspunt voor het betrouwbaar maken van de software. Dit vereist dat bij de discussie over softwarebetrouwbaarheid verder wordt gekeken dan alleen naar het ontwerp van de software: bij het opstellen van eisen en het daaropvolgende ontwerp van een elektromechanische of hydraulische besturing moet de eenvoud van besturing het uitgangspunt zijn. Alhoewel software in theorie alles kan besturen, is een belangrijke vraag hoe groot het afbreukrisico van een project wordt naarmate de complexiteit toeneemt.

2.3 Taakuitvoering van software

Voordat je het over betrouwbaarheid en falen van software hebt, moet je eerst de taak van software vaststellen. Software wordt gemaakt om een **taak uit te voeren**. Daarom gaan alle veiligheidsstandaarden uit van de definitie dat software faalt wanneer het zijn taak niet of te laat vervult. Software die geen taak heeft, kan

daarin ook niet falen. Het vaststellen van de taak van software is dan ook een belangrijk onderdeel van het definiëren van softwarefalen.

De taakuitvoering van software is vaak een technisch afgeleide van de functie van het object. De functie (missie) van een object beschrijft de gewenste rol, zoals het veilig laten doorstromen van het verkeer, of het regelen van de waterstand. De taak van software binnen de functie van zo'n object is om ervoor te zorgen dat die functie kan worden uitgevoerd. Wanneer het de functie van een waterkering is om te sluiten bij stormvloed, heeft een stuk software bijvoorbeeld de taak om het sluitcriterium te bepalen, het sluiten van een deur aan te sturen of de seinen om te zetten om de scheepvaart te stremmen. Een functiebeschrijving van het object, die ook onderdeel is van een faalkansanalyse, is noodzakelijk voor het vaststellen van de taak die een specifiek stuk software moet uitvoeren.

2.4 Kwaliteit van software

Een belangrijk onderwerp in softwareontwikkeling en -beheer is de kwaliteit van de software. Nu is kwaliteit een vaag begrip, en daarom biedt ISO 25010 een standaardvocabulaire waarmee de kwaliteit van software beter kan worden beschreven.

Zo bevat ISO 25010 kenmerken als 'gebruiksvriendelijkheid' en 'onderhoudbaarheid'. Punt van aandacht is wel dat 'onderhoudbaarheid' binnen ISO 25010 betrekking heeft op het gemak waarmee de software gewijzigd kan worden, wat afwijkt van de traditionele RAMS-definitie.

Ook de 'betrouwbaarheid' van de software is onderdeel van ISO 25010, en ook hier wijkt de definitie af van de definitie binnen het RAMS-werkveld zoals Rijkswaterstaat die hanteert. In dit document hanteren we dan ook de definitie uit de RAMS-handleidingen, omdat deze preciezer is en beter aansluit op de rest van het werkveld van Rijkswaterstaat.

2.5 Fouten, falen, veiligheidskritisch falen en betrouwbaarheid

Alvorens het te hebben over betrouwbaarheid van software, moeten we onderscheid maken tussen fouten in software, falen van software en veiligheidskritisch falen van software.

Alle software bevat **fouten**. Uit statistieken blijkt dat geen enkel stuk software foutloos is. Er zijn verschillende soorten fouten: van het verkeerd afronden van getallen tot het crashen van de totale applicatie of het nemen van verkeerde beslissingen. Maar niet elke fout heeft ook tot gevolg dat software faalt (ofwel: dat de taakuitvoering wordt verstoord): zo kan een fout in de software tijdelijk leiden tot een afwijkende waterstand, maar wanneer bijvoorbeeld een redundante meting wordt uitgevoerd, hoeft dit niet direct uit te monden in falen. De relatie tussen fouten en falen kan dan ook heel situatiegevoelig zijn.

Softwarefalen gaat verder dan een fout: de fout 'escaleert', waardoor de software zijn taak niet of te laat vervult. Dat is dus iets anders dan dat de software 'het niet meer doet'. Alhoewel dit soms wel het geval is doordat de software crasht, is dit niet het meest voorkomende scenario. Het overgrote deel van softwaregerelateerde ongevallen wordt veroorzaakt doordat de software anders reageert dan bedoeld door de gebruikers. De taakuitvoering van software wordt in de praktijk in de meeste gevallen verstoord doordat deze de verkeerde beslissing neemt, onterecht ingrijpt of te vroeg of te laat is. Vaak heeft dit soort falen ook verdergaande gevolgen dan softwarefouten. We spreken dus van het falen van software wanneer

de software anders, later of vroeger reageert dan bedoeld door de gebruikers met het verlies van de functie tot gevolg.

2.5.1 *Betrouwbaarheid van software*

De **betrouwbaarheid van software** kan worden omschreven als de mate van aanwezigheid/tijdigheid/juistheid van een taakuitvoering die door software wordt gerealiseerd. Het is hier van belang om scherp onderscheid te blijven maken naar de taakuitvoering om zo de betrouwbaarheid goed te kunnen beschouwen. Zo kent een waterkering twee functies: openblijven als het kan (de rivier niet blokkeren), en dicht zijn als het moet (beschermen van het achterland). Falen van software kan de betrouwbaarheid van een van deze of beide functies aantasten. Daarom is het van belang om goed te onderscheiden welke taakuitvoering van software men in een analyse onder de loep neemt.

Dit onderscheid tussen de verschillende functies van een object, en daarmee de taakuitvoering van software, is essentieel omdat dit vaak de geldigheid van velddata bepaalt. Zo laten de historische prestaties van de waterkeringen vaak heel goed zien dat de software heel goed is in niets doen: het niet onterecht sluiten van de kering. De goede prestaties voor deze taakuitvoering zeggen niets over de te verwachten prestaties van de taakuitvoering als men wél tot sluiting over moet gaan: tijdens sluiting moeten soms zeer complexe processen actief worden beheerst, die alleen dan actief worden. Hierdoor is er van deze processen zeer weinig velddata. Dit laatste raakt ook aan de kern van de problematiek van veel veiligheidssoftware binnen het Rijkswaterstaat-domein: de belangrijkste taak (het garanderen van de veiligheid als deze in het geding komt) hoeft zo zelden te worden uitgevoerd dat de betrouwbaarheid niet of zeer moeilijk op basis van historische gegevens kan worden bepaald.

Er zijn praktische getallen over de maximaal haalbare betrouwbaarheid van software. Industriestatistieken laten zien dat er minimaal 2 fouten in elke 10.000 regels broncode zitten. Vanuit de industrie wordt aangegeven dat een praktische maximaal haalbare faalkans aan het veilig falen van software zo rond de 10^{-5} per vraag voor on-demand-systemen, of 10^{-9} per uur voor continu-systemen ligt [IEC 61508, DO-178B].

2.5.2 *Veiligheidskritisch falen en missiekritisch falen*

Nu hoeft niet elke vorm van falen gevaar voor de omgeving of de primaire functie van een object op te leveren. Als bijvoorbeeld de Oosterscheldekering door falende software onterecht sluit, levert dit geen direct overstromingsgevaar voor de bewoners van Zeeland op, dus is dit niet veiligheidskritisch. Daarom is het van belang om ook **veiligheidskritisch falen** te onderscheiden: softwarefalen dat (direct) gevaar voor de omgeving oplevert. Daarnaast onderkennen we ook **missiekritisch falen**: falen dat de primaire doelstelling van een object in gevaar brengt.

Met name in veiligheidskritisch falen speelt 'fail to safe'-gedrag (veilig falen) een belangrijke rol. In sommige omgevingen kan men er door ontwerpkeuzes voor zorgen dat het systeem bij falen zo reageert dat het terugvalt in een veilige situatie. Let wel: 'fail to safe' beïnvloedt niet de kans op falen, maar alleen de kans op gevaarlijk falen¹. In essentie wordt ervoor gezorgd dat als het systeem faalt, dat op een veilige manier gebeurt.

¹ Voor het gebruik van veiligheidscomponenten geldt overigens ook het omgekeerde: de leverancier doet alleen een gecertificeerde uitspraak over de kans op veiligheidskritisch falen. Door het implementeren van een 'fail to safe'-

Nu hangt het van de situatie af of '**fail to safe'-gedrag** kan worden toegepast. Zo kennen vliegtuigen gedurende de vlucht geen **veilige toestand** en alles wat daarmee samenhangt (luchtverkeersleiding, vliegtuigmotoren, etc.) dus ook niet. Maar een trein kun je zonder directe gevolgen stilzetten, waardoor er eenvoudig 'fail-to-safe'-gedrag te realiseren is. De mogelijkheid van een veilige toestand wordt ook deels bepaald door de context en de soort constructie. Kijken we bijvoorbeeld naar de waterkeringen, dan zien we dat de ene waterkering wel een veilige toestand kent, en de andere niet. Zo kent de Oosterscheldekering wel een veilige toestand: deze kering kan altijd sluiten zonder de veiligheid van de omgeving direct in gevaar te brengen. De Maeslantkering kent echter geen veilige toestand: onterecht sluiten brengt directe risico's voor het waterverkeer met zich mee (zeker omdat dit verkeer niet eenvoudig remt) en bij hoge (rivier)afvoeren een verhoogd overstromingsrisico voor het te beschermen achterland (dus verlies van primaire functie). De aan- of afwezigheid van 'fail-to-safe'-gedrag is dus voor een deel afhankelijk van de omgeving.

In veel gevallen zijn veiligheid en beschikbaarheid deels strijdig met elkaar. In een tunnel is de veilige toestand een afgesloten tunnel. Met name het afsluiten van de tunnel levert een beschikbaarheidsprobleem van de tunnel voor de gebruikers op. Een RAMS-analyse helpt inzichtelijk te maken waar deze strijdigheid ligt en waar veiligheid en beschikbaarheid in elkaars verlengde liggen.

2.6 Faalgedrag van software in RAMS-termen

Nu we falen van de taakuitvoering door software beschreven hebben, gaan we het positioneren binnen de kaders van de RAMS-analyses. Eerst kijken we naar de vraag of het systematisch of random falen betreft. Daarna kijken we naar de effectiviteit van testen, het faalgedrag van software als functie van de tijd, en de rol van hersteltijden.

2.6.1

Systematisch of random falen

In theorie is software deterministisch: in eenzelfde situatie zal software hetzelfde reageren en zal deze dus ook altijd op dezelfde wijze falen. Daarmee is er dus sprake van systematisch falen. Falen zal, zoals ook blijkt uit de praktijk, dan voornamelijk voortkomen uit ontwerpfouten.

Systematisch falen betekent ook dat er geen waarde mag worden gehecht aan het dubbel uitvoeren van dezelfde software om softwarefalen te voorkomen. Twee identieke stukken software zullen, gegeven dezelfde input met identieke timing, draaiend op foutloze hardware, beide op dezelfde manier falen. In faalkansen termen is er sprake van een β -factor van 1 (Common Cause Factor binnen het β -factor model). Redundantie van software heeft alleen zin als er sprake is van 'multi-programming': meerdere alternatieve uitvoeringen van software die zo min mogelijk dingen delen, zodat common cause factoren kunnen worden uitgesloten.

De praktijk laat wel zien dat software een complexe opeenstapeling is van firmware, besturingssysteem, softwarebibliotheken en een applicatie. Ook al is het falen daarvan theoretisch gezien systematisch, de complexiteit zorgt ervoor dat de specifieke condities van falen regelmatig zeer moeilijk te reconstrueren zijn. Dit hangt veelal samen met timing-issues, maar ook met complexe interacties tussen de verschillende lagen van de oplossing. Het gevolg is dat software voor de

aanpak kan dit leiden tot een hele veilige maar wellicht ook relatief vaak onbeschikbare component. De eis is immers gesteld aan het veilig falen. In de praktijk blijkt dan ook dat de faalkansverdeling verre van symmetrisch is.

observerende mens wel degelijk een random faalcomponent heeft. Het gedrag is zo complex dat we het systematische element van falen niet meer herkennen. Let wel: de software is nog steeds deterministisch, maar de processen die tot falen leiden zijn zo complex dat hier deels een random faalmodel op van toepassing is.

Voor een betrouwbaarheidsanalyse van een taakuitvoering door software gaan we dan ook altijd uit van een β -factor van 1, tenzij er uitdrukkelijke redenen zijn om daarvan af te wijken. In praktijk gaan we er dus altijd van uit dat identieke stukken software ook altijd gelijktijdig zullen falen. Echter, voor het modelleren van faalgedrag beschouwen we het faalgedrag als random, als het systeem tenminste goed is ontworpen en getest. Dat houdt in dat, ook als een object routinematig vrijwel identieke handelingen uitvoert, er altijd een kans is dat de software 'spontaan' faalgedrag vertoont. Er moet dus altijd rekening worden gehouden met een faalkans voor software, alhoewel deze door maatregelen tot acceptabele niveaus kan worden teruggebracht. De essentie is echter dat software nooit foutloos is.

2.6.2

Effectiviteit van testen

De eis dat "het systeem goed ontworpen en getest" moet zijn, roept de vraag op wanneer er voldoende is getest. De complexiteit van software maakt dat de waarde van testen relatief is. Vanwege die complexiteit is volledig testen namelijk onmogelijk: door de exponentiële groei van het aantal mogelijkheden krijg je al snel hele lange doorlooptijden. Kijken we naar het aantal testen dat nodig is om statistisch een faalkans in geval van 'random falen' met 95% of meer zekerheid te bepalen, dan zien we dat voor iedere faalkans die een orde van grootte beter is, steeds 10 keer zoveel getest moet worden.

Tabel 2.1: Confidence level bij gegeven faalkans en aantal foutloze testen/gebruik.

Faalkans	Aantal foutloze testen van typerend gebruik											
	1	10	30	100	300	1.000	3.000	10.000	30.000	100.000	300.000	1.000.000
10^0	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
10^{-1}	10,00%	65,1%	95,8%	100,0%	100%	100%	100%	100%	100%	100%	100%	100%
10^{-2}	1,00%	9,6%	26,0%	63,40%	95,1%	100%	100%	100%	100%	100%	100%	100%
10^{-3}	0,10%	1,0%	2,9%	9,52%	25,9%	63,2%	95,0%	100%	100%	100%	100%	100%
10^{-4}	0,01%	0,1%	0,3%	1,00%	3,0%	9,5%	25,9%	63,2%	95,0%	99,9%	100%	100%
10^{-5}	0,00%	0,01%	0,03%	0,10%	0,30%	1,0%	3,0%	9,5%	25,9%	63,2%	95,01%	99,9%

Wil je een faalkans van 10^{-3} voor een specifieke taakuitvoering aantonen, dan heb je dus meer dan 3000 verschillende testen van dezelfde taakuitvoering nodig. Dit brengt een dusdanige hoeveelheid testen met zich mee dat dit vaak niet realistisch is. Dit probleem speelt eigenlijk bij alle systemen die een zeer hoge betrouwbaarheid moeten hebben: testen is gewoonweg te inefficiënt om de betrouwbaarheid aantoonbaar te maken². Het statistisch aantonen van softwarebetrouwbaarheid is dan ook zeer lastig.

2.6.3

Merkbaarheid falen in gebruik

Een probleem met softwarebetrouwbaarheid is dat bepaalde taken slechts zeer sporadisch worden uitgevoerd. Softwarefouten kunnen dus vrij lang ongemerkt aanwezig zijn, zeker als je de eerdergenoemde complexiteit in ogenschouw neemt.

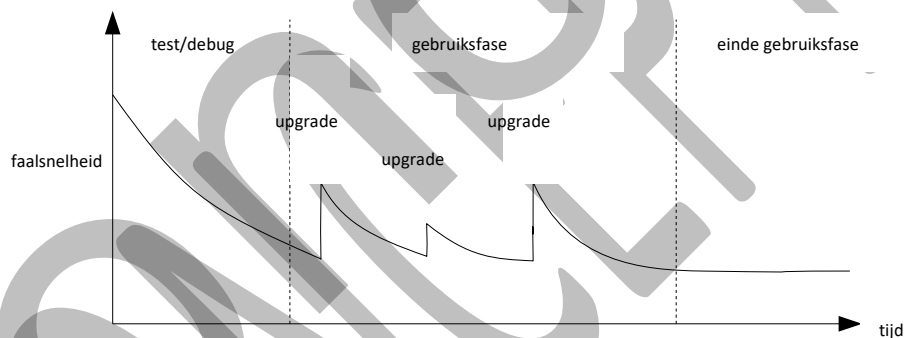
² Dijkstra heeft dit ook geconstateerd en typeerde het als volgt: "program testing can be used very effectively to show the presence of bugs but never to show their absence".

Falen openbaart zich dan ook vaak op het ongunstigste moment: als men de software nodig heeft om een object te bedienen/besturen. Dus in veel gevallen is er sprake van onmerkbaar of latent falen: falen openbaart zich pas als men de taakuitvoering nodig heeft voor de missie of veiligheid van het object.

2.6.4 *Faalgedrag van software in de tijd*

Hoewel de uiteindelijke betrouwbaarheid van hardware en software redelijk vergelijkbaar is, ontwikkelt die zich op een andere manier door de tijd heen. Elektromechanische installaties gaan meestal zo'n 35 jaar mee, en zijn onderhevig aan veroudering. De hardware moet dus vervangen worden voordat veroudering de beschikbaarheid van de installatie nadelig beïnvloedt (de opgaande curve van het badkuipmodel).

Software slijt niet, waardoor het geen fysieke vermoeidheidsverschijnselen vertoont. Dus zolang men de software niet wijzigt en het gebruik ervan ook niet, zal de faalkans van software gelijk blijven. De opgaande curve van het badkuipmodel is dan ook afwezig. Door het doorvoeren van bugfixes zal de software door de tijd heen zelfs betrouwbaarder worden. Maar updates en upgrades kunnen ook fouten introduceren, wat negatieve gevolgen heeft voor de betrouwbaarheid van software. Grafisch ziet dit er als volgt uit:



Figuur 2.1: Faalkans software over de tijd.

Deze grafiek is alleen van toepassing als de programmacode functioneel niet wordt gewijzigd en het gebruiksprofiel constant blijft.

De aanname dat de code gedurende de levensduur van een systeem hetzelfde blijft, klopt echter niet. Soms worden er fouten in de installatie ontdekt die in de software verholpen moeten worden. Maar ook als de fysieke installatie gewijzigd wordt, moet de software eveneens worden gewijzigd. Daarnaast zien we dat de wens aan te sluiten bij globale trends, zoals bediening op afstand en informatiebeveiliging, evenzo leidt tot wijziging van bestaande software. Binnen het Rijkswaterstaat-domein komt het dan ook vrijwel niet voor dat software 15 jaar ongewijzigd blijft.

Ondanks dat de software zelf geen last heeft van slijtage, heeft de software wel last van de slijtage van de omliggende fysieke installatie. Daar komt bij dat het gedrag van de omgeving aan verandering onderhevig is: zo veranderen waterstanden en hun gedrag bijvoorbeeld door zeespiegelstijging en extremer weer. Indirect wordt hierdoor de software onbetrouwbaarder. De software moet namelijk meer foutafhandelingsroutines gaan gebruiken die zich minder bewezen hebben, met mogelijk falen tot gevolg. We zien dus in de praktijk dat software door de tijd heen

onbetrouwbaarder lijkt te worden, met name doordat die meer problemen uit de omliggende installatie en omgeving moet verhelpen.

Het idee dat software niet aangetast wordt door de tijd is echter fout. In werkelijkheid is software zelfs zeer gevoelig voor de tijd, en zelfs gevoeliger dan andere onderdelen van een fysieke installatie. Een typische PLC bevindt zich niet in een vacuüm: hij is voorzien van een hardware design, firmware, een besturingssysteem en een engineering-station. Doordat in de loop van de tijd de (commerciële) beschikbaarheid van die onderliggende componenten verandert, wordt de betrouwbaarheid van de software aangetast. Dit fenomeen wordt ook wel **software-rot** genoemd, wat inhoudt dat de vervanging van hardware ernstig wordt bemoeilijkt. Dit maakt het bijvoorbeeld noodzakelijk om proactief de beschikbaarheid van fysiek identieke reservedelen in de gaten te houden, en om ook proactief te kijken naar de impact van upgrades van firmware, besturingssysteem en engineering-stations. In de praktijk leidt dit fenomeen tot een vervangingscyclus van software van minimaal eens per 15 jaar.

2.6.5

Softwareherstel

Een aandachtspunt voor de beschikbaarheid van een systeem is de aangenomen reparatietijd van software. Waar bij traditionele elektromechanische systemen de reparatietijd bestaat uit het vervangen van een gefaalde component, is dit bij software anders. Het is onrealistisch om het herstel gelijk te stellen met de correctie van de softwarefout. Het is beter en gangbaarder om uit te gaan van de functionele hersteltijd. Deze functionele hersteltijd wordt beïnvloed door de architectuur van de totale installatie. Als een systeem niet afhankelijk is van een door de software intern bijgehouden toestand, dan zou bijvoorbeeld het resetten van de software al tot functioneel herstel kunnen leiden. Maar als de installatie eerst handmatig op een veilige manier in een vooraf gedefinieerde toestand moet worden gebracht voordat die kan worden gereset, dan loopt de hersteltijd al snel op tot uren. Een goed doordacht herstelgedrag van de installatie is dus van belang voor de totale softwarebetrouwbaarheid.

2.7 Aanvaardbaarheid van risico

Geen enkel systeem is vrij van risico's. Het ontwerpen en onderhouden van een systeem betekent ook dat individuen en organisaties veelvuldig beslissingen nemen over risico's. Men maakt keuzes over welke risico's worden geaccepteerd en voor welke risico's mitigerende maatregelen nodig zijn. Deze keuzes worden beïnvloed door wetgeving, de bereidheid van de opdrachtgever om risico's te accepteren, de sociale normen binnen het domein, een verantwoordelijke houding naar de maatschappij, maar ook door de met de maatregelen gemoeide kosten.

In de praktijk is er brede overeenstemming over wat **onaanvaardbare risico's** zijn. Onaanvaardbare risico's moeten volgens de wettelijke of sociale norm altijd gemitigeerd worden. Een voorbeeld hiervan is dat risico's voor medewerkers tijdens onderhoudswerkzaamheden moeten worden vermeden door het gebruik van werkschakelaars. Er zijn ook **aanvaardbare risico's**. Dit zijn risico's waarover algemene overeenstemming bestaat dat een leverancier redelijkerwijs niets kan doen om ze te voorkomen, of dat het verhoudingsgewijs te veel geld zou kosten om ze te voorkomen.

Dit lijken vastomlijnde kaders, maar dat is niet zo. De aanvaardbaarheid van een risico hangt sterk samen met de context, de kans van optreden, de impact en de kosten van mogelijke maatregelen. Op primaire waterkeringen gaat het groepsrisico van de bewoners van het achterland bijvoorbeeld soms voor de veiligheid van de

individuele medewerker. Ook de kosten van een maatregel spelen een rol: het is niet uitlegbaar aan de maatschappij dat eenvoudige en goedkope maatregelen die de risico's verder kunnen terugdringen, niet genomen worden.

Risicomanagement heeft als doel om het totale pakket van risico's van een installatie in kaart te brengen en tot een aanvaardbaar niveau terug te brengen. Een installatie kan pas in productie als ze vrij is van onaanvaardbare risico's. Dit betekent niet dat de installatie vrij is van risico, maar dat de al dan niet gemitigeerde risico's aanvaardbaar zijn. Dit worden ook wel **restrisico's** genoemd.

Binnen Rijkswaterstaat wordt de handreiking Prestatiegestuurde Risicoanalyses (PRA³) gebruikt om risico's in termen van faalkansen in te schatten. Daarbij leidt een onacceptabel hoge faalkans tot de noodzaak om risicobeperkende maatregelen te nemen totdat een aanvaardbaar prestatieniveau is bereikt.

2.8 De rollen van software in de installatie

Traditioneel bestaat de opzet van een technische installatie uit drie onderdelen:

- Een (elektro)mechanische installatie (Equipment Under Control): de installatie zoals deze bediend wordt. Bij Rijkswaterstaat is deze vaak opgedeeld in 'Logische FunctieVervullers' (LFV).
- De **besturing** van de (elektro)mechanische installatie: deze zorgt ervoor dat de installatie functioneel doet wat nodig is om de gewenste dienst te leveren. Bijvoorbeeld de besturing van een brug waarmee de brugwachter de brug opent.
- De **bewaking** van de (elektro)mechanische installatie: de beveiliging van essentiële parameters in het systeem om zo schade aan mens, installatie, milieu en omgeving te voorkomen of te beperken. De stereotype rol is de bewaking van kritieke eigenschappen van de installatie en ingrijpen indien nodig door middel van een 'Fail-to-safe'-reactie. Een voorbeeld is de overbelastingsbeveiliging van een elektrische installatie.

Rijkswaterstaat onderkent naast de besturing en de bewaking van een (elektro)mechanische installatie nog een rol: de **bediening** waarmee de bedienaar de besturing opdracht kan geven een proces te starten of te stoppen. Tezamen worden deze rollen binnen Rijkswaterstaat ook wel de **3B** (Bediening, Besturing en Bewaking) genoemd.

Traditionele veiligheidsstandaarden, zoals de IEC 61508, gaan uit van de gedachte dat de bewaking een aanvullende maatregel is om de risico's van de totale installatie terug te brengen tot een aanvaardbaar niveau, wat betekent dat de bewaking buiten de besturing wordt geplaatst. Die bewaking wordt gerealiseerd door veiligheidsfuncties. Modernere standaarden accepteren dat deze veiligheidsfuncties een geïntegreerd onderdeel van de besturing van een installatie kunnen zijn, maar stellen dan wel dat de besturing aan dezelfde betrouwbaarheidseisen moet voldoen als de veiligheidsfuncties.

De technische scheiding van bewaking en besturing heeft als voordeel dat over het algemeen de betrouwbaarheid van de bewaking eenvoudiger aan te tonen is, waardoor ook eenvoudiger kan worden aangetoond dat het restrisico aanvaardbaar is. Dit laatste gebeurt door middel van een risicoanalyse. In de kwantitatieve risicoanalyse (QRA) van de landelijke tunnelstandaard is dit duidelijk te zien. Het nadeel is dat in sommige installaties de bewaking veel kennis vereist van de

³ Pas op dat T-Map de afkorting PRA ook gebruikt, maar dan als productrisico-analyse

toestand van het proces, waardoor de bewaking dus van nature verweven is met de besturing.

Zoals hierboven beschreven gaan veiligheidsstandaarden onder meer uit van de gedachte dat het de rol van de bewaking is om de risico's die gepaard gaan met een installatie tot een acceptabel niveau terug te dringen. Dit gaat bij Rijkswaterstaat slechts ten dele op: in veel Rijkswaterstaat-installaties, zoals tunnels en waterkeringen, zijn de LFV's integraal onderdeel van een actieve veiligheidsfunctie. Zo is bij brand in een tunnel het activeren en in stand houden van de noodstand van de ventilatie essentieel. Bij een waterkering is het actief sluiten van de kering de veiligheidsfunctie.

Dit houdt in dat veiligheidsfuncties een wezenlijk andere rol hebben. Traditioneel worden veiligheidsfuncties ingezet om de omgeving te beschermen tegen negatieve effecten van de installatie. Bij Rijkswaterstaat-installaties worden veiligheidsfuncties ook ingezet om de installatie aan te sturen om negatieve effecten uit de omgeving te beperken. Hierbij heeft een veiligheidsfunctie dus een actieve rol in het direct besturen van een installatie ten behoeve van veiligheid.

2.9 Bronnen van softwarefalen

Als we betrouwbare software willen maken, dan moeten we zien te voorkomen dat deze faalt. Gebeurt dit toch, dan moet de software veilig falen.

Uit industriegegevens⁴ weten we waar de oorsprong van de fouten in software ligt. Als we ervan uitgaan dat de software zeer grondig is ontworpen en ontwikkeld, hebben de eerdergenoemde 2 fouten per 10.000 regels de volgende oorsprong:

Tabel 2.2: Aantal fouten vrijgegeven aan het veld per 10.000 regels code

Oorsprong fout	Initieel aanwezig	Effectiviteit QA proces	Uiteindelijk aantal fouten	Procentueel
Eisen	3,20	77%	0,74	37%
Ontwerp	4,00	85%	0,60	30%
Programmeren	5,60	95%	0,28	14%
Slechte bugfixing	1,28	70%	0,38	19%
Totaal	14,08		2	100%

Uit dit overzicht blijkt dat de meeste fouten in software worden gemaakt in de programmeerfase. Dit soort fouten is door reviews en testen echter ook het meest effectief te vinden en te verwijderen. Het overgrote deel van de resterende fouten in software wordt geïntroduceerd in de gestelde eisen en het ontwerp.

Fouten in de eisen en het ontwerp zijn voor een belangrijk deel te wijten aan complexiteit. Als de complexiteit te hoog wordt, overzien de ontwerpers het ontwerp niet en maken ze ontwerpfouten. In de praktijk is dit de belangrijkste bron van fouten in het gedrag van software. De belangrijkste manier om de betrouwbaarheid of de beschikbaarheid te verhogen, is het verlagen van de complexiteit van de software, zowel in eisen (eenvoudige taakuitvoering) als in code. Voor de eisen zou dat betekenen dat zeer specifieke uitzonderingssituaties anders aangepakt zouden moeten worden (buiten de software om) om zo de software te ontlasten. Voor de resulterende code is de cyclomatische complexiteit van belang, die laat zien hoeveel verschillende paden door de software beschikbaar zijn. Het vroegtijdig beheersen

⁴ Capers Jones 'Software Assessments, Benchmarks and Best Practices'.

van het aantal mogelijke paden door software, is dan ook een belangrijke stap in de beheersing van de betrouwbaarheid van software.

Hergebruik van software, bijvoorbeeld door middel van standaardbouwblokken, kan de betrouwbaarheid helpen verbeteren. Bij de spoorwegen leunt men voor de beveiliging van het spoor grotendeels op gestandaardiseerde softwarebouwstenen. Er zijn echter ook gevallen bekend (Ariane 5) waarin hergebruik van software catastrofale gevolgen had, omdat de nieuwe omgeving te veel afweek van het initiële gebruik. Ook kunnen dit soort standaardbouwstenen voorzieningen bevatten die onnodig zijn voor de specifieke oplossing, waardoor ze ongewenst gedrag kunnen veroorzaken. Het is dus zeker geen gegeven dat een bewezen oplossing op één locatie gelijk goed functioneert op een andere. Een vereiste is dan ook om goed vast te stellen of een oplossing naar behoren werkt in een nieuwe installatie. Een voordeel van gestandaardiseerde bouwstenen is wel dat men eenvoudiger lering kan trekken uit fouten op andere locaties. Over het algemeen is op de langere termijn het gebruik van standaardcomponenten gunstiger.

2.10 Geaccepteerde methodes voor het realiseren en behouden van de betrouwbaarheid

Het beheersen van de betrouwbaarheid van software vereist een goed ingericht proces voor zowel softwareontwikkeling als -onderhoud. Er is brede consensus binnen de industrie over de benodigde aanpak: de IEC 61508 beschrijft hoe de betrouwbaarheid van veiligheidsfuncties moet worden bewerkstelligd en vormt daarmee de basis voor een familie van internationale industrienormen.

Door toepassing van deze normen wordt discussie over de relevantie en compleetheid van de uit te voeren maatregelen voorkomen: experts uit de industrie hebben immers gezamenlijk consensus bereikt over de maatregelen, die daarmee fungeert als de 'mening van de industrie'. Daarnaast schept de toepassing van deze normen ook helderheid naar de leverancier over de verwachte grondigheid van de uit te voeren werkzaamheden.

De IEC 61508-familie heeft bovendien in de afgelopen 20 jaar de status van de facto wetgeving bereikt: zij staat voor wat de industrie als gangbare praktijk beschouwt om een veiligheidsgerelateerd systeem te ontwerpen, bouwen, testen en beheren. Deze normen beantwoorden daarmee ook een indirect de vraag wanneer de industrie een ontwikkelaar/beheerder als 'verwijtbaar nalatig' beschouwt.

Deze normen-familie bestaat uit de volgende normen:

- IEC 61508:2010, de 'moederstandaard' voor alle functionele veiligheidsstandaarden
- IEC 61511:2016, de variant voor veiligheidskritische toepassingen in de procesindustrie
- IEC 61513:2011, de variant voor veiligheidskritische toepassingen in het nucleaire domein
- IEC 62061:2005, de variant voor veiligheidsverhogende functies in productiemachines (machinerichtlijn)
- IEC 62279:2015, de variant voor veiligheidssystemen gebruikt bij de spoorwegen
- ISO 26262:2011, de variant voor veiligheidskritische applicaties in de automotive industrie

Hierbij fungeert de IEC 61508 ook als vangnet: in domeinen waar geen industrie-specifieke standaard aanwezig is, zoals in het RWS-domein, valt men automatisch terug op de generiek toepasbare IEC 61508.

De rol van de IEC 62061 vraagt specifieke aandacht. Veel installaties van Rijkswaterstaat vallen ook onder de machinerichtlijn. Het lijkt dus voor de hand liggend om één standaard voor alle veiligheidstoepassingen te hanteren. Dit is echter onverstandig omdat:

- de ISO 62061 erop is gericht het individuele risico dat medewerkers lopen, te beperken als onderdeel van een pakket van beschermende maatregelen in een machinaal productieproces. Bij veel Rijkswaterstaat-objecten wordt het risico dat Nederlandse burgers als groep lopen, beveiligd door het activeren van de machine (zoals waterkeringen, maar ook tunnelventilatie), en gaat dit groepsrisico in voorkomende gevallen boven het risico van de individuele medewerker.
- deze norm als principe hanteert dat het (veilig) stilzetten van de installatie een veilige situatie oplevert. Zoals al eerder beschreven, geldt dit niet voor veel veiligheidssystemen in het Rijkswaterstaat-domein. Deze norm is gelimiteerd tot een faalkans van 10^{-3} , wat bijvoorbeeld in het domein van waterkeringen niet toereikend is.

Voor dit domein valt men dus terug op de generieke IEC 61508:2010.

Binnen de IEC 61508 behandelt deel 1 de generieke processen voor alle activiteiten, en deel 3 de specifieke eisen voor softwareontwikkeling en -beheer. In TOPAAS wordt een duidelijke relatie gelegd met de IEC 61508. Dit houdt in dat je voor het maken van betrouwbare software de IEC 61508-3 moet toepassen. Aspecten uit de IEC 61508-3 komen ook naar voren in TOPAAS, denk bijvoorbeeld aan maximale samenhang en minimale koppeling (maximum cohesion and loose coupling).

2.11 De levenscyclus van software

Deze paragraaf zet uiteen hoe de aanpak van softwarebetrouwbaarheid conform de breed geaccepteerde software-engineeringmethode in alle fasen van de levenscyclus van software toegepast kan worden. Bij het ontwerpen van software worden min of meer dezelfde stappen in de levenscyclus gevolgd als bij het ontwerpen van infrastructurele systemen (systems engineering), met als verschil dat bij de ontwikkeling van software de tijdlijnen korter zijn. De SE-conceptfase bestaat onder meer uit planstudies. Softwareontwikkeling begint pas tegen het eind van de ontwerpfase van het object zelf.

In de softwarelevenscyclus wordt per fase aangegeven welke activiteiten moeten worden uitgevoerd en welke producten moeten worden opgeleverd om uiteindelijk tot betrouwbare software te komen en om die software over de gehele levensduur te kunnen beheren en onderhouden.

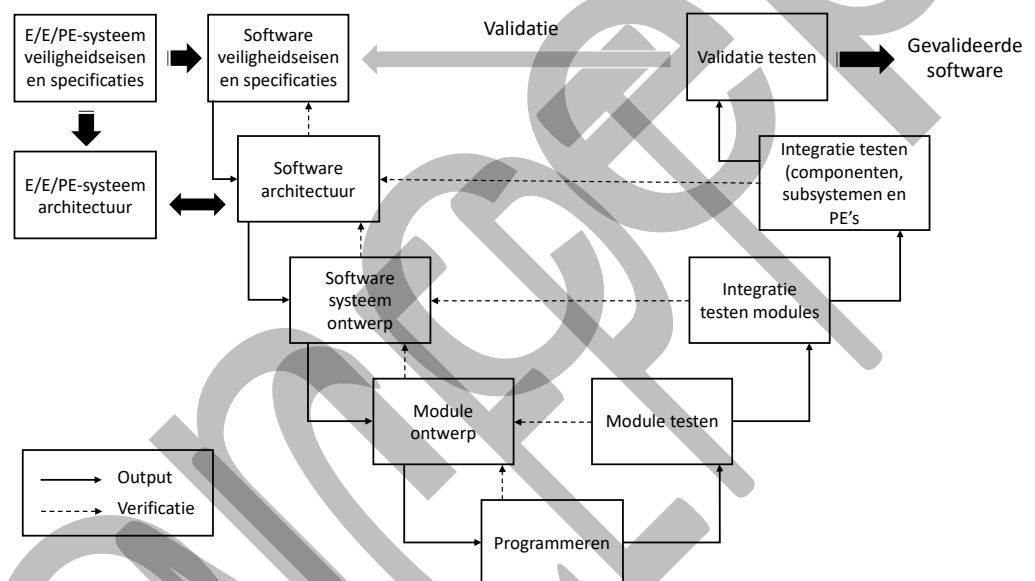
De SE-werkwijze onderscheidt vijf levenscyclusfasen:

1. Concept
2. Ontwikkeling
3. Realisatie
4. Gebruik, beheer & onderhoud
5. Sloop

We kijken in dit document alleen naar de realisatie van de software die een rol speelt in de bediening van het onderhavige object. Software heeft als doel om de

functie van het object (mede) te bewerkstelligen. Dat vindt op verschillende plaatsen binnen het object plaats, en zelfs binnen aparte omgevingen (industriële automatisering, bedienruimte, monitoring, beslis- en ondersteuningssystemen). De concept- en ontwikkelingsfase (SE) brengen de verschillende subfuncties in kaart waarin software mogelijk een rol speelt. Vervolgens gaat voor al deze (eventueel gegroepeerde) subfuncties een softwarelevenscyclus van start.

In ieder softwaretraject wordt een soortgelijke indeling van levenscyclusfasen gehanteerd, maar veelal bevat die meer stappen en meer details. In de IEC 61508-3 (figuur 6) wordt het V-model voor software weergegeven waarbij ook de relatie met de E/E/PE-systemen wordt weergegeven. Hieronder wordt deze figuur in het Nederlands weergegeven:



Figuur 2.2: V-model conform de IEC 61508-3.

In de rest van deze richtlijn wordt in hoofdlijnen deze indeling gevolgd.

3 Fase 0: Aantoonbaarheid van softwarebetrouwbaarheid

3.1 Inleiding

Een groot probleem binnen projecten in het Rijkswaterstaat-domein is de aantoonbaarheid van de betrouwbaarheid en beschikbaarheid van software; hoe toont de opdrachtnemer aan dat de oplossing voldoet aan de gestelde eisen? Dit wordt des te complexer als het om systemen gaat waarvoor een hoge beschikbaarheid of veiligheid is vereist. Er zal altijd een restrisico zijn. Zelfs in het geval van zeer volwassen software is er bij het repareren van iedere bevinding een kans dat deze reparatie onvolledig is of dat deze weer een nieuw probleem introduceert. Het gevolg is dat absolute veiligheid en beschikbaarheid onhaalbare doelen zijn. Er kunnen dan ook geen bewijs worden geleverd of garanties worden gegeven dat de oplossing veilig of beschikbaar is, maar het is wel mogelijk om vertrouwen creëren. In feite vertegenwoordigt een faalkanscijfer dat is onderbouwd met gegevens, meer een mate van vertrouwen dan dat het een statistisch hard cijfer is. Vertrouwen wordt opgebouwd door analyses, mogelijk gemaakt door een transparant proces en door het aantoonbaar voldoen aan de beste industriële praktijken. De ontwerp- en beheerprocessen streven daarom twee doelen na: het bereiken van veiligheid/beschikbaarheid en het onderbouwen van de geleverde prestatie. Het is de verantwoordelijkheid van de beheerder (en in de realisatiefase de ontwerper namens deze beheerder) om aan te tonen dat de claim van een voldoende veilig en beschikbaar systeem geldig is.

Nu laten de betrouwbaarheid en beschikbaarheid van missie-/veiligheidskritische functies zich niet eenvoudig aantonen door testen, zelfs als onder testen ook het volledig nalopen van de eisen en ontwerpdocumentatie wordt verstaan. Aan het eindproduct is niet te zien welk proces gevolgd is. De bewijsvoering moet voortkomen uit de combinatie van een (aantoonbaar) gedegen proces en het (aantoonbaar) voldoen aan producteigenschappen. Deze bewijsvoering moet worden opgebouwd gedurende de ontwikkeling en uiteindelijk overhandigd worden aan de opdrachtgever. Er blijft echter een groot risico voor de opdrachtnemer dat de onderbouwing achteraf onvoldoende blijkt voor het verkrijgen van voldoende vertrouwen van de opdrachtgever.

3.2 Dossiervorming

De meest eenvoudige aanpak bestaat uit het opstellen van een **veiligheidscasus/ beschikbaarheidscasus**. Een dergelijke casus beschrijft welke maatregelen een ontwikkelaar/beheerder genomen heeft, en waarom ze naar zijn mening voldoende zijn. Bij voorkeur hanteert men een reeds bewezen aanpak om zo te voorkomen dat er discussie ontstaat over de effectiviteit van de maatregelen. Reeds bij het opstellen van eisen worden keuzes gemaakt. Het specificatieteam dient dus al een eerste versie van deze casus op te leveren. Gedurende iedere fase wordt de casus aangevuld.

Rijkswaterstaat wordt bij voorkeur in een vroeg stadium, namelijk al bij het opstellen van de maatregelen, erbij betrokken om de geschiktheid van het pakket aan maatregelen samen met de opdrachtnemer vast te stellen. Het is bijvoorbeeld gebruikelijk een zelfstandig 'Safety Management Plan' op te stellen of de maatregelen te integreren in een projectplan of Software Development Plan (SDP). Door de discussie al aan het begin te laten plaatsvinden, kan relatief eenvoudig worden bijgestuurd, terwijl correctie achteraf daarentegen vaak onhaalbaar blijkt.

In dit verband hangen technische keuzes en procesmaatregelen met elkaar samen: een oplossing die 'slechts' de configuratie is van een reeds bewezen COTS-component, moet uiteraard anders afgehandeld worden dan een grootschalig zelfbouwproject. Technische keuzes hebben gevolgen voor de procesinrichting van de opdrachtnemer. En het is juist die afweging die vooraf moet worden afgestemd, omdat hiermee ook de basis wordt gelegd voor de acceptatie van de genomen maatregelen.

Een SDP (Software Development Plan) wordt uiteraard onderworpen aan procestoetsing. Rijkswaterstaat richt zijn borging en toetsing risicogestuurd in, en verwacht dit ook van zijn leveranciers. Dit betekent dat een dergelijke toetsing plaatsvindt op basis van het in het SDP beschreven proces met aanvullende maatregelen om veiligheid en beschikbaarheid te realiseren. Deze toetsing moet worden uitgevoerd door onafhankelijke veiligheidsbeoordelaars, waarbij het niveau van onafhankelijkheid afhangt van het veiligheids-/beschikbaarheidsniveau dat men nastreeft. Je kunt bijvoorbeeld denken aan een onafhankelijke persoon, een onafhankelijke afdeling of een onafhankelijke organisatie. Als de wederzijds geaccepteerde processen met voldoende diepgang door een (onafhankelijke) deskundige worden getoetst, biedt dit een sterke basis voor een veiligheidskasus/beschikbaarheidskasus.

4 Fase 1: Eisen en specificaties

4.1 Eisen aan betrouwbaarheid en beschikbaarheid

Voor Rijkswaterstaat als opdrachtgever geldt dat er een zekere noodzaak is tot het specificeren van de betrouwbaarheid en beschikbaarheid van software. Op basis van de standaard J-STD-016 moet men daarnaast eisen opstellen die kritisch zijn voor veiligheid of de missie expliciet markeren en moet men ook een plan maken voor de afhandeling daarvan.

Rijkswaterstaat onderscheidt standaard drie functies van een systeem: bedienen, besturen en bewaken. Kenmerkend voor RWS-objecten is dat bewaken veiligheidskritisch is, en dat bedienen en besturen missiekritisch zijn.

Onderdeel van de beschrijving van functies die kritisch zijn voor veiligheid of de missie is het beschrijven van de precieze taakuitvoering. Daarbij hoort het vaststellen van de beoogde beschikbaarheid (in de RAMS-methodiek uitgedrukt als een Q^5) en de beoogde betrouwbaarheid (in de RAMS-methodiek uitgedrukt als een λ^6) die de taakuitvoering dient te hebben. Een Q wordt doorgaans gebruikt als een taakuitvoering op aanspraak wordt ingeroepen (de slagboom die een tunnel moet afsluiten, de noodventilatie die aan moet springen). Een λ wordt doorgaans gebruikt voor systemen die continu worden gebruikt (bijvoorbeeld luchtkwaliteitsmeting in een tunnel).

Een λ of Q staat voor de betrouwbaarheid waarmee een (veiligheids- of beschikbaarheidsgerelateerde) taakuitvoering wordt uitgevoerd, waarbij het dus essentieel is dat er een eenduidig beeld ontstaat van wat het systeem moet doen (de functie van het systeem). Het uitvoerig beschrijven van een taakuitvoering, en van de situatie waarin Rijkswaterstaat die als gefaald beschouwt (faaldefinities), is daarmee een integraal onderdeel van de beschrijving van een functie.

4.1.1

Kwaliteit van de beschreven eisen

Eisen vormen het uitgangspunt voor de werking van het systeem. Zij beschrijven WAT het systeem in welke situatie aan dynamisch gedrag hoort te vertonen. Zij moeten daarom voor zowel gebruikers als ontwikkelaars en testers begrijpelijk, verifieerbaar en voor één uitleg vatbaar zijn. Voor wat betreft de beschrijving van de eisen geldt dat deze SMART geformuleerd moeten worden. Meer specifiek moeten deze aan de volgende kwaliteitseigenschappen voldoen⁷:

- De eisen zijn compleet.
- De eisen zijn correct.
- De eisen zijn ieder voor zich duidelijk en er is beschreven in welke context de eisen gelden.
- De eisen zijn concreet en limitatief (zo mag bijvoorbeeld niet worden geëist dat "alle vormen van falen van een tunnel worden gemitigeerd").
- De eisen zijn maar op één manier te interpreteren.
- De eisen zijn vrij van interferentie met andere eisen.
- Er is beschreven wat de beschikbaarheids- en veiligheidsimpact is van de eis (dit kan op eiseniveau, maar bijvoorbeeld ook op use caseniveau).

⁵ Q staat voor kans op falen per aanspraak op de functie.

⁶ λ staat voor de kans op falen per tijdseenheid.

⁷ Zie IEC 61508-3, §7.2.2 en IEEE SWEBoK.

- Er is beschreven wanneer de eis als geslaagd wordt beschouwd (ook dit kan op eisniveau, maar ook op use case niveau).
- De eis is testbaar.

Een genummerde lijst van eisen is altijd handig, mede omdat het nummeren⁸ van eisen helpt bij de verdere traceerbaarheid in het proces, maar de beschrijving van de beoogde werking van het systeem heeft prioriteit. Eisen aan software zijn namelijk altijd gekoppeld aan taakuitvoeringen. In het algemeen wordt er bij softwareontwikkeling dan ook gewerkt met een lopende tekst, of use cases, waarin de eisen gemarkeerd en genummerd zijn. Het begrip van het te bouwen systeem staat dus voorop, maar is er wel een context voor elke eis.

Niet-functionele eisen

Naast de beschrijving van het dynamische gedrag moet men ook nadenken over niet-functionele eisen. Niet-functionele eisen spelen een belangrijke rol bij de inpassing van software in zijn context. De belangrijkste niet-functionele eisen in deze richtlijn zijn betrouwbaarheid en beschikbaarheid, maar er zijn andere niet-functionele eisen die in hoge mate bepalend zijn voor de kwaliteit van het product. ISO 25010 beschrijft een set kwaliteitseigenschappen (niet-functionele eisen) van software die men als eis zou kunnen opvoeren, afhankelijk van de behoefte van de gebruikers en doelstellingen van het systeem.

Het is belangrijk dat opdrachtgevers zich realiseren dat sommige eigenschappen op gespannen voet staan met elkaar. Hoge eisen aan gebruiksvriendelijkheid kunnen strijdig zijn met hoge eisen aan beveiliging. Het is dus van belang bij de formulering van de eisen hier een keuze in te maken en prioriteiten te stellen.

Om van de in ISO 25010 geformuleerde kwaliteitseigenschappen tot eisen te komen is een extra stap nodig waarin deze eigenschappen worden voorzien van een meetschaal met daarop criteria. Op deze wijze kan men ook voor software optimale waarden, marges en toleranties aangeven.

Er zijn meer eisen die in het verlengde van betrouwbaarheid en beschikbaarheid liggen:

- Reactiesnelheid: een te vroege of te late reactie van software is in principe een faalwijze van software;
- Foutbestendigheid: hoe robuust is een systeem als de omgeving fouten bevat;
- Nauwkeurigheid: de betrouwbaarheid is zo goed als de nauwkeurigheid van de cruciale berekeningen die aan het handelen van software ten grondslag liggen. Als je niet nauwkeurig genoeg bent, kun je daarmee onbetrouwbaar worden;
- Begrijpelijkheid: de begrijpelijkheid van de userinterface is van belang wanneer mensen handelingen moeten uitvoeren om het proces tot een goed einde te brengen.

Het beschrijven van de rol van deze kwaliteitseigenschappen in de beoogde taken en use cases is van belang voor de implementatie hiervan. Bijvoorbeeld: als RWS de mens als hersteller wil zien in een technische installatie, dan moeten de handelingen voor die gebruiker begrijpelijk zijn, moet de software die rol faciliteren en is de kwaliteit van de implementatie medebepalend voor de slagingskans van

⁸ Nummeren kan ook vervangen worden door unieke identificatie van de eisen.

herstelacties. Maar als RWS vindt dat het proces geen menselijke interventie toe mag laten, dan is begrijpelijkheid van de GUI dus niet van belang.

4.1.2 *Greenfield-bepaling geambieerde faalkans*

Bij een nieuw te ontwerpen systeem (greenfield-situatie) is er geen ervaring met het systeem, en dus ook niet met de beoogde rol van software daarin. Normaliter zal het totale project een faalkans of beschikbaarheid moeten garanderen, en kan er via de RAMS-methodiek een budget voor de desbetreffende systemen worden vastgesteld.

4.1.3 *Brownfield-bepaling faalkans*

Bij de vervanging van een bestaand systeem (brownfield-situatie) ligt het voor de hand om te eisen dat de faalkans van het nieuwe systeem niet groter mag zijn dan van het oude. Wel is het zo dat de kans aanwezig is dat met een aanpassing fouten in de software worden geïntroduceerd. Het is dus van belang je zelfs bij zeer kleine wijzigingen/verbeteringen de vraag te stellen of die eis realistisch is.

In veel situaties is er sprake van een bestaand systeem met bijbehorende oude specificaties en/of veldprestaties op basis waarvan men de beoogde betrouwbaarheid kan bepalen. De ervaring leert dat dit geen eenvoudige opgave is. Een belangrijk uitgangspunt is dat de prestaties van het systeem door de wijziging niet mogen verslechteren. Dat betekent dat als de veldprestaties beter zijn dan de specificaties, het verstandig lijkt om de huidige veldprestaties als basis voor de beoogde betrouwbaarheid te hanteren. Hiervoor is wel nodig dat men een goed beeld heeft van het werkelijke aantal storingen van de software in een object. De realiteit leert dat veel hardwarestoringen afgewenteld worden op de software, waardoor het hanteren van deze slechtere faalkans tot een verslechtering leidt van het gebouwde softwaresysteem. Een vereiste is dan ook om de gegevens die gebruikt worden voor de veldprestaties, op te schonen.

Een belangrijk onderwerp is het opnieuw inpassen van de faalkans in de RAMS-analyse. Bij stormvloedkeringen worden foutenbomen gebruikt voor de RAMS-analyse. Het vermijden van 'faalkansverlies' is daarin van belang. Doordat er vaak sprake is van mogelijke common cause failures (CCF) met andere systemen in een installatie, bijvoorbeeld met PLC-hardware die al in de installatie aanwezig is, kan het zijn dat in theorie de ene hardwareleverancier andere eisen krijgt dan de andere, omdat bij de ene leverancier er een CCF in het overall object ontstaat. Dit zijn onderwerpen die vroegtijdig in het specificatietraject aandacht behoeven.

4.1.4 *Safety Integrity Levels*

In de IEC 61508 norm worden vier Safety Integrity Levels (SIL-levels) voor software onderscheiden. Elke niveau staat voor een bepaalde mate van vrijwaring van onacceptabele risico's. In oplopende volgorde horen bij elk niveau dan ook strengere eisen aan de faalkans van software. Bij SIL-4 bijvoorbeeld is er sprake van een faalkans per aanspraak die ligt tussen de 10^{-4} en 10^{-5} . Dat is het niveau dat gehanteerd wordt voor kerncentrales.

De IEC-norm beschrijft voor elk SIL-level zowel aanbevolen specifieke maatregelen in de software als aanbevolen onderdelen voor het softwareontwikkelp proces waarmee het mogelijk is om de bijbehorende faalkanseis te halen⁹. Het zal duidelijk zijn dat de werkwijze voor SIL-4 aanmerkelijk meer en uitgebreidere aanbevelingen

⁹ Het volgen van de aanbevelingen is geen garantie voor het behalen van de bijbehorende softwarefaalkans. De norm kan dus evenmin gebruikt worden om een bepaalde faalkans aan te tonen.

omvat dan voor SIL-1. Daarmee is de keuze voor een te behalen SIL-level dus ook een kosten/baten afweging.

De beste aanpak is dan om aan de individuele functies/use cases specifieke faalkansen toe te kennen om daarmee een SIL-level voor een geheel (sub)systeem vast te stellen. Zo wordt sturing gegeven aan het ontwikkelproces, en kan de softwareontwikkelaar ook de slimme verdere decompositiekeuzes maken in het ontwerpproces. Op basis van de eisen aan de faalkans per uur (λ) of de faalkans per aanspraak (Q) kan men het SIL-level bepalen. Hiervoor gelden de volgende tabellen:

Tabel 4.1: SIL-niveaus voor continu werkende systemen (λ)

Safety Integrity Level (SIL)	Continu/frequent aangesproken operatiemodus (Kans op gevaarlijk falen per uur)
4	$\geq 10^{-9}$ tot $< 10^{-8}$
3	$\geq 10^{-8}$ tot $< 10^{-7}$
2	$\geq 10^{-7}$ tot $< 10^{-6}$
1	$\geq 10^{-6}$ tot $< 10^{-5}$

Tabel 4.2: SIL-niveaus voor op aanvraag werkende systemen (Q)

Safety Integrity Level (SIL)	Laag frequent aangesproken operatiemodus (Gemiddelde kans op falen per aanspraak)
4	$\geq 10^{-5}$ tot $< 10^{-4}$
3	$\geq 10^{-4}$ tot $< 10^{-3}$
2	$\geq 10^{-3}$ tot $< 10^{-2}$
1	$\geq 10^{-2}$ tot $< 10^{-1}$

Een stuk software vervult vaak meerdere taakuitvoeringen in meerdere use cases. Bijvoorbeeld, de besturing van een afsluiter handelt de primaire opening van de afsluiter af als dat nodig is, zorgt dat hij dichtstaat als de afsluiter niet nodig is, maar detecteert ook wanneer de afsluiter faalt. Zo'n stuk software moet dan voldoen aan meerdere λ 's en Q's.

Omdat dit soort taakuitvoeringen vaak ook dezelfde code delen (bijvoorbeeld de communicatie naar de afsluiter toe), ontstaat al snel de situatie dat men per stuk software meerdere faalkansen heeft. In dat geval bepaalt natuurlijk de zwaarste eis het regime dat voor het bouwproces wordt gehanteerd. Vanuit technisch oogpunt mogen echter de individuele faalkansen niet vergeten worden.

Hierdoor zijn de volgende overwegingen van belang:

1. Als een softwaremodule of (sub)systeem meerdere functies vervult, kan aan elke afzonderlijke taakuitvoering een specifieke faalkanseis worden gesteld. Bij het bepalen van het van toepassing zijnde SIL-level voor een systeem is de zwaarste eis dominant.
2. Vanuit technisch oogpunt mogen echter de individuele faalkansen niet vergeten worden: die geven richting voor de identificatie van het fail-to-safe-gedrag, aangezien ze aangeven waar de minste pijn geleden wordt bij falen. Door de beoogde faalkansen van de individuele taakuitvoeringen met elkaar te vergelijken, kun je zien waar de fail-to-safe toestand zit (indien aanwezig).

4.2 Omgang met een-op-eenrenovatie van systemen

Binnen Rijkswaterstaat worden veel vervangingen vaak geïnitieerd doordat de PLC-hardware (in de toekomst) niet meer beschikbaar is. Hieronder verstaan we ook het vervangen van hardware door een nieuwere versie van dezelfde hardware (hardware-upgrade).

De ervaring leert dat bij een zuivere hardware-upgrade (dus vervanging door hetzelfde merk en type hardware, maar dan een recentere versie van de hardware/softwareconfiguratie van de PLC) het risico goed te beheersen is. Bij een hardware-upgrade is het NIET voldoende om alleen de hardware te vervangen. Wijzigingen in de software zijn nodig omdat de firmware en libraries in de tussenliggende jaren zijn gewijzigd. Ook is de CPU vaak sneller, waardoor er mogelijk timing-problemen kunnen ontstaan.

Is een upgrade binnen hetzelfde merk en type niet mogelijk, dan moet in de regel de software (volledig) worden herschreven. Het is dan vaak niet voldoende om alleen toegang tot de broncode te geven: hierin wordt namelijk vaak geleund op proprietary libraries, waar andere softwareontwikkelaars geen inzage in hebben. Er moet in dat geval een gedetailleerde functiereconstructie plaatsvinden om de gewenste functionaliteit goed te beschrijven.

Uitgangspunt is in alle gevallen dat er bij een hardware-upgrade niet ook functionele wijzigingen worden uitgevoerd. Vanuit het oogpunt van softwarebetrouwbaarheid heeft het sowieso sterk de voorkeur om bewezen software zo min mogelijk te wijzigen. Maar ook omwille van de beheersbaarheid van het project moet worden voorkomen dat functionele wijzigingen gelijktijdig worden meegenomen. Verificatie van de upgrade kan dan plaatsvinden aan de hand van herijkte initiële testplannen en een gerichte vergelijking met de originele hardwareoplossing.

4.3 Activiteiten

Dit hoofdstuk beschrijft hoe software-eisen opgesteld moeten worden. Hierbij zijn de volgende hoofdactiviteiten van belang.

Hoofdactiviteit 1: Stel functionele en niet-functionele eisen aan de software op een dusdanige wijze op dat ze SMART zijn conform par. 4.1.1.

Er zijn verschillende manieren om betrouwbaarheids- en beschikbaarheidseisen op te stellen.

Deelactiviteit 1.1: Bepaal welke manier het beste past bij het systeem waarvoor software moet worden gerealiseerd (zie paragraaf 4.1.2, 4.1.3 en 4.1.4).

Hoofdactiviteit 2: Gebruik hoofdstuk 9 bij het opstellen van eisen bij aanbestedingen van softwaregerelateerde projecten.

Hoofdstuk 9 geeft handvatten die helpen bij het opstellen van eisen bij een aanbesteding van een project voor de vervanging van software.

4.4 Op te leveren producten/documenten

Document met eisen en specificaties voor software.

5 Fase 2: Ontwerp van de oplossing

5.1 Oriëntatie

Het doel van de oriëntatiefase is het ontwikkelen van een gezamenlijk beeld van de oplossing en de eisen die daaraan gesteld worden. Hier legt de opdrachtgever (OG) uit wat de doelstelling van het systeem is, en heeft opdrachtnemer (ON) het doel om de opdracht zo helder en scherp mogelijk te krijgen.

5.1.1

Acceptatie van de eisen

Voor de aanvang van een project is het noodzakelijk dat de opdrachtnemer de set van eisen toetst op compleetheid, eenduidigheid, bouwbaarheid en testbaarheid. Men kan hier denken aan het proces van de System Requirement Review (SRR)¹⁰.

Een belangrijk onderdeel hiervan is het bespreken/ter discussie stellen van de eisen die veel invloed hebben op het uiteindelijke ontwerp. Het is de rol van de opdrachtnemer om in de SRR de impact van deze eisen te benoemen, denk aan de noodzaak tot trade-offs, hoe de ontwerper kan besluiten wat het optimale ontwerp is. De opdrachtgever heeft in de SRR tot taak de achtergrond toe te lichten en aanvullende informatie aan te dragen om het ontwerpproces te faciliteren. In het algemeen hebben veel beschikbaarheids- en veiligheidseisen veel invloed op het ontwerp. Deze eisen moeten goed met de opdrachtgever worden afgestemd. Waar mogelijk worden ook de gevolgen voor het ontwerp en de daarmee gemoeide kosten expliciet besproken. Het doel is samen tot een gedragen oplossing te komen voor de belangrijkste ontwerpkeuzes. Pas als alle eisen tezamen eenduidig, bouwbaar en testbaar zijn, kan men met het ontwerpen beginnen.

Een belangrijk expliciet uitgangspunt hierbij is dat er concrete duidelijkheid van opdrachtgever naar de opdrachtnemer wordt gecreëerd en vice versa, zonder dat de contractuele verantwoordelijkheid van opdrachtnemer en opdrachtgever verwatert. Die duidelijkheid kan via eenvoudige procesafspraken worden geborgd. Notuleer het besprokene en stel gezamenlijk vast welke informatie een concretisering van de eisen behelst, welke informatie als toelichting is bedoeld en welke opmerkingen suggesties zijn. Zo voorkom je een bekende valkuil, namelijk dat de opdrachtgever de neiging krijgt terug te grijpen op de vaak wat abstractere eisen van de vraagspecificatie, die vaak voor meerdere interpretaties vatbaar zijn. Het kan bijvoorbeeld zijn dat een gebruikelijke eis vanuit de vraagspecificatie, zoals "alle 1^e orde falen dienen gemitigeerd te worden" in de SRR wordt doorvertaald naar een concrete opsomming van te mitigeren faalwijzen. De abstractere variant is niet-limitatief, en dus niet bouwbaar en testbaar. Als gevolg van de concretisering moeten nieuwe inzichten via een wijzigingsproces ingebracht worden, wat realistisch is in de contractuele verhoudingen en ook resulteert in een betere beheersing van deze vernieuwde inzichten.

5.2 Systeemarchitectuur

Het ontwerp van een systeem begint bij de architectuur: hoe wordt de functionaliteit opgedeeld in meerdere deelsystemen, en hoe interacteren die met elkaar? Hierbij is het primaire doel dat deelsystemen goed afgebakende delen van de functionaliteit afdekken, waarbij men het principe van "maximale cohesie en minimale koppeling" nastreeft. Maar de architectuur moet zich ook richten op de belangrijkste risico's,

¹⁰ Zie SMC Standard SMC-S-21 'Technical Reviews and Audits for Systems, Equipment and Computer Software'.

waaronder beschikbaarheid/betrouwbaarheid. Tot slot is een correcte beschrijving van dit alles noodzakelijk.

5.2.1

Overwegingen in de systeemarchitectuur en betrouwbaarheid/beschikbaarheid

Een vereist onderdeel van de architectuur is de beschrijving (view) van de invulling van beoogde beschikbaarheid/betrouwbaarheid, wat vaak een belangrijke bepalende factor is in architectuurbeslissingen. De architect beschikt hier over twee basisstrategieën: de kritieke taakuitvoering toewijzen en de benodigde faalkansen delegeren, of de faalkans verminderen via een specifieke verdeling en orkestratie van systeemonderdelen (denk aan redundantie via 3-out-of-2 multiprogramming units). In deze stap worden dan ook de eerste keuzes voor de budgettering van Q en λ over de verschillende systemen gemaakt.

Redundantie, multiprogramming en betrouwbaarheid/beschikbaarheid

Een belangrijke stap in het ontwerpen van missiekritische of veiligheidskritische systemen is de toewijzing van de faalkans aan de verschillende deelsystemen. Voor elke bedrijfskritische/veiligheidskritische taakuitvoering moet de ontwerper een verdeling maken over de verschillende deelsystemen, waarbij ook de andere architecturale overwegingen (zoals separation of concerns en maximum cohesion) bewaakt moeten worden.

Zeker in de architectuurfase is een focus op het realistisch budgetteren van de faalkans van groot belang. In theorie zou men daar vrij complexe RAMS-berekeningen op los moeten laten. De praktijk leert echter dat door relatief eenvoudige aannames voor de betrouwbaarheid van de individuele deelsystemen te combineren met betrouwbaarheidsnetwerken je een gedegen systeemontwerp kan opzetten. Door het ontwerpproces te beginnen met betrouwbaarheidsnetwerken (Reliability Block Diagram) en realistische aannames/budgetten voor de faalkans per deelsysteem, kan men vrij eenvoudig tot een goede architectuur komen die betrouwbaarheidseisen kan realiseren.

Kijken we naar de aannames voor de budgettering van de faalkans, dan is in de praktijk een faalkans van 10^{-2} realistisch, maar brengt een aantoonbare faalkans van 10^{-3} al snel hoge eisen met zich mee, waarvan het de vraag is of een softwareleverancier die eenvoudig kan waarmaken. Voor een aantoonbare faalkans van 10^{-4} is het bijzonder lastig om het daarvoor vereiste proces te realiseren. De machinerichtlijn acht een dergelijk niveau van betrouwbaarheid dan ook onrealistisch. Het is mogelijk wel te realiseren door middel van het betrouwbaarheidsnetwerk: het toepassen van multiprogramming-achtige oplossingen waardoor meerdere componenten gelijktijdig moeten falen om het totale systeem te laten falen. De praktijk wijst uit dat het vaak goedkoper is om twee goede multiprogrammed systemen te bouwen met elk een onafhankelijke faalkans van 10^{-2} dan om één bijna-perfect systeem met een faalkans van 10^{-4} te bouwen. Ook het introduceren van de mens als hersteller, wat inhoudt dat overgestapt kan worden op manuele bediening als de besturing faalt, kan op vrij eenvoudige wijze een grote verbetering in de faalkans bewerkstelligen.

Single points of failure en graceful degradation

Zeker voor veiligheidskritische en bedrijfskritische toepassingen is het van belang in het ontwerp vast te leggen in hoeverre systemen autonoom kunnen handelen en hoe andere deelsystemen omgaan met het falen van een deelsysteem. De bijbehorende kwaliteitseigenschap is 'foutbestendigheid' (zie par. 4.1.1 **Fout!** **Verwijzingsbron niet gevonden.**). Aan de gestelde eisen op dit punt wordt

voldaan als de ontwerper rekening houdt met het falen van deelsystemen om zo te voorkomen dat het falen van een deelsysteem uiteindelijk leidt tot het falen van het totale systeem. Deze systeemeigenschap kan actief gemodelleerd, en dus in de architectuurontwerpen getoetst worden door middel van modellering in betrouwbaarheidsnetwerken op functieniveau¹¹. Op basis hiervan kan ook goed worden gekeken naar 'graceful degradation', het doorfunctioneren met beperkte functionaliteit gedurende een storing. Hiervan is bijvoorbeeld sprake bij lokale deelsystemen die per tunnelsegment in onderhoud genomen kunnen worden zonder dat de gehele tunnel uit bedrijf wordt genomen.

De betrouwbaarheidsanalyse (variërend van een Reliability block diagram tot een Fault Tree Analysis) laat ook zien of bepaalde deelsystemen een Single Point of Failure (SPOF) vormen. RWS eist in de regel dat SPOFs in een object of systeem niet mogen voorkomen. Een typisch voorbeeld van een dergelijke SPOF is het arbitragesysteem dat voor veel installaties is voorgeschreven. Dit systeem bepaalt welke bedienvorm leidend is. Falen van die component blokkeert alle bedienvormen, wat voor veel functionaliteit tot onbeschikbaarheid zal leiden. Door deze taakuitvoering te decentraliseren en bewust safe defaults te introduceren wordt dit vermeden. In het voorbeeld van het arbitragesysteem werkt dit als volgt: bij het falen van besturingscomponenten wordt toegestaan dat deze op handbediening geforceerd worden overgenomen. Het voor het faalgedrag optimaal verdelen van taakuitvoeringen over (deel)systemen is dan ook een belangrijke taak in het architectuurontwerp.

5.2.2

Beschrijving van de architectuur

Het architectuur- en ontwerpproces leidt tot een beschrijving van de architectuur, die pas het gewenste effect heeft als die aantoonbaar en compleet de eisen afdekt, van alle kanten gecheckt is op omissies, fouten en onuitgesproken aannames en bovenal op de beschreven wijze geïmplementeerd is. De [ISO 42010] pakt deze uitdaging op door te specificeren dat de architectuurbeschrijving bestaat uit viewpoints en views, die de stakeholderbelangen afdekken. De afdekking van de gestelde eisen is in ieder geval een van de stakeholderbelangen. Gezien de variatie in stakeholders en hun belangen is er niet één vast template van viewpoints en views op te stellen. Het SSDD moet wel expliciet de gekozen viewpoints benoemen in relatie tot de stakeholders en hun belangen.

Een goed voorbeeld van een set views is het 4+1 model, dat verschillende views onderscheidt:

- **Logische view:** geeft een beschrijving van de functionele onderdelen van een systeem en de functionaliteit voor de eindgebruiker. Ook geeft deze view de randvoorwaarden voor het goed functioneren van de software en zijn onderdelen.
- **Development view:** geeft een beschrijving van het systeem vanuit de ontwikkelaar. Deze view beschrijft de organisatie van de software gedurende de ontwikkeling in termen van pakketten broncode, gebruikte standaardbibliotheken van code, compilers, linkers en de onderlinge afhankelijkheden daartussen.
- **Process view:** geeft een beschrijving van alle actieve processen gedurende executie, inclusief alle benodigde onderlinge communicatie, synchronisatie, redundantie en concurrency-control.

¹¹ Zie bijvoorbeeld <https://www.openmodelica.org/images/docs/2012-02-Hossain-Nyberg-Rogovchenko-Fritzson-mathmod2012-Functional-Safety.pdf> en <https://pdfs.semanticscholar.org/92a6/1b770301444d0d81862618588e0d924d1018.pdf>, die gebaseerd zijn op OpenModelica. Deze tool is rechtstreeks te koppelen aan Enterprise Architect

- **Physical view:** geeft een beschrijving van de hardware-infrastructuur (PLC's, servers en netwerken) en de allocatie van de runtime onderdelen daarop, vanuit het perspectief van de system engineer.
- **Use case view/Scenario's:** geeft, door middel van een kleine set voorbeelden (use cases) of scenario's, inzicht in de samenhang tussen de verschillende componenten. Hierbij wordt expliciet niet alleen het primaire scenario beschreven, maar ook de alternatieve scenario's die ontstaan als gevolg van redundantie of monitoring met restartfunctie (watchdog resets). Deze view moet ten minste alle uit te voeren taken weergeven die voor de betrouwbaarheidsanalyse van belang zijn. De scenario's zullen als basis dienen voor het te bouwen systeem en het testprotocol.

Een belangrijk onderdeel van de beschrijving van de architectuur is het beschrijven van de rationale van de architectuur. Hierin worden de belangrijkste ontwerpbeslissingen vastgelegd, evenals de minder voor de hand liggende keuzes, inclusief de achterliggende overwegingen. Doel hiervan is te voorkomen dat bij toekomstig onderhoud fouten worden geïntroduceerd doordat men de architectuur niet begrijpt.

5.2.3

Traceerbaarheid van eisen

Bij het opzetten van de architectuur is het noodzakelijk de eisen traceerbaar te maken. Er zijn twee soorten traceerbaarheid van eisen:

- **Forward traceability:** het structureel vastleggen waar elke eis is geïmplementeerd. Dit is het eenvoudigst te realiseren door middel van een Verification Cross Reference Index, waarbij voor elke eis beschreven wordt waar deze geïmplementeerd is en hoe.
- **Backward traceability:** hier gaat het erom vast te leggen dat ontwerpbesluiten door de eisen worden ondersteund.

De kerngedachte bij de 'Forward Traceability' van eisen is dat deze integraal onderdeel is van het ontwerpproces, en zo de compleetheid en gedegenheid van het ontwerp waarborgt. Ook voorziet 'Forward Traceability' het project van een helder afgebakend eindpunt. Het is daarmee een belangrijke registratie, omdat hij aantoont dat alle eisen geïmplementeerd worden, wat tevens de essentie is van de TOPAAS-scoring. De IEC 61508 beveelt deze vorm van traceerbaarheid aan voor SIL-2, maar laat ruimte om hiermee te stoppen bij de architectuur, het detailontwerp of de code.

De kerngedachte van 'Backward Traceability' is dat er geen overbodige zaken worden gebouwd. De praktijk leert echter dat rechtstreekse 'Backward Traceability' naar de eisen vaak niet haalbaar is, omdat de eisen onvoldoende specifiek zijn. Dit moet dan gecompenseerd worden door een tussenlaag van expliciete ontwerpbesluiten, zodat alle zaken die ontworpen en gebouwd worden omdat ze noodzakelijk zijn voor de oplossing, te traceren zijn.

5.2.4

Competenties van de ontwerpers

Een ontwerp is zo goed als het ontwerpteam, en daarmee zijn de competenties van de ontwerpers een belangrijk onderdeel van de kwaliteit van het uiteindelijke product. Deze moeten niet alleen goed kunnen ontwerpen en kennis hebben van de gebruikte technologie, maar ook het domein, en de problemen die daar doorgaans in spelen, goed kunnen doorgronden. Dit kan worden gewaarborgd doordat de ontwerpers zelf voldoende ervaring hebben in het domein, óf door ontwerpers in te zetten die die kennis wel hebben in een coachende rol.

Waar het gaat om de inbreng van domeinkennis blijft er altijd een taak liggen bij opdrachtgever. Het proces van SRR (opdrachtgever legt uit, opdrachtnemer stelt vragen) wordt in de ontwerpfase omgekeerd: in de PDR en CDR legt de opdrachtnemer uit en stelt de opdrachtgever de vragen. De opdrachtgever heeft de verantwoordelijkheid om de opdrachtnemer op domeinkennis te bevragen, vooral in PDR.

5.2.5 *Ontwerprichtlijnen*

Ontwerprichtlijnen bieden input voor het ontwerpproces en een goede ondersteuning van het ontwerpteam. Dergelijke richtlijnen zijn niet bedoeld om beginnende ontwerpers voor beginnersfouten te behoeden, maar dienen meer als een collectief geheugen voor ontwerpers, waarin eerdere ervaringen met oplossingsrichtingen zijn opgeslagen. Ze kunnen bijvoorbeeld gebruikte ontwerpoplossingen bevatten voor veelvoorkomende problemen, en de ervaringen die daarmee zijn opgedaan (zogenaamde architectuurpatronen). Ook kunnen ze afgeraden oplossingen bevatten (ook wel anti-patterns genoemd).

5.2.6 *Hergebruik van componenten*

In een architectuur kan het handig zijn (deel)oplossingen te hergebruiken. Dit heeft als voordeel dat het een component kan opleveren met een bewezen betrouwbaarheid. In het geval van hergebruik is het van belang vast te stellen dat componenten geschikt zijn. Daarvoor moet men vaststellen of de omgeving waarin de oplossing succesvol opereerde, zowel technisch als qua beoogde toepassing voldoende overeenkomt met de toekomstige toepassing.

5.2.7 *Ontwerpreviews*

Een ontwerppreview vervult een belangrijke rol bij de borging van de kwaliteit van een ontwerp. Thema's die hierin aan de orde komen, zijn:

- Voldoet het ontwerp aan de eisen?
- Wat zijn de risico's die voortkomen uit het ontwerp?
- Is het ontwerp testbaar?
- Is het ontwerp bouwbaar?

Aanwezigen die bij een dergelijke review betrokken zijn, nemen inhoudelijk verantwoordelijkheid voor deze thema's.

Het Fagan-inspectieproces helpt bij het verbeteren van de effectiviteit van de reviews. De essentie van Fagan-inspecties is dat iedere reviewer vanuit een andere invalshoek naar het document kijkt, en de specifieke rol die de auteur en de moderator daarin hebben. Binnen de inspectie wordt de aandacht gericht op het vinden van 'major' fouten, waarvan tijdige reparatie een grotere hoeveelheid rework voorkomt. Hierdoor is het mogelijk documenten met meer diepgang te inspecteren. Het Fagan-inspectieproces zorgt ervoor dat eerst al het commentaar wordt geïnventariseerd en gecategoriseerd, en dat alleen (discussie)tijd wordt besteed aan essentiële discussiepunten.

Inhoudelijk zijn er meerdere methodes beschikbaar, zoals de CDR vanuit SMC-S-21 en SEI's ATAM.

Het is verstandig de opdrachtgever te betrekken bij de review van de architectuur, om zo ook te borgen dat de eisen juist zijn geïnterpreteerd en correct zijn doorvertaald naar het architectuurontwerp. Het doel hiervan is dus vast te stellen of het architectuurontwerp voldoet aan de eisen. Een te beheersen risico is dan wel dat de opdrachtgever mogelijk zal willen streven naar een nog veel beter product,

waardoor deze vorm van 'gold plating' al snel oneindige reviewsessies kan opleveren.

Algemene opmerking over Agile-methoden

Projecten bij Rijkswaterstaat kenmerken zich door een sterke nadruk op de aantoonbaarheid van betrouwbaarheid en veiligheid. In de softwaredevelopmentwereld is een werkwijze gestoeld op Agile-principes de standaard. Veel van de elementen daarvan (early integration, daily standup, visual management, retrospective) zijn een-op-een bruikbaar. Een aantal principes echter niet (emerging architecture en de embracing change, met daarbij de insteek dat ver vooruitkijken niet realistisch is). Alhoewel er aanpassingen mogelijk zijn op de Agile-werkwijze (zoals *SafeScrum* en *ScrumR*) zodat deze beter zou passen binnen de context van een Rijkswaterstaat-project, blijft de registratie van aantoonbaarheidsinformatie binnen het proces een probleem. Wij hanteren daarom een waterval-achtige aanpak.

5.2.8

Omgang met Model driven development

In plaats van een aanpak waarbij men handmatig code schrijft, kan men ook kiezen voor Model driven development (MDD). Dit is een veelbelovende aanpak binnen software engineering om software te ontwikkelen. Zeker omdat binnen het RWS-domein veel objecten gestandaardiseerd worden (door de landelijke tunnelstandaard, landelijke bruggenstandaard, etc.), lijkt het genereren van een oplossing op basis van herbruikbare en bewezen bouwstenen een gezonde aanpak. Door de oplossing te beperken tot het ontwikkelen van een model wordt het project overzichtelijker en dus beheersbaarder. Ook kan men op basis van het wiskundig gesloten model modelverificatie doen en testcases genereren. Hierdoor kan men in een vroegtijdig stadium al de werking van systemen modelleren en testen, wat in de regel de kwaliteit verhoogt, zeker als men gebruik kan maken van codegeneratie en/of bewezen bouwstenen.

Het inzetten van codegeneratie is een architecturale beslissing: het verandert het risicoprofiel ten opzichte van handmatig overzetten, verlegt de allocatie van activiteiten en vereist andere competenties binnen het realisatieproject. De codegenerator zelf dient ook onderwerp van verificatie te zijn.

5.2.9

Independent Verification and Validation (IV&V)

Naast procesmatige borging kan men ook kijken naar onafhankelijke verificatie en validatie. Doel hiervan is om onafhankelijk te laten vaststellen dat producten (of processen) aan hun kwaliteitscriteria voldoen. Kernvragen hierbij zijn of eisen daadwerkelijk zijn aangetoond, of documenten compleet, duidelijk en consistent zijn, etc. Door deze controles buiten de reguliere organisatie te plaatsen, spelen de aannames die ontwerpers normaal maken, geen rol, wat de kwaliteit ten goede komt.

Een discussiepunt is hoe onafhankelijk een IV&V-organisatie moet zijn. Tot en met SIL-2 is een andere afdeling van de organisatie acceptabel, vanaf SIL-3 moet dit een externe partij zijn.

Om 'besmetting' met de ontwerpaannames te voorkomen is het van belang dat de IV&V-organisatie deels zelf zijn informatie vergaart bij de opdrachtgever.

5.3 Borging van processen en producten

5.4 Activiteiten

Hoofdactiviteit 1: De opdrachtnemer toetst in de oriëntatiefase de vraagspecificatie en stelt deze, waar nodig, ter discussie om tot een bouwbare en testbare set aan eisen te komen. Leg de resultaten vast in een document. (zie par. 5.1).

Het is van belang dat de opdrachtnemer uiteindelijk de eisen kan accepteren, waarbij bouwbaarheid en testbaarheid in beschouwing zijn genomen. Daarnaast is het van belang dat de opdrachtgever na de oriëntatiefase weet waar de software aan gaat voldoen.

Hoofdactiviteit 2: Stel de systeemarchitectuur dusdanig op dat die invulling geeft aan de vraagspecificatie. Neem in de overwegingen expliciet redundantie-overwegingen, multiprogramming, single-points-of-failure en graceful degradation mee (zie par. 5.2.1).

Onderdeel van deze activiteit is de toewijzing van eisen aan deelsystemen. Onder andere faalkansanalyses kunnen een manier zijn om het effect van toewijzingen goed te beschouwen en single-points-of-failures te identificeren.

Hoofdactiviteit 3: Stel een duidelijke architectuurbeschrijving op en gebruik daarbij expliciete stakeholders en verschillende viewpoints (zie par. 5.2.2).

Er zijn verschillende views die hiervoor gebruikt kunnen worden, zoals beschreven in par. 5.2.2.

Hoofdactiviteit 4: Zorg dat de invulling aan eisen traceerbaar is (zie par. 5.3).

Hoofdactiviteit 5: Voer een goede ontwerppreview uit en gebruik hiervoor bekende methoden, zoals Fagan-inspecties (zie par. 5.2.7).

Hoofdactiviteit 6: Borg processen en producten door het toepassen van een goed intern kwaliteitssysteem.

5.5 Op te leveren producten/documenten

Planfase:

- Software Development Plan (SDP)
- (Review van) Test Master Plan (TMP)
- Validatieplan softwarebetrouwbaarheid [1]

Ontwerp:

- SRR (notulen/verslag)
- SSDD
- Hazops/FTA
- (bijdrage aan) Product Risico analyses
- SRS/SDD
- (Review van) STD
- PDR (meetmoment), evt. meerdere
- CDR (mijlpaal)

6 Fase 3: Realisatie

6.1 Ondersteunende tools en softwarebibliotheken

Softwarebibliotheken en ondersteunende tools hebben invloed op de software, en moeten dus ook beheerst worden. Eerst gaan we in op de zekerheid van de kwaliteit van standaardcompilers, dan bespreken we de betrouwbaarheid van de compiler en uiteindelijk bespreken we gehanteerde softwarebibliotheken.

6.1.1 *Zekerheid van de kwaliteit van de compiler*

De compiler zet de broncode om in executeerbare code voor een specifiek target platform. Voor veiligheidskritische code is het gebruikelijk om code-optimalisaties in een compiler uit te zetten, omdat de vaak complexe optimalisatieslag als potentiële bron van fouten wordt gezien. Ook moeten ontwikkelaars de kwaliteit van de compiler zelf goed in beschouwing nemen; een kritische blik op de herkomst en het track-record van de compiler is noodzakelijk. Er zijn compilers die (veelal door TÜV) zijn gecertificeerd voor een bepaald SIL-niveau. Tot SIL-2 is het heel acceptabel om een compiler te gebruiken waarmee een ontwikkelaar en de industrie goed bekend zijn, en waar men dus de problemen wel van kent.

Een compiler wordt in de regel gebruikt met een safe subset van de taal: een set van instructies waarvan is vastgesteld dat deze goed functioneren (bijvoorbeeld door intensief gebruik) en geen onverwachte bijeffecten hebben. De safe subset wordt doorgaans beschreven in een codeerstandaard en expliciet geborgd via reviews en/of statische code-analysetools.

Bij gecertificeerde compilers gebruikt men bij voorkeur een validatieset: een stuk broncode dat bij wijze van test gecompileerd kan worden en waar de uitkomst van bekend is. Zo kan men door middel van een goed gekozen test aantonen dat alle instellingen goed staan en er geen gekke dingen zijn gebeurd met de compiler en/of gebruikte softwarebibliotheken.

6.1.2 *Zelfgemaakte generatietools*

Als men voor een project zelf generatietools bouwt (of laat bouwen), zoals compilers, dan is dit product ook onderdeel van het project. Hier maken we onderscheid tussen twee soorten ondersteunende tools:

- **On-Line tools** (in de IEC 61508 ook wel T1 en T2 supporting tools genoemd). Denk daarbij aan runtime interpreters en vergelijkbare oplossingen.
- **Off-line tools** (in de IEC 61508 ook wel T3 supporting tools genoemd). Hierbij moet je denken aan zelfgebouwde converters die compileerbare code genereren, zoals de omzetting van I/O-lijsten naar typicals, etc.

Het verdient aanbeveling dit soort ondersteunende tools ook te specificeren. Met name (beperkingen aan) de input, als ook de output. Voor conversietools moet je denken aan compilertechniek, en dan is een minimumvereiste in de vorm van het ontwikkelen van een LL1-grammatica van de verwachte input wel noodzakelijk.

Online tools moeten aan dezelfde eisen voldoen als de rest van de oplossing: ze vormen immers een integraal onderdeel van de totale oplossing. Omdat ze in de

oplossing zeer waarschijnlijk bij meerdere componenten/TUB¹²s ingezet worden, leveren ze mogelijk zelfs een common-cause bijdrage aan de faalkans.

Offline tools zijn eenvoudiger te beheersen: dit zijn vrij vaak conversie-achtige tools waarvan het grootste risico is dat ze code produceren die op onmerkbare wijze niet goed functioneert. Dit wordt deels gemitigeerd door:

- te zorgen dat de converter geen output produceert als er fouten in de input worden geconstateerd;
- het conversieproces inhoudelijk te testen, wat betekent dat op basis van een bestand met alle mogelijke constructies een vooraf gedefinieerde output wordt genereerd;
- de output van deze tools inhoudelijk te (laten) reviewen, mogelijk met een steekproef;
- de tools onder versiebeheer te houden en te waarborgen dat de geproduceerde code met goedgekeurde versies van de tools is geproduceerd;
- het testen van de geproduceerde code.

Ook leert de ervaring dat het gebruik van dergelijke conversietools het noodzakelijk maakt de mogelijke input te specificeren.

6.1.3

Softwarebibliotheken

Softwarebibliotheken zijn vaak onderdeel van de compiler (engineering station) en het gebruikte target platform. Door het intensieve gebruik is dit vaak een zeer betrouwbare functionaliteit. Wel moet je vooraf zekerstellen dat de functionaliteit gebruikt wordt zoals die bedoeld is, en er bijvoorbeeld geen onnodige afhankelijkheden gecreëerd worden met het SCADA doordat de softwarebibliotheken verwachten dat alarmen gekwiteerd moeten worden voordat processen in gang gezet kunnen worden.

Wat betreft het configuratiemanagement is het van belang goed in kaart te hebben welke functies van softwarebibliotheken gebruikt worden, met name om bij updates van de softwarebibliotheken te kunnen toetsen of daar wijzigingen in hebben plaatsgevonden. Er zijn gevallen bekend waarbij wijzigingen in softwarebibliotheken de oplossing nadelig beïnvloeden. Dus het upgraden van bibliotheken dient met beleid te worden uitgevoerd en vereist vaak een regressietest om zeker te stellen dat er niet ongemerkt fouten worden geïntroduceerd.

6.2 Externe borging codekwaliteit door SIG/Expleo/Etc.

Externe partijen (zoals bijvoorbeeld SIG, Expleo, IFsQ en Omnext) bieden diensten aan om de kwaliteit van de broncode van een project te borgen. In de praktijk zien we dat SIG met name naar codemetriekeken kijkt die van belang zijn voor het onderhoud van de broncode. Wanneer je structureel de omvang en complexiteit monitort, krijgt je een beeld van waar de hotspots van de oplossing zitten. Wel moet worden opgemerkt dat de definities van dergelijke partijen vaak afwijken van die welke bij TOPAAS gebruikt worden. Zeker als het om omvang en complexiteit van de software gaat. De door deze partijen geproduceerde getallen zijn niet direct bruikbaar voor TOPAAS-analyses.

Codemetriekeken zeggen zeker niet alles, zelfs niet als het alleen om onderhoudbaarheid gaat. Daarom is soms een bredere blik noodzakelijk. Expleo biedt een wat ruimere blik, en kijkt bijvoorbeeld ook naar de architectuur van de oplossing.

¹² TUB=Taakuitvoeringsblok, zie ook TOPAAS

We tekenen hierbij wel aan dat alle genoemde organisaties relatief weinig ervaring met PLC-oplossingen hebben.

6.3 Activiteiten

Hoofdactiviteit 1: Borg de kwaliteit en de toepasbaarheid van de compiler of zelfgemaakte generatietool om fouten bij de vertaling naar het target platform te voorkomen.

Hoofdactiviteit 2: Pas externe borging van codekwaliteit toe door gebruik van externe partijen als SIG, Expleo, IFsO of Omnext.

6.4 Op te leveren producten/documenten

Realisatiefase:

- Test software/scripts (executable test suite)
- Code/unit test
- Componenttesten (SW-SW-integratie)

7 Fase 4: Testen

Testen is een integraal onderdeel van het project, en vormt ook een belangrijk onderdeel van de aantoonbaarheid van de oplossing.

Testen levert de informatie die nodig is om te bepalen of en in hoeverre het systeem voldoet. Men kan door testen niet aantonen dat het systeem vrij is van fouten. Ook is systeembetrouwbaarheid in de regel niet aan te tonen door testen; testen kennen een bias naar specifieke situaties en de statistiek is vaak te beperkt om gedegen uitspraken te kunnen doen.

De basis voor de testplannen wordt, net als de meest essentiële processen, uitgezet vanuit het Master Testplan (MTP). In het MTP wordt de algehele teststrategie voor de verschillende testsoorten vastgelegd conform T-Map. Deze testsoorten worden vervolgens in onderliggende STP's en STD's beschreven. Ook worden in het MTP de verschillende (OTAP-)omgevingen beschreven.

7.1 Onafhankelijkheid van de testorganisatie

Testers controleren de kwaliteit van de producten van de ontwikkelaars, en moeten daarom onafhankelijk van de ontwerpers en het ontwerp opereren. Dit geldt voor alle activiteiten die V&V-gerelateerd zijn. De IEC 61508¹³ vereist voor SIL-2 dat de onafhankelijkheid minimaal op persoonsniveau geborgd moet worden, en bij systemen met een hoger SIL-niveau moet zelfs een onafhankelijke afdeling of organisatie worden ingeschakeld.

Het doel van deze scheiding is het voorkomen van groepsdenken, waarbij (foutieve) aannames van ontwerpers door testers worden overgenomen. Door de ontwerpen en testen van hetzelfde niveau gelijktijdig en onafhankelijk te beschrijven (ook een vereiste van de IEC 61508¹⁴), kan de ON dit ook borgen. Het detecteren van interpretatieverschillen in ontwerp en testen is dan typisch een onderdeel van de kruislingse reviews van de documenten nadat deze documenten beide af zijn. Deze aanpak vereist dat vanuit hetzelfde uitgangspunt zowel ontwerp als test uitgewerkt moet kunnen worden.

7.2 Teststrategie

Een belangrijk onderdeel van een teststrategie is het methodisch bepalen van de diepgang van de testen. Voor systemen met veiligheidseisen en/of beschikbaarheidseisen is de risicoanalyse daarbij leidend. De complexiteit van software is in die systemen dermate hoog dat het volledig doortesten van alle toestanden waarin de software zich kan bevinden, praktisch onmogelijk is. Projecten hebben niet oneindig veel tijd en geld tot hun beschikking en dus is het noodzakelijk om een prioritering aan te brengen waarbij niet-(veiligheids)kritische functies ondergeschikt zijn aan de (veiligheids)kritische functies. Deze aanpak wordt Risk Based Testing (RBT) genoemd. Een goede teststrategie wordt (ook) meegenomen in het ontwerpproces. Zo kunnen speciale testinterfaces worden ingevoegd om anders onmogelijke testen effectief uit te voeren.

Beproefde werkwijzen zoals T-Map geven door middel van een Product Risico Analyse (PRA) richting aan het aanbrengen van de prioritering, iets wat ook door de

¹³ Zie IEC 61508, deel 1, §8.2.18

¹⁴ Zie IEC 61508, deel 3, §7.4.7.1, §7.4.8.1, §7.9.2.1

IEC 61508 wordt ondersteund. Hierbij moet dus per beschikbaarheids-/veiligheidsrisico bepaald worden:

1. wat de sequentie aan gebeurtenissen is die aanleiding geven tot gevaarlijk gedrag;
2. hoe groot de kans van optreden is;
3. wat de gevolgen van optreden zijn.

In essentie is dit een Failure Mode, Effect (and Criticality) Analysis (FME(C)A) of een HAZard and OPERability studie (HAZOP) waarmee de impact vastgesteld kan worden. In een PRA worden minimaal de volgende impactcategorieën onderscheiden:

1. **Catastrofaal falen:** falen met vele doden en/of brede economische/maatschappelijke impact tot gevolg.
2. **Veiligheidskritisch falen:** falen met enkele doden of lokale economische/maatschappelijke impact tot gevolg. Hieronder vallen de faalwijzen waarbij de machineveiligheid in het geding is.
3. **Kritiek falen:** falen van het object zonder direct effect op de veiligheid van mens en omgeving, maar wel met (langdurige) onbeschikbaarheid tot gevolg.
4. **Hoge ernst:** schade aan het object die het functioneren in de toekomst belemmert.

Op basis van een dergelijke risicoanalyse is initieel het te realiseren faalkansniveau per taakuitvoering beschreven, waarna deze ook op basis daarvan getest moet worden. Voor de kans van optreden wordt vaak de complexiteit en omvang van de uitvoerende software als een proxy gehanteerd. Alhoewel op (deel)systeemniveau het maximale niveau bepaald wordt, kan/moet men dus per (veiligheids)kritische functie differentiëren in het testregime.

Vanuit de IEC 61508 wordt voor SIL-2 het volgende vereist voor veiligheidsfuncties:

- Voor unit-tests:
 - Dynamische analyse en testen, en dan specifieker het bepalen van de testgevallen op basis van grenswaardeanalyse.
 - Er moet gebruik gemaakt worden van 'Functional and Black Box testing', en meer specifiek van equivalentieklassen van input-parameters en grenswaardeanalyse.
- Integratietesten:
 - Er moet gebruik gemaakt worden van 'Functional and Black Box testing', en meer specifiek van equivalentieklassen van input-parameters en grenswaardeanalyse.
- Acceptatietesten:
 - Er moet gebruik gemaakt worden van 'Functional and Black Box testing', en meer specifiek van equivalentieklassen van input-parameters en grenswaardeanalyse.

7.3 Bepaling dekkingsgraad testen

Een belangrijk onderdeel van elke teststrategie is het bepalen van de minimale dekkingsgraad van testen. Middels een methodische testaanpak kan een requirements, design coverage of code coverage maat worden opgenomen. Denk hierbij aan gedrag dat is gespecificeerd via een state machine: de dekkingsgraad van testen kan op basis van alle transities en grenswaarden van de zogenaamde guards worden ingesteld. In de IEC 61508 gaat men uit van een code coverage van 100% (óf entry points óf statements óf branches óf condities) voor het testen van

units. Hierbij hanteert men wel het 'Comply or Explain'-principe: als het niet mogelijk is de dekkingsgraad te realiseren moet er een verklaring worden opgesteld die uitlegt welke situaties niet getest worden en welke aanvullende maatregelen zijn genomen om de daardoor ontstane risico's te mitigeren.

Op testen die zich op hogere abstractieniveaus bevinden, zoals SAT en FAT, is een lagere dekkingsgraad vaak acceptabel: het is in de regel niet haalbaar om in een keten alle vertakkingen van de code te raken.

7.4 Tracerbaarheid van eisen naar testen

Bij het opzetten van de testplannen is het noodzakelijk de eisen traceerbaar te maken naar de testplannen. Dit gebeurt door middel van 'Forward Traceability'. De kerngedachte is dat zo eenvoudig kan worden aangetoond dat aan de eisen is voldaan, wat van groot belang is voor de acceptatie door de opdrachtgever. Een beschrijving van de manier waarop de eisen met betrekking tot de betrouwbaarheid van de taakuitvoering zijn ingevuld, vormt hier een belangrijk onderdeel van.

Dit is ook de essentie is achter de TOPAAS-scoring: als alle eisen expliciet en aantoonbaar getest zijn, geeft dat vertrouwen in de goede werking van de oplossing. De IEC 61508 beveelt deze vorm van traceerbaarheid aan voor SIL-2, maar laat ruimte om hiermee te stoppen bij de verschillende testniveaus.

7.5 Testomgevingen

Er zijn in een project vaak meerdere testomgevingen in gebruik, de OTAP-straat. OTAP staat voor Ontwikkeling, Test, Acceptatie en Productie. Unit-testen worden veelal in de ontwikkelomgeving uitgevoerd (vaak geautomatiseerd). Alle integratietesten worden in de testomgeving uitgevoerd en de acceptatietesten in de acceptatieomgeving. Uiteindelijk wordt de software op de productieomgeving geïnstalleerd. Een softwareversie 'promoveert' dus van de ene omgeving naar de andere.

Er worden geen specifieke eisen aan deze omgevingen gesteld, behalve dat ze technisch geschikt moeten zijn voor het uitvoeren van de tests waarvoor ze bedoeld zijn. In theorie zou de A-omgeving technisch zo dicht mogelijk bij de P-omgeving moeten komen om zo, tijdens de integratie van het systeem in het object, te kunnen focussen op fouten vanuit het aansluitschema. De acceptatieomgeving lijkt dan ook in alle essentiële configuratie-items op de productieomgeving. Dat betekent dat hier ten opzichte van de testomgeving ook geen 'vervuiling' bestaat in de vorm van bijvoorbeeld diagnose-tooling, en dat de relevante interfaces exacte replica's zijn van de productieomgeving. Met 'essentieel' en 'relevant' wordt bedoeld dat deze aspecten bijdragen aan het kritische gedrag van de TUB's. Opgemerkt wordt wel dat de praktijk vaak weerbarstig is: het gedrag van werktuigen en sensoren die onder fysieke load daadwerkelijk handelingen verrichten/monitoren is vaak op subtiële wijze anders dan de test-stubs. Een realistische testomgeving, zeker in de A-omgeving, is wel degelijk relevant, maar het belang ervan kan ook overdreven worden. Bij veel systemen kan men volstaan met een hele serie schakelaars en een goed draaiboek voor de simulatie van een productieomgeving. Een praktisch aandachtspunt is wel het introduceren van false-positives, zeker als het om performance en timing van het systeem gaat. Het is dus bij een analyse van de representativiteit van een testomgeving van belang om vast te stellen of de afwijkingen ten opzichte van de P-omgeving onterecht tot goedkeuring kunnen leiden, en hoe dat risico gemitigeerd kan worden.

De acceptatieomgeving valt, net als de rest van de OTAP-straat, onder configuratie- en wijzigingsbeheer. Acceptatietesten vinden zowel plaats op de werking van het geïntegreerde systeem als ook op de uitrol en rollback.

Op basis van de 'Cybersecurity implementatierichtlijn objecten RWS' is men in de praktijk verplicht om na oplevering van de productieomgeving de acceptatieomgeving in stand te houden ten behoeve van analyse en het faciliteren van upgrades. Traditionele aanpakken waarbij de A-omgeving gepromoveerd wordt naar de P-omgeving, leveren in dat opzicht gelijk een probleem op.

7.6 Registratie van testresultaten

Een belangrijk onderdeel van de herhaalbaarheid van testen is het vastleggen van gegevens. Uit de praktijk blijkt dat een combinatie van geautomatiseerd testen en logging op dit vlak het meest effectief is. Handmatig testen is onvermijdelijk (denk aan power cycles, opstart, gedrag bij losse connectoren), maar de vastlegging van gegevens moet even secuur gebeuren als een automatisch script zou doen.

De IEC 61508 schrijft voor dat van alle units, subsystemen en systemen moet worden bijgehouden aan welke testen ze onderworpen zijn, wat de resultaten zijn, welke (testontwerp- en acceptatie-)beslissingen genomen zijn en welke problemen zijn ontdekt. In praktijk betekent dit dat eindresultaten van testen moeten worden geregistreerd in het Software Test Report (STR). Deze resultaten worden vervolgens via het configuratiemanagementsysteem gekoppeld aan een specifieke versie van het configuratie-item. Op deze wijze ontstaat er, na herhaalde integratie en testen, een structuur in het configuratiemanagementsysteem waarbij de bewijsvoering voor vrijgave automatisch wordt opgebouwd.

Om betrouwbaarheidsgroeimodellen te kunnen toepassen (zie 8.7.4) moet van testintervallen tevens worden bijgehouden op welke tijden een interval is gestart en geëindigd en op welk moment defecten zijn opgetreden. Waar van toepassing moet daarbij bekend zijn of dit geautomatiseerde (test)runs waren, hoeveel parallelle operaties er zijn gestart en/of hoeveel gebruikers er gelijktijdig handelingen met het systeem verrichtten.

7.7 Afhandeling van afwijkingen

Het detecteren van afwijkingen van de waargenomen resultaten ten opzichte van de verwachte resultaten is de primaire start van een analyse. Deze analyse moet uitwijzen of er sprake is van een verkeerde verwachting bij de testers en/of of de implementatie door de ontwikkelaars niet klopt. Na deze stap wordt de echte oorsprong van de fout opgespoord. Als de implementatie niet kloppend is ten opzichte van het ontwerp, is dit een issue 'geïnjecteerd in development, gevonden in een test'.

Voor de classificatie van de afwijkingen wordt sterk aanbevolen de classificatie van de Product Risico Analyse (PRA) over te nemen.

Ongeacht of de afwijking nu wel of niet conform specificatie is, legt de IEC 61508 hier de verplichting op om vooraf vast te stellen of het corrigeren van het issue niet meer risico introduceert dan het wegneemt. Dit is dan ook een vereiste aan het afhandelingsproces, waarbij in het geval van veiligheidskritische functies alleen de veiligheidsverantwoordelijke van het object mag beslissen over acceptatie.

De meest praktische aanpak is dat gevonden afwijkingen via het change advisory board (CAB) en de wijzigingsprocedure afgehandeld worden, een aanpak die

conform de IEC 61508 is: volgens deze norm moet de veiligheidsverantwoordelijke betrokken zijn bij het oplossen en/of accepteren van risico's die voortkomen uit deze afwijkingen (zie ook paragraaf §8.3 'Wijzigingsbeheer').

7.8 Releasemanagement en vrijgave naar de productie

In de standaard T-Map-aanpak wordt vaak uitgegaan van grenswaarden van aantallen fouten waarbij de testmanager vrij mag geven naar de volgende integratietest of productie. Bij (veiligheids)kritische omgevingen is deze aanpak echter uit den boze: alleen de veiligheidsverantwoordelijke mag, onafhankelijk van de testmanager, beslissen of een (sub)systeem de volgende testronde in mag.

Voor vrijgave naar een volgende fase worden de volgende zaken meegewogen:

- Welke testen hebben plaatsgevonden en met welk resultaat?
- Welke testen hebben niet plaatsgevonden, en waarom niet, en wat zegt dit over het risico?
- Wat is de dekkinggraad van de uitgevoerde testen?
- Welke afwijkingen zijn geïdentificeerd?
- Welke afwijkingen zijn gecorrigeerd en hoe zijn deze na correctie geanalyseerd of getest?

Met name de vrijgave van een veiligheidskritisch systeem naar productie mag uitsluitend gebeuren door de (veiligheids-)verantwoordelijke van de opdrachtgever. De reden hiervoor is onder meer dat de (veiligheids)verantwoordelijke van de opdrachtgever de enige is die de impact kan inschatten. Daarnaast kan de combinatie van meerdere afwijkingen alsnog tot een onacceptabele situatie leiden.

Voor vrijgave moeten op zijn minst de volgende zaken bekend zijn:

- Geïdentificeerde afwijkingen en hun ingeschatte impact
- Welke testen hebben plaatsgevonden en met welk resultaat?
- Welke testen hebben niet plaatsgevonden en waarom niet?
- Aan welke eisen is voldaan en aan welke niet?
- De resulterende dekkinggraad van testen, en welke taakuitvoeringen niet zijn getest

7.9 Testen van modificaties en regressietesten

Bij modificatie van een stuk software, ongeacht de reden, wordt in principe een compleet nieuw product ter keuring aangeboden aan de tester. Een wijziging van slechts één regel broncode leidt zeer waarschijnlijk tot een software product (de executable) met meer dan alleen een minimale lokale wijziging. Gezien de noodzaak om dit product volledig te testen, kan een impactbeoordeling nuttig zijn om te bepalen welke delen van het product niet gewijzigd zijn en dus minder zouden hoeven worden getest. Om zeker te zijn dat een wijziging geen bijeffecten heeft, moet ook de omliggende functionaliteit gericht worden getest. Dit vereist vaak een analyse van de code, die identificeert welke functionaliteit nog meer geraakt zou kunnen worden door een wijziging. Op basis van de IEC 61508 is regressietesten van een wijziging een vereiste op SIL-2-niveau.

7.10 Activiteiten

Hoofdactiviteit 1: Opstellen MTP/Teststrategie

Hoofdactiviteit 2: Inrichten testorganisatie, alle zaken als PMP/QMP

Hoofdactiviteit 3: Uitwerken testontwerp (STD), logische testcases

Hoofdactiviteit 4: Uitwerken fysieke testen, scripts, testdata, omgevingen

Hoofdactiviteit 5: Uitvoeren testen

Hoofdactiviteit 6: Analyseresultaat en rapportage

7.11 Op te leveren producten/documenten

MTP

--- PDR ---

Review FMEA
Review SSDD
PRA

--- CDR---

PRA

STP/STD per CI (review SRS/SDD)

STR per test

8 Fase 5: Gebruik, Beheer & Onderhoud

8.1 Inleiding

De fase 'Gebruik, Beheer & Onderhoud' start op het moment dat het object of systeem door RWS is geaccepteerd. Op dat moment wordt het geheel overgedragen aan de lijnorganisatie en de verantwoordelijke beheerder. De focus ligt op de instandhouding en het blijvend voldoen aan de prestatie-eisen van het systeem of object.

Omdat een object onderhevig is aan veroudering, slijtage en veranderende omstandigheden, zijn onderhoud, renovatie en vervanging of uitbreiding essentieel. De beheers- en onderhoudsmaatregelen die voortvloeien uit de objectrisicoanalyse (ORA) en onderhoudsanalyse zijn opgenomen en geborgd in het instandhoudingsplan (IHP). Levenscycluskosten (Life Cycle Costs, LCC) vormen een belangrijke parameter daarin^[15]. De uitvoering van dit plan omvat onder meer gepland onderhoud, inspecties, testen, verzamelen van storingsdata, uitvoeren van een Root Cause Analysis (RCA, zie par. 8.7.1) en een analyse van de werkelijke beschikbaarheid^[16].

Door de toenemende verwevenheid van hardwarecomponenten met onderliggende IT-systemen heeft het onderhoud op deze componenten ook in toenemende mate gevolgen voor de software, en daarmee voor de softwarebetrouwbaarheid. Bij het selecteren van een onderhoudsstrategie dient de impact op de software(betrouwbaarheid) mee begroot te worden. Daarbij moet ook in acht worden genomen wat nodig is voor het beheer van software. In de volgende paragrafen zijn de verschillende vormen van onderhoud beschreven¹⁷. Verder wordt het proces van wijzigingsbeheer toegelicht, waarbij een richtlijn wordt gegeven voor het betrekken van betrouwbaarheidsrisico's bij de impactanalyse van voorgestelde wijzigingen.

Daarnaast levert de analyse van de werkelijke beschikbaarheid van het object inzicht in de prestaties van het IT-systeem en in de betrouwbaarheid van de software. Zo wordt antwoord gegeven op de volgende vragen: in hoeverre komt de werkelijkheid overeen met de uitkomsten van de oorspronkelijke faalkansanalyse en is er aanleiding om een of ander bij te stellen? De werkwijze voor die bijstelling is beschreven in paragraaf 8.5 waarin ook het principe van Bayesiaans updaten wordt toegelicht. Dit vereist dat de operationele gegevens die inzicht bieden in de prestaties van het systeem, worden bijgehouden. De paragraaf 'Logging' geeft daar een richtlijn voor.

Tot slot is een paragraaf gewijd aan de terugkoppeling van operationele ervaringen naar het ontwikkelproject en de leverancier. Daarin wordt onder meer beschreven hoe betrouwbaarheidseisen in een Service Level Agreement (SLA, zie par. 8.7.5) voor tweedelijnslijnondersteuning kunnen worden opgenomen. Ook komen het belang van de Root Cause Analysis (RCA) en het onderhoud aan een regressietestset en acceptatieomgeving aan de orde.

8.2 Vormen van onderhoud en consequenties voor software

¹⁵ Kader Life Cycle Cost (LCC) 2014, ww-nummer #1485, Rijkswaterstaat, december 2002

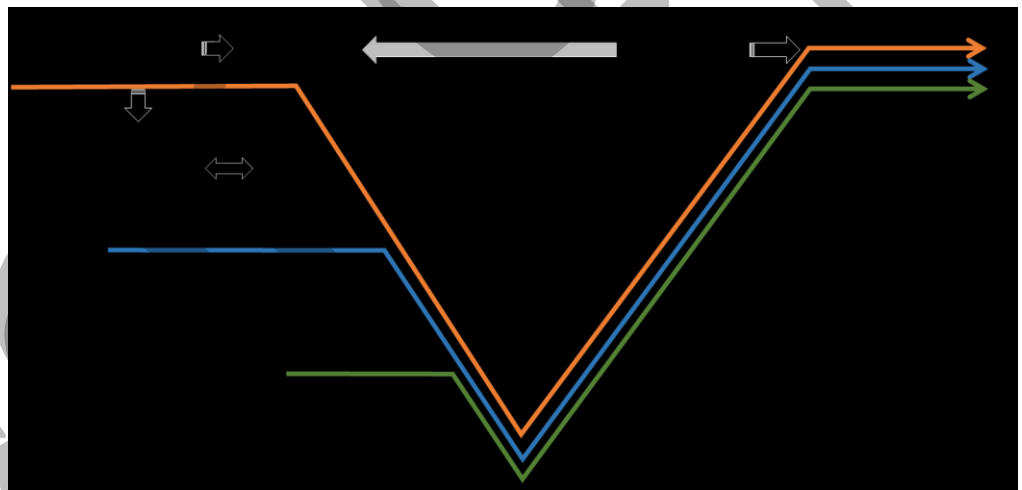
¹⁶ Handreiking Prestatiegestuurde Risicoanalyses, 1.0.1, maart 2018, Steunpunt ProBo

¹⁷ Met betrekking tot het wijzigen van software wordt geen onderscheid gemaakt tussen 'onderhoud' en 'modificatie'.

Op basis van de risicoanalyse en Life Cycle Costs (LCC) analyse vindt een onderhoudsanalyse plaats waarmee de optimale onderhoudsstrategie van een hardwarecomponent wordt bepaald. Mogelijke onderhoudsstrategieën zijn:

- SAO (StoringsAfhankelijk Onderhoud): vervanging van onderdelen op het moment dat zich een storing voordoet
- GAO (GebruiksAfhankelijk Onderhoud): vervanging van onderdelen op basis van kalendertijd of aantal gebruiksmomenten
- TAO (ToestandsAfhankelijk Onderhoud): vervanging van onderdelen op basis van vooraf bepaalde storingsvoorspellende grootheden (bijvoorbeeld de hoeveelheid ijzerdeeltjes in de olie van een motor)

Modificaties van het object en onderhoud aan hardwarecomponenten leiden in veel gevallen tot noodzakelijke aanpassingen aan het IT-systeem, en daarmee aan de software. Voor elke aanpassing worden in principe alle stappen van het V-model doorlopen, zoals in de vorige hoofdstukken beschreven: stel de specificaties vast, stel de architectuur vast, stel een ontwerp op, programmeer de software, enz. We zeggen met nadruk 'in principe', want afhankelijk van de aard en de impact van de wijziging start men niet altijd helemaal links bovenaan in het V-model. Soms gelden er nieuwe eisen. Soms zijn de bestaande eisen onverkort van toepassing, maar moet het ontwerp worden aangepast. En soms volstaat alleen een wijziging in de code. Het komt dus hierop neer: je bepaalt het startpunt in het V-model en doorloopt vanaf dat punt de rest van de stappen in de juiste volgorde (dat kan ook iteratief zijn).



Figuur 8.1: Onderhoud en het V-model

Er worden verschillende vormen van onderhoud aan software onderscheiden. De bekendste zijn:

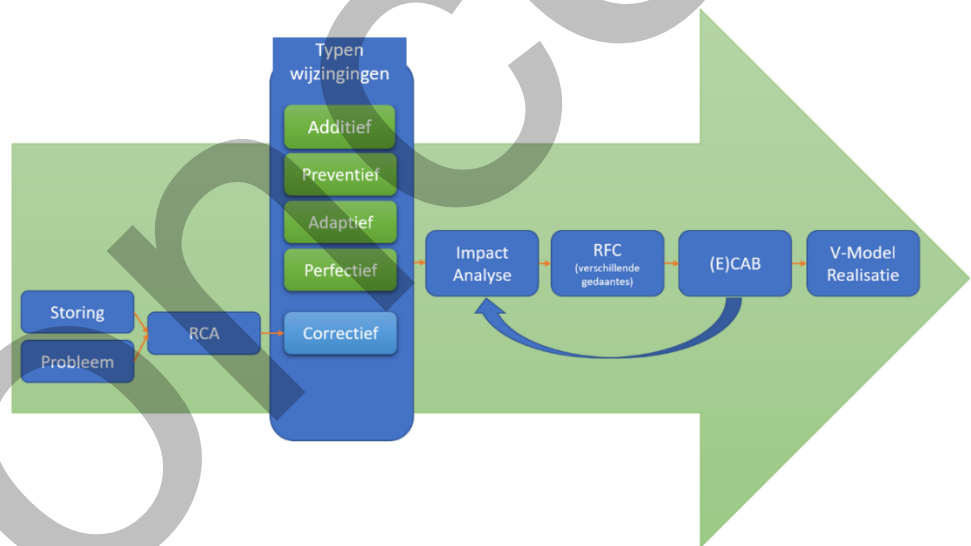
- **Correctief onderhoud:** vindt plaats naar aanleiding van een fout of verstoring in een functie van de software.
- **Perfectief onderhoud:** hierbij worden prestaties en kwaliteit van de software verbeterd zonder functionele wijzigingen aan te brengen.
- **Preventief onderhoud:** voorkomt mogelijk toekomstige problemen, zoals het vollopen van diskruimte, het ontstaan van kwetsbaarheden.
- **Adaptief onderhoud:** vindt plaats naar aanleiding van een wijziging in de technische installatie of wensen van de gebruikers.
- **Additief onderhoud:** hierbij worden nieuwe functionaliteiten toegevoegd.

Voor alle soorten wijzigingen moet de vaste procedure voor wijzigingsbeheer doorlopen worden. Daarin is vastgelegd hoe het besluitvormings- en acceptatieproces rond een wijziging dient te verlopen. Deze procedure wordt in de volgende paragraaf toegelicht.

8.3 Wijzigingsbeheer

Wijzigingsbeheer ziet toe op de besluitvorming rond het doorvoeren van voorgestelde aanpassingen aan het IT-systeem. De procedures daarvoor zijn gebaseerd op standaardwerkwijzen, die beschreven staan in ITIL¹⁸. Een cruciaal onderdeel daarvan is het analyseren van de impact van elke wijziging op het systeem en met name op zijn softwarebetrouwbaarheid.

Hoewel lineair afgebeeld, is een impactanalyse een processtap die tot aanpassingen van bijvoorbeeld ontwerpscenario's kan leiden, waarna opnieuw een impactanalyse uitgevoerd wordt, net zolang tot de impact van alle benodigde wijzigingen compleet is.



Figuur 8.2: Behandeling van wijzigingsverzoek (RFC).

Wijzigingsverzoeken worden volgens het standaardwijzigingsbeheersproces afgehandeld (zie figuur 8.2). Daartoe voer je de volgende vijf stappen uit:

1. Bepaal de wijziging.
2. Bepaal het wijzigingsgebied.
3. Voer de risicoanalyse uit.
4. Verwerk de maatregelen risicoanalyse in de RFC (Request For Change).
5. Besluit over de RFC.

1. Bepaal de wijziging

Breng de gewenste wijziging in kaart. Wat is het kerndoel van de wijziging, wat is de nieuwe situatie?

Wat zijn de omgevingsparameters, eventueel nieuwe randvoorwaarden, etc.?

Definieer de prioriteit van de wijziging en sluit alle referenties bij, bijvoorbeeld naar

¹⁸ Generieke Procesbeschrijvingen ITIL voor de Industriële Automatisering in Objecten van RWS.

een probleem-ID uit de ITIL-omgeving of een voorgenomen wijziging aan het object.

Geef aan wat de gewenste oplossingsrichting is of de mogelijke oplossings-scenario's zijn.

2. Bepaal het wijzigingsgebied

Breng het complete wijzigingsgebied in kaart. Het wijzigingsgebied betreft alle direct betrokken configuratie-items in de betreffende onderdelen van het V-model.

Denk hierbij niet alleen aan de software en hardware, maar ook aan alle bijbehorende configuratie-items, zoals beschrijvingen, handleidingen, testware, trainingsmaterialen, processen en procedures, logfiles en geautomatiseerde interpretaties daarvan, processormodel, architectuur, personen, (onderhouds)-contracten, etc.

3. Voer de risicoanalyse uit

Aan de hand van stap 1 en 2 wordt een risicoanalyse uitgevoerd. Kijk voor de faalkans ook expliciet naar de impact op:

- de gevarenanalyse en FMECA;
- de Foutenboomanalyse (FTA);
- de prestatiegestuurde risicoanalyse (PRA)¹⁹ van het gehele object;
- de vigerende TOPAAS-afschatting;
- de compliance met IEC 61508.

In paragraaf 8.4.5 wordt een toelichting gegeven op de verschillende risicoanalyses.

Bepaal voor ieder onderdeel van de wijziging op welke plek in het V-model begonnen moet worden.

4. Verwerk de maatregelen risicoanalyse in de RFC

Maak op basis van de stappen 2 en 3 een overzicht van de risico analyse maatregelen voor de RFC. Dit kan een iteratief proces zijn totdat oplossingsalternatieven zijn gevonden die een optimum vormen tussen risico, impact, kwaliteit, kosten, doorlooptijd en gewenste functionaliteit.

Veranker een eventueel nodige bijwerking of complete uitvoer van de FMECA, FTA, PRA en TOPAAS-afschatting in de RFC middels de juiste processtap in het V-Model.

5. Besluit over de RFC

De RFC wordt voorgelegd bij de Change Advisory Board (CAB), die over de doorvoering van de wijziging besluit. Eventueel kan met het advies van de CAB de wijziging nog worden aangepast.

Bij het beoordelen van de mogelijke oplossingen wordt de risicoanalyse beschouwd in het licht van het totale object. Een van de afwegingen is of het probleem inderdaad een oplossing vereist: soms is het beter een bekend probleem te laten voortbestaan, dan het risico te lopen nieuwe, onbekende problemen te introduceren.

Rol van de objectbeheerder en CAB

Wanneer een wijzigingsverzoek wordt ingediend, treedt het wijzigingsadviesorgaan in werking. De 'Change Advisory Board' (CAB) beoordeelt een aangevraagde wijziging op noodzaak en impact. Een impactanalyse omvat ook de gevolgen voor de softwarebetrouwbaarheid. De CAB bestaat in de regel voor zover aanwezig uit vertegenwoordigers van de leverancier of de instantie die de wijziging aanbrengt, de verantwoordelijke technisch manager van RWS, de faalkansmanager van RWS en de

¹⁹ Prestatiegerichte Risico Analyse (dus niet de Product Risico Analyse uit ISTQB).

verantwoordelijke objectbeheerder. Op basis van de analyse adviseert de CAB de objectbeheerder over het al dan niet doorvoeren van de wijziging.

Rol van de IEC 61508 binnen het besluit

Punt van aandacht bij de besluitvorming is dat in IEC 61508 is opgenomen dat wijzigingen in systemen die in een netto veiligheidsverlies op objectniveau resulteren, in principe niet geaccepteerd mogen worden. Dit kan dus betekenen dat een wijziging wordt geweigerd omdat het middel mogelijk erger is dan de kwaal.

8.4 Proces van wijziging

Geaccepteerde wijzigingen worden op een beheerste manier doorgevoerd. Het team dat de wijziging doorvoert, doorloopt de relevante processtappen van het V-model. Welke stappen dat zijn, hangt af van de aard en impact van de wijziging.

8.4.1

Correctief onderhoud

Storingen in het object dienen te worden opgelost als ze niet zonder interventie verdwijnen of kunnen worden vermeden. Voor (hardware)componenten, kan dit plaatsvinden door middel van een reset, reparatie of vervanging. Wanneer storingen (incidenten) in het onderliggende IT-systeem escaleren tot een storing in het object, kan de beheerder besluiten een wijzigingsverzoek in te dienen om de oorzaak van de softwarestoring weg te nemen. Root Cause Analysis (zie par. 8.7.1) geeft inzicht in de oorzaak van storingen en problemen (zie par. 8.7.1). Te denken valt bijvoorbeeld aan een vlinderklep die niet sluit vanwege een ontbrekend signaal vanuit de besturingssoftware²⁰. De ernst van de fout, het belang van de functie en de impact van de oplossing bepalen de keuze voor en de prioriteit van het correctief onderhoud.

Van nieuwe, vernieuwde of aangepaste componenten wordt de nieuwe faalkans opgenomen in de foutenboom. Met behulp van TOPAAS wordt bepaald of en hoe de faalkans van de betreffende TUB's wordt beïnvloed. Belangrijke aandachtspunten daarbij zijn:

- de kennis en ervaring van de medewerkers die de correcties aanbrengen
- de veranderde complexiteit of omvang van de TUB
- de toepassing van geldende architectuurconcepten
- de dekkingsgraad van de uitgevoerde (regressie-)testen²¹

Het optreden van de fout zelf kan aanleiding zijn om een niet eerder onderkende faalmodus op te nemen in de FMECA. Dat kan van invloed zijn op de structuur van de foutenboom, die dan dienovereenkomstig wordt aangepast.

8.4.2

Preventief en perfectief onderhoud

Preventief of perfectief onderhoud aan het IT-systeem betreft bijvoorbeeld het vervangen van de PLC's, het upgraden van operatingsystemen of het vervangen van switches. Het kan hierbij gaan om zeer ingrijpende wijzigingen of zelfs complete nieuwbouw van de onderliggende software.

Preventief onderhoud aan software lijkt minder voor de hand liggend. Software is op zichzelf immers niet aan slijtage onderhevig. Toch kan planmatig onderhoud aan software noodzakelijk blijken als een bepaalde versie van de programmeeromgeving in de toekomst niet langer door een leverancier wordt ondersteund. Als wordt

²⁰ Fouten in de infrastructuur van het object, zoals een vlinderklep die niet sluit vanwege slijtage aan een draaipunt, worden hier buiten beschouwing gelaten.

²¹ Regressietesten zijn testen waarmee wordt geverifieerd of ongewijzigde componenten nog op dezelfde wijze werken als voorheen.

besloten de software niet te upgraden naar een nieuwe versie, doet dit weliswaar geen afbreuk aan de bestaande functionaliteit en betrouwbaarheid, maar kan dit andere vormen van onderhoud in de toekomst ernstig bemoeilijken.

8.4.3 *Adaptief en additief onderhoud*

We spreken van adaptief of additief onderhoud in het geval dat er wijzigingen aan het object worden aangebracht die wijzigingen in het functioneel of technisch ontwerp en in het IT-systeem met zich meebrengen.

Dergelijk onderhoud grijpt binnen het V-model in op het niveau van systeemontwerp, soms zelfs op het niveau van eisedefinitie. Dat betekent dat alle stappen in het V-model vanaf dat punt doorlopen worden, als ware het een nieuwbouwtraject (zie Figuur 8.1).

8.4.4 *Andere aanleidingen voor onderhoud*

Analyse van gebruiksgegevens en logging

Periodiek worden de gebruiks- en loggingsgegevens geanalyseerd (zie par. 8.5). Dit geeft inzicht in de prestaties van het systeem. Trends daarin kunnen wijzen op toekomstige problemen; tabellen die 'vollopen', performance die afneemt, toenemend aantal fouten in communicatie, etc. Het aanbrengen van wijzigingen naar aanleiding hiervan zou je ook 'correctief onderhoud' kunnen noemen, zij het dat de eventuele fout zich nog niet heeft geopenbaard. Voorstellen voor verbeteringen kunnen door diverse partijen (opdrachtgever, gebruiker, opdrachtnemer) worden geformuleerd en worden ter beoordeling voorgelegd aan de CAB (zie 8.3).

Ervaringen met andere objecten

Wanneer componenten in meerdere installaties en objecten worden gebruikt, kunnen ook gebruiks- en storingsgegevens van die andere objecten aanleiding zijn om onderhoud uit te voeren. Bestaande (regionale) overlegorganen wisselen gegevens hieromtrent uit op grond waarvan wijzigingsvoorstellen worden geformuleerd. De wijzigingsvoorstellen kunnen betrekking hebben op zowel hardware- als softwarecomponenten. Ook deze voorstellen worden voorgelegd aan de CAB van het specifieke object.

Aanbevelingen door leveranciers

Naast voorstellen tot verbetering vanuit de eigen organisatie, kunnen ook wijzigingsvoorstellen worden gedaan door leveranciers van componenten. Wanneer zo'n voorstel betrekking heeft op softwarecomponenten, bestaat dit doorgaans uit de release van een nieuwe versie of patch op een bestaande versie van de software in het IT-systeem. Het kan ook gaan om een nieuwe release naar aanleiding van upgrades van onderliggende operatingsystemen of apparatuur, of ervaringen van de leverancier met andere objecten. Leveranciers van hardware en firmware melden gewoonlijk bekende problemen aan hun klanten. Met name bij security-problemen worden klanten soms proactief gewaarschuwd. Andere leveranciers raden klanten aan om periodiek hun website te raadplegen om te kijken of er nog nieuwe waarschuwingen publiek gemaakt zijn. Ook worden via diezelfde kanalen end-of-life waarschuwingen voor producten uitgegeven. Het is dus zaak om deze informatie structureel te monitoren omdat die softwareonderhoud noodzakelijk kan maken.

Het wijzigingsvoorstel dat wordt ingediend bij de CAB bevat alle eventuele aanpassingen en alle (release)informatie die nodig is om een adequate impactanalyse uit te voeren.

Monitoring van de leverbaarheid van reservedelen

De hardware waar de software op draait, is niet oneindig leverbaar. Ook worden er vaak subtiele wijzigingen doorgevoerd in firmware en libraries. Alhoewel er in de regel op de locatie van een object wel een reservedelenvoorraad is, is er altijd het risico dat doordat een product van de markt wordt gehaald (end-of-life), de reservevoorraad te beperkt wordt om tijdens de vervangingsperiode van de hardware (een modificatietraject) de beschikbaarheid van het object te garanderen. Zeker nadat is aangekondigd dat specifieke hardware-/firmwareversies van de markt worden gehaald, is het zaak om het moment waarop deze voor het laatst besteld kunnen worden (last time buy) in de gaten te houden.

8.4.5

Specifieke stappen in het V-model

Bij het aanbrengen van wijzigingen in het IT-systeem en onderliggende software zijn specifieke acties met betrekking tot de softwarebetrouwbaarheid van toepassing.

Gevarenanalyse en FMECA

Wijzigingen in de eisen aan het object of daaraan gerelateerde wijzigingen in het ontwerp van de installatie brengen mogelijk nieuwe gevaren of faalmodi met zich mee. Daarom adviseren we om een evaluatie uit te voeren van de bestaande gevarenanalyse en FMECA. Wanneer er door vervanging van componenten geen nieuwe faalmodi worden geïntroduceerd, wijzigt de structuur van de foutenboom niet.

Ook zonder nieuwe of gewijzigde gevaren en faalmodi dient de FMECA geëvalueerd te worden; bij het aanbrengen van bepaalde wijzigingen in het systeem kan een reeds onderkende faalmodus tot andere gevolgen leiden.

Foutenboomanalyse

Bij nieuwe of gewijzigde gevaren of faalmodi is het duidelijk dat de bestaande foutenboom moet worden geëvalueerd en waar nodig aangepast. Gewijzigde componenten kunnen eveneens nieuwe faalkansen met zich meebrengen die in de foutenboom verwerkt moeten worden. De faalkansmanager van het object is verantwoordelijk voor de foutenboomanalyse en de afweging of en in hoeverre wijzigingen in het object de kans op de Ongewenste Top Gebeurtenis (OTG) doen toenemen.

Basic Software Event

De kans op basic software events²² wijzigt mogelijk wanneer de inhoud van de TUB²³ wijzigt, bijvoorbeeld door een aanpassing van de betreffende programmatuur, of door een wijziging in de begrenzing van de TUB. In beide gevallen wordt aan de hand van een evaluatie van de TOPAAS-score bepaald of en in hoeverre de faalkans wordt beïnvloed (in positieve of negatieve zin!), hierbij wordt in ieder geval meegewogen:

- Kennis en ervaring van de medewerkers die de wijzigingen uitvoeren
- Veranderde complexiteit of omvang van de TUB
- De toepassing van geldende architectuurconcepten
- De dekkingsgraad van de uitgevoerde (regressie-)testen

Borgen van compliance met IEC 61508

²² Hier: een basic software event dat in een foutenboom een softwarefaalmodus beschrijft.

²³ TUB=Taakuitvoeringsblok, zie ook TOPAAS

Wanneer in het oorspronkelijke (nieuwbouw)project sprake was van toepassing van een Safety Integrity Level (SIL) en de bijbehorende werkwijze en organisatie ook daadwerkelijk zijn doorgevoerd, dan is dat in de TOPAAS-analyse van de TUB's meegenomen. Bij het aanbrengen van wijzigingen in de software dienen dan eveneens de richtlijnen voor het gehanteerde SIL te worden gevolgd!

8.5 Logging en bewaking in relatie tot softwarebetrouwbaarheid

De doelstelling van logging is om, net als in een vliegtuig, over een black box te beschikken, waarmee chronologisch kan worden gereconstrueerd hoe de softwarepaden feitelijk worden doorlopen. Dit leidt tot informatie over hoe vaak en wanneer softwaredelen (subroutines of functies) werden aangeroepen in het feitelijke gebruik. Tot deze softwaredelen behoren zowel de (sub)routines voor de primaire taakuitvoering als ook secundaire functies, zoals de foutafhandeling. Daarnaast geeft logging een goed beeld van de feitelijke input aan de code, zodat bij geconstateerde fouten een realistische testcase kan worden geconstrueerd. Dit maakt het mogelijk om uitspraken te doen over de betrouwbaarheid van de software.

Net als bij een black box wordt zeer selectief omgegaan met de te loggen gegevens.

De (software)risicoanalyses geven aan welke gebeurtenissen en gegevens, naast de minimaal vereiste gegevens, gelogd moeten worden, hoe lang logs bewaard moeten worden, en met welke regelmaat de logs aan een review worden onderworpen. Zorg er bij gedistribueerde softwaremodules voor dat de timestamps nauwkeurig zijn gesynchroniseerd.

Loggingseisen hebben dezelfde status als functionele eisen en dienen hetzelfde proces in het V-model te doorlopen, inclusief architectuur, design, ontwikkelen en testen.

De logs zijn een onmisbare bron voor de (periodieke) beoordeling van de goede werking van het systeem als ook voor een RCA (zie par. 8.7.1) naar aanleiding van storingen en problemen, en fungeren na elke uitrol als controlemiddel.

8.5.1

Inhoud Logs

Minimale logging omvat Event logging en Monitoring logging, waarbij geborgd moet worden dat de logging compleet, correct, precies en niet-gemanipuleerd plaatsvindt. De genoemde vormen van logging leggen de volgende gegevens vast:

Event logging:

- ID van het CI, de routine/module of TUB (een TUB is doorgaans te grofmazig)
- timestamp-log
- versie van de module
- relevante omgevingsparameters/statusinformatie
- correlation ID: een ID dat het event koppelt aan events in andere logs (bij HTTP-verkeer is dat bijv. een session id)
- event start
- event exit
- exit-code van de routine
- foutroutine: timestamp, aanroepend CI, kritische fouten, waarschuwingen, uitvoeringsfouten, informatie

Monitoring logging:

- uptime-informatie van de in de FMECA en FTA benoemde modules/taakcodes die bij een TUB horen
- notificaties bij afwijkingen van de verwachte monitoringwaarde
- checksum van de software- en hardwareconfiguratie om wijzigingen te detecteren
- geautomatiseerde auditresultaten van bijvoorbeeld controles, zoals een vergelijking van de actuele configuratie met CMDB

8.5.2 *Bewaartermijn logs*

Logs moeten bewaard worden gedurende een periode die lang genoeg is om gebeurtenissen op een relevante manier te reconstrueren. Hierbij moet ook worden zeker gesteld dat er voldoende data uit incidenteel aangesproken taken vergaard worden. De bewaartermijnen moeten per TUB worden bepaald; zij volgen uit de ontwerpfase en risicoanalyse.

Denk bij de bepaling van gepaste bewaartermijnen goed na of je cyberbedreigingen wilt mee-evalueren. Doorgaans is een aanvaller maandenlang of langer 'slappend' aanwezig, voordat hij toeslaat. In dat geval wil men nog steeds op de logs kunnen terugvallen voor analyse.

8.5.3 *Periodieke review*

Naast de geautomatiseerde afhandeling van audit-checks op de logs en het geautomatiseerde beheer daarvan (verplaatsen, consolideren, back-up, etc.) is periodieke handmatige controle, bijvoorbeeld eens per maand, noodzakelijk. De wijze waarop deze plaatsvindt en de precieze frequentie volgen wederom uit de functionele en risicogedreven eisen.

Telkens wanneer de foutafhandeling wordt aangesproken, treedt de procedure voor storingen in werking. Dat houdt in dat storingen geregistreerd worden in de ITIL/CMDB-managementomgeving en dat ze in het geordende storingsproces worden opgenomen.

Aanvullende opmerkingen

Logging kan op elke omgeving in de OTAP-straat plaatsvinden voor test en verificatie van het systeem, om omgevingen te kunnen vergelijken en omdat de primaire logging onderdeel is van het systeem.

Op andere omgevingen dan de productieomgeving kan gebruik gemaakt worden van uitgebreidere logging. Deze uitgebreidere logging mag bij een uitrol naar productie niet overgenomen worden, bijvoorbeeld omdat het uitrol proces configuraties overschrijft.

Voor de RCA (zie (8.7.1)) van storingen en problemen worden logging-gegevens gebruikt. Als de logging voor deze analyse onvoldoende lijkt, kan besloten worden om via het wijzigingsproces de logging uit te breiden.

8.6 Bayesiaans updaten

Bij de oplevering van een systeem wordt een initiële faalkansafschatting van TOPAAS over de betreffende configuratie-baseline afgegeven. Deze inschatting blijft geldig tot de configuratie-baseline wijzigingen ondergaat die om een nieuwe TOPAAS-afschatting vragen.

Tegelijkertijd worden door het gebruiken en testen van het systeem observatiedata gegenereerd. Deze observaties (in principe faalgebeurtenissen) kunnen gebruikt

worden voor een vergelijking met de TOPAAS-afschatting: zo kan beoordeeld worden of er, gezien de observaties, meer of minder vertrouwen is in de eerder ingeschatte faalkans. Hiervoor mogen uiteraard geen observaties worden gebruikt die reeds meegenomen zijn bij de oorspronkelijke TOPAAS-afschatting.

Voor de vergelijking wordt dus de TOPAAS-afschatting als initiële faalkans gebruikt (à priori faalkans: P_T). Een observatie bestaat uit data over het gedrag van het systeem.

Aan de hand van deze observatiegegevens wordt nagaan hoe groot de waarschijnlijkheid is dat de observaties overeenkomen met een systeem van faalkans (P_O), met andere woorden: komen de aantallen succesvolle en falende aanspraken op de software overeen met de verwachting? Een dergelijke analyse noemen we triangulatie. Voor deze analyse is het nodig dat we zowel faal- als succesgedrag van de software vastleggen.

Om ook over voldoende data te beschikken van laagfrequent aangesproken TUB's zal gebruik gemaakt worden van representatieve data uit de simulatieomgeving - doorgaans de acceptatieomgeving.

Uitgaande van de twee kansen (P_T en P_O) bepalen we niet een 'hogere' kans, maar een mate van consistentie. In het geval van onvoldoende consistentie kan ertoe besloten worden de TOPAAS-bepaling te onderzoeken en/of om onderzoek te doen naar het datacollectieproces. In principe blijft P_T de voor de foutenboomanalyse geldende faalkans.

Binnen het triangulatieproces zijn twee invalshoeken voor het gebruik van de data mogelijk: sequentieel en cumulatief.

Bij sequentieel berekenen worden de data in incrementen verwerkt, zodat het resultaat steeds opnieuw wordt bijgesteld. Bij voldoende data kan dat leiden tot een tussentijdse bijstelling van de kans (zie figuur 8.3).

Wij behandelen hier alleen de cumulatieve manier.

Voor de cumulatieve berekening wordt de Bayesiaanse update-formule gebruikt:

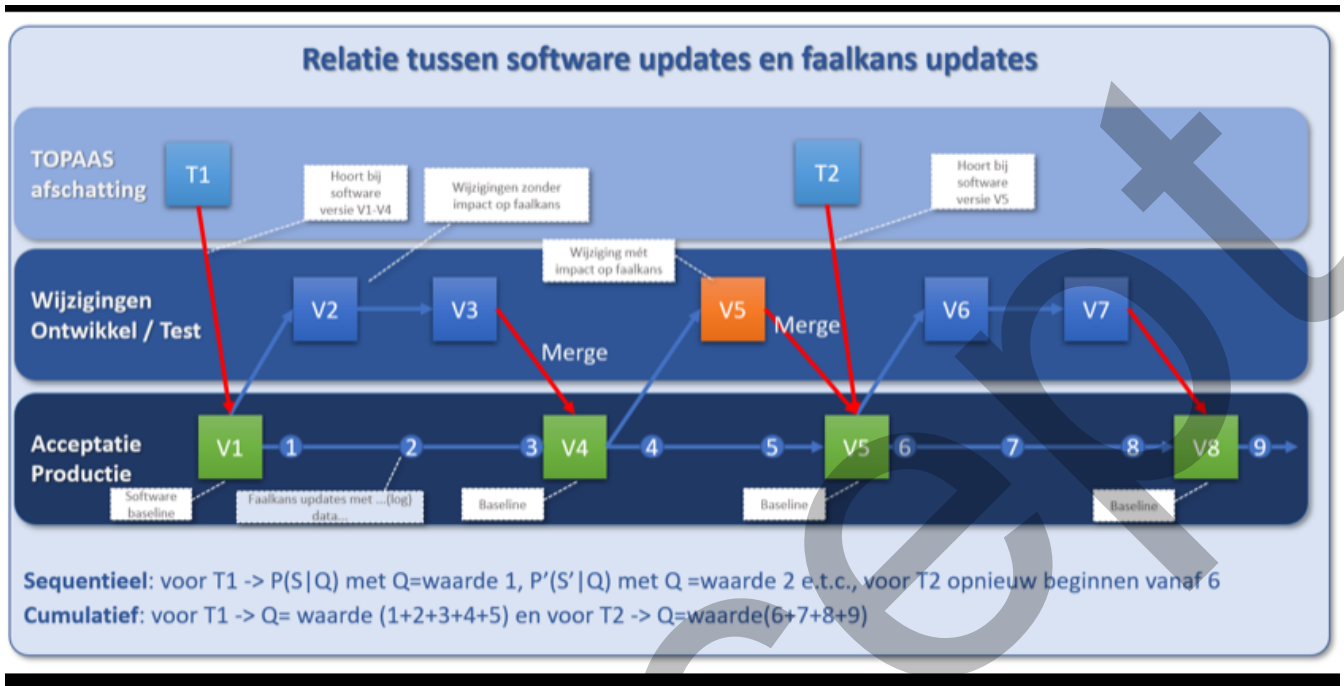
$$P(T|O) = \frac{P(O|T) * P(T)}{P(O)}$$

waarbij:

$P(T|O)$ de kans is dat je een score T hebt bij een Observatiereeks (O), ofwel de likelihood dat score T hoort bij observatiereeks O .

Deze kans wordt bepaald met:

- $P(O|T)$: de kans dat in het laatste increment een faalevent staat, gegeven TOPAAS-score T . Deze wordt bepaald door de dataset met faal events wordt beschouwd als een niet-homogeen Poisson-proces data distributie. Daarbij wordt in eerste instantie de eventuele samenhang tussen bijvoorbeeld de gebruikte tools, testen en het optreden van fouten (clusters) genegeerd. De clusteranalyse vindt plaats op basis van loggingdata.
- $P(T)$ = TOPAAS-kans
- $P(O)$ = de faalkans als bepaald binnen de observatieset O



Figuur 8.3: Trianguleren faalkans met updates

8.7 Overig gebruik, beheer en onderhoudsaspecten

8.7.1

Root Cause Analysis (RCA)

Root Cause Analysis (RCA) is het analyseren van de onderliggende oorzaak van een opgetreden storing of defect. Een RCA wordt zo nodig tijdens het ontwikkeltraject uitgevoerd op faaldata in testresultaten van het systeem. Maar de analyse moet in ieder geval plaatsvinden bij storingen in de productie. De uitkomst van de analyse helpt mogelijk andere storingen te voorkomen en kan als basis dienen voor maatregelen die soortgelijke storingen in andere objecten en systemen moeten voorkomen. Daartoe moet de analist toegang hebben tot alle documentatie en ontwerpen van het systeem. Uit de analyse kunnen aanbevelingen volgen voor het uitvoeren van preventief onderhoud en het aanpassen van engineeringdocumenten en richtlijnen.

8.7.2

Acceptatie-omgeving

De acceptatie-omgeving lijkt voor wat betreft alle essentiële configuratie-items op de productieomgeving. Dat betekent dat hier ten opzichte van de testomgeving geen 'vervuiling' in de vorm van bijvoorbeeld diagnose-tooling voorkomt, en dat de relevante interfaces exacte replica's zijn van die in de productieomgeving. Met 'essentieel' en 'relevant' bedoelen we dat deze aspecten aan het kritische gedrag van de TUB's bijdragen, of anders gezegd: no risk = no test.

De acceptatieomgeving valt, net als de hele OTAP-straat, onder configuratie- en wijzigingsbeheer. Acceptatietesten worden zowel uitgevoerd op de werking van het geïntegreerde systeem als op de uitrol en rollback.

Omgevingen zijn doorgaans kostbaar, zodat een acceptatieomgeving ook gebruikt wordt voor onder andere trainingsdoeleinden en het repliceren van storingen in de productie. Door in de acceptatieomgeving de kritische functies, maar vooral ook de

functionaliteiten die in de productie slechts incidenteel aangesproken zullen worden, langdurig en onder verschillende testscenario's, leidend tot een optimale Testmaat-N, aan te spreken, worden data verkregen over zowel het gewenste gedrag als het faalgedrag van de software. Zie ook **Fout! Verwijzingsbron niet gevonden.**

8.7.3 *Regressietesten*

Met regressietesten wordt bepaald of en in hoeverre het totale IT-systeem na het aanbrengen van wijzigingen op onderdelen nog aan alle eisen voldoet. Een onderdeel daarvan is dat ook wordt geverifieerd of de ongewijzigde delen van de software nog onveranderd werken. Dit is een cruciaal onderdeel van het testtraject bij elke vorm van onderhoud. De mate van effectiviteit van de test hangt sterk af van de mate waarin de regressietestset up-to-date is. Bij onderhoud op het object en het onderliggende IT-systeem moet altijd ook het onderhoud op de regressietestset worden meegenomen. Zie verder paragraaf 7.9.

8.7.4 *Reliability Growth Modelling (RGM)*

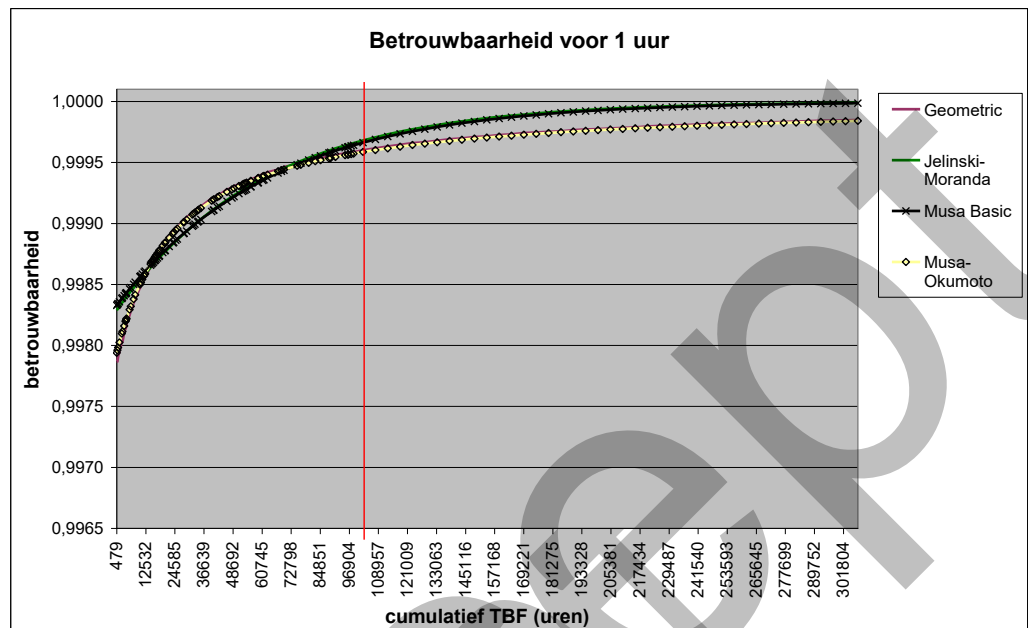
Reliability Growth Modelling (RGM) is gebaseerd op het uitgangspunt dat de betrouwbaarheid van software toeneemt met het herstellen van gevonden fouten. Betrouwbaarheids-groeimodellen beschrijven de curve van opgetreden (en opgeloste) fouten in een bepaald test- of gebruiksinterval. Een interval is in dit geval een periode in de tijd waarin het systeem functies uitvoert en eventuele incidenten worden verzameld om aan het eind van het interval te worden opgelost.

Door de plaats van het betreffende systeem (of TUB) op de curve te bepalen, kan een inschatting worden gemaakt van de faalkans of faalintensiteit (Q of λ), het totale aantal resterende fouten of de time-to-next-failure.

Welk model van toepassing is, hangt af van de situatie van het IT-systeem. Er zijn ongeveer 15 kenmerken aan de hand waarvan de meest waarschijnlijke modellen geselecteerd worden. We noemen er enkele:

1. Zijn de processen vooral sequentieel (zoals veelal in procesbesturing) of parallel van aard (zoals veelal in administratieve automatisering)?
2. Staan opgetreden fouten los van elkaar?
3. Worden fouten opgelost zonder nieuwe fouten te introduceren?

Vervolgens wordt met behulp van controles op de uitkomsten bepaald of en in hoeverre de resultaten van een model verworpen kunnen worden.



Figuur 8.4: Voorbeeld toepassing RGM

Toepasbaarheid

Om betrouwbaarheids-groeimodellen te kunnen toepassen, moet van test- of gebruiksintervallen worden bijgehouden op welke tijden een interval is gestart en geëindigd en op welke momenten fouten zijn opgetreden. Waar van toepassing moet daarbij bekend zijn of dit geautomatiseerde (test)runs waren, hoeveel parallelle operaties er zijn gestart en/of hoeveel gebruikers er gelijktijdig handelingen met het systeem verrichtten.

Ondanks het feit dat veel test- en gebruiksintervallen op IT-systemen van RWS geen statistisch significant aantal fouten per TUB opleveren, is het aan te bevelen de noodzakelijke gegevens voor invoer in de modellen vanaf de start van een project te verzamelen. Toepassing achteraf kan zinvol blijken als aanvulling op en ondersteuning van TOPAAS-analyses (zie ook **Fout! Verwijzingsbron niet gevonden.**).

8.7.5

Service Level Agreement (SLA)

Een Service Level Agreement (SLA) is een vorm van prestatiecontract dat RWS met een (IT-systeem)leverancier afsluit. Het stelt een opdrachtnemer voor een bepaalde periode verantwoordelijk voor het onderhoud van een (deel van het) IT-systeem. In de opdrachtoomschrijving ligt de nadruk op instandhouding en het blijvend voldoen aan de prestatie-eisen, ook wat betreft de softwarebetrouwbaarheid.

Omdat opdrachtnemers in de praktijk vaak nog niet risicogestuurd werken, wordt het vereiste onderhoud taakstellend uitgevraagd [16]. Daarbij wordt vastgelegd onder welke condities de opdrachtnemer bepaalde (herstel)werkzaamheden moet uitvoeren (met bijvoorbeeld maximale functionele en technische hersteltijden). De eisen aan de softwarebetrouwbaarheid zijn daarin op dezelfde wijze verwoord als in het oorspronkelijke D&C-contract²⁴.

²⁴ D&C: Design & Construct

Voor sommige objecten en projecten wordt met een opdrachtnemer een DBFM²⁵-contract gesloten. Een dergelijke overeenkomst omvat tevens het onderhoud. Een aanvullend Prestatiecontract met betrekking tot het onderhoud van het IT-systeem is dan niet nodig.

8.8 Activiteiten

Dit hoofdstuk beschrijft hoe software gebruikt en onderhouden moet worden om op elk moment te voldoen aan de gespecificeerde softwarebetrouwbaarheid. Hieronder worden de hoofdactiviteiten samengevat weergegeven.

Hoofdactiviteit 1: Bepaal de onderhoudsvorm per component en bepaal de consequenties voor de software (zie par. 8.2).

In de basis bepaald de onderhoudsvorm hoe onderhoud gepleegd wordt en hoe met wijzigingen dient te worden omgegaan.

Hoofdactiviteit 2: Richt een gedegen wijzigingsbeheersproces in (zie par. 8.3).

Wijzigingsbeheer bestaat uit de volgende deelactiviteiten:

- Deelactiviteit 2.1: Bepaal de wijziging.
- Deelactiviteit 2.2: Bepaal het wijzigingsgebied.
- Deelactiviteit 2.3: Voer een risicoanalyse uit.
- Deelactiviteit 2.4: Verwerk de maatregelen in een Request for Change (RFC).
- Deelactiviteit 2.5: Besluit over de RFC.

Hoofdactiviteit 3: Voer een gedegen proces van wijzigingen in (zie par. 8.4).

Als een wijziging geaccepteerd wordt, dan is het van belang dat deze wijziging op een beheerste wijze doorgevoerd wordt.

Hoofdactiviteit 4: Leg in het onderhoudsbeleid vast dat overige onderhoudsactiviteiten uitgevoerd worden (zie par. 8.4.4).

Naast het reguliere onderhoud conform de onderhoudsvormen die bepaald zijn in hoofdactiviteit 1, is er een aantal belangrijke onderhoudsactiviteiten die relevant zijn om de softwarebetrouwbaarheid te borgen. De volgende deelactiviteiten dienen opgenomen te worden in het onderhoudsbeleid:

- Deelactiviteit 4.1: Analyseer gebruiksgegevens en logging.
- Deelactiviteit 4.2: Verzamel data en ervaringen van andere objecten met dezelfde componenten, die aanleiding kunnen geven tot wijzigingen in het voorliggende systeem.
- Deelactiviteit 4.3: Registreer aanbevelingen van leveranciers en neem deze op in het wijzigingsbeheer.
- Deelactiviteit 4.4: Monitor de leverbaarheid van reservedelen en anticipeer hierop in het wijzigingsbeheer.

Hoofdactiviteit 5: Zorg voor goede logging en bewaking van het proces (zie par. 8.5).

Hoofdactiviteit 6: Voer op vooraf bepaalde criteria een Bayesiaanse update-analyse uit (zie par. 8.6).

²⁵ Design, Build, Finance and Maintain

8.9 Op te leveren producten/documenten

Een aantal documenten fungeert als input voor de gebruiks-, beheers- en onderhoudsfase. Deze documenten dienen veelal geüpdatet te worden tijdens deze fase. Documenten en producten die in ieder geval beschikbaar moeten zijn en bijgehouden moeten worden, zijn:

- een beschrijving van het software- en hardwaresysteem (softwarearchitectuur, bibliotheken, etc.)
- testbeschrijvingen, testomgeving en testresultaten uit voorliggende fasen
- Service Level Agreement (contract tussen leverancier en opdrachtgever)
- risicoanalyses
- een procesbeschrijving van het beheer en het onderhoud, waaronder het wijzigingsbeheer en wijzigingsproces
- een procesbeschrijving van logging en bewaking en Logfiles
- een reservedelenlijst
- een bijgewerkte Hazop, FMECA en faalkansschatting per TUB

9 Opstellen en gebruik van aanbestedingseisen

9.1 Inleiding

De functionaliteit van een object wordt mede bepaald door de software, terwijl deze software, vergeleken met het civiele werk, slechts een fractie van de kosten vertegenwoordigt. Eisen aan de software zijn meestal sterk afhankelijk van keuzes binnen systems engineering. Daardoor is het lang niet altijd mogelijk de software apart aan te besteden. Bij integrale aanbesteding is een goede voorbereiding noodzakelijk om te borgen dat betrouwbaarheid en faalkans de juiste aandacht krijgen.

Rijkswaterstaat werkt met systeemgerichte contractbeheersing (SCB) en selecteert opdrachtnemers die aantoonbaar werken onder kwaliteitsborging. Onder SCB worden systeemtoetsen, procestoetsen en producttoetsen risicogestuurd uitgevoerd. Het project wordt aan RWS-zijde gestuurd door een multidisciplinair (IPM)-team bestaande uit een contractmanager, projectmanager, technisch manager en technisch specialisten. Er worden verschillende contractvormen toegepast (denk aan E&C, D&C, DBFM). De eisen die per contractvorm (kunnen) worden gesteld zullen verschillen. Wanneer een eis uit dit hoofdstuk bij een bepaalde contractvorm niet kan worden gesteld, dient daar een alternatief voor te worden opgesteld (bijv. in de vorm van een hoofdeis, die nader wordt ingevuld door de opdrachtnemer en moet worden goedgekeurd door de opdrachtgever).

De contractbeheersing van een opdracht waarin missiekritische/veiligheidskritische software dient te worden gerealiseerd of aangepast, verschilt in de basis niet van een andere opdracht. Een IPM-team met voldoende capaciteit en expertise kan deze begeleiding op zich nemen op basis van goed geformuleerde eisen.

Eisen komen in verschillende verschijningsvormen, en hebben betrekking op verschillende onderwerpen. We noemen de Vraagspecificatie Proces (VSP) dat de projectmanagementprocessen en technisch managementprocessen volgens ISO 12207 beschrijft en de Vraagspecificatie Eisen (VSE), die systeemeisen en technische processen bevat (ISO 12207). De technische processen bestaan uit drie categorieën: creërend, controlerend en beschouwend. Bij de invulling van deze processen moet altijd rekening worden gehouden met professionele beste praktijken. Afhankelijk van het risicoprofiel van een softwaremodule (SIL level) geeft de IEC 61508 standaard richtlijnen (recommendations) voor de technische en procesmaatregelen die genomen moeten worden opdat bij de creërende processen zo min mogelijk fouten worden geïntroduceerd, bij de controlerende processen zo veel mogelijk fouten worden verwijderd en bij de beschouwende processen een transparant, begrijpelijk en zo volledig mogelijk beeld van de risico's wordt gegeven.

9.2 Probleemstelling

Tijdens de uitvoering zijn (contract)eisen een belangrijk instrument als basis voor de toetsing en, indien nodig, voor bijsturing van het werk van de opdrachtnemer. Binnen softwaregerelateerde projecten is hier echter sprake van een dilemma. De totstandkoming van software is (in vergelijking met civiel werk of de realisatie van elektro- of werktuigbouwkundige systemen) eigenlijk te vergelijken met een zeer uitgebreid ontwerptraject. Het eindproduct (de executable(s)) wordt in enkele minuten (of uren) door een computer (de zogenaamde build-omgeving) volledig opnieuw geproduceerd. Als je vooraf veel eisen opstelt, beperk je de ontwerpruimte

en leidt dat tot een minder geschikt product. Als je werkt met globale eisen, geeft dat veel discussie over de interpretatie daarvan tijdens de bijsturing of toetsing. Het RWS-team moet actief betrokken zijn met inachtneming van de contractuele afspraken.

Bij missiekritische software is het totstandkomingsproces afhankelijk van het SIL level van de onderdelen (IEC 61508), dus van engineeringkeuzes die binnen het contract gemaakt worden. Bij te veel eisen vooraf kan een ongewenste verstrengeling van contract-, proces-, functionaliteits- en technische eisen ontstaan. Te veel vrijheid vooraf kan leiden tot ongewenste meerwerkdiscussies om de gewenste prestatie van het object te realiseren.

De uitdaging is dus om te komen tot een gebalanceerde eisenset, die voldoet aan de eisen van alle rollen binnen het IPM-team. Het stellen van meer eisen is niet per se beter: er ligt ook een taak bij het IPM-team om ervoor te zorgen dat het over de juiste expertise en voldoende capaciteit beschikt om producten te toetsen en onzekerheden, zorgen en aannames te beoordelen op de impact ervan. Dit alles om RWS in staat te stellen indien nodig tijdig binnen het contract bij te sturen.

9.3 Eisen in het voortraject

In de voorbereiding op de uitbesteding, de periode waarin VSE/VSP worden opgesteld, zijn wijzigingen in eisen goedkoop en snel uit te voeren. In deze periode moet RWS serieus tijd investeren in het inzichtelijk maken van onzekerheden in het onderhavige traject.

De ideale situatie met het oog op de softwarebetrouwbaarheid is wanneer de VSE/VSP in nauw overleg met de opdrachtnemer worden opgesteld. Deze kan als beste de impact van de eisen aangeven, zodat VSE/VSP kunnen worden bijgesteld voor de contractvorming. Deze situatie dient in de reviewfase zo goed mogelijk te worden gesimuleerd. Benoem binnen het IPM-team een (sub)team met een opdrachtnemersrol, dat op basis van de eisen een ontwerp, raming en planning maakt. Op die manier kunnen, door het doen van aannames, omissies in de VSE/VSP expliciet gemaakt worden en gerepareerd worden. Bij de review en vaststelling van de VSE/VSP is het verstandig om een SRR conform [SMC Standard SMC-S-21, 15 September 2009] uit te voeren. Merk hierbij op dat deze standaard is geschreven op basis van een volledige uitbesteding van al het systems-engineeringwerk en dat bij SRR ook vooruitgekeken wordt naar het vervolgtraject (indeling in componenten en planning). Het genoemde sub-team is dus nodig om deze standaard goed te kunnen toepassen.

Het voortraject is pas geslaagd als er een echte SRR met de opdrachtnemer is uitgevoerd. Naast een gedegen VSE/VSP speelt de selectie van de opdrachtnemer een cruciale rol in het succesvol realiseren van het werk. Een IPM-team kan ervoor kiezen om eisen minder gedetailleerd uit te vragen en deze middels een gunningsproces door de leverancier te laten uitwerken tot een uitvoerbaar plan. Het gunningsproces moet dan wel dusdanig ingericht zijn dat het beste plan wordt gekozen. In dit document gaan we niet verder in op de te hanteren gunningscriteria.

9.4 Algemene aandachtspunten

Na gunning zal het RWS-team zijn rol als begeleider moeten oppakken. Het eerdergenoemde sub-team dat de raming en planning heeft gemaakt, kan hier een rol in krijgen, maar dient zich ervan bewust te zijn dat de opdrachtnemer nu aan zet is. Het sub-team zal de mogelijkheid moeten krijgen (in PSU en PDR) om zijn planning en ontwerp toe te lichten. Het RWS-team moet zelf de inschatting maken

welke meetings tijdens het opstarten van het project nodig zijn voor optimale transparantie.

Meer in het algemeen geldt dat er een risico is dat RWS onvoldoende zicht heeft op het ontwikkelproces. Daarom is het nodig om expliciet eisen te stellen aan de transparantie. Op die manier kan RWS, ten behoeve van het risicomanagement, risicogestuurd technische zorgpunten op impact en kans beoordelen. Ook kunnen er op basis van de specifieke producten en reviews ter plekke of achteraf nadere analyses uitgevoerd worden.

In de paragrafen hierna is een aantal eisen opgenomen die de basis vormen voor specifieke sub-eisen die eventueel kunnen worden opgenomen in de VSE/VSP. Bij de uitbesteding van louter software vallen functionele specificaties of SRS onder de VSE.

Voor een heldere, goed gedefinieerde communicatie zijn de engineering-basisprocessen als configuratiemanagement (baselining van systeemeisen en wijzigingen daarop) nodig. De correcte inrichting van deze processen volgt uit de hieronder geformuleerde eisen.

Tot slot: zelfs de strakste eisen zijn geen garantie voor een goed product. Waar gewerkt wordt, worden ook fouten gemaakt. Niet alle tussenproducten zijn even perfect, soms wordt er met reden afgeweken van standaarden en richtlijnen. En bovendien zijn niet alle bevindingen even schadelijk voor het geleverde werk. Van belang is dat bevindingen altijd met de specialist van de andere partij worden doorgenomen alvorens over te gaan tot een bespreking of signalering ervan op project-/contractniveau.

9.5 Specifieke eisen aan processen

EIS P.1: De opdrachtnemer dient het softwareontwikkelproces in te richten met inachtneming van een SIL-classificatie van de softwaremodules. Gebaseerd op de IEC 61508.

EIS P.2: De opdrachtnemer dient bij het opstarten van het project zijn technische processen zodanig in te richten dat gedurende de ontwerpfase en achteraf de overwegingen, onzekerheden en aannames van ontwerpbesluiten getoetst kunnen worden door een derde partij.

SUBEIS P.2.1: De opdrachtnemer dient resultaten van controlerende processtappen (reviews/inspecties/processen) vast te leggen, evenals de corrigerende maatregelen.

Opmerking: Basishygiëne als configuratiemanagement volgt uit de bovenstaande EISen.

EIS P.3: De opdrachtnemer dient in de planning een PDR- en CDR-mijlpaal op te nemen.

TOELICHTING P.3: De acceptatiecriteria van PDR- en CDR-reviews zijn beschreven in [SMC Standard SMC-S-21,15 September 2009]. Deze standaard bevat een systeembenadering. Dat betekent dat in PDR en CDR de software nadrukkelijk ook in het kader van de taakuitvoering van het object beschouwd wordt.

SUBEIS P.3.1: De opdrachtnemer dient als basis voor de PDR en CDR een consistente baseline bestaande uit SSS, SSDD, SDP en FMEA/FTA te leveren.

TOELICHTING: Het RWS-team dient deze eis aan te vullen met een door dit team gewenste procesinvulling. Denk hierbij aan walkthroughs of vraag-/antwoordsessies.

SUBEIS P.3.2: De diepgang van deze onderdelen wordt bij de PDR-mijlpaal geacht risicogestuurd te zijn, waarbij deze diepgang op verzoek van de opdrachtgever toegelicht wordt.

EIS P.4: De opdrachtnemer dient software die kan leiden tot falen of niet-beschikbaar zijn van infrastructuur van RWS, per module te kwantificeren. De kwantificering dient te worden uitgevoerd conform de laatste vigerende versie van TOPAAS.

SUBEIS P.4.1: De opdrachtnemer dient RWS op aanvraag toegang te verlenen tot alle benodigde documentatie en functionarissen ten behoeve van het uitvoeren van een TOPAAS-analyse, die eventueel door RWS wordt gedelegeerd aan een externe partij.

De reden dat we deze eisen stellen is dat software-onbetrouwbaarheid een samenspel is van allerlei elkaar versterkende factoren. In de basis wordt softwarefalen veroorzaakt door fouten die door mensen in software worden geïntroduceerd. Vooral de fouten die niet herkend worden door inspecteurs/reviewers en testers (omdat ze bijvoorbeeld met dezelfde foute aannames werken), blijven zonder een zeer transparante werkwijze ondetecteerbaar.

9.6 Specifieke eisen aan de organisatie

EIS O.1: De opdrachtnemer dient aantoonbaar gekwalificeerd personeel in te zetten.

EIS O.2: De opdrachtnemer dient maatregelen te nemen die borgen dat technische klokkenluiders gehoord en beschermd worden.

TOELICHTING: Technische klokkenluiders zijn engineers (inhoudelijke medewerkers) die zorgen hebben over de software of het software-ontwikkeltraject.

EIS O.3: De opdrachtnemer dient in de projectstructuur functiescheiding toe te passen tussen (System) Engineering, Software Development, Verification/Validation en RAMS rollen.

SUBEIS O.3.1: De opdrachtnemer dient in zijn processen te borgen dat discussies (onenigheid, aannames, zorgpunten) tussen deze gescheiden functies nagelopen kunnen worden door het RWS-team.

TOELICHTING: Het doel van deze functiescheiding is dat er structureel diversiteit wordt ingebouwd in het project, waardoor binnen het projectteam scherp gecontroleerd kan worden op onbenoemde aannames. Daarnaast kan het projectteam risicogestuurd geholpen worden met een onafhankelijke view vanuit het RWS-team of een derde partij.

OPMERKING: Als deze discussie concurrentie- of privacygevoelig is, mag de opdrachtnemer hier nadere afspraken over maken. Bijvoorbeeld een onderzoek door een derde partij onder een non-disclosure agreement.

9.7 Te toetsen producten

EIS T.1: De opdrachtnemer dient de oplossing te beschrijven in een aantal 'views'. Deze views betreffen een context, een functionele (dynamische) beschrijving (hoe werkt het) en een systeem-decompositie (statische) beschrijving (waar bestaat het uit). Het begrip view is beschreven in de ISO 42010.

SUBEIS T.1.1: De context-view dient de FTA Top events en tevens alle externe actoren te beschrijven als input voor FTA en FMEA.

SUBEIS T.1.2: De functionele view dient alle taakuitvoeringen te benoemen die nodig zijn voor de (kritische) functionaliteit. Inclusief hun afhankelijkheid van (bijdrage aan) de top-events.

SUBEIS T.1.3: De systeemdecompositie-view dient alle te realiseren of aan te sluiten onderdelen te benoemen.

EIS T.2: Alle onderdelen uit de systeemdecompositie-view dienen uitgewerkt te worden in formele eisen, interfacespecificaties en ontwerp, en via vastgelegde testcases op de interfacespecificatie getest te worden.

EIS T.3: Het SDP dient te beschrijven hoe het totale softwarepakket wordt gerealiseerd, getest en in gebruik genomen.

TOELICHTING: Het SDP wordt doorgaans aan het begin van het project opgesteld, mogelijk als specialistische bijlage bij het projectmanagementplan (PMP). Op dat moment zijn het ontwerp en ook de specifieke eisen aan de softwaredeelsystemen niet bekend. Het SDP dient bij CDR in lijn te zijn met SSDD. Het is verstandig om het SDP bij de PDR bij te stellen.

SUBEIS T.3.1: Het SDP dient te benoemen hoe de onderdelen uit de decompositie worden gerealiseerd, welke processen hiervoor vanuit de IEC 61508 van toepassing zijn.

SUBEIS T.3.2: Het SDP dient de inzet van inspecties, reviews en testen te benoemen, zowel op basis van de benoemde eisen als op basis van aanvullende risico's.

SUBEIS T.3.3: Het SDP dient (in aanvulling op het gestelde in J-STD-016) de integratiestrategie te benoemen, welke modules tezamen worden getest en in welke contexten. Het doel bij deze aanpak is dat er een transparante beoordeling wordt gemaakt van de mate van betrouwbaarheid waarmee de modules samen met de hardware hun taakuitvoering in het totale object gaan uitvoeren.

SUBEIS T.3.4: Het SDP beschrijft keuzes en hun achtergrond bij de ontwikkeltooling, met name standaardbibliotheken en compiler(s).

EIS T.4: De opdrachtnemer dient foutenbomen zodanig te coderen dat deze aansluiten op eerder gebruikte coderingen van de faalwijzen uit de FMEA en de analyse externe gebeurtenissen.

SUBEIS T.4.1: De opdrachtnemer dient software die kan leiden tot falen of niet-beschikbaar zijn van infrastructuur van RWS, per module in de vorm van aparte basisgebeurtenissen in de foutenboom op te nemen.