LECTORALE REDE

SOFTWARE VOOR SLIMME SYSTEMEN

Een systeemvisie op de softwareontwikkeling van (mobiele) robots en sensornetwerken

DR.IR. TEADE PUNTER | 12.10.2018

Fontys
LECTORAAT
High Tech Embedded Software

SOFTWARE VOOR SLIMME SYSTEMEN

Lectorale rede, in verkorte versie uitgesproken bij de aanvaarding van de functie van lector High Tech Embedded Software aan Fontys Hogeschool ICT op 12 oktober 2018.

Dr.Ir. Teade Punter

Fontys
LECTORAAT
High Tech Embedded Software

INHOUDSOPGAVE

1. EEN ONDERWIJSDAG BIJ FHICT IN 2023

Als inleiding op het lectoraat High Tech Embedded Software (HTES) neem ik u mee naar een willekeurige onderwijsdag bij Fontys Hogeschool ICT (FHICT) in 2023. Het lectoraat HTES is dan net met haar tweede ronde begonnen; de resultaten van de eerste jaren praktijkgericht onderzoek zijn zichtbaar. We kijken even in mijn agenda als docent die dag.

- **8:30 -** Aankomst bij Fontys op Strijp TQ. Ik word welkom geheten door Porti, de service robot. Porti herkent mij als medewerker en wenst mij een prettige dag. Gasten zal Porti welkom heten en naar de gewenste afspraak brengen.

- **9.00 uur -** Stand-up met 20 studenten Technology die in het *software lab* 'Autonome, mobiele robots' meedoen. Vroeger noemde ik dit een klas. Tegenwoordig zijn het drie project-groepen die elkaar informeren over de voortgang. We nemen meteen inhoudelijk de *architectuur* van de nieuwe robot software met elkaar door.

- **10.00 uur -** Instructie geven over *integratie en test*. Ik leg een nieuwe methode voor integratietesten uit. De studenten gaan dit toepassen voor het testen van een sensornetwerk voor biomonitoring.

- **11.00 uur -** *Modelleren* met collega docent Suzana. We bespreken een nieuwe versie van het model waarmee het opvangen van een bal door de voetbalrobot wordt beschreven. Het overleg gaat erover of het vangen van de bal door de robot *geleerd* kan worden.

- **13.00 uur -** Ik heb een kort gesprek met student Jan over het bepalen van de *software kwaliteit* van onze nieuwe sensornode om een paprikaziekte te meten. Wij kijken naar de veiligheids-aspecten van de node.

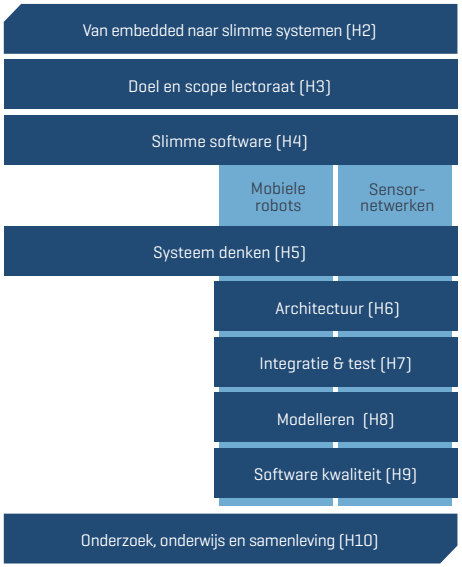
- **14.00 uur -** Overleg met Ton van het bedrijf DDL. Het bedrijf is twee jaar geleden partner van het lectoraat geworden. Ton, en collega's, zijn regelmatig een dag aanwezig in de onderwijsruimte. Je werkt samen aan een nieuw apparaat dat DDL maakt. Ton heeft de systeem kennis, maar de studenten passen een nieuwe programmeertechniek toe om zo te onderzoeken of dit performance voordeel oplevert.

- **15.00 uur -** Bijwonen van afstudeer-presentatie door student Lilian. Zij heeft haar afstudeeropdracht uitgevoerd bij bedrijf Conexis, als onderdeel van de onderzoekslijn ‘Sensornetwerken’. Het lectoraat stuurt zo het onderzoek, door afstudeerders op een thema te begeleiden.

- **16.00 uur -** Afspraak met de drie groepen studenten uit het *software lab* waarmee ik als docent vanochtend begonnen ben. De studenten hebben een Telco met een onderzoeksgroep in Spanje. Ik help om de antwoorden van de Spanjaarden op de gestelde vragen te interpreteren naar verbeteringen in de architectuur.

Deze agenda toont onderdelen waar het lectoraat HTES de komende jaren mee aan de slag gaat, namelijk: architectuur, integratie & test, modelleren en kwaliteit en dan toegepast op mobiele robots en sensornetwerken. Het laat ook zien dat we dit willen bereiken via het opzetten van software labs en het opstellen en uitwerken van onderzoeksthema’s waarbij bedrijven en maatschappelijke organisaties betrokken zijn.

Dit essay werkt eerst het begrip embedded software uit aan de hand van het begrip Cyber Physical Systems waarvoor het nodig is om software te maken voor slimme systemen die gaan samenwerken (hoofdstuk 2). Deze slimme software zien we in (mobiele) robots en sensornetwerken ontwikkeld worden (hoofdstukken 3 en 4). Omdat het in al deze toepassingen om systemen gaat en omdat de apparaten zelf en het ontwikkelen ervan complexer worden is er systeemdenken nodig. Het kader voor systeemdenken wordt eerst geïntroduceerd (hoofdstuk 5). Daarna worden de aspecten ervan uitgewerkt en steeds toegepast op mobiele robots en sensornetwerken. De aspecten zijn: architectuur (hoofdstuk 6), integratie en test (hoofdstuk 7), modelleren (hoofdstuk 8) en software kwaliteit (hoofdstuk 9). Tenslotte wordt het begrip software lab in de context van onderzoek, onderwijs en samenleving geïntroduceerd (hoofdstuk 10).



FIGUUR 1 - STRUCTUUR VAN DIT ESSAY

2.
VAN EMBEDDED NAAR SLIMME SYSTEMEN

> De waarde van embedded software is beperkt... het staat niet apart op de factuur... Als de hoeveelheid software groeit, dan zal het product zich openen en zal software als zelfstandig product verkocht worden.
>
> – MICHIEL VAN GENUCHTEN (2006)

Dit inleidende hoofdstuk legt uit wat embedded software is, en dat het onderdeel is van een embedded systeem. Het vertelt ook dat embedded systemen steeds meer Cyber Physical Systems worden, die intelligentie bevatten, waardoor we ze ook aanduiden als slimme systemen, die we weer kunnen opvatten als high tech embedded systemen waarvoor high tech embedded software wordt gemaakt.

"Embedded systems zijn geïntegreerde hardware/software systemen die worden ingebouwd in systemen en apparaten[1] die niet noodzakelijkerwijs zijn te herkennen als geautomatiseerde apparaten of computers. Ze omvatten platforms van hardware en software waarmee beslissingsvermogen en intelligentie kunnen worden toegevoegd aan bijna elk product, systeem, infrastructuur of service.

Als gevolg hiervan definiëren en beheersen embedded systemen de functionaliteit en kwaliteit van deze producten en applicaties. Ze vormen een integraal en onontbeerlijk onderdeel van complexe hightech systemen." (Geelen e.a, 2018).

Al sinds de jaren 50 van de vorige eeuw worden embedded systemen gebruikt voor het aansturen van apparaten en processen. Voorbeelden zijn regelaars voor verwarmingssystemen en lopende banden. De computer is daarbij onderdeel van het proces of apparaat: het is ingebed in het proces of apparaat. Vandaag de dag treffen we embedded systemen overal om ons heen aan: in consumentenelektronica, medische apparatuur, communicatie-infrastructuur, voertuigen, beveiliging, robots, industriële productie, enzovoorts.

Embedded systemen zijn niet meer weg te denken uit onze moderne samenleving.

1. Apparaten worden ook aangeduid als devices, bijvoorbeeld mobiele- en hardware devices.

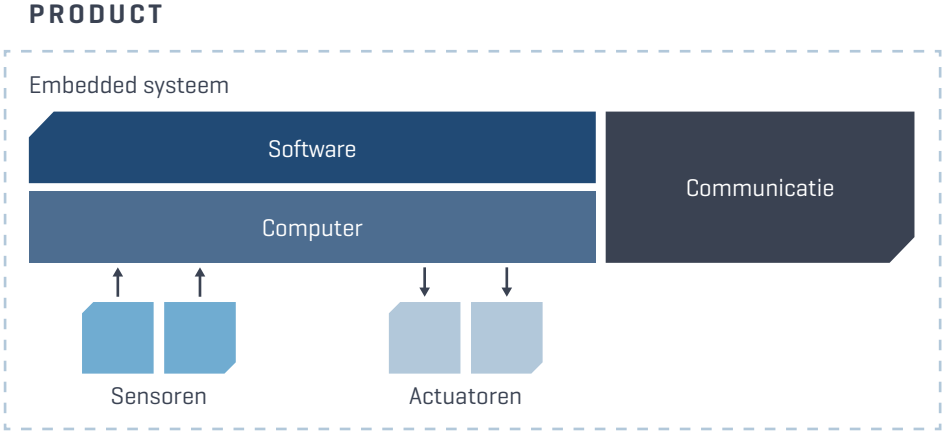
Zij spelen een belangrijke rol in de steeds slimmere producten en diensten die nodig zijn bij het ontwikkelen en voortzetten van veel maatschappelijke ontwikkelingen - zoals in gezondheid en welzijn, transport en logistiek, veiligheid, energie en milieu. Embedded systemen zijn een zogenaamde 'enabling technology': een faciliterende of ondersteunende functie, die noodzakelijk is, vaak zonder dat de gebruikers zich daarvan in het dagelijks gebruik bewust zijn. De samenleving, zowel individuen als organisaties, is in al haar facetten, in toenemende mate, afhankelijk geworden van de betrouwbaarheid en veilige werking van deze voortdurend in complexiteit en verbondenheid groeiende systemen.

In een embedded systeem werken hardware en software samen. Om embedded systemen aan te sturen zijn vier ingrediënten nodig: sensoren, actuatoren, computers én software. De sensoren zijn de ogen en oren van een embedded systeem. Zij geven gegevens over de toestand van het systeem. De besturingsopdrachten van het embedded systeem worden uitgevoerd met actuatoren. Zij regelen bijvoorbeeld de temperatuur, of het zijn de motoren die een robot laten bewegen. De embedded computer is bijvoorbeeld een microcontroller of een processor die specifieke taken uitvoert (Lee & Seshia, 2015). De computer ontvangt de data van de sensoren en stuurt de actuatoren aan. De embedded software voert deze aansturing uit. Het is een operationele beschrijving van het systeem en bestuurt dit systeem volgens bepaalde (systeem)regels. Embedded software kan in verschillende talen worden geprogrammeerd. Vaak is dat de taal C, een imperatieve taal waarin het nagaan en wijzigen van variabelen centraal staat. De combinatie van variabelen bepaalt een programmatoestand. Ook wordt regelmatig de object georiënteerde taal C++ gebruikt. Hierin wordt de globale programmatoestand vervangen door een verzameling objecten. Tegenwoordig zien we Python vaker gebruikt worden. Dit is een scripting taal, die door een zogenaamde interpreter wordt uitgevoerd.[2]

Een embedded systeem kan onderdeel zijn van een groter systeem waarin de losse delen met elkaar samenwerken. Door alle losse systemen met elkaar te verbinden ontstaat een groter systeem. Er is dan sprake van een netwerk van embedded systemen. Een sprekend voorbeeld zijn sensornetwerken, waar elke sensorapparaat – een embedded systeem op zichzelf – een knooppunt of 'node' is.

Door deze knooppunten met elkaar te verbinden wordt het sensornetwerk opgesteld, waardoor de nodes gegevens aan elkaar door kunnen geven en het mogelijk wordt om te meten over grotere oppervlaktes of aan objecten die in een gebouw of in de openlucht staan. Een voorbeeld is het uitvoeren van temperatuur- en vochtigheidsmetingen in aardappelvelden om meer te weten te komen over een aardappelziekte (Langendoen e.a., 2006). Omdat de netwerken die de verschillende embedded systemen koppelen in essentie de communicatie tussen de systemen regelen duiden we netwerk verder aan als communicatie in onderstaande figuren. Communicatie is de extra ingrediënt van embedded systemen. Daarom spreken we van 4+1 ingrediënten.

De communicatie in embedded systemen maakt het mogelijk om software toepassingen op meerdere computers uit te voeren. Een voorbeeld is de moderne auto, waarin de software verdeeld wordt over verschillende *electronic control units* (ECU). De software componenten, en samenstellingen ervan in de vorm van subsystemen, communiceren via een databus. Om de verdeling van de software goed te regelen wordt een architectuur opgesteld. Autosar is een zogenaamde referentie architectuur die de automobielindustrie gebruikt om software over hardware te verdelen. Het definieert de verdeling van de software componenten, zodat deze goed met elkaar kunnen communiceren via standaard interfaces (Webers e.a., 2008).



FIGUUR 2 - DE 4 + 1 INGREDIËNTEN VAN EEN EMBEDDED SYSTEEM.

2. Een overzicht van programmeertalen en hun populariteit, ook voor het embedded domein, is te vinden in de Tiobe index https://www.tiobe.com/tiobe-index/.

Een ander voorbeeld van het belang van communicatie zijn machines in productiestraten die aan elkaar gekoppeld worden. De embedded computers communiceren met elkaar en zijn als geheel op elkaar afgestemd. Het verbinden van apparaten is onderdeel van wat wordt aangeduid als Smart Industry of Industrie 4.0.[3]

We hebben nu gezien dat in het originele begrip van embedded systemen de nadruk ligt op de relatie tussen hard- en software, waarbij met hardware elektronica wordt bedoeld. De aandacht voor embedded systemen gaat tegenwoordig echter steeds meer in de richting van de relatie tussen computer(s) en de fysieke wereld én de communicatie tussen de systemen. De interactie tussen (real-time) computing en fysieke systemen is van belang en wordt mogelijk gemaakt door de integratie van informatie, software, elektronische en mechanische onderdelen. Een voorbeeld is de aansturing van motoren vanuit de software waarbij rekening wordt gehouden met de fysieke aspecten van de motor, zoals de slip tijdens het versnellen en afremmen van de motor. Ook de communicatie tussen de systemen via het internet, de zgn. cyberspace, wordt belangrijker. In plaats van embedded systemen spreken we daarom steeds vaker van Cyber Physical Systems (CPS). In het vervolg van dit essay zijn embedded systemen synoniem met Cyber Physical Systems.

Door vanuit de software en elektronica rekening te houden met de aansturing van de fysieke aspecten wordt het mogelijk om nieuwe systemen te maken die de functionaliteit, autonomie, bruikbaarheid, veiligheid en betrouwbaarheid van de losse systemen overstijgen. Voorbeelden van Cyber Physical Systems (CPS) zijn robots, medische apparatuur, auto's en energie centrales. In het begrip CPS speelt de verbinding tussen systemen, via de zogenaamde cyberspace, oftewel het internet, ook een belangrijke rol[4,5] (Lee & Seshia, 2015). Immers, vaak worden via het internet de systeemdelen met elkaar verbonden. Sommigen definiëren daarbij het Internet of Things (IoT), waarin alle dingen met elkaar verbonden worden, als de ultieme vorm van dergelijke verbindingen.[6]

3. Smart industry is een trend in de Nederlandse maakindustrie waarin apparaten meer op elkaar worden afgestemd om zo efficiency en effectiviteit van productie en distributie te verbeteren. Deze slimme bedrijven ontstaan onder andere door hun apparaten met elkaar te laten communiceren waardoor een informatieketen tot stand komt. Zo kan men flexibeler aanpassen en werken. In Duitsland en ook internationaal wordt deze trend aangeduid als Industrie 4.0. Naast de trend zijn er initiatieven vanuit overheid en bedrijfsleven om Smart industry te ontwikkelen en te promoten, zie bijvoorbeeld https://www.smartindustry.nl/
4. Europa: https://ec.europa.eu/programmes/horizon2020/en/h2020-section/smart-cyber-physical-systems
5. Verenigde Staten: https://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf15541
6. Soms wordt er zelfs gesproken over het Internet of Everything (IERC, z.j.)

Internet of Things is een concept dat de Smart Industry mogelijk kan maken door de manier waarop machines met machines gaan communiceren.

Regelmatig wordt het begrip System-of-Systems (SoS) gebruikt. Hierbij gaat het om de integratie van losse systemen die onafhankelijk van elkaar kunnen functioneren, maar die gezamenlijk als netwerk verbonden zijn (Jamshidi, 2009), (Incose, z.j.). De systemen zijn bij wijze van spreken 'lid van een groter systeem' en hebben daarmee meerwaarde ten opzichte van wat zij als losse systemen afzonderlijk kunnen leveren.

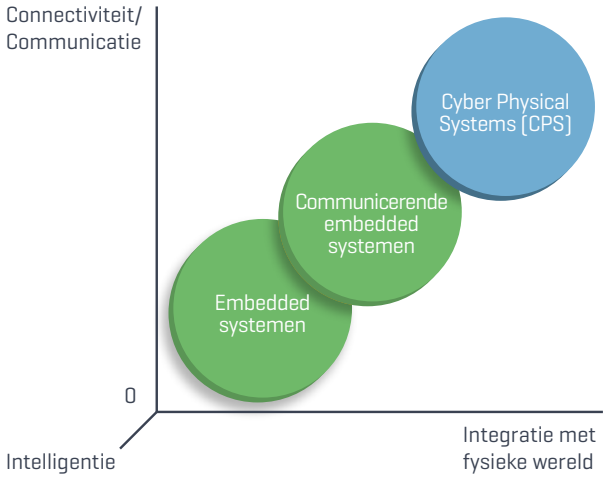
Het lidmaatschap van een individueel systeem kan over de tijd veranderen: een systeem kan ook weer worden verwijderd uit het netwerk. Een System-of-Systems evolueert over de tijd en is daarmee continu in verandering (Engel, 2014), (Horváth, 2012). De hiervoor genoemde voorbeelden van embedded systemen, in auto's en productiestraten, en ook de thema's van het lectoraat, mobiele robots en sensornetwerken, kunnen elk als System-of-System én als Cyber Physical System worden opgevat.

We onderkennen nu twee dimensies in de ontwikkeling van embedded systemen. De eerste gaat over de communicatie en de mate waarmee systemen en systeemonderdelen verbonden zijn.

Dit is de ontwikkeling van losse systemen naar System-of-Systems die mogelijk wordt door het internet aspect van Cyber Physical Systems. De tweede as gaat over de mate waarin het systeem met de fysieke wereld is geïntegreerd. Dit verwijst naar de hiervoor beschreven ontwikkeling van embedded systemen als hardware/software tot Cyber Physical Systems. Aan deze twee dimensies voeg ik nog een derde dimensie toe, namelijk de intelligentie van systemen.

Deze derde dimensie gaat over de toename van intelligentie in embedded systemen. Het betreft de toenemende 'slimheid' van embedded systemen door het toepassen van kunstmatige intelligentie. Zelfrijdende auto's zijn een voorbeeld van de ontwikkeling in deze dimensie. In een zelfrijdende auto zijn alle rij-taken geautomatiseerd en het voertuig kan zelfstandig zijn weg vinden.[7] Er zijn sensoren en camera's aan de auto toegevoegd die met speciale technieken de omgeving in de gaten houden. De data die zo binnenkomen worden geanalyseerd en de software neemt zelf beslissingen om te reageren op dreigend gevaar. De auto kan ook bij het naderen van langzamer rijdend verkeer een signaal naar achteropkomend verkeer geven om langzamer te gaan rijden. Deze slimheid verbetert de verkeersveiligheid. Zelfrijdende auto's opereren autonoom, beslissen zelf op basis van hun slimheid. Overigens is deze slimheid ook terug te voeren op hun mate van verbondenheid (dimensie connectiviteit), omdat de auto's data over verkeerssituaties aan elkaar overbrengen en elkaar zo slimmer maken.

In figuur 3 worden de drie dimensies weergegeven door ze elk aan de hand van een as uit te drukken. Er is een globale waarde voor elk van de dimensies toe te kennen aan de hand van de positie op elke as, die oploopt vanuit de oorsprong (O). Vanuit de oorsprong neemt elk van de dimensies toe. De figuur geeft ook de aandachtsgebieden van het lectoraat High Tech Embedded Software (HTES) aan. De twee groene bollen zijn gebieden waar de opleiding ICT & Technology al ervaring mee heeft en al onderwijs voor verzorgt. De blauwe bol is het aandachtsgebied van het lectoraat HTES waarop het lectoraat zich verder richt. Het zijn Cyber Physical Systems, die ook vaak intelligent zullen zijn, en die we dan opvatten als slimme systemen. Deze slimme systemen vatten wij op als high tech embedded systemen, zoals waferscanners, printers en medische apparaten, omdat de ontwikkeling ervan vergelijkbare uitdagingen kent. Voorbeelden van slimme systemen zijn robots en sensor-netwerken. Een concreet voorbeeld is de Marsrover Curiosity, zie kader A.



FIGUUR 3 - DRIE DIMENSIES OM (HIGH TECH) EMBEDDED SYSTEMEN UIT TE DRUKKEN.

7. De SAE international standard onderkent vijf niveaus om de automatisering van de auto uit te drukken (Hyatt en Paukert, 2018)

KADER A
MARSROVER CURIOSITY ALS VOORBEELD VAN EEN INTELLIGENT CYBER PHYSICAL SYSTEM

Een Marsrover is een rijdend laboratorium, dat ontwikkeld is om op de planeet Mars rond te rijden en observaties te doen. Het wagentje kan monsters van de bodem nemen, analyseren en de resultaten naar de aarde zenden (Onderwater, 2018). Ook maakt de Rover (panorama) foto's[8] die het doorstuurt naar de aarde.

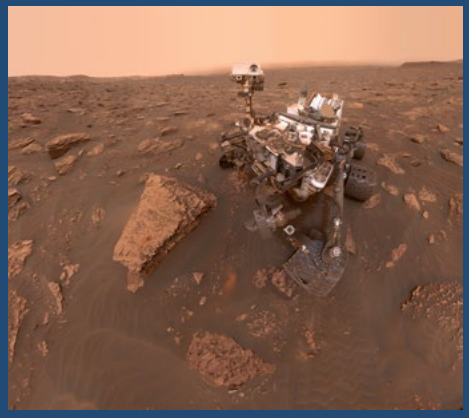
Voor de diverse Marsreizen is er per missie een nieuwe Rover gemaakt. Hierin is elke keer de hard- en software verbeterd op basis van inzichten uit de voorgaande reizen. Curiosity is de meest recente Marsrover. Het is in augustus 2012 op Mars geland. Deze Rover is tot op heden bezig met het ontdekken van de planeet. Een Marsrover is een Cyber Physical System.

Marsrovers dienen betrouwbaar te zijn om bijvoorbeeld met grote temperatuurwisselingen om te kunnen gaan en bestand te zijn tegen kosmische straling. Het zijn autonome robots omdat ze door de afstand tussen aarde en Mars niet vanaf de aarde kunnen worden aangestuurd.

Als de aarde en Mars het dichtst bij elkaar staan is de afstand 55 miljoen kilometer. De tijd tussen zenden en ontvangen van een signaal duurt dan 3 minuten. Dit kan oplopen tot 21 minuten.

De vereiste betrouwbaarheid en de benodigde autonomie van de Marsrover leveren uitdagingen op voor het ontwikkelen van de software van deze robot. Een Marsrover is een 'one-of-kind system', die maar één keer op missie gaat. Alle (software) onderdelen worden specifiek voor de missie gebouwd en moeten van tevoren, op aarde, goed ontwikkeld en getest worden. Daarom is voor Curiosity een modulaire software architectuur opgezet, met duidelijk gedefinieerde interfaces, die het mogelijk maakt om onderdelen uit te wisselen. Ook is de software zo opgezet dat het gehele systeem beschermd wordt tegen fouten die optreden in (software) onderdelen. Er is een gedegen ontwikkeltraject gevolgd, met dagelijkse integratie en grondige unit- en integratietesten (Holzman, 2014). Verder wordt een coding standard toegepast die gebaseerd is op risico's die zijn bepaald aan de hand van software afwijkingen[9] die men in eerdere marsmissies vond. Voor de regels van de coding standard is tijdens ontwikkeling continu met tools nagegaan of ze correct werden toegepast. Een andere maatregel was het opnieuw inrichten van het review proces om de code te analyseren. De code is zo 'schoon geschrobt' (Holzman, 2014). De sterkste maatregel in de software ontwikkeling is het toepassen van model checking. In de Curiosity code worden 120 parallelle taken uitgevoerd. Het kan dan gebeuren dat meerdere processen bewerkingen uitvoeren waarvan het uiteindelijke resultaat afhangt van de relatieve snelheid van de uitvoering ervan.[10] Model checking helpt om fouten op te sporen die voortkomen uit de parallelle uitvoering van processen (Holzman, 2014).

Een vaak beproefde aanpak voor het realiseren van een betrouwbaar systeem in veiligheid kritieke (fail safe) systemen is het dubbel uitvoeren van componenten. Enkele hardware componenten in de Rover zijn daarom dubbel uitgevoerd, zodat in het geval van haperen of falen de andere component het werk kan overnemen. In de ruimtevaart wordt dit vaak gedaan door dezelfde software op nog een tweede PC parallel mee te laten draaien. Echter, over het algemeen is deze aanpak voor software beperkt, omdat de juiste werking van de software bepaald wordt door de toestand waarin de software verkeert. Als de software niet kan omgaan met deze toestand, er een niet bedoelde actie mee uitvoert, dan functioneert het systeem alsnog niet.[11] Een aanpak die bij software wel functioneert is het updaten van software. Zo is de Curiosity al snel na de landing voorzien van nieuwe software voor het beter functioneren van de robotarm op het karretje en een beeldverwerkingsroutine om obstakels te detecteren. Deze software is na de landing van de Rover gemaakt en overgezonden vanaf de aarde.



SELFIE VAN MARSROVER CURIOSITY, OKTOBER 2015. BRON: NASA JPL

8. https://mars.jpl.nasa.gov/msl/multimedia/images/?ImageID=4421

9. Zogenaamde software anomalies.
10. Zogenaamde race condities.
11. Zie crash Ariane 5 (Lions e.a., 1999)
12. https://mars.jpl.nasa.gov/msl/

3. HET LECTORAAT HIGH TECH EMBEDDED SOFTWARE (HTES)

> "Een lectoraat is bijna synoniem aan groei. Een lector initieert en coördineert praktijkgericht onderzoek naar vraagstukken die leven bij het bedrijfsleven of in de maatschappij."
>
> – ZESTOR[13]

Dit hoofdstuk introduceert het lectoraat en beschrijft op welke toepassingsgebieden het zich richt als het gaat om slimme systemen.

Het lectoraat High Tech Embedded Software (HTES) van Fontys Hogeschool ICT (FHICT) houdt zich bezig met onderzoek naar methodes en technieken voor het verbeteren van de software ontwikkeling voor high tech embedded systemen.

Embedded systemen spelen een belangrijke rol in de economie van de regio Zuid-Nederland. Deze regio is de hotspot in Nederland op het gebied van innovatie. Meer dan 40% van de Research and Development (R&D)-investeringen wordt gedaan in Zuid-Nederland en wordt uitgevoerd bij bedrijven zoals Philips, ASML, NXP, Océ/Canon en DAF Trucks (RIS3, 2013).

Het lectoraat HTES wil graag een rol spelen in dit ecosysteem. Dit sluit goed aan bij de ambitie van FHICT om haar onderzoek optimaal te verbinden met het bijbehorend werkveld om zo te komen tot optimale kenniscreatie en uitwisseling.

Naast de economische relevantie van embedded systemen, hebben ze ook een brede invloed op mens en maatschappij. Het gaat immers om slimme en intelligente systemen die invloed kunnen hebben op ons werk, ons leven. Het zijn systemen die autonoom kunnen opereren in ongestructureerde omgevingen en daarin hun eigen weg moeten zoeken. Ook deze interactie met techniek heeft de aandacht van het lectoraat.

Het onderzoek van het lectoraat begint vanuit systeemdenken.

13. Zestor is arbeidsmarkt- en opleidingsfonds van het hbo (Zestor, 2018).

Dit systeemdenken is nodig omdat de embedded software zelf en de ontwikkeling ervan steeds complexer wordt en meer interacties tussen software en andere disciplines heeft. Het lectoraat ziet vele uitdagingen op het gebied van het ontwikkelen van embedded systemen, zoals: het maken van betrouwbare embedded software, software die systemen veilig laat werken, software die goed samenwerkt met andere software, enzovoorts. Voor het systeemdenken en om met systeemcomplexiteit om te gaan hanteert het lectoraat een denkkader dat bestaat uit vier aspecten die zijn gebaseerd op eigen ervaringen met en toegepast onderzoek rondom het thema high tech systemen: architectuur, integratie & test, kwaliteit en modelleren. Deze aspecten zijn ook terug te vinden in de literatuur, zoals (Kim & Kumar, 2012), (Muller, 2012) en (Serebrenik & Mens, 2015). Het kader wordt uitgewerkt in hoofdstuk 5.

Dat het systeembegrip centraal staat om embedded software te begrijpen zien we ook terug in het ICT & Technology onderwijs, waar de semesters zijn

georden aan de hand van soort systeem. In semester 2 staat de hardware-software interactie centraal. In semester 3 is het thema event driven systems waarbij het gaat om toepassingen die verdeeld zijn over meerdere processen. Thema van het vierde semester is meet- en regelsystemen die over meerdere computers zijn verdeeld.

Het enorm brede toepassingsgebied van embedded systemen maakt het voor het lectoraat noodzakelijk hierin nadere keuzes te maken. Om focus aan te brengen richt het lectoraat zich op de software ontwikkeling van specifieke embedded systemen, namelijk die van (mobiele) robots en sensornetwerken. Beide toepassingen zijn gekozen na overleg met bedrijven, marktverkenningen en overleg met docenten en studenten binnen FHICT. Dit is gebeurd door de onderzoeksgroep Research Group Technology, die zich sinds oktober 2015 gebogen heeft over de koers van het praktijkgericht onderzoek voor de studierichting ICT & Technology.

De keuze voor mobiele robots is ingegeven omdat het een aansprekende vorm van Cyber Physical Systems zijn, zoals bij robot voetbal en Automated Guided Vehicles (AGV's). Er is ook voldoende robotkennis aanwezig om op voort te bouwen. Zo heeft Eric Dortmans, oud Fontys lector, veel kennis rondom de software ontwikkeling van robots, zoals het Robot Operating System (ROS), ontwikkeld en uitgedragen.

Sensornetwerken maken het mogelijk om over het verbinden van systemen onderling na te denken. Ook hiermee is al ervaring opgedaan. Zo heeft de voorganger van het huidige lectoraat, het lectoraat Architectuur van Embedded systemen onder leiding van Wim Hendriksen, met het project Firebee gewerkt aan draadloze netwerken (Hendriksen e.a., 2013).

Het doel van het lectoraat HTES is het verbeteren van kennis en vaardigheden op gebied van embedded software, van medewerkers en studenten van FHICT en de bedrijven waar we mee samenwerken.

Het praktijkgerichte onderzoek richten we op:

- Het realiseren van mobiele autonome robots.
- Slimme sensornetwerken waarmee metingen kunnen worden uitgevoerd (in de openbare ruimte, gebouwen en woningen en in natuur).
- Ontwerptechnieken om deze systemen te kunnen maken.

Al deze onderwerpen vereisen informatici die goed kunnen communiceren met andere disciplines, en goed kunnen bepalen wat er echt toe doet in de toekomstige (complexe) systemen. Dit zijn precies het soort informatici die de opleiding ICT & Technology probeert op te leiden.

4. SLIMME SOFTWARE IN MOBIELE ROBOTS EN SENSORNETWERKEN

> Voorlopig is het grootste gevaar niet dat computers zo slim worden dat ze ons leven gaan beheersen, maar juist dat computers nog steeds zo ontzettend stom zijn, en toch ons leven al beheersen.
>
> – DANIEL DENNETT (BROCKMAN, 2015)

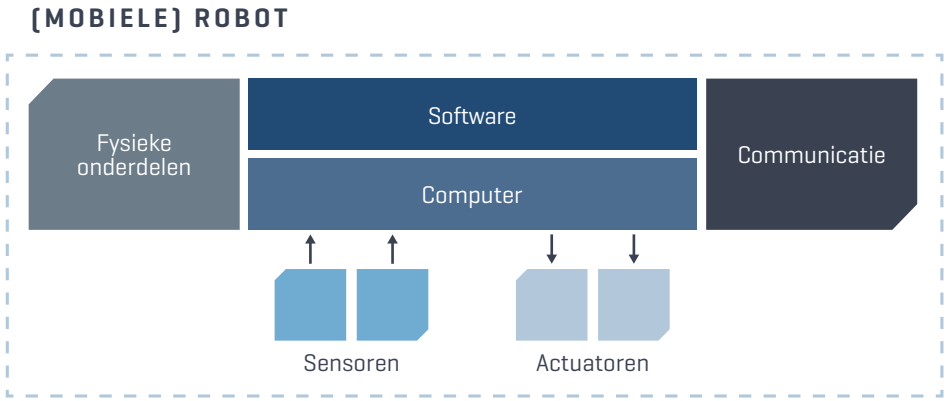
Dit hoofdstuk gaat in op de twee toepassingsgebieden van het lectoraat, mobiele robots en sensornetwerken. Vervolgens komt het maatschappelijke belang en de uitdagingen van beide aan de orde.

4.1. TWEE TOEPASSINGS-GEBIEDEN

4.1.1 Mobiele robots

Robots bevatten de vier plus één ingrediënten van een embedded systeem. Ze hebben sensoren als zintuigen, waarmee ze in hun omgeving gebeurtenissen (events) kunnen waarnemen. Voorbeelden zijn een camera, een gyroscoop en wielbewegingssensoren.

Robots gebruiken actuatoren om te reageren op hun omgeving, om acties uit te voeren; bijvoorbeeld de motoren om sneller of langzamer te rijden. De software om de robot aan te sturen wordt uitgevoerd op de besturingscomputer. Communicatie is belangrijk voor robots om met mensen, andere robots of computers samen te werken. Ze zullen bijvoorbeeld contact hebben met een (draadloos) netwerk, of geven signalen door via licht en/of geluid. Ook wordt (draadloos of bedrade) communicatie gebruikt voor het bijwerken van de software op de robot. Verder hebben veel robots fysieke onderdelen, zoals bewegende armen om voorwerpen vast te pakken en te stapelen of wielen om voort te bewegen.



FIGUUR 4 - ELEMENTEN VAN EEN ROBOT.

FIGUUR 5 - ROBOT PLATFORMEN WAAR HET LECTORAAT AAN WERKT.

Het robot domein is divers. Er zijn rijdende robots, robots die aan mensen doen denken, zoals de Nao[14] en Pepper[15] robots, of juist mensen mogen opereren, zoals de RoboSculpt[16].

Er zijn robots die producten kunnen assembleren, of de onderdelen kunnen transporteren en kunnen oppakken. Robots kunnen op verschillende manieren worden ingedeeld. Bijvoorbeeld op basis van de taken die ze uitvoeren of hun werkomgeving, zoals voor de zorg van mensen, medische operaties, entertainment of productie van auto's. Ook kunnen robots worden ingedeeld op basis van hun fysieke vorm, zoals een arm met grijper, een mobiel platform of een delta robot[17]. Verder kunnen robots worden ingedeeld op basis van hun omgeving, bijvoorbeeld of ze in de lucht gebruikt worden (drones), op het land (Marsrover) of in het water, zoals de Octopus robot (Chianetti e.a., 2015).

Het lectoraat HTES richt zich op mobiele robots en dan specifiek op de robots die zich voortbewegen op twee of meer wielen (Siegwart e.a, 2011). Dit zijn dynamische robots die zich snel kunnen voortbewegen, zoals de Robocup wedstrijden laten zien. Ook komen tijdens de softwareontwikkeling van dit type robots, de multidisciplinaire uitdagingen van Cyber Physical Systems aan de orde, zoals het versnellen van de robot. Het lectoraat kijkt naar mobiele robots van verschillende complexiteit, waarbij we ook praktische aspecten als opslag, onderhoud en kosten meenemen. Daarom zien we de eenvoudige miniatuurrobots zoals EV3 van Lego Mindstorms[18] en Turtlebot3[19] als geschikte platformen voor een deel van ons onderzoek. Verder kijken we naar AGV's en de Turtle5K voetbalrobot[20]. Complexiteit en daarmee de te realiseren intelligentie op de robot hangt af van factoren zoals de gebruikte sensoren, de manier van besturen en de mate van zelfstandig redeneren. Figuur 5 toont vier robot platformen waar het lectoraat de komende tijd aan werkt. We zien hierin een stijgende complexiteit om de software ervoor te ontwikkelen. De ontwikkelingen op het gebied van zelfrijdende auto's, door bijvoorbeeld Waymo[21], en Marsrovers – door Nasa en ESA – vormen een belangrijke inspiratie bron. Ze vormen daarmee de 'stip op de horizon' voor het ontwikkelen van mobiele robots.

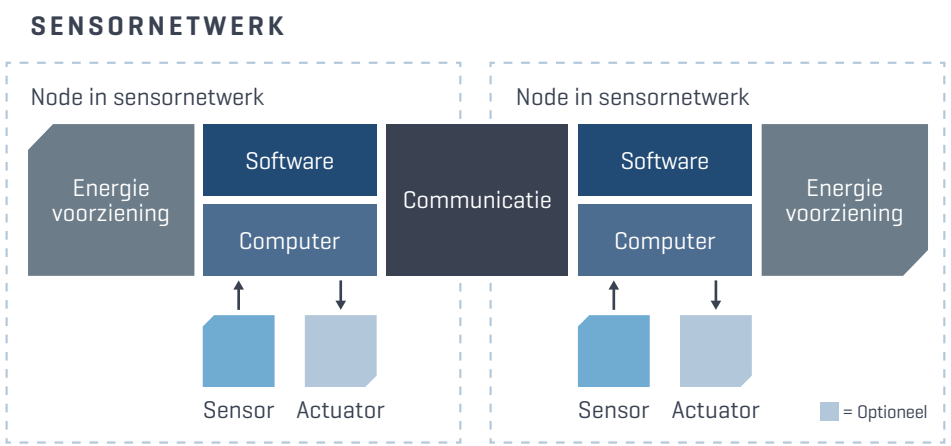
4.1.2 Sensornetwerken

Een sensornetwerk bestaat uit vele kleine, vaak goedkope, sensoren die over een gebied of gebouw verspreid zijn. De sensoren registreren fysieke veranderingen zoals temperatuur, luchtdruk of licht en geven de data hierover door binnen het netwerk. Sensornetwerken kunnen bedraad en draadloos zijn uitgevoerd. De focus van het lectoraat ligt op draadloze netwerken omdat sensornodes dan flexibel te plaatsen zijn op mobiele robots en omdat dergelijke netwerken specifieke uitdagingen voor software engineers kennen, zoals energievoorziening.

Deze wireless sensor networks (WSN) bestaan uit tientallen tot honderden sensoren die met elkaar communiceren via radiofrequenties en zo gegevens uitwisselen. Vaak liggen de sensoren op verschillende afstanden van elkaar. De afstanden kunnen variëren tussen centimeters en kilometers. Elke sensornode[22] bevat een microprocessor en is daarmee een klein embedded systeem, met hardware en embedded software (Buratti, e.a., 2009). Sensornetwerken bevatten veelal drie van de vier ingrediënten van een embedded systeem: sensoren, software en hardware, zie figuur 6.

Soms zijn de nodes in een sensornetwerk voorzien van actuatoren, waardoor ze in staat zijn om in de fysieke wereld in te grijpen. Dit zijn zogenaamde wireless sensor and actuator (actor) networks (WSAN).

14. https://www.softbankrobotics.com/emea/en/robots/nao
15. https://www.softbankrobotics.com/emea/en/robots/pepper
16. https://www.tue.nl/universiteit/nieuws-en-pers/nieuws/16-04-2018-nieuwe-robot-voor-schedelbasischirurgie-is-zeer-nauwkeurig-en-ontlast-chirurgen/
17. https://www.youtube.com/watch?v=6JIjCpW35CM
18. https://www.lego.com/nl-nl/mindstorms/products/mindstorms-ev3-31313
19. http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
20. http://roboticopenplatform.org/wiki/TURTLE
21. https://waymo.com/
22. Soms ook aangeduid als mote.



FIGUUR 6 - ELEMENTEN VAN EEN SENSORNETWERK.

Een voorbeeld is een WSAN dat met sensoren wandelaars op straat detecteert en op basis daarvan de straatverlichting in- en uitschakelt. Zo zorgt het systeem ervoor dat de openbare ruimte alleen verlicht wordt als er mensen aanwezig zijn.

Elke node in een sensornetwerk heeft communicatie en een (eigen) energievoorziening. Met de communicatie worden gegevens in het netwerk doorgegeven. Ook zal deze voorziening worden gebruikt om software bij te werken. In een WS(A)N zorgt de (radio)zender voor de communicatie tussen de nodes in het netwerk. Zenders zenden met een bepaalde frequentie – tussen de 400 MHz en 6 GHz – en bandbreedte, zie bijvoorbeeld de wifi standaard IEEE 802.11. De combinatie frequentie en bandbreedte wordt bepaald door de afstand tussen de nodes (wifi zendt bijvoorbeeld uit tot 100 meter), en de benodigde zendenergie om over deze afstand data te sturen. De eventuele demping en reflectie van de (radio)signalen spelen een rol bij deze keuze. Bluetooth, Zigbee, Wifi en Ultra Wide Band (UWB) zijn voorbeelden van draadloze protocollen die onderling verschillen in bandbreedte en frequentie en daarmee in zendafstand, hoeveelheid en snelheid van dataoverdracht (Lee e.a., 2007). Energieverbruik is een beperkende factor voor sensornodes. Vaak werken nodes met een batterij, die niet regelmatig kan worden opgeladen. Daarom is het interessant om nodes zelf energie aan hun omgeving te laten onttrekken, zie paragraaf 9.2.

Sensornetwerken worden op vele manieren ingezet:

- Landbouw - Sensoren in aardappelvelden om ziekte en groei te registeren.
- Industrie - Gas sensoren die worden gebruikt om in industriële processen snel gevaarlijke gassen op te sporen, of sensoren die temperatuurverschillen in een productielijn meten.
- Domotica - Sensoren die registreren of het donker wordt om vervolgens de gordijnen van de woning automatisch te sluiten.
- Verkeer - Sensoren in het wegdek om te bepalen of er in de winter zout gestrooid moet worden.

4.1.3 Combinatie van mobiele robots en sensornetwerken

Mobiele robots en sensornetwerken zijn op zichzelf twee onderzoeksgebieden. Elk thema is op zich al voldoende toepassingsgebied voor een lectoraat. Het lectoraat HTES pakt beide gebieden aan omdat deze verbanden met elkaar hebben en elkaar beïnvloeden. Dit zien we al terug in de systeemplaatjes van beide in de figuren 4 en 6. Sensoren spelen een centrale rol in beide toepassingsgebieden. In sensornetwerken genereren ze de data. In robots genereren sensoren de gegevens waarmee het robotsysteem redeneert en beslissingen neemt. Sensoren worden steeds beter bruikbaar voor robottoepassingen. Dit zien we bijvoorbeeld bij robotvraagstukken, zoals indoor positioning[23]. De robot moet daar bepalen waar het apparaat staat. Hiervoor kan de robot zijn eigen sensoren gebruiken om de omgeving waar te nemen, maar we zien ook dat de robots sensornetwerken in hun omgeving gaan gebruiken om hun eigen plaats te bepalen. Een voorbeeld is de mogelijkheid om vanuit een 'wolk van sensor punten' de positie van de robot te lokaliseren. Andersom worden met sensoren op mobiele robots ook data verzameld. De gyroscoop en accelerometer op de robot registreren bijvoorbeeld hoek en versnelling. Deze data kunnen aan andere robots of andere afnemers worden doorgegeven. Er ontstaan zo mobiele sensornetwerken, zoals een Robocup voetbalteam.

4.2. DE MAATSCHAPPELIJKE INVLOED VAN SLIMME SYSTEMEN

Het lectoraat ziet dat slimme systemen steeds meer impact hebben op de maatschappij. "Robots zullen steeds vaker flexibel worden ingezet, waarbij de samenwerking tussen robot en mens en robots onderling belangrijker wordt" (BOM, 2015).

"De aansturing van robots gaat steeds vaker decentraal gebeuren" (Holland Robotics, 2018). Een sprekend voorbeeld zijn de twee robots die samen een Ikea-stoel in elkaar zetten, zonder ruzie te krijgen![24]

23. Als radar gebruikt wordt als sensor dan bepaalt de gebruikte bandbreedte de resolutie/nauwkeurigheid. Met een grotere bandbreedte kunnen kleinere objecten beter worden gezien.
24. https://www.nytimes.com/2018/04/18/science/robots-ikea-furniture.html

Ook ontwikkelingen in de zorg en de robotica maken dat mensen en robots steeds meer gaan samenwerken. Robots moeten daardoor steeds beter ontwikkeld worden om met mensen te leren omgaan. Daarmee is er in toenemende mate behoefte aan het voorspellen van robotgedrag en het beter interpreteren van data van deze systemen.

Diverse partijen, zoals het samenwerkingsverband Holland Robotics, verwachten dat 'robots uit hun kooi komen'. Waar tot nu toe robots vooral werkten in afgesloten omgevingen, omdat zij niet veilig genoeg waren om in de buurt van en naast mensen te opereren, zal dat de komende jaren anders worden. Steeds meer robots zijn in staat met mensen samen te werken en weten wat ze wel en niet moeten doen om veilig met mensen om te gaan. De zogenaamde collaborative robots (cobots) zijn dan ook met een opmars bezig. De komende jaren zal het steeds normaler worden dat mensen met een robot samenwerken. Dat robots zo nauw betrokken zijn in ons leven, maakt het ook belangrijk dat iedereen is voorbereid op deze samenwerking (Holland Robotics, 2018), (Sparc, 2016). Voor cobots betekent dit dat er data uit de omgeving van de robot verwerkt moet worden om daaruit snel en betrouwbaar conclusies te trekken over de aanwezigheid van mensen en hun intenties.

Robots zullen ook steeds vaker in ongestructureerde omgevingen moeten functioneren. Het gaat dan bijvoorbeeld om robotarmen die voorwerpen van verschillende grootte of vorm uit dozen moeten pakken. Een ander voorbeeld zijn de warehouses en productiestraten in de Smart Industry waar Automated Guided Vehicles (AGV's) goederen oppakken (picken) en verwerken voor een vervolgstap, zoals het op een pallet zetten van artikelen. Hier worden robots geacht veilig te reageren wat betekent dat ze bijvoorbeeld om moeten kunnen gaan met situaties waar er onverwacht voorwerpen of zelfs mensen op hun pad staan die hun geplande beweging belemmeren.

Robots zijn een vergaande vorm van automatisering. Ze zijn daarmee het vervolg op vele menselijke initiatieven om repetitief werk te verlichten en te vervangen, zoals Jacquard's weefgetouwen (1801), Holwirth's ponskaartenmachine (1890) en de eerste computers (1941-1948). Robots hebben ook invloed op werkgelegenheid. Frey en Osborne (2013) voorspelden in 2013 dat tot 2033 bijna de helft van het aantal banen in Amerika door computers of robots gaat verdwijnen. Er zijn websites waarop je zelf kunt nagaan wat de kans is dat jouw baan wordt weg geautomatiseerd[25].

Daar staat tegenover dat robots nieuwe mogelijkheden en daarmee nieuwe banen genereren (Est e.a., 2015), (Royakkers e.a., 2012). Een analyse van de ING-bank[26], die als reactie op het onderzoek van Frey en Osborne werd uitgevoerd, concludeerde dat de robot en andere arbeidsbesparende technologie volop kansen biedt voor Nederland, waardoor verschuivingen optreden in de opleidingsbehoeftes van mensen.

De manier waarop robots worden toegepast verschillen sterk. Robots kunnen mensenlevens redden door in rampgebieden gevaarlijk terrein voor ons te verkennen, maar ze kunnen ook worden ingezet als zelfstandig opererende aanvalswapens (Huët, 2017). Hierover komen al snel ethische kwesties rondom zaken als veiligheid en privacy naar voren. Isaac Asimov (2017) stelde in de vorige eeuw drie gedragsregels op waaraan uiteenlopende robots zich zouden moeten houden,[27].

Ik verwacht dat dergelijke regels de komende jaren verder zullen worden uitgewerkt per toepassingsdomein. In het domein van zorgrobots speelt bijvoorbeeld de discussie over de verantwoordelijkheid van ingrepen waarbij een standpunt is dat "een zorgrobot geen zelfstandige dokter mag zijn: er moet bij een operatie altijd een menselijke arts aanwezig zijn die de baas blijft over de situatie. Op deze manier blijft er ook altijd iemand, een mens, verantwoordelijk op momenten dat het fout gaat" (Schouten, 2017). Ik denk dat het aan individuele burgers en de *politiek* is om te bepalen hoe onze samenleving met de komst van robots in te richten. Dit is een politieke discussie waarbij het van belang is om onszelf te realiseren dat deze discussie wordt beïnvloed door de technologie zelf. De technologie verandert mogelijkheden en daarmee ons denken (Pous, 2008), (Eynikel, 2018). Ik denk dat het lectoraat HTES een bijdrage aan de maatschappelijke discussies over slimme systemen kan geven door duidelijk te maken wat er op technisch gebied mogelijk is. De (on)mogelijkheden van deze systemen spelen immers een belangrijke rol bij de vraag hoe wij tegen de techniek aan kijken, zie ook kader B.

Ook *sensornetwerken* hebben maatschappelijke impact. In het verleden zijn sensornetwerken ook wel als *smart dust* aangeduid. De miniaturisering van de elektronica zou het mogelijk maken dat talloze kleine draadloze sensoren, ter grootte van stofdeeltjes, alom worden toegepast om van alles te meten.

25. http://www.bbc.com/news/technology-34066941

26. https://www.ing.nl/media/ING_ebz_economisch_bureau_analyse_robots_zijn_voor_nederland_vooral_een_kans_5_mei_2014_tcm162-38954.pdf
27. https://nl.wikipedia.org/wiki/Drie_wetten_van_de_robotica

KADER B
KUNSTMATIGE INTELLIGENTIE MIST NOG ZOIETS ALS GEZOND VERSTAND[28]

Slimme apparaten worden onder andere slimmer door het gebruik van artificial intelligence (AI). Er is al langer een discussie gaande over de invloed van AI op mens en maatschappij, zie bijvoorbeeld (Mols, 2012). Tijdens het AI-congres CogX in juni 2018 kwam dit weer aan de orde. Journalist Wouter van Noort (2018b) beschrijft daarbij twee kampen.

Het eerste kamp denkt dat de ontwikkelingen in de AI zo snel gaan dat het onvermijdelijk is dat het mensen op vrijwel alle fronten gaat inhalen. Misschien al binnen enkele decennia. De onderbouwing die daarvoor wordt aangehaald zijn IBM's schaakcomputer Deepblue die Gary Kasparov in 1996 versloeg en Google's AlphaGo computer die het spel Go in 2016 van Lee Sedol won. De singularity beweging onder leiding van Ray Kurzweil, met ook volgelingen in Nederland, maakt ook deel uit van dit kamp. Singularity is het theoretische moment dat zgn posthumans (verbeterde mensen met een opgevoerde intelligentie) hun ontwikkeling in eigen hand nemen en zichzelf zo snel gaan verbeteren dat de wereld onbegrijpelijk wordt voor normale, hedendaagse mensen.

Het tweede kamp denkt dat het zo snel niet zal gaan. Dit kamp ziet een groot aantal technische beperkingen. Eén daarvan is de vereiste rekenkracht. Ook kan AI alleen specifieke taken uitvoeren. AI is te vergelijken met de software van een broodrooster, dingen die één ding goed kunnen, maar verder niks. AI heeft nog lang niet zoiets als gezond verstand.

28. Titel is overgenomen van het gelijknamige artikel (Noort, 2018b).

Sensoren zijn inderdaad alsmaar kleiner en breder toepasbaar geworden, o.a. door MEMS technologie[29] en de opkomst van draadloze communicatie. Dit heeft ervoor gezorgd dat er steeds meer sensornetwerken ontstaan en geïntegreerd worden in het dagelijks leven van de mens, het ubiquitous computing, waarbij de sensoren reageren op veranderingen in hun omgeving. Het wordt mogelijk om data te registreren wat en waar er voorheen niet gemeten kon worden, het pervasive computing[30].

Deze trend heeft met name invloed op de *privacy* van mensen. Hoe kleiner de embedded systemen met de sensoren worden, des te beter het mogelijk wordt om de gewoonten van mensen te registeren zonder dat mensen daar zelf weet van hebben. Het sensornetwerk kan daarmee worden opgevat als een spionage middel. Met sensornetwerken kan de locatie van mensen bepaald worden, zoals dat nu bijvoorbeeld al op grote schaal met GSM-technologie gebeurt. Immers met mobiele telefoons is de weg waarlangs gereisd wordt eenvoudig te traceren. Mensen gebruiken deze toepassingen ook vaak bewust. Hardlopers kunnen hun route aan de hand van een app volgen, om zo hun prestaties te verbeteren. Recent werd duidelijk dat de locaties van meer dan zesduizend mensen op meer dan tweehonderd militair gevoelige plekken waren te achterhalen via een mobiele fitness-applicatie (Peek, 2018).

Bepalend voor het gevoel van het verlies van privacy is het gevoel van controle over wat er gemeten wordt (Lederer e.a., 2003). Privacy betekent dat mensen weten wie wat over hen weet en als zij zelf controle hebben over welke informatie zij over zichzelf weggeven. Het is bij wijze van spreken het recht of de gunst om met rust gelaten te worden. De controle over de technologie is dus belangrijk voor de acceptatie van sensornetwerken. Een ander aspect van de maatschappelijke impact van sensornetwerken is de *beveiliging* er van. Beveiliging van sensornetwerken is te vergelijken met de informatieverspreiding over het internet en de beveiliging van deze informatie, de cyber security.

Ontwerpers van sensornetwerken die 'gevoelige' of vertrouwelijke data registreren worden geacht het systeem zo te maken dat deze data niet in handen kan komen van organisaties en personen voor wie deze data niet bedoeld zijn. In een draadloze omgeving is het mogelijk om via een 'hack' onderdeel te worden van het netwerk en zo ongeoorloofd data te achterhalen. Alle componenten van het sensornetwerk dienen daarom goed beveiligd zijn.

29. https://www.mems-exchange.org/MEMS/what-is.html
30. https://en.wikipedia.org/wiki/Ubiquitous_computing

4.3. UITDAGINGEN VOOR HET ONTWIKKELEN VAN ROBOTS EN SENSORNETWERKEN

De algemene verwachting is dat robots steeds vaker te maken gaan krijgen met minder gestructureerde omgevingen. Tevens zullen robots steeds vaker gaan samenwerken, met elkaar of met mensen. Robots zullen ook veel vaker autonoom gaan functioneren, wat betekent dat ze een eigen beslissingssysteem nodig hebben dat snel genoeg kan redeneren op basis van de data afkomstig uit eigen sensoren (Holland Robotics, 2018).

De maatschappelijke impact van robots laat zien dat het voor robot software ontwikkeling relevant is om na te denken over autonome systemen en hun intelligentie; zie kader C voor een kunstzinnige interpretatie.

Daarbij komen vragen naar boven, zoals:
- Wat maakt robots intelligent zodat ze autonoom kunnen functioneren?
- Hoe wordt intelligentie van een robot geprogrammeerd?
- Hoe kan kunstmatige intelligentie in robots worden toegepast? In welke mate? En wanneer bijvoorbeeld niet?

Op het gebied van sensornetwerken zien we een vergelijkbare ontwikkeling richting intelligente systemen, de smart systems. Het gaat om toepassingen die steeds meer omvattend worden omdat er technisch meer mogelijk wordt. Een grote uitdaging voor sensornetwerken is het ontwikkelen van netwerken die jaren gebruikt kunnen worden. Zo dient het netwerk te kunnen worden uitgebreid, terwijl het in de lucht blijft. Daarnaast wil men graag dat de nodes beperkt energie verbruiken omdat batterijen anders te vaak vervangen moeten worden.

Onderzoeksvragen zijn onder andere:
- Wat maakt een sensornetwerk intelligent?
- Wordt de intelligentie decentraal op nodes geprogrammeerd of wordt het centraal geregeld? Hier komen dan tal van sub vragen bij kijken, zoals: Waar de verwerking van gegevens plaatsvindt? Waar de rekenkracht wordt geplaatst? Hoe met het energie verbruik van de nodes om te gaan? Dit gaat over de autonomie van nodes in een sensornetwerk.

In het werkveld zien wij een behoefte aan praktische kennis en vaardigheden voor het ontwikkelen van embedded software voor mobiele robotica en sensornetwerken. Ontwikkelaars zijn daarbij deelnemers in de transitie naar slimme apparaten, waarbij zij werken aan ‘machines die denken’ (Brockman, 2015).

KADER C
SAM (SYMBIOTIC AUTONOMOUS MACHINE) ALS VOORBEELD VAN EEN INTELLIGENT SYSTEEM

Het is een robot die frisdrank maakt en ook alle functies daaromheen uitvoert. Het is een voorbeeld van hoe wij in de toekomst met autonome robots kunnen omgaan. SAM past de smaak van drankjes aan, regelt de voorraad, doet de financiën en houdt social media bij. SAM heeft een eigen bankrekening en bepaalt zelf de prijs van het drankje. Deze zou afhankelijk kunnen zijn van wat jouw voorgangers overhadden voor het drankje. Pin een frisje en SAM bepaalt wat voor glaasje prik jij ontvangt. Het frisje verkregen op 28 oktober 2017 smaakte lekker. Je kunt na betaling en drinken een oordeel over het drankje geven, waarmee de smaak van het drankje wordt beïnvloed. De frisdrank machine is o.a. tijdens de Dutch Design Week 2017 in Eindhoven gepresenteerd. Ontwerpers Marie en Arvid experimenteren met de mogelijkheid van machines om autonoom te werken en meer rechten te verwerven als zelfstandig acterende technologie. Er is gekeken om SAM een juridische entiteit te geven. Dat is vooralsnog niet mogelijk, al woedt er binnen de Europese Commissie een flinke discussie over elektronische entiteiten (Van Noort, 2018a). Recent kwam robot Sophia, een robothoofd, in het nieuws omdat Saoedi-Arabië de robot haar staatsburgerschap toekende. Als robots slimmer worden dan ontstaan er vergelijkbare ethische vraagstukken (Mols, 2017).

SAM

5. EEN KADER VOOR SYSTEEMDENKEN OVER SLIMME SOFTWARE

> Alle levende systemen zijn complex...
>
> – FRITJOV CAPRA (CAPRA & LUISI, 2014)

Dit hoofdstuk introduceert een kader voor systeemdenken om praktijkgericht onderzoek naar slimme software te kunnen doen. Dit kader omvat vier aspecten: architectuur, integratie & test, modelleren & leren en software kwaliteit.

We hebben hiervoor gezien dat embedded software functioneert in high tech systemen. Omdat deze systemen steeds vaker samen werken en ingrijpen op de fysieke wereld zijn het Cyber Physical Systems geworden. Deze systemen worden slimmer. Om over deze systemen te redeneren is systeemdenken nodig. Dit raakt aan systems engineering (Incose, z.j.) en het begrip systems architecting (Muller, 2012). Systeemdenken kent een rijke traditie en heeft verschillende invloeden ondergaan, zie bijvoorbeeld (In't Veld, 1988), (Checkland, 1981).

Dit essay behandelt systeemdenken vanuit het belang van het beheersen van de complexiteit en het multidisciplinaire karakter van de software, waarbij het eerste leidend is. Bij de ingrediënten van embedded systemen in hoofdstuk 2 kwam al aan de orde dat er andere dan de software discipline zijn die van belang zijn bij het ontwikkelen van embedded systemen, zoals elektronica en fysica. Software zou daarbij de integrerende discipline moeten zijn die rekening houdt met en invloed neemt op de andere disciplines. Om het systeem correct te ontwikkelen en te integreren, dienen software ontwikkelaars te redeneren vanuit het systeem, niet alleen vanuit de software. Complexiteit staat vaak centraal bij het ontwikkelen van embedded software. Het gaat zowel om de complexiteit van de software zelf (het product) als de complexiteit van het proces om software te maken (het proces).

De complexiteit van software als product heeft met de randvoorwaarden en de mate van distributie te maken.

KADER D
COMPLEXITEIT VAN SOFTWARE

In het algemeen worden software-systemen steeds groter en ingewikkelder. Dit is terug te voeren op de toename van code en de onderlinge afhankelijkheden in de software. Een veelgebruikte maatstaf (metriek) om software complexiteit uit te drukken is het aantal regels code (LoC) van een systeem. Hieronder is deze omvang voor enkele systemen uitgedrukt (McCandless, 2015):

- Space Shuttle (1981) = 400.000 LoC
- Boeing 777 (1995) = 4.000.000 LoC
- Marsrover Curiosity (2012) = 5.000.000 LoC
- Boeing 787 (2011) = 14.000.000 LoC
- Android Operating System (vanaf 2008) = 12.000.000 LoC
- Personen auto, zoals Mercedes of Lexus (2015) = 100.000.000 LoC

Dat de omvang en complexiteit van software toeneemt is logisch vanuit de groei van systemen en toename in functies die ze vervullen. Veel van de software in een embedded systeem beschrijft het gewenste systeem gedrag, de bedoelde software functies. Vaak wordt er ook veel software gemaakt om het systeem om te laten gaan met mogelijke problemen –soms wel 30% tot 70% van de totale software. Het is software die nodig is om het systeem bestuurbaar te houden in allerlei foutsituaties, de zogenoemde exception handling (Bruntink e.a., 2006). Deze software kan ingewikkeld zijn omdat het vaak te maken heeft met verschillende systeemonderdelen (actuatoren en sensoren), terwijl de afhandeling vaak binnen een bepaalde tijdsafloop moet plaats vinden.

Software code wordt ook complex omdat het over jaren door verschillende mensen wordt ontwikkeld. Het overdragen van kennis over de gemaakte software verloopt niet altijd even goed, terwijl ook inzichten en ideeën over wat het systeem moet kunnen veranderen. Zo kan *legacy code* ontstaan die vaak moeilijk is te onderhouden. Het begrip *software bloating* legt de nadruk op niet noodzakelijk groeiende code die wordt veroorzaakt door o.a., slechte specificaties, zwak ontwerp, ondersteuning voor niet gebruikte functies. Sommige bedrijven proberen hun code te reduceren, want code onderhouden kost veel geld. Om code te reduceren is een *refactoring* of *redesign* van de code nodig (Sommerville, 2015).

Embedded systemen moeten voldoen aan een grote variëteit beperkende randvoorwaarden, zoals rondom energieverbruik, geheugenruimte, verwerkingssnelheid, beschikbare bandbreedte, tot aan afmetingen en gewicht toe. Tegelijkertijd moeten ze voldoen aan strikte eisen betreffende (real-time) prestaties, betrouwbaarheid, privacy van gegevens, beveiliging en integriteit, (netwerk) integratie en systeem interoperabiliteit. Deze beperkte middelen en strikte niet-functionele eisen onderscheiden embedded systeemtechnologie van ICT-systemen in het algemeen. De complexiteit neemt ook toe door de verschuiving van stand-alone systemen naar System-of-Systems. Dit levert geheel nieuwe uitdagingen op, waarbij innovatieve strategieën en processen nodig zijn bij het productontwerp, om deze complexe producten en diensten te kunnen ontwikkelen (Geelen e.a., 2018).
Zie kader D voor meer over complexiteit.

Ook het software maakproces is complex. Er werken vaak meerdere mensen en teams tegelijkertijd aan software. Het wordt ook vaak op verschillende plaatsen, '24 uur per dag/7 dagen in de week', in de wereld ontwikkeld. Verder zijn er vaak verschillende andere disciplines betrokken bij het software maakproces. Deze multidisciplinariteit vereist van softwareontwikkelaars dat ze de keuzes uit de andere disciplines, zoals elektronica, begrijpen en meenemen. In de loop der tijd zijn vele programmeertalen ontwikkeld om betere software te kunnen maken. Ook zijn de methoden en technieken, zoals Continuous Integration en geautomatiseerd testen, geïntroduceerd, om softwareontwikkeling te verbeteren. Software ontwikkelaars zijn zich continu aan het bijscholen. Organisaties verliezen nogal eens het overzicht over wat de software doet en wat er moet gebeuren als er ergens iets moeten worden aangepast (Beck, 2003), (Sommerville, 2015).

Om complexiteit beheersbaar te maken zijn er twee algemene richtlijnen (Basten, 2011). De eerste is het *decomponeren* van 'iets' groters, ingewikkelds, tot kleinere eenvoudigere delen die begrepen worden en het aanbrengen van structuur tussen de delen. Het decomponeren is een beproefd principe tijdens ontwerpen, waar begrippen bij horen als functionele decompositie, separation of concerns, compositionaliteit en divide and conquer. Het resultaat van het decomponeren, de systeemdecompositie, wordt aangeduid met begrippen als systeemarchitectuur, work breakdown structure, en systeemontwerp.

Na het decomponeren dienen de onderdelen, nadat zij ontwikkeld zijn, weer te worden samengebracht. Het integreren van een systeem is essentieel om een werkend systeem te krijgen. Echter, vaak zien we dat er onvoldoende aandacht is voor integratie.

Er zijn legio voorbeelden van systemen waarvan het geplande ontwerp niet inzichtelijk was of niet werkte zoals bedoeld[31].

Samenstellen van de systeemdelen, oftewel *integreren*, is daarom erg belangrijk in een systeembenadering. Integreren is 'de andere kant van decomponeren'; het is het spiegelbeeld van de architectuur. En net als de architectuur wordt ook de integratie gedefinieerd door de architect. In grotere systemen zal echter vaak een rolverdeling plaatsvinden en zal de integratie door een team van architecten, of integratoren worden gedefinieerd en onderhouden.

De tweede richtlijn voor het beheersen van complexiteit is *abstraheren*. Daarbij wordt niet-essentiële informatie weggelaten om de essentie, dat waar het om gaat in een ontwerp, over te houden. Met de gevonden abstracties worden vragen beantwoord voor het ontwerpproces. Het resultaat van abstraheren zijn modellen. Modellen spelen een belangrijke rol in de architectuur omdat zij antwoorden kunnen geven op de structuur van de decompositie.

Het decomponeren, integreren en modelleren zijn activiteiten die leiden tot een product. Om te weten of het product goed is zullen we de *kwaliteit* ervan vast willen stellen. Software kwaliteit zullen we vaststellen aan de hand van de *kwaliteitseigenschappen* van het systeem.

We hebben nu vier aspecten gevonden die ons helpen om naar slimme software in Cyber Physical Systems te kijken.

Activiteit (Werkwoord)	Resultaat (Zelfstandig naamwoord)	Betrokken persoon/rol
Decomponeren Syn: Ontwerpen, Definiëren van de architectuur Architecting, Structureren	Decompositie Syn: Ontwerp, Architectuur, Structuur	Ontwerper, Architect, Syn: Ontwikkelaar
Samenstellen en testen Syn: Integreren, Componeren	Integratie (plan, volgorde) en Test	Architect, Integrator
Abstraheren, Syn: Modelleren	Abstractie, Syn: model	Architect, ontwikkelaar
Systeem eigenschappen bepalen	Kwaliteit, Syn: System/software quality, System performance, System KPI.	Architect, ontwikkelaar, Quality assurance

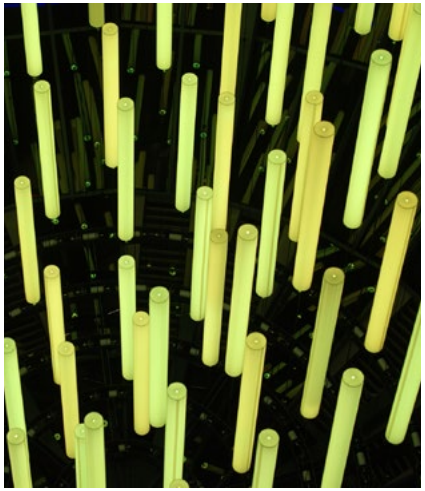
TABEL 1 – BEGRIPPEN VOOR SYSTEEMDENKEN IN HIGH TECH EMBEDDED SOFTWARE.

31. Voorbeelden zijn crash Ariane 5 (Lions e.a., 1999) en de ontwikkeling van de Boeing 787 Dreamliner (Allworth, 2013).

Deze vier aspecten vormen het kader voor het systeemdenken. Ze vormen perspectieven, bij wijze van spreken 'de bril', waarmee het lectoraat HTES naar de wereld kijkt. Vanuit deze perspectieven stellen we vragen over mobiele robots en sensornetwerken, hoofdstuk 6 tot en met 9, en geven daarmee invulling aan de onderzoekslijnen, zie paragraaf 10.1. Het kader bestaat dan uit de volgende vier aspecten, met de daarbij behorende onderzoeksvragen:

1. Architectuur - Het decomponeren, en daarmee het ontwerpen van het systeem, wordt in het kader aangeduid als architectuur. Dit gaat over de structuur die nodig is om een systeem te maken. Het definiëren van functies en interfaces van mobiele robots en sensornetwerken helpt om de structuur ervan te vinden en te behouden.
   De eerste onderzoeksvraag die het lectoraat zich hier stelt is: hoe krijgen software engineers een beter begrip van de functies van Cyber Physical Systems, waarvan de ontwikkeling kennis uit diverse disciplines vereist?

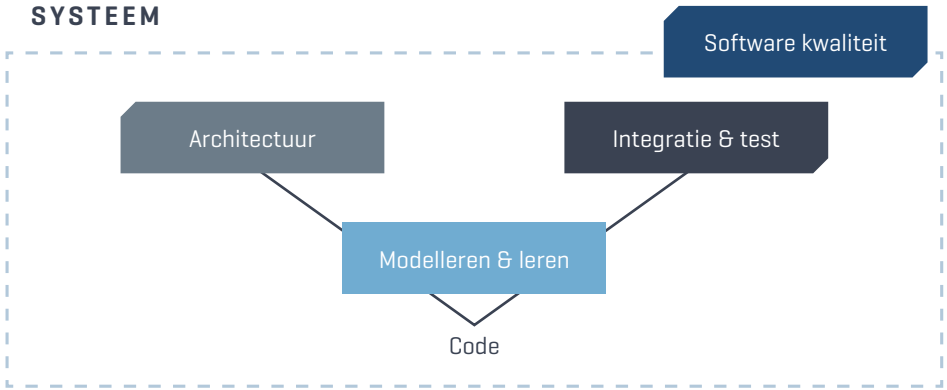
2. Integratie en test - Het samenstellen van systemen op basis van de architectuur duiden we verder aan als integratie en test. Wetenschappelijk onderzoek heeft tot op heden weinig aandacht besteed aan systeem- en software integratie. Integreren wordt immers vaak gezien als een kwestie van organiseren. Wij denken dat het meer omvat dan het organiseren en werken deze invalshoek in hoofdstuk 7 uit. Daarbij is integratie juist een gebied waar het hbo met haar praktische insteek en denkwijze toegevoegde waarde heeft. De tweede vraag die het lectoraat hier stelt is: hoe kunnen functies in slimme systemen het beste gesimuleerd en getest worden?



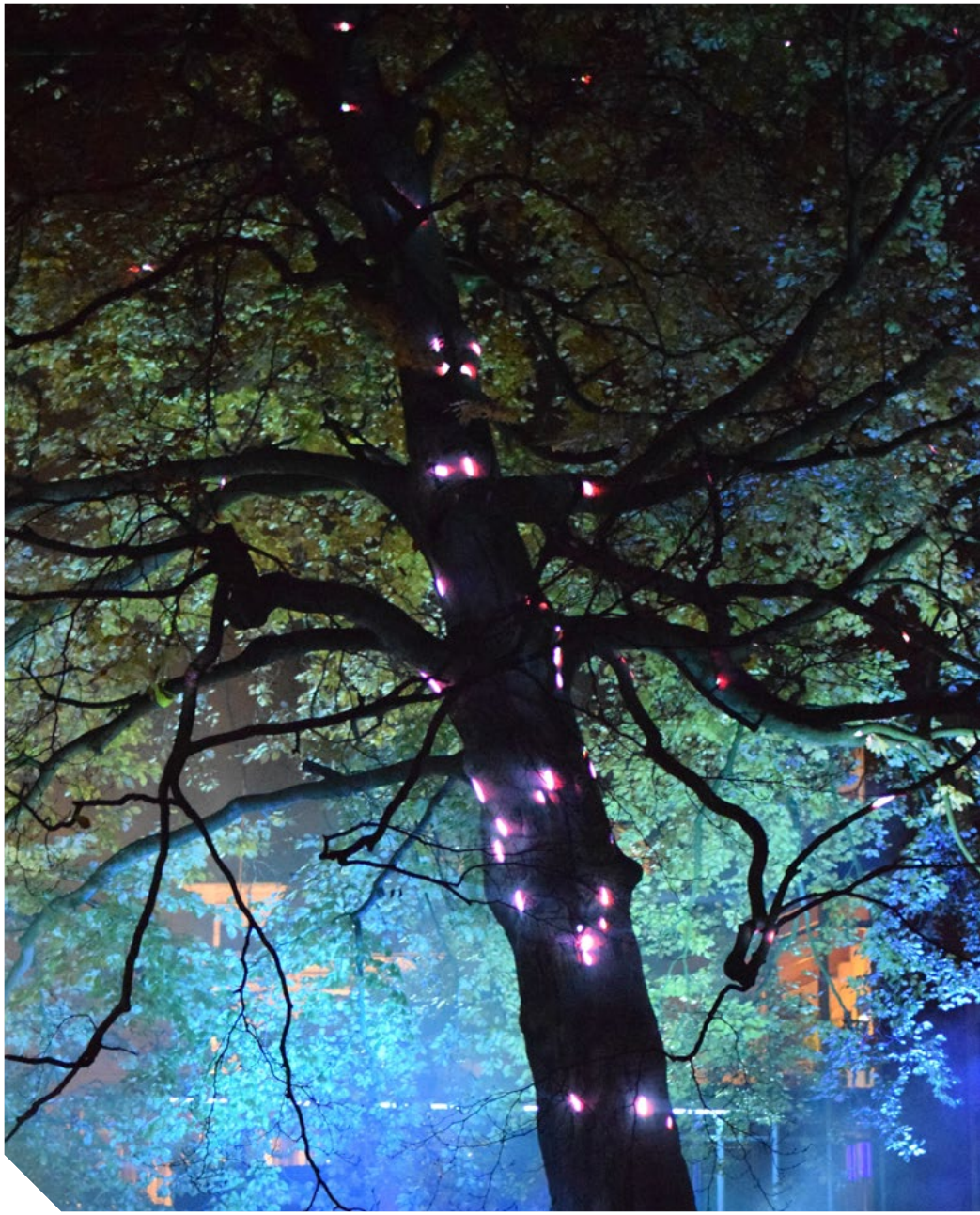
3. Modelleren en leren, oftewel (model) leren - Het abstraheren, het maken van modellen, wordt in het denkkader aangeduid als modelleren en leren. Modellen spelen een belangrijke rol in de architectuur en integratie en test omdat zij antwoorden geven op de vragen over de structuur van de decompositie en daarmee over de architectuur en integratie. Modellen

voor embedded software beschrijven het systeemgedrag in termen van gebeurtenissen en toestanden. Er zijn diverse typen modellen mogelijk op basis van abstractieniveaus en modellerings-aspecten. De derde centrale vraag die het lectoraat hier stelt is: welke modellen zijn nodig om de vragen rondom robots en sensornetwerken te beantwoorden? Hierbij wordt naast ontwerp/architectuur ook naar modellen in combinatie met testen en simuleren gekeken. Als we modelleren opvatten als een manier om gedrag te definiëren, dan komt bij slimme systemen de vraag boven of en hoe zij dit gedrag kunnen aanleren?
De rol van kunstmatige intelligentie is bij het ontwikkelen van intelligente systemen van groot belang. Daarom is leren als thema toegevoegd aan modelleren. De vierde vraag die het lectoraat hier stelt is: wat is de relatie tussen model en leren? Hoe kan robot software worden gemodelleerd zodat robots gedrag kunnen leren?

4. Software kwaliteit - Kwaliteits-eigenschappen worden in het denkkader verder aangeduid als software kwaliteit. Kwaliteit is het product van architectuur, integratie en test en het modelleren en leren. Voor dit aspect formuleert het lectoraat HTES (nog) geen onderzoeksvragen. Software kwaliteit wordt gebruikt om vragen en deel-aspecten rondom de slimme software te ordenen, bijvoorbeeld door aandacht te geven aan (cyber)security van robots.



FIGUUR 7 - ASPECTEN VAN SYSTEEMDENKEN RELEVANT VOOR HIGH TECH SOFTWARE: HET DENKKADER, DE BRIL VAN HET LECTORAAT HTES.



6. ARCHITECTUUR – KADER, ROBOTS EN SENSORNETWERKEN

> Alle architectuur is ontwerp (design), maar niet elk ontwerp is architectuur. Architectuur omvat de relevante ontwerpbeslissingen die het systeem bepalen, waarbij relevant wordt gemeten in de kosten om te wijzigen.
>
> – GRADY BOOCH (BUSCHMAN E.A., 2007)

In dit hoofdstuk wordt het eerste aspect van het kader voor systeemdenken uitgewerkt. Het begrip architectuur wordt eerst beschreven. Daarna wordt op de betekenis ervan voor mobiele robots en sensornetwerken ingegaan.

We gebruiken architectuur in dit essay zowel in de zin van het proces om te komen tot de architectuur (*architecting*) als het resultaat ervan (*architecture*)[32]. De nadruk ligt op het resultaat, de architectuur, omdat we met name naar de beschrijving van de architectuur kijken. Binnen de software engineering zijn diverse talen ontwikkeld om software architectuur te beschrijven. Vaak wordt de Unified Modeling Language (UML) notatie gebruikt om de structuur en het gedrag van een softwaresysteem te beschrijven (Selic, 2000), (Booch, e.a., 1999). Een aanvulling daarop is SysML[33]'. Dit is een modelleertaal die voortkomt uit UML, maar breder, voor het gehele systeem, is te gebruiken. Deze leveren verschillende diagrammen, zoals klasse diagrammen, message sequence charts en state diagrammen. In de literatuur wordt vaak benadrukt dat een software architectuur beschreven moet worden aan de hand van perspectieven (views) en belanghebbers (stakeholders). Het 4+1-model in het Rational Unified Process (RUP) beschreven door Kruchten (1995) is hier een voorbeeld van, maar er zijn meerdere manieren (Hofmeister e.a., 2007), (Brown, 2014). In de praktijk zien we deze perspectieven vaak al terug in de diagrammen waarin we structuur en gedrag van het systeem vastleggen.

32. ISO 42010 standaard geeft terminologie voor architectuur(ISO/IEC 42010).
33. http://www.omgsysml.org/what-is-sysml.htm

In de UML notatie wordt voor de structuur vaak het klasse diagram gebruikt. Voor het beschrijven van gedrag gebruiken we dan state machine diagrammen en sequence charts.

Op een hoger abstractieniveau wordt de software architectuur vaak uitgedrukt aan de hand van componenten. Dit kunnen de onderdelen van het systeem zijn, zoals database, driver, client en server. Vaker zal worden geprobeerd om ook de functie van de component aan te geven, zodat duidelijk is wat de component doet. Voorbeelden zijn beeldverwerking en positionering. Deze functies zijn soms één-op-één met de fysieke onderdelen van het embedded systeem verbonden, zoals de fysieke camera in het geval van beeldverwerking. Een software functie kan ook door het samenwerken van verschillende (sub)functies/componenten worden gerealiseerd. De beeldverwerking bestaat bijvoorbeeld uit bewerkingen als beeld uitlezen, beeld opslaan, beeld analyseren.

Tussen software componenten wordt gecommuniceerd. Data die door de sensorcomponent worden verzameld zullen door de besturingscomponent worden gebruikt om commando's aan de actuatorcomponent te geven. Om deze communicatie goed te laten verlopen is het belangrijk om goede interface definities op te stellen.

Zo wordt duidelijk welke data een component nodig heeft en welke het aan andere componenten levert.

De componenten tezamen realiseren een systeem dat op een bepaalde manier functioneert. Deze manier duiden we aan als de kwaliteit van het systeem, die we uitdrukken aan de hand van kwaliteitseigenschappen, zie hoofdstuk 9. Deze eigenschappen verwijzen naar de systeemfuncties. Bijvoorbeeld: de beeldverwerkingsfunctie realiseert de beeldkwaliteit. Dit kwaliteitsattribuut kunnen we bijvoorbeeld meten aan de hand van beeldresolutie.

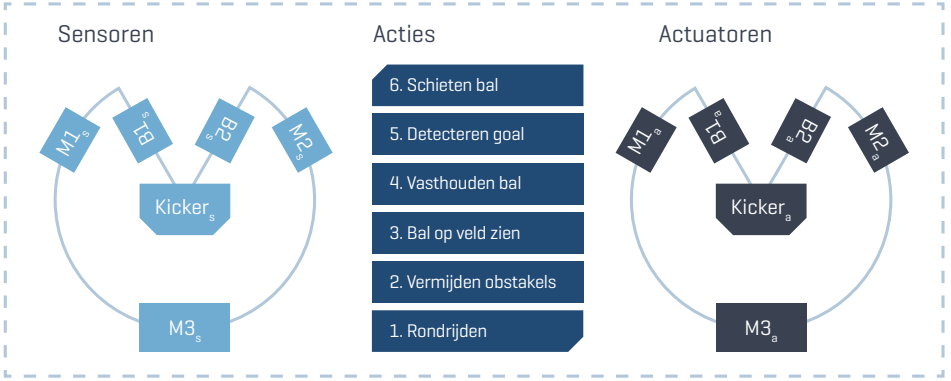
Als een systeem onderdeel is van een System-of-Systems zal een ontwerper/architect nadenken over hoe haar systeem samenhangt met andere, al bestaande of nog te bouwen systemen. De interfaces van het systeem met andere systemen worden dan beschreven. In de architectuur van een samenwerkend systeem zal over een aantal zaken moeten worden nagedacht. Allereerst het delen van data. De verschillende sensoren op de verschillende systemen zullen ongelijksoortige gegevens (met verschillende data structuren) gaan opleveren en deze aan elkaar doorgeven, terwijl de systemen tegelijk parallel aan elkaar werken. Een tweede uitdaging is het opdelen van functionaliteit. De functies gedefinieerd voor systeem A hebben mogelijk invloed op de functies van systeem B. Dit kan resulteren in *emergent behavior*: onverwacht systeemgedrag. De uitdaging voor de architectuur van System-of-Systems is om de samenwerkende systemen zo te ontwerpen dat er voorspelbaar gedrag resulteert.

In de volgende paragrafen worden de bovenstaande architectuur aspecten uitgewerkt voor mobiele robots en sensornetwerken.

6.1. ARCHITECTUUR VAN ROBOTS – ACTIES EN GEDRAG

Een robotarchitectuur gaat over de softwarestructuur om de robot acties te kunnen laten uitvoeren, zodat vastgelegd is hoe de robot reageert op de data die hij met zijn sensoren opvangt. Zo ontstaat robot gedrag. We illustreren dit hieronder aan de hand van de acties, sensoren en actuatoren van een voetbalrobot[34].

Een voetbalrobot heeft diverse sensoren en actuatoren, zie figuur 8. Hij heeft sensoren om te detecteren of hij een bal vast heeft. Verder heeft hij een camera om zijn omgeving te registreren en vast te stellen of hij binnen de lijnen van het speelveld staat en om te weten waar de bal en het doel zijn. Ook zal de robot kunnen bijhouden hoe ver er gereden is met behulp van zijn wielsensoren ($M1_s$, $M2_s$, $M3_s$). De actuatoren van de voetbalrobot zijn drie motoren ($M1_a$, $M2_a$, $M3_a$), die omniwheels (Siegwart e.a., 2011) aandrijven, zodat de robot bijvoorbeeld met een bepaalde snelheid kan bewegen.



FIGUUR 8 - TWEE KEER BOVENAANZICHT VOETBALROBOT: SENSOREN, ACTUATOREN EN ACTIES.

34. Voorbeeld is afgeleid van het vak Robotics dat in de Embedded Systems minor in semester 7 wordt gegeven.

Verder heeft de robot twee balwiel-motoren ($B1_a$, $B2_a$) die in de 'mond' van de robot, op bal niveau zijn gemonteerd. Om een bal te schoppen is de robot uitgerust met een schopsysteem, de kicker.

Om te kunnen reageren op wat robots 'zien' in hun omgeving zijn er acties geprogrammeerd in de robotsoftware. Onze robot heeft bijvoorbeeld als basisactie dat hij gaat rondrijden totdat het een bal detecteert. De robot stuurt hiervoor de drie motoren $M1_a$, $M2_a$ en $M3_a$ aan zodat er bijvoorbeeld een rechthoekig parcours over het veld met een snelheid van 1 meter/seconde wordt gereden. Omdat de robot niet tegen zijn teamgenoten en tegenstanders mag aanrijden, is een tweede actie geprogrammeerd om obstakels te vermijden. Hiervoor gebruikt de robot zijn camera. De derde actie die de robot uitvoert is het actief opsporen van de bal. Dit wordt gedaan met behulp van de camera. Zodra de bal is gespot, rijdt de robot er op af en probeert de robot de bal te pakken met zijn balwielmotoren $B1_a$ en $B2_a$. Dit is de vierde actie, het vasthouden van de bal. De balwielmotoren gaan, op het moment dat de bal detectoren $B1_s$ en $B2_s$ – die ook in 'mond' van de robot zijn gemonteerd – een bal zien, ronddraaien zodat de bal wordt vastgeklemd. Als de voetbalrobot de bal in bezit heeft, dan kan zijn ultieme doel worden gerealiseerd, namelijk het maken van doelpunten.

Hiervoor wordt de actie detecteren van goal begonnen, wat de robot met behulp van de camera doet. De robot houdt het doel continu in de gaten. Als de plaats van het doel is vastgesteld wordt de actie schieten van bal begonnen. Eerst wordt aan de hand van de wielsensoren bepaald hoe snel de robot rijdt, tegelijkertijd wordt de afstand tot het doel gemeten zodat kan worden bepaald hoe hard het schot moet zijn. Bij het schieten richt de robot zich met zijn voorkant in de richting van het doel en gebruikt het zijn kicker.

Het geheel van acties beschrijft de functionele architectuur van de voetbalrobot. De genoemde acties vatten we op als (systeem)functies of taken van de robot. In de robotica worden dergelijke functies vaak op algemeen niveau onderkend (Siegwart e.a., 2011):

- Perception - de robot observeert de omgeving, de wereld van de robot. Hiervoor worden met behulp van verschillende sensoren metingen gedaan.
- Localisation - de robot bepaalt zijn positie (bijvoorbeeld uit te drukken in x,y coördinaten) in zijn omgeving.
- Cognition - de robot beslist hoe hij gaat reageren om zijn doel(en) te bereiken.
- Motion control - besturen van de beweging. De robot verandert zijn motor aansturing om een traject te kunnen volgen.

Het maken van dergelijke functies is ingewikkeld en vereist inzicht in de algoritmes en fysieke aspecten van de robot. Een voorbeeld is het ontwikkelen van perception, om te bepalen waar de bal in het veld ligt en waar het doel en waar de robot en zijn teamgenoten en tegenstanders staan. In het RIF Robocup team, zie kader F, werken Peter Lambooij en Eric Dortmans, beiden docentonderzoekers van het lectoraat HTES, aan het calibreren van de software voor een 360-graden spiegel. Hiermee leren we wat de pixels betekenen en hoe de echte afstand tot de robot in het veld wordt afgebeeld op de spiegel en daarmee op de pixels van de camera.

De bovenstaande wijze om robotgedrag te definiëren aan de hand van parallelle acties is Behavior Based Control (Brooks, 1986), (Mataric, 2001). Het bepaalt de manier van software programmeren voor robots. Acties worden gelijktijdig gestart op basis van de gebeurtenissen in de omgeving van de robot. Er zijn zogenaamde arbiters nodig om te bepalen welke taken met voorrang moeten worden uitgevoerd op het moment dat er tegelijkertijd triggers voor verschillende, tegengestelde, acties binnenkomen. Deze arbiters geven met hun prioritering niveaus in een architectuur aan. Behavior Based Control wordt tegenwoordig algemeen toegepast in moderne, service gebaseerde robots. Er zijn ook andere invloeden op het maken van software code. Eén daarvan ontstaat door de onzekerheid over de toestand van robot mee te nemen. In kader E wordt dit geïllustreerd aan de hand van onzekerheid over toestanden in een cobot met een robotarm.

Robotacties zullen over het algemeen in lagen worden beschreven, om abstractie-niveaus en de interfaces tussen de componenten/functies duidelijk te kunnen beschrijven. De lagen onderscheiden dan verschillen in prioriteit en tijd. In de RIF voetbalrobot architectuur wordt bijvoorbeeld een basis hardware abstractie laag (HAL) onderkend, waar de sensoren data genereren en actuatoren werken; voor een indruk zie figuur 9. Voor deze laag is het tijdsbudget 1000 Hz, wat betekent dat de software van deze laag de motoren met deze kloksnelheid moet kunnen aansturen. De tweede laag in de architectuur betreft de perceptie (wat ziet de robot?) en robotacties, zoals rijden, schieten en vasthouden van de bal. De software in deze laag werkt op een lagere frequentie (25 Hz), hiervoor wordt software uit het Robot Operating System (ROS)[35] gebruikt (O'Kane, 2014), (Quigley e.a., 2015).

35. www.ros.org

KADER E
VOORBEELD VAN HET OMGAAN MET ONZEKERHEID OVER TOESTAND ROBOT IN CODE

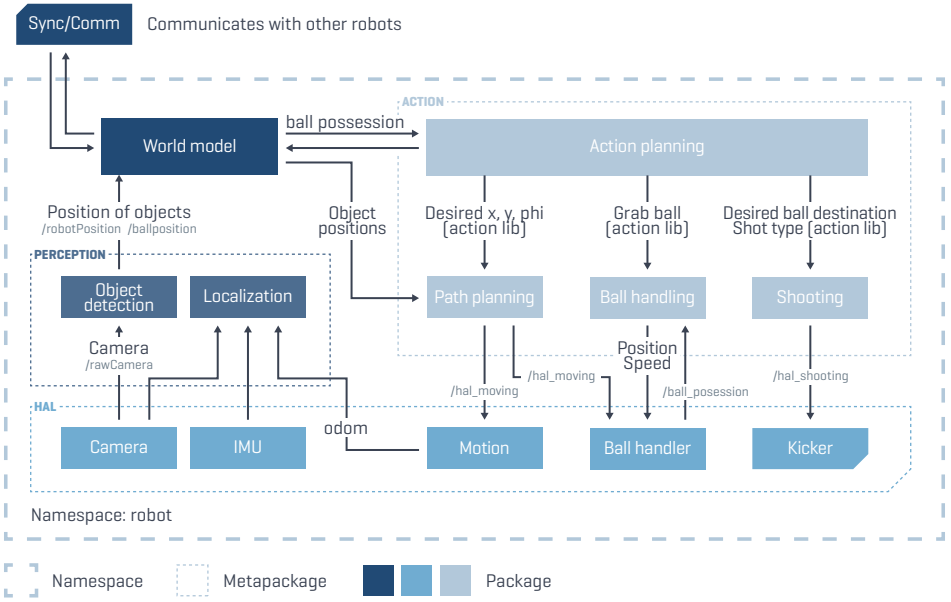
Het volgende voorbeeld van het meenemen van onzekerheid over toestanden in een cobot met een robotarm is afkomstig uit de praktijk van het bedrijf SmartRobotics[36]. In embedded systemen wordt vaak procedureel geprogrammeerd, er wordt een procedure van instructies uitgevoerd, zoals:

- Move_to_object()
  //beweeg [de robot arm] naar het object
- Grab_object ()
  //pak het object op
- Move_object_to_destination
  //vervoer het object naar een bestemming

In de praktijk zijn er aanvullingen nodig om dergelijke instructies te programmeren. De robot moet bijvoorbeeld stoppen met de instructie Move_to_object als het geen object ziet. Omdat het om een cobot gaat mag de instructie niet worden uitgevoerd als de mens met wie de robot samenwerkt niet gedetecteerd kan worden. Op deze manier worden er allerlei condities aan het programma toegevoegd. De instructie Move_to_object[] wordt dan bijvoorbeeld[37]:

```
Move_to_object()                    //beëindig actie als geen mensen
if NO-HUMAN or USER-STOPS:          zichtbaar zijn of als gebruiker de
    stop()                          actie stopt.
while not OBJECT-PRESENT:           //wacht met naar het object te
    if NO-HUMAN or USER-STOPS       bewegen als object niet zichtbaar is.
        stop()
    else:
        wait_some_time()
```

36. Voorbeeld is overgenomen uit de presentatie 'The software behind robots for production and logistics' gegeven door Heico Sandee (CTO Smart Robotics) tijdens het onderzoeksymposium ICT in Practice op 8 maart 2018.
37. Het voorbeeld dient hier voor het omkleden van procedurele aanpak met voorwaarden, maar kan verder worden uitgewerkt richting een 'beslissingsmachine' zoals Sandee in zijn presentatie liet zien.



FIGUUR 9 - SOFTWARE ARCHITECTUUR RIF ROBOCUP MSL TEAM, JUNI 2018[38].

Robots worden vaker uitgevoerd met een architectuur die bestaat uit twee tot vier lagen om de abstracties in taken af te dekken, zie bijvoorbeeld andere voetbalrobots (ASML Falcons, z.j.), (Tech United, z.j.), (VDL Robotsports, z.j.) en andere robots (Waibel e.a., 2011).

Het bovenstaande betreft de architectuur van één robot. Als we naar meerdere robots kijken dan vermenigvuldigen de architecturen zich, waarbij voor samenwerkende robots wel duidelijk moet zijn wie, wanneer, welke beslissingen op teamniveau neemt. Hiervoor zijn globaal gesproken twee manieren van sturing: centraal of decentraal.

Bij een centrale sturing van robots is er een coördinatie laag die alle acties van de robots regelt. Elke robot voert deze acties gehoorzaam uit. Deze situatie zien we vaak bij fleet managers voor AGV's).

38. RIF Robocup MSL team, Mathijs Frenken en Rick Pijnenburg.

Dit is onderwerp van onderzoek voor het lectoraat in het Let's Move IT-project, zie kader G. We zien dat AGV's in distributiecentra al enige intelligentie hebben en bijvoorbeeld zelf in staat zijn om obstakels te ontwijken, maar de overige functies, zoals het plannen van de route en het bijhouden van omgevingskaart worden door de fleetmanager geregeld.

Bij een decentrale aansturing zijn alle robots slim. Ze hebben elk een eigen besturingssysteem en kunnen zelfstandig beslissen. Iedere robot is een *agent*. De robots zijn gelijkwaardig aan elkaar en vormen tezamen een System-of-Systems dat in feite een gedistribueerd systeem is. De Robocup MSL is een voorbeeld van decentrale aansturing. De voetbalrobots zijn agents, die elk bepalen welke actie(s) ze uitvoeren. Voetbal is een teamsport, dus als robots samenspelen, zijn er afspraken nodig over elkaars intenties, wie welke rol heeft (spits, verdediger, etc.) en de daarbij behorende taken. Als drie robots A, B en C bijvoorbeeld een aanval doen, dan moet bij het afspelen van de bal door robot A op één of andere wijze duidelijk zijn voor robot B en C wie de bal gaat opvangen[39].

Vergelijkbare software configuraties zijn verdeeld over de robots en worden parallel door elke robot van het team uitgevoerd.

De besturing is gedistribueerd over de robots. De beslissingen op individuele robots veranderen op basis van rol, positie en spelmogelijkheden. Tegelijkertijd is de afstemming tussen robots zodanig, dat er bijvoorbeeld wel informatie over elkaars posities wordt uitgewisseld. Iedere robot is daarmee in staat om zijn eigen wereld model over de posities van de anderen, bepaald door camera beelden, te verrijken met positie informatie aangereikt door andere robots. Bij een decentrale aansturing is de mate van autonomie en intelligentie op de individuele robots dan ook groter dan in de decentrale aansturing. Een interessante vorm van decentrale aansturing zijn de voetbalrobots in de Robocup Small Size League (SSL)[40] die voor hun informatie afhankelijk zijn van één camera die boven het veld hangt. De spelverdeling gebeurt centraal, waarna iedere robot zijn eigen rol oppakt.

In het Let's Move IT-project onderzoekt het lectoraat hoe centrale en decentrale aansturing zich tot elkaar verhouden. Bij een centrale aansturing is er een fleet management systeem dat de planning uitvoert, maar zijn er ook functies, zoals routeplanning, centraal geregeld. Bij decentrale planners heeft iedere AGV een planningsfunctie aan boord.

39. Voor de duidelijkheid: in het Robocup voetbal mogen robots onderling niet communiceren over plannen.
40. http://www.robocup.org/leagues/7

KADER F
RIF ROBOCUP MSL VOETBAL TEAM

RIF is een consortium van bedrijven en onderwijsinstellingen, waarin Fontys Hogeschool ICT het voortouw neemt. RIF is een afkorting van resistance is futile, oftewel Robocup Initiatief Fontys.
Het doel van het team is om robot voetbal te spelen in de zogenaamde Robocup Midsize League[41] (MSL).
Robocup is een open gemeenschap, er wordt informatie uitgewisseld.
Het is ook een competitie, waardoor verschillende benaderingen in het robotvoetbalspel met elkaar kunnen worden vergeleken (Stone e.a., 2014).
Het tweede doel van RIF is om gestructureerd software te ontwikkelen voor samenwerkende mobiele robots met meerdere mensen en partijen. RIF is daarmee een consortium waarbij er naast Fontys ook bedrijven meedoen.

Met RIF komen diverse robot software uitdagingen aan de orde, zoals het opstellen van software architectuur, het samenwerken van robots in minder gestructureerde situaties en het ontwerpen en programmeren van robotfuncties, zoals vision. Het lectoraat ondersteunt het onderwijs waarin studenten kunnen werken aan de voetbal robots. Hiervoor werken we deels binnen het reguliere onderwijs.

Het Fontys RIF-team is onderdeel van het lectoraat HTES. Docenten in het RIF-team zijn ook betrokken bij het lectoraat. Studenten doen facultatief mee aan RIF of voeren een onderdeel van hun onderwijsprogramma uit in het team. RIF is daarmee een software lab, zie paragraaf 10.3.

41. https://nl.wikipedia.org/wiki/RoboCup_Middle_Size_League

KADER G
LET'S MOVE IT PROJECT[42]

Op dit moment leveren robotleveranciers specifieke verkeersmanagementsystemen (fleetmanager) en implementeren zij ook de interactie van robots met ERP (Enterprise Resource Planning) systemen, liften, deuren etc. op hun eigen wijze. Er bestaat geen generiek of geïntegreerd fleetmanagement systeem, waardoor de diverse typen robots nu letterlijk onafhankelijk van elkaar opereren. Dit resulteert in inefficiënt robotverkeer met een groot risico op verkeersproblemen, hinder voor personeel en dure parallelle koppelingen met interfaces (met deuren, liften etc.). Dit leidt tot verwarring, onzekerheid en potentiële veiligheidsproblemen bij werknemers op de werkvloer.

In het Let's Move IT-project wordt daarom gewerkt aan integraal verkeers-management, modulaire interfaces en slimme gecombineerde omgevings-interpretatie. Zo kunnen logistieke robots veilig, flexibel, robuust en adaptief opereren in een steeds veranderende productieomgeving in aanwezigheid van mensen. In het Let's Move IT-project wordt gewerkt aan een referentie architectuur om hiermee om te gaan.

Afhankelijk van de slimheid van de robots kan voor centrale of decentrale sturing worden gekozen. Zo onderzoeken wij of de keuze van de aansturing afhangt van de omgeving waarin de robot gaat opereren. De ruimte waarin de robots opereren speelt waarschijnlijk een rol. Op een kleine productievloer wordt over het algemeen een centrale planner gebruikt, omdat er een grotere kans is op verkeersopstoppingen. Ook kan het aantal objecten in de ruimte of de verschillen tussen de robots van invloed zijn.

6.2. ARCHITECTUUR VAN SENSORNETWERKEN – NETWERK EN NODES

De architectuur van een (draadloos) sensornetwerk gaat over het doorgeven van data uit de sensoren die elke node in het netwerk bevat. De opgevangen data worden via de nodes verstuurd binnen het netwerk. Het onderzoek van het lectoraat rondom de architectuur van sensornetwerken gaat over het opzetten van de netwerken en het programmeren van de nodes.

Netwerk - bij het opstellen van een sensornetwerk speelt de netwerk topologie en de routering van de data door het netwerk een centrale rol. De topologie gaat over de volgorde waarin en de manier waarop de sensornodes gekoppeld zijn. Dit is belangrijk om bijvoorbeeld te garanderen dat een netwerk het gehele gebied kan afdekken en de nodes makkelijk met elkaar te verbinden zijn. De architect zal bij het bepalen van de afstand tussen de nodes het bereik moeten afwegen rondom effecten dat als de nodes te dicht bij elkaar staan, meer energie verbruikt wordt en data vaker interfereren.

Afhankelijk van de gekozen topologie kan ook in sensornetwerken het onderscheid centraal en decentraal worden gemaakt, zie kader H. Star en cluster zijn centrale architecturen, terwijl mesh een decentrale opzet heeft. In de decentrale aanpak is de autonomie van de nodes groter dan in de centrale aanpak. Voor de software betekent dit dat op autonome nodes meer ingewikkelde functies worden geprogrammeerd. Dit is bijvoorbeeld nodig als data die in de openbare ruimte verzameld worden, persoonsgegeven bevatten die de levenssfeer van mensen aangaan. Deze data mogen allen geanonimiseerd of geaggregeerd verzonden worden zodat er geen privacy risico's zijn. Lokale bewerking van data kan ook nodig zijn om de te verzenden hoeveelheden data te bewerken.

In sensornetwerken geven de nodes gegevens aan elkaar door. Deze worden verzonden via de radiofrequenties, waarbij de nodes over het algemeen weinig energie verbruiken (ordegrootte 0,5 milliwatt). Dit betekent vaak dat nodes zoveel mogelijk in een slaaptoestand verkeren en toch op tijd wakker moeten worden om te meten, gegevens te ontvangen en deze door te zenden. Door het beperkte zendvermogen van radiomodules kunnen vaak slechts enkele nodes rechtstreeks data naar het volgende verbindingspunt sturen. Het totaal aantal nodes in een sensornetwerk is vaak groot, geregeld gaat het om honderden tot duizenden nodes. Op netwerk niveau is de software die de routes van de data door netwerk aanstuurt belangrijk. Er zijn diverse routeringsalgoritmes ontwikkeld waar hier rekening mee kan worden gehouden (Devloo, 2014).

De netwerk topologieën maken duidelijk dat voor de architectuur van sensornetwerken bepaald moet worden waar en hoe gegevens worden doorgestuurd en waar de data verwerking plaatsvindt, oftewel waar de berekeningen plaatsvinden. Als de gegevensverwerking centraal gebeurt moeten alle data via nodes worden doorgestuurd naar de centrale sink of router, hierdoor kan vertraging en verlies van data optreden, maar is er minder rekencapaciteit nodig op de nodes zelf. In een decentrale aanpak wordt de rekenkracht verdeeld. Bij grote hoeveelheden data is dit bijvoorbeeld nodig om de verwerking ervan aan te kunnen. Elke node in het netwerk bevat een

42. https://www.nwo.nl/onderzoek-en-resultaten/onderzoeksprojecten/i/89/29489.html

processor waarop software draait. De software zorgt voor het (herhaald) opvragen van de meetgegevens bij de sensor en het doorsturen van de meetgegevens in het netwerk. Afhankelijk van de netwerktopologie worden de data gestuurd naar nodes, sinks en/of routers. De software kan ook bewerkingen op de data uitvoeren. Zo dient de software op de nodes om te kunnen gaan met uitzonderingssituaties, zoals het verzenden van gegevens naar nodes die tijdelijk niet beschikbaar zijn, waarbij de verzendende node er voor moet zorgen dat dezelfde data herhaald verzonden worden totdat het bericht komt dat de data ontvangen zijn.

Verder dient in de architectuur worden nagedacht over bewerkingen op nodes. Er zijn meerdere bewerkingen mogelijk. Eén daarvan is optimalisatie van nodes, om de hardware beter te gebruiken en om de node beter in staat te zijn om data pakketten te ontvangen en door te sturen. Omdat de microprocessoren van de nodes beperkte geheugen- en processor capaciteit hebben draaien ze vaak een speciaal operating systeem, zoals ContikiOS[44] of TinyOS[45]. De sinks en centrale verwerkingseenheden in een sensorwerk en de gateways die de verbinding met het internet maken, kunnen meer gegevens, van meerdere nodes, verwerken. Zij hebben meer processor capaciteit en meer uitgebreide software dan de nodes. Met de komst van sterkere en goedkope processoren, zoals ESP8266[46], wordt het mogelijk om meer bewerkingen op een node uit te voeren. Op een node worden soms meerdere sensoren gebruikt. Er zijn vele soorten sensoren beschikbaar, afhankelijk van wat men wil meten. Voorbeelden zijn infrarood, laser, (lucht)druk en licht sensoren. De fysieke eigenschappen van dergelijke sensoren is van invloed op de te maken software. Het gaat daarbij om de precisie en nauwkeurigheid van de sensoren. Een laser sensor is bijvoorbeeld heel precies: herhaalde metingen komen op hetzelfde punt uit. De sensor kan tegelijkertijd onnauwkeurig zijn, omdat de overeenkomsten tussen de gemiddeld gemeten waardes laag is. Een infrarood sensor is daarentegen vaak niet erg precies, maar wel nauwkeurig, de sensor heeft een bias.

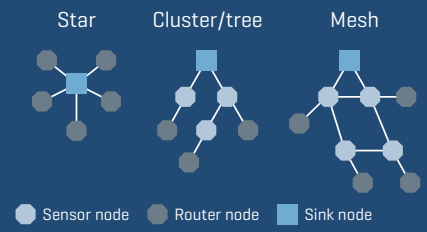
Om met de verschillende karakteristieken van sensoren om te kunnen gaan, zullen filters moeten worden toegepast. Een voorbeeld zijn *particle filters* die aan de hand van de meetwaarden globale optima bepalen die dan de toestand schatting zijn, zie bijvoorbeeld (Kang e.a., 2018).

44. http://www.contiki-os.org/
45. http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Documentation_Wiki
46. https://www.espressif.com/en/products/hardware/esp8266ex/overview

KADER H
NETWERKTOPOLOGIEËN

Er worden over het algemeen drie netwerk topologieën onderkend:

- Star – één-op-één verbinding. Elke sensor is direct verbonden met een sink node, waarin de data single-hops nemen. Eén sink node verzamelt de data van een groep nodes. Voordeel van deze architectuur is het over het algemeen lagere energieverbruik. Het nadeel is echter dat het maximale oppervlakte dat het netwerk kan afdekken beperkt wordt door de maximale afstand tussen twee nodes.
- Mesh – één-op-veel verbinding waarin elke router verschillende nodes combineert en zo koppelt aan meerdere nodes. Voordeel van deze architectuur is dat het netwerkoppervlakte groter kan zijn, dat er minder data verlies is en het netwerk zichzelf kan herstellen.
- Cluster/boom – dit een combinatie van de star en mesh architectuur. Het is een hiërarchische topologie waarin het netwerk in clusters wordt verdeeld, waarbij de clusters met elkaar via zgn. clusterhoofden communiceren. Het combineert de voordelen van beide architecturen, maar levert mogelijk ook vertraging van het verzenden van data op.



NETWERK TYPOLOGIEËN VOOR SENSORNETWERKEN (XU E.A., 2014)

Het filter wordt gebruikt om grootheden te schatten in niet-lineaire dynamische systemen, het is een op een Monte Carlo simulatie gebaseerde implementatie van de regel van Bayes voor dynamische systemen. Deze filters worden gebruikt voor het volgen en schatten van de positie van objecten, bijvoorbeeld het schatten van een positie van een mobiele robot. Voor dergelijke filters wordt dan software op de nodes ontwikkeld. De kennis over filters in sensoren kan ook voor mobiele robotica worden gebruikt omdat daar immers vergelijkbare situatie inschattingen moeten worden gedaan.

7.
INTEGRATIE EN TEST – KADER, ROBOTS EN SENSORNETWERKEN

> "Het doel van (systeem)integratie is het vroegtijdig vinden van onvoorziene problemen, om deze op tijd te kunnen oplossen."
>
> - GERRIT MULLER (MULLER, 2012)

In dit hoofdstuk wordt het tweede aspect van het kader voor systeemdenken uitgewerkt. Het begrip integratie en test wordt eerst beschreven. Daarna wordt op de betekenis ervan voor mobiele robots en sensornetwerken ingegaan.

Integratie & test is een activiteit die het spiegelbeeld is van architectuur & ontwerp. Tijdens integratie wordt nagedacht over hoe de losse componenten/functies tot een systeem worden samengesteld. Nadenken over integratie helpt echter ook om vroegtijdig eventuele problemen in het systeem te signaleren. Testen sluit aan op integratie omdat daarmee problemen ten opzichte van de gestelde eisen worden bepaald[47].

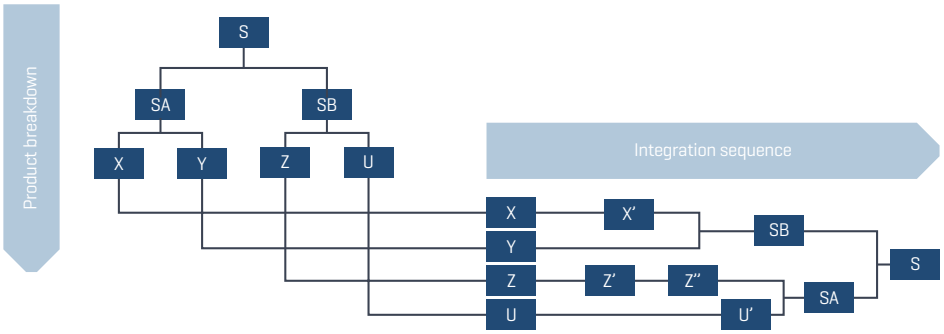
Vanuit het integratie perspectief denken architecten en ontwikkelaars anders dan vanuit het ontwerp perspectief, omdat er rekening wordt gehouden met tegenvallers, mogelijke problemen en benodigde tussenresultaten. Vaak gaan ontwerpers en architecten er vanuit dat het design dat zij maken goed/correct is. Echter, in de praktijk blijkt vaak het tegendeel. Dit komt doordat het geplande ontwerp niet goed is doordacht, of omdat het ontwerp in de praktijk anders werkt dan voorzien is op de tekentafel.

De kern van integratie is om zo vroeg mogelijk alle onderdelen van het uiteindelijk te maken systeem te laten werken om zo eventuele problemen vroegtijdig op te sporen.

47. Mijn visie op integratie en test komt voort uit mijn werk bij Embedded Systems Institute, dat met onder andere Frans Beenker (ESI), Tom Brugman (ASML), Andreas Schoenwaldt (ASML) vorm is gegeven in de cursus System Integration and Test., http://www.esi.nl/innovation-support/competence-development/courses/system-integration-and-test.dot. Deze kennis is vervolgens met Jac Jamar (TNO) en Evert van der Plassche (ESI TNO, Nobleo) toegepast in ESI integratieprojecten bij ASML tussen 2013 en 2015.

Immers dan kan er nog tijdig gerepareerd worden. Hiervoor is het nodig om terug te redeneren, het *backwards reasoning*, vanuit het uiteindelijke systeem naar de componenten. Zo wordt de integratie volgorde vastgesteld, waarin eerst de meest kritische systeemonderdelen worden opgeleverd en geïntegreerd. In termen van figuur 10: wat is er nodig om S te maken?, en vervolgens wat is nodig voor SB en SA?, enzovoorts. Zo zijn er voor een systeem integratie vaak tussenproducten nodig die niet in het ontwerp staan, maar wel nodig zijn om de deelsystemen, die wel in het ontwerp staan, te kunnen maken. In figuur 10 zijn deze tussenproducten aangegeven met Z' en Z''. Het zijn tussenversies van deelsysteem Z. De tussenproducten zijn te vergelijken met de steigers die nodig zijn tijdens het bouwen van een huis. Zo wordt bijvoorbeeld het metselen van een bovenverdieping mogelijk. Steigers worden niet door architecten ingepland, maar zijn toch nodig om het systeem te kunnen maken.

Het integratie en test perspectief vindt zijn oorsprong in de systems engineering, waar integratoren sterk te maken hebben met tegenvallers bij het tijdig opleveren van vaak fysieke onderdelen. Het is echter ook nuttig voor software engineering. Ook bij software levert de integratie vaak de echte pijn op. Grote fouten duiken pas op als alles in elkaar wordt gezet en alle componenten met elkaar moeten gaan praten. Menig FHICT-student kent deze situatie als alle slim bedachte onderdelen vlak voor de eindoplevering bij elkaar gebracht worden. Ook in software willen we daarom zo vroeg mogelijk inzicht in een werkend systeem hebben.



FIGUUR 10 - RELATIE TUSSEN ARCHITECTUUR EN INTEGRATIE.

48. Tekening door Teade Punter en Jac Jamar, 2013.

Daarom moeten basisfuncties van een systeem, bijvoorbeeld een motoraansturing, zo snel mogelijk worden geïmplementeerd, zodat kan worden aangetoond dat dit werkt. Daarna worden de andere functionaliteiten toegevoegd.

Integreren moet daarom zo vroeg mogelijk tijdens ontwikkeling beginnen en het dient ook continu te gebeuren. De software discipline onderkent daarvoor *Continuous Integration* (CI) (Humble en Farley, 2010). Software wordt automatisch gebouwd en de nadruk ligt er op geautomatiseerd testen. Zo kan software eerder op een omgeving worden uitgevoerd die lijkt op de uiteindelijke hardware omgeving. Dit helpt om software fouten eerder te vinden. Dit sluit aan op een Agile manier van software ontwikkelen (Boehm & Turner, 2004), (Kniberg, 2015), (Van Solingen en Rustenburg, 2010), (Angelov e.a., 2016). Aandacht voor integratie is ook belangrijk voor High Tech Embedded Software omdat software programma's met elkaar verbonden moeten worden om deze als systeem te kunnen laten werken en ze in een System-of-Systems werkend te krijgen. Daarbij is integratie een gebied waar het hbo met haar praktische insteek en denkwijze toegevoegde waarde heeft.

Vanuit het integratie perspectief zijn er vier met elkaar samenhangende richtlijnen om met de complexiteit van Cyber Physical Systems, zoals robots en sensornetwerken, om te gaan: acteren op risico, testen, simuleren en het run-time aanpassen van een systeem. Deze benaderingen worden hieronder toegelicht.

Acteren op risico - kan integratie helpen omdat het vroegtijdig vaststellen van risico's in de software architectuur helpt om op tijd maatregelen te kunnen nemen. In het geval een robot wordt ontworpen waarbij het van belang is om op elk moment te weten waar de robot zich bevindt, dan is het logisch om het positioneringsdeelsysteem goed te bekijken. Als dan uit experimenten blijkt dat de verschillen tussen de werkelijke en geplande robot coördinaten vaak acceptabel, maar regelmatig, bijvoorbeeld om de minuut, te groot zijn, dan is dat een risico. Het is vervolgens de vraag wat hier aan te doen: positionering nauwkeuriger maken of het systeem om laten gaan met de regelmatig optredende, te grote, afwijkingen in de positionering.

De uitdaging van risico denken is om dit niet alleen aan projectleiders over te laten, maar juist ook de architecten en ontwikkelaars er bij te betrekken. Het probleem is nogal eens dat risico-inventarisaties nietszeggende lijstjes worden. Het blijft dan bij inventariseren zonder acties te nemen om de risico's op te lossen (risk mitigation) of er mee om te gaan (risk contingency).

Ik denk dat het risico-denken in het praktijkgericht onderzoek aan het hbo[49] bijdraagt aan het duidelijk maken van het belang van onderzoek voor studenten. Veel van ons onderwijs richt zich op het maken van oplossingen, die in proftaken over langere tijd (10 tot 20 weken) worden ontwikkeld. Door studenten na te laten denken over de risico's van hun ontwerp – lees: de risico's van een niet-functionerend eindproduct – worden de onzekerheden in het ontwerp duidelijk. Dit zijn dan ook de punten waar onderzoek nodig is. Het risico denken helpt ook om testprioriteiten te stellen en kan helpen om keuzes in de architectuur bij te stellen.

Software testen - is het vaststellen van de mate waarin de software voldoet aan de gestelde eisen, de requirements (Graham e.a. 2007). Er zijn diverse testuitdagingen, namelijk:

1. Volledig testen bestaat niet - Algemeen wordt erkend dat het compleet testen van embedded software niet mogelijk is (Kaner, 1997), (Koopman, 2014). Of zoals vaak wordt geformuleerd: "testen toont wel de aanwezigheid van fouten, maar niet de afwezigheid ervan". Dit komt omdat systemen te groot en te complex zijn en omdat de tijd en de middelen om te kunnen testen tot uitlevering er niet is. Daarom worden er in de praktijk testprioriteiten gesteld en wordt er akkoord gegaan met het incompleet testen van systemen.
2. Snel feedback geven - een belangrijk gegeven in de software ontwikkeling is dat naarmate fouten later in het proces worden opgelost, het meer tijd en geld kost om ze te verhelpen; de wet van Boehm[50]. Testen is zo een filter voor het vangen van fouten. Naarmate het filter beter is worden fouten eerder gevonden, wat over het algemeen meer testtijd en daarmee geld kost.
3. Samenhang vinden tussen testniveaus - Vaak worden in systeemontwikkeling verschillende testniveaus onderkend, zoals unit tests, component- en systeemtests. Deze niveaus hangen samen met de omvang van het deel van het *system-under-test* (SUT). De algemene vraag hierover is hoe vanuit de lagere testniveaus, zoals unit tests, uitspraken te doen over de hogere testniveaus. Dus hoe voorspellen de unit tests iets over de software component, een deelsysteem of misschien wel het gehele systeem? (Leitner e.a. 2013)[51].

Simuleren (visualiseren) - is het nabootsen van de werkelijkheid waarbij een model van de werkelijkheid wordt gebruikt. Simulaties laten vanuit een beginsituatie zien hoe deze situatie verandert en zich ontwikkelt over de tijd. In het model zijn de regels opgesteld, volgens welke de verandering plaatsvindt. Door delen van een systeem te simuleren en/of te emuleren krijgen ontwikkelaars eerder inzicht in hun systeem.

Simulaties gebruiken een model van het systeem zonder dat er daadwerkelijk een systeem aanwezig is. Simulatoren worden ook wel aangeduid als Software-in-the-loop (SIL) of Hardware-in-the-loop als respectievelijk de hardware of software niet aanwezig zijn. Het gaat hier om emulaties waar hardware of software worden geïmiteerd, waardoor het externe gedrag van systeem kan worden geobserveerd. Bedrijven zoals Océ Technologies (Canon) en Vanderlande Industries (Toyota) gebruiken simulatoren tijdens hun software ontwikkeling (Roos, 2015).

Een nieuw begrip rondom simulatie, sterk verbonden met het de opkomst van Industrie 4.0 / Smart industry is *Digital Twin*. Het is een digitale representatie van het fysieke systeem. Beide worden naast elkaar gecreëerd (Grieves & Vickers, 2017). De digitale versie van het systeem kan dan helpen om problemen in de echte wereld op tijd op te lossen.

Testen en simuleren vullen elkaar aan. Testen wordt ook wel aangeduid als verifiëren, het geeft een bevestiging dat het systeem aan de gestelde eisen voldoet: *building the system right.* Simuleren ondersteunt het valideren, het geeft de bevestiging dat het systeem juist is, dat het voldoet aan de klantwensen: *building the right system*. Testen en simuleren zijn daarmee beide manieren om software- of systeemkwaliteit te bepalen; het vierde aandachtsgebied voor systeemdenken.

Runtime system adaptation - systemen zoals sensornetwerken en robots functioneren in veranderende omgevingen. De systeemomgeving zelf kan dan stabiel worden gehouden, maar men kan ook proberen om systemen zichzelf te laten aanpassen aan hun omgeving. Dit is run-time system adaptation. Het vereist intelligentie op de systemen die zowel de toestand in hun omgeving als hun eigen toestand kan waarnemen en vervolgens kan redeneren wat er aangepast moet worden om weer te functioneren door middel van mechanismen zoals zelf herstel (self healing). De uitdaging is om dit real time uit te voeren, en geen zichtbare vertraging in het systeemgedrag op te lopen. Hiervoor zijn modellen nodig die de huidige en toekomstige combinaties van toestanden in de omgeving beschrijven. Dergelijke modellen zullen op integratie/ architectuur niveau worden opgesteld.

49. HBO = hoger beroepsonderwijs
50. Wet van Boehm, https://www.toets-je-parate-kennis-over.nl/software-testen/wet-van-boehm/
51. Overigens is deze vraag te vergelijken met het toepassen van software metrics om software kwaliteit te bepalen (Punter, 2001), (Punter e.a., 2004).

In de volgende paragrafen worden de bovenstaande punten rondom integratie en test uitgewerkt voor zover van toepassing op mobiele robots en sensornetwerken.

7.1. INTEGRATIE EN TEST VAN MOBIELE ROBOTS

Bij integratie en test van mobiele robots gaat het steeds om de vraag of de robots gaan werken zoals dat bedacht is. Gaan de robots op het gewenste moment de juiste acties uitvoeren? Werken de robots goed samen? Dit zijn vragen die het lectoraat met testen en simuleren probeert aan te pakken.

Voor het *testen* van robotsystemen kan de robot architectuur als uitgangspunt worden genomen. Deze biedt immers het overzicht van de componenten en de acties die de robot moet gaan uitvoeren. Voor elk van de robotacties kunnen dan unit tests worden geschreven. Een voorbeeld is een test voor de actie 'rijden' die betrekking heeft op de motor component. De test kan dan zijn dat er gekeken wordt of output snelheid van de test elke keer hetzelfde is voor bepaalde input waardes. Unit tests zijn belangrijk omdat ze de basisacties van de robot afdekken. Als de robot verder wordt ontwikkeld en er komen nieuwe acties bij dan kan met unit tests eenvoudig worden bepaald of de al gerealiseerde acties ook nog kunnen worden uitgevoerd en de robotontwikkeling vooruit gaat[52]. Een aanpak om zo snel mogelijk feedback te krijgen, is het testen van fysieke robotonderdelen door ze na te bootsen (Graham e.a., 2007). De hardware onderdelen worden dan vervangen door mocks, die het gedrag van de betreffende onderdelen nabootsen. Zo kan de robot zonder de echte hardware worden getest. In het RIF voetbalteam is dit bijvoorbeeld gedaan door de hardware onderdelen, zoals kicker en motoren, softwarematig na te bootsen.

De uitdaging voor het praktijkgerichte onderzoek rondom het testen van robots is om te komen van losse unit tests tot meer overkoepelende tests, waarmee het gedrag van de robot kan worden getest. Met het leren van de robot functies, zoals aangegeven bij de architectuur, zie paragraaf 6.1, denken we dat het mogelijk wordt om software tests op het niveau van de functies te definiëren en te onderhouden. Op deze manier kan kennis over robot testen worden opgebouwd. Met dit inzicht over de verschillende testniveaus wordt het waarschijnlijk beter mogelijk om meer inzicht te krijgen in emergent behavior van robots, bijvoorbeeld als ze met elkaar samenwerken zoals beschreven in hoofdstuk 6.

Simuleren kan ook helpen om eerder inzicht in robotgedrag te krijgen. Eerder in dit hoofdstuk is al aangeven dat simuleren ook een manier is om eerder feedback te krijgen. Het lectoraat zoekt naar diverse methoden om robots te simuleren. Een voorbeeld is het gebruik van het software pakket Unity dat veel wordt toegepast in de game industrie. René Bakx, docentonderzoeker van het lectoraat HTES, heeft in het kader van het Let's Move IT-project een virtuele wereld in dit pakket gemaakt waarmee een robot in een virtuele (game) wereld rondrijdt. Er is ook een koppeling gemaakt met een mini robot die op de vloer, in de echte wereld, rondrijdt. De robot wordt via commando's door de simulator aangestuurd. De positie waar de robot zich werkelijk bevindt, wordt op zijn beurt weer bijgehouden door de simulator. Dit gebeurt aan de hand van een marker die zich op bovenkant van de robot bevindt en die wordt uitgelezen middels een camera aan het plafond. Op deze manier stuurt de simulator de robot aan. Er kan een pad in de simulator worden opgegeven dat wordt gevolgd in de echte wereld. In het onderzoek zijn wij bijvoorbeeld geïnteresseerd in de overeenkomsten en verschillen tussen de virtuele wereld in de simulator en het echte pad. Deze aanpak kan worden opgevat als een digital twin. Deze toepassing laat zien dat er met bestaande kant-en-klare technologieën, een goedkope simulator is te bouwen die breed inzetbaar is.

De uitdaging voor het praktijkgerichte onderzoek rondom simuleren van mobiele robots is de vraag wat wel en wat niet te mee te nemen bij het opstellen van de simulator. Dit is één van de vragen die speelt bij de gemeenschappelijke Robocup MSL simulator[53] (Yao e.a., 2015). In een simulatie van de voetbalrobot kan bijvoorbeeld de robot als object worden gezien, zonder de wielrotaties van het aantal wielen mee te nemen. De keuze van het abstractieniveau bepaalt tot op welk niveau een simulator wordt uitgewerkt. Hierbij raakt het thema simulatoren het thema modelleren, zie hoofdstuk 8.

Het lectoraat heeft vooralsnog geen concrete plannen voor de integratie en test thema's: acteren op risico en het runtime aanpassen van robots. Dit laatste is wel een interessante denkrichting om met samenwerkende robots in ongestructureerde omgeving om te kunnen gaan.

52. Deze tests zijn dan regressietests.

53. https://github.com/RoboCup-MSL/MSL-Simulator/wiki

7.2. INTEGRATIE EN TEST VAN SENSORNETWERKEN

Om een sensornetwerk volgens ontwerp op te bouwen moeten de nodes aan elkaar gekoppeld worden tot een netwerk. Het integreren van de nodes in een netwerk is in de praktijk vaak een uitdaging, omdat er combinaties van problemen optreden die alle moeten zijn opgelost om het netwerk te laten functioneren. Zo kunnen bij het ontwerp kleine fouten zijn gemaakt in het communicatie protocol die de data niet door het netwerk laten stromen. Als dit dan is opgelost kunnen weer andere problemen optreden, zoals elektrische voedingsproblemen in enkele van de nodes waardoor het netwerk alsnog niet werkt. Zo beschrijven Langendoen e.a. (2006) wat er allemaal mis kan gaan bij het opzetten van een sensornetwerk in een aardappelveld. Werner-Allen e.a. (2006) presenteren problemen bij metingen rondom een vulkaan. De lessen hiervan zijn dat inzicht nodig is in allerlei aspecten, zoals protocollen, hardware en mogelijke oorzaken van verstoringen, om een sensornetwerk goed te laten werken. Het onderzoek van het lectoraat rondom de integratie en test van sensornetwerken gaat over simuleren van netwerken, intelligente autonome nodes en deployment van software in netwerken.

Simuleren - Simuleren van een sensornetwerk is nodig om de werking van de verschillende nodes na te gaan. Het lectoraat wil ervaring opdoen met het uitvoeren van simulaties om aspecten als de betrouwbaarheid van een netwerk en het energieverbruik ervan te bepalen. Er zijn verschillende simulatoren voor sensornetwerken beschikbaar, zoals Cooja voor ContikiOS[54] en Tossim voor TinyOS[55]. Voor het simuleren kan naar diverse parameters worden gekeken, zoals: de gebruikte protocollen, het aantal nodes, de oppervlakte van het gebied dat wordt afgedekt door de nodes, capaciteit van de nodes (hoeveelheid data buffering). Simulaties helpen dan bij het opzetten van foutdiagnoses in de netwerken om zo in het uitgerolde netwerk snel de oorzaken van problemen te vinden. Dit is verbonden met modelleren, omdat we ons hier af moeten vragen welke wiskunde/modellen hiervoor te gebruiken is.

Intelligente autonome nodes - Bij de architectuur van sensornetwerken zijn autonome nodes in een netwerk geïntroduceerd, zie paragraaf 6.2. Dergelijke autonome nodes kunnen worden uitgebreid met functies om beter met veranderingen in het netwerk om te gaan, zodat netwerken zich run time kunnen aanpassen.

Daarbij kan men denken aan functies, zoals (Delicato e.a., 2014):

- Self-configuration - De nodes passen zich dan aan veranderingen in het netwerk aan, bijvoorbeeld wanneer er plotseling meer data worden gegenereerd.
- Self-healing - Het ontdekken van ontwrichtingen van het netwerk en deze herstellen. De nodes beschikken dan over ‘policies’ om op de ontwrichtingen te reageren.
- Self-protection - Het detecteren en beschermen van het netwerk tegen bedreigingen op node niveau. Dit is nodig om het netwerk te beveiligen tegen cyber aanvallen.

Deployment van software - Als een netwerk veel nodes heeft dan is het belangrijk dat op elke node de juiste software staat en dat er eenvoudig updates gemaakt kunnen worden. Deployment is de denkwijze om er voor te zorgen dat de juiste software versies en onderdelen op de juiste hardware versies en onderdelen draaien. De vraag die het lectoraat zich hier stelt is hoe deployment van software goed geregeld te krijgen bij ontwikkelaars en daarmee de hele ontwikkelorganisatie op het zelfde professionele niveau te krijgen.

54. https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja
55. http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM

8. MODELLEREN EN LEREN – KADER, ROBOTS EN SENSORNETWERKEN

> We bestuderen en ontwerpen systemen door er modellen van te maken. Modellen helpen ons om uitspraken te doen over het systeem dat we bestuderen of ontwerpen.
>
> – METHODOLOGIE VAN INFORMATIE-SYSTEEMONTWIKKELING (LEMMEN E.A., 1993).

In dit hoofdstuk wordt het derde aspect van het kader voor systeemdenken uitgewerkt. Eerst wordt het begrip model uitgelegd, waarna vier manieren om te modelleren en te leren aan de orde komen. Vervolgens wordt ingegaan op de betekenis van modelleren en leren voor mobiele robots en sensornetwerken.

Abstraheren, één van de twee richtlijnen die we in hoofdstuk 5 introduceerden om met de complexiteit van Cyber Physical Systems om te gaan, is de aanleiding voor modelleren. Een model beschrijft een systeem op een meer eenvoudige wijze, om zo een beter begrip van bepaalde aspecten ervan te krijgen. Zo krijgen FHICT-studenten in het eerste semester de opdracht om voordat ze gaan programmeren een stroom-schema van hun programma te maken. Het doel hiervan is om voordat de programma code wordt gemaakt eerst de beslissingen van het programma op te schrijven, zodat duidelijk is welke software constructies, zoals een while- of for-loop, te gebruiken.

Hieronder behandelen we het begrip model eerst uitgebreider, door in te gaan op de vraag ‘wat is een model?’ Dit doen we aan de hand van twee dimensies, namelijk de mate van detaillering (abstractieniveau) van het model, en de aspecten van het systeem dat het model beschrijft.

Het *(abstractie)niveau* van een model gaat over de mate waarin details van het systeem beschreven worden. Als er minder details in een model zijn opgenomen, dan is het eenvoudiger om het systeem globaal te beschrijven en is de scope van de systeembeschrijving breder. Er kan zo een algemeen systeemplaatje worden opgesteld zonder dat men zich over details hoeft te bekommeren.

Het gekozen abstractie niveau bepaalt ook de uitdrukkingskracht (expressiviteit) van het model. Het conceptuele model is vaak een schets, bij wijze van spreken de schets op de achterkant van het bierviltje. Modellen op implementatieniveau hebben daarentegen over het algemeen een sterkere uitdrukkingskracht, omdat ze vaak meer formeel genoteerd kunnen worden. We spreken dan over berekenbare modellen: models of computation[56]. Hiervoor zijn verschillende formalismes te gebruiken, zoals eindige toestandsautomaten (finite state machines), waarmee aan de hand van een eindig aantal toestanden, overgangen daartussen en acties van het systeem wordt beschreven. (Groote & Mousavi, 2014), (Theelen e.a., 2011).

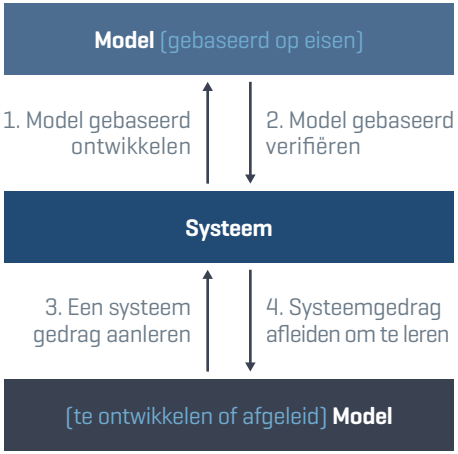
Aspecten van het model gaan over de manier waarop we het systeem modelleren[57]. We kunnen een model bijvoorbeeld definiëren door naar de structuur of juist naar het gedrag van het systeem te kijken.

Aspecten kunnen onderverdeeld worden aan de hand van de volgende dimensies:

- Werking systeem en daarbij naar de binnenkant of de buitenkant/de interacties: whitebox versus blackbox.
- Afhankelijkheden in tijd en daarbij wel of niet naar het tijdsverloop kijken: statisch versus dynamisch.
- Afhankelijkheden van tijd of gebeurtenis en daarbij naar opeenvolgende tijdstippen en gebeurtenissen (discreet) of gebeurtenissen van te voren in een tijdsverloop (continu) meenemen.
- Waarschijnlijkheden en daarbij de kansen op verschillende toestanden niet (deterministisch) of juist wel (probabilistisch) meenemen.

Voor de robots en sensornetwerken waar het lectoraat mee werkt zullen de modellen vooral dynamisch, discreet zowel als continu en probabilistisch zijn. De verschillende modellen onderzoekt het lectoraat bijvoorbeeld in het ToLeRo-project, zie paragraaf 10.1.

Om met modelleren te beginnen zal het startpunt vaak zijn om de diverse toestanden in het systeem te onderscheiden, bijvoorbeeld in een mobiele robot: `motor_aan`, `motor_uit`. Gebeurtenissen (events) in een systeem in de systeemomgeving veranderen de systeemtoestanden. Hierbij kan de nieuwe toestand weer aanleiding zijn voor opeenvolgende gebeurtenissen met veranderingen tot gevolg. Een voorbeeld is de toestand `motor_aan` die de robot doet rijden waardoor er informatie over een naderend obstakel binnenkomt die de toestand `versnellen_motor` oproept. Modellen kunnen op vier verschillende manieren worden gebruikt, namelijk om: modelgebaseerd te ontwikkelen, modelgebaseerd te verifiëren, een systeem gedrag aan te leren en om gedrag van een systeem af te leiden om te leren. Modellen vervullen daarbij verschillende rollen voor architectuur en integratie en test. Figuur 11 vat deze vier toepassingen van modellen aan de hand van pijlen samen.



FIGUUR 11 - VIER MANIEREN OM MODELLEN TE GEBRUIKEN; MODELLEREN EN LEREN.

Deze pijlen hebben betrekking op het modelleren of leren: de situatie dat een model van te voren is opgesteld (modelleren) of dat het model tijdens gebruik wordt ontwikkeld of afgeleid (leren).

De vier toepassingen van modellen, oftewel het modelleren en leren worden hieronder uitgewerkt.
1. *Modelgebaseerd ontwikkelen* is een werkwijze om software te ontwikkelen. Model Driven Architecture (MDA)[58] en Model Driven Software Engineering (MDSE) zijn benaderingen binnen de software engineering die eerst een model van het systeem, het gedrag, of onderdelen ervan, opstellen om er vervolgens code uit te genereren. Hiervoor is het van belang om de juiste modelleringstaal en – concepten te gebruiken (Van Gool e.a., 2006). Zo kunnen we een software programma zien als een reeks toestandsovergangen, dat voldoet aan regels (policies), die geprogrammeerd worden in de software. Hierbij kunnen twee vragen worden gesteld die onderzoekers in het verleden hebben bekeken en die nog regelmatig aan de orde komen:

Vraag 1: In welke taal wordt het systeem gemodelleerd?
Vraag 2: Kan het hele systeem met een model beschreven worden?

56. Het Ptolemy-project past verschillende computational models toe in Cyber Physical Systems: https://ptolemy.eecs.berkeley.edu/
57. Denk aan de Dreamplanes van C.W. Miller (Nicolai en Carichner (2010).
58. http://www.omg.org/mda/

De eerste vraag gaat onder meer over de vraag of het systeem gemodelleerd kan worden door een algemene taal te gebruiken of dat een taal speciaal voor het systeem wordt gedefinieerd. Het gaat daarbij om de eerder genoemde uitdrukkingskracht (expressiviteit) van de modelleringstaal, die vaak van invloed is op de ondersteuning door tools. Over het algemeen geldt hoe expressiever de taal, des te beter deze door een tool ondersteund kan worden. Deze vraag zien we terug in de discussie over het gebruik van algemene talen, zoals UML of SysML, en het eventueel maken van aanpassingen op de taal (van Gool, e.a., 2006), versus het gebruik van een Domain Specific Language (DSL) die speciaal voor een machine (een domein) wordt opgesteld.

De tweede vraag is of het nuttig is om het hele systeem met een model te beschrijven. Of kan het systeem, of delen ervan, beter met verschillende modellen worden beschreven? Er zijn dan verschillende modellen, waarvan de resultaten invloed op elkaar hebben. Het is dan belangrijk na te gaan hoe de modellen samenhangen en of er bijvoorbeeld modeltransformaties nodig zijn, zie bijvoorbeeld (Moneva e.a., 2010) en (Hendriks e.a., 2011). Bij Philips Healthcare, en andere bedrijven, zijn modelleerbenaderingen toegepast, waarbij de component interfaces precies worden beschreven. Er is dus op systeemniveau een complete modelbeschrijving, maar het onderliggende niveau, de componenten worden niet alle gemodelleerd. Het wordt dan mogelijk om sommige componenten wel te modelleren, en sommige juist niet (Kurtev e.a., 2017).

2. *Modelgebaseerd verifiëren* is een geavanceerde manier om het te ontwikkelen systeem, of onderdelen ervan, meteen (bewijsbaar) correct te beschrijven en de werking van het systeem aan de hand van eigenschappen als ontbreken van deadlock meteen te bewijzen. Voor dynamische systemen wordt in de industrie en het onderwijs regelmatig Matlab/Simulink gebruikt voor het continue controle domein. Verder wordt door verschillende bedrijven in de regio Eindhoven het Dezyne tool van Verum gebruikt. Deze tool helpt bij het correct beschrijven van dynamische systemen in het discrete domein, zodat de controle aansturing van systemen kan worden gedefinieerd en bewezen (Osaiweran e.a., 2016).

In deze benadering wordt het embedded systeem uitgedrukt als een communicerend systeem waarin parallel processen worden uitgevoerd (Groote en Mousavi, 2014). Het systeemgedrag wordt beschreven als een verzameling toestanden en een verzameling toestandsovergangen (transities), waarbij men in de praktijk vaak in activiteiten in plaats van states denkt. Hieraan worden acties gekoppeld die de (systeem)toestanden met elkaar verbinden. Over het proces van toestandsovergangen kan vervolgens worden geredeneerd, of elke toestand bereikbaar (reachable) is, of dat een toestand niet definitief beëindigt, d.w.z. dat het systeem in een zogenaamde deadlock brengt. De proces algebra mCRL[59] (Moussavi en Groote, 2014) wordt tegenwoordig toegepast in Dezyne, zodat de correctheid van het gemaakte ontwerp bewezen kan worden. Een uitdaging van het toepassen van formele methodes is de praktische berekenbaarheid. Met de omvang van de software neemt ook de grootte van de zogenaamde toestandsruimte toe, met als gevolg een state space explosion[60]. Het lectoraat past bijvoorbeeld het Dezyne tool regelmatig toe in haar projecten om zo meer te weten te komen hoe systemen te verifiëren.

3. *Een systeem gedrag aanleren* - Met de toenemende complexiteit van systemen is het aantrekkelijk om een systeem gewenst gedrag aan te leren in plaats van het gedrag van tevoren volledig in het systeem vast te leggen. Vaak kennen ontwikkelaars niet alle parameters die het systeemgedrag beïnvloeden. In de praktijk leren lijkt een goed alternatief (Riedmiller, 2016). Daarom wil het lectoraat in het verlengde van de modelleren, kijken naar leren, als alternatief voor modelleren. De regels voor het uitvoeren van processen worden dan door het systeem aangeleerd. Hiervoor kijken we naar machine learning, dat een onderdeel is van de kunstmatige intelligentie (Stone e.a., 2016) en daarbinnen op het zogenaamde reinforcement learning, een onderdeel van Machine Learning (Jordan en Mitchell, 2015).

4. *Systeemgedrag afleiden* - Ook kan een bestaand systeem worden bestudeerd en kan daar systeemgedrag uit worden afgeleid om daar vervolgens over te leren. Het gaat dan om leren, door systeemgedrag af te leiden en dat vervolgens te modelleren en/of te testen. Degenen die het systeem observeren zijn dan de leerlingen. Dit kan een software engineer zijn die onderhoud pleegt aan het systeem of er nieuwe onderdelen aan toe gaat voegen. Het systeem vatten we daarbij op als een black box. Door deze black box van inputs te voorzien en vervolgens de outputs te registreren kan het gedrag van het systeem worden achterhaald. Dit gedrag wordt dan beschreven in een model. Het systeem is daarmee ‘de leraar die het model als leerling leert’ (Vaandrager, 2017) Deze benadering is mogelijk voor bestaande systemen die gewenst

59. http://www.mcrl2.org/web/user_manual/showcases/Patient_support_platform.html
60. Dit betekent dat vaak maar delen van een compleet systeem kunnen worden gemodelleerd. Het beperkt de bruikbaarheid van formele technieken in de industrie (Parnas, 2010). Daartegenover staan meer positieve geluiden (Oseiweran e.a., 2016).

systeemgedrag vertonen. Immers alleen dan loont het zich om model af te leiden, de model inference. Deze benadering kan gebruikt worden om in complexe systemen, waarvan het overzicht verloren is gegaan, het overzicht terug te krijgen, en zo te 'reverse engineeren' en te 'refactoren'. Het lectoraat wil onderzoeken of een dergelijke aanpak nuttig kan zijn om met emergent behavior van mobiele robots om te gaan.

Modelbased testing - Is een vijfde manier om modellen toe te passen. Deze aanpak is niet hiervoor en niet in figuur 11 weergegeven. In deze aanpak ontwikkelt men parallel aan eisen voor het systeem een apart model om het systeem te testen (Tretmans, 2008). De ontwikkelingen binnen deze interessante invalshoek volgen we de komende jaren, maar laten we voor onze onderzoeksprojecten voorlopig buiten beschouwing.

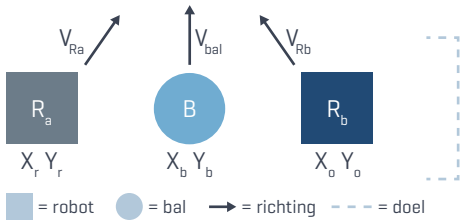
Samenvattend stellen we dat modellen een belangrijke rol spelen voor het uitwerken van de architectuur en integratie en test omdat zij antwoorden geven op de vragen over de structuur van de decompositie en daarmee over de architectuur en integratie. Modellen voor embedded software beschrijven het systeemgedrag in gebeurtenissen en toestanden. Er zijn diverse typen modellen mogelijk op basis van abstractieniveaus en modelleringsaspecten. De centrale vragen die het lectoraat zich hier stelt zijn:

- Welke modellen (en ondersteunende tooling) zijn nodig om embedded systemen, zoals robots en sensornetwerken, te modelleren?
- Hoe kan robot software worden gemodelleerd zodat robots gedrag kunnen leren, zodat zij samen een opdracht kunnen uitvoeren?

Wetenschappelijke onderzoek heeft diverse tools, ideeën en toepassingen voor het modelleren en verifiëren van systemen opgeleverd. De impact van het modelleren van systemen op het hbo-ict-onderwijs is groot. De uitdaging voor het hbo is hoe hierop aan te sluiten en een eigen interpretatie (meerwaarde) te geven.

8.1. MODELLEREN EN LEREN BIJ ROBOTS

Het modelleren van robots is gericht op robotgedrag, om zo robots in ongestructureerde situaties te laten (samen) werken. Robots vertonen gedrag waarbij ze bijvoorbeeld aan de lopende band lassen of in het geval van robotvoetbal, snel op een bal toerijden en samenspelen om een goal te maken. De standaard aanpak om deze systemen aan te sturen is door hier vooraf één of meerderde modellen voor op te stellen. Dit gebeurt vaak met behulp van (eindige) toestandsmachines waarbij voor elke situatie een actie is gedefinieerd. De robot 'weet' dus vooraf welke acties, wanneer worden genomen. Maar wat doen als we de omgeving nog niet van te voren kennen, of als we flexibel gedrag willen aanbrengen?



FIGUUR 12 - SITUATIE IN HET VOETBAL-ROBOT WAAR HET NUTTIG VOOR IS EEN MODEL TE MAKEN.

Een voorbeeld van het nut van een robotmodel is de volgende situatie in het robotvoetbal, zie figuur 12: robot R_A rijdt op de voetbal af die voor het doel ligt. Om een bal in een doel te kunnen scoren zijn er een aantal vragen waarvoor we een model willen gebruiken om antwoorden te vinden:

- Waar is de robot? Wat zijn de coördinaten (x_r y_r) van de robot op tijdstip t?
- Hoever ligt de bal voor het doel? Wat zijn de coördinaten (x_b y_b) van de bal?
- Met welke snelheid (v_r) komt de robot op de bal afrijden?
- Kan de tegenstander Robot R_B de bal tegenhouden?

Er zijn dus verschillende parameters in een model die we mee kunnen nemen voor een goede beschrijving van deze situatie in het voetbalspel. Parameters zijn onder andere: positie van robot, bal, doel en tegenstander.

Het model dat voor deze situatie nuttig is beschrijft de diverse toestanden: er is sprake van verschillende objecten, met versnellingen en vertragingen.

Onze robot kan meerdere mede- en tegenstanders in het spel tegenkomen die elk richting en snelheid hebben. In het model zouden we dit met vectoren uit willen drukken.

Voor voetbalrobots zullen we over het algemeen het *dynamische gedrag* willen modelleren: hoe de robots reageren op gebeurtenissen in hun omgeving. Met het model bekijken we dan hoe specifieke robots op acties reageren. Het gedrag kunnen we dan vastleggen in een zgn. state machine. Het gedrag in deze state machine kan formeel worden genoteerd en vervolgens geverifieerd. Dit wordt uitgevoerd als onderdeel van project Toestanden en Leren voor Robots (ToLeRo), gefinancierd door Centre of Expertise HTSM. Hierbij kijken we naast de state machine, ook naar behavior trees en reinforcement learning. Het inzicht dat we hier mee hopen op te doen helpt om spelgedrag van voetbalrobots te definiëren. We verwachten dat dit inzicht weer nuttig is om planning van AGVs te onderzoeken en omgekeerd[61].

De omgeving van een robot kan volledig bepaald zijn als een volgorde van opeenvolgende toestanden. Echter, de toestanden waar robots mee te maken krijgen zijn gebaseerd op de gegevens die zij opvangen met hun sensoren. Deze sensorwaardes hebben een bepaalde mate van precisie waarover er een mate van vertrouwen (confidence) bestaat, zie ook paragraaf 6.2. Zo hebben infrarood (IR) sensoren een brede variëteit van waardes die zij terug geven ten opzichte van bijvoorbeeld de lasersensor. Het geloof over de juistheid van de sensorwaarden (belief) wordt daarmee door de sensoren en de omgeving waarin ze opereren bepaald. Door dergelijke karakteristieken van sensoren mee te nemen in modellen, zullen waarschijnlijkheden worden meegenomen. Hierbij kunnen sensorwaardes bijvoorbeeld ook met elkaar gemengd kunnen worden om tot een meer betrouwbare resultaten te komen. Ook is veelal filtering van meetwaarden nodig om afwijkingen door ruis te elimineren. Het lectoraat richt zich op het bruikbaar maken van bestaande filtermethoden, zoals Kalman, Particle filtering (Kang e.a., 2018) voor het modelleren van robot (posities).

Leren, reinforcement learning - De modellen die we tot nu bespraken, modelleren robotacties op basis van de toestanden waarin de robot zich bevindt. Het robotgedrag komt voort uit een gedragslijn (policy) die van te voren wordt vastgelegd en in de robot software wordt geprogrammeerd. De gedragslijnen en hun bruikbaarheid worden in een state machine vastgelegd. Dit is echter bepaald door de zienswijze en kennis die er is op het moment van het maken van het model. Hoe de robot zich in de echte omgeving moet en zal gedragen kan weer anders zijn. Een meer flexibele benadering is als de robots hun gedrag wordt geleerd terwijl ze werken en ze hun gedrag te laten leren. Hiervoor wil het lectoraat onderzoek doen met het zogenaamde reinforcement learning (Sutton en Barto, 2012).

In het robotvoetbal bestaat veel interesse voor machine learning (Riedmiller, 2016).

De robot wordt hierbij opgevat als een rationele agent, die te vergelijken is met de arbiter, die in paragaaf 6.1 werd geïntroduceerd. De arbiter zal dus continu keuzes maken in het gedrag dat hij wil uitvoeren. Hij streeft daarbij naar de optimale bruikbaarheid van de gedragslijnen. Door de robot per actie selectie een beloning (reward) te geven kan gewenst gedrag worden aangeleerd.

61. Bij hogere niveaus modelleren past de nuancering dat we bijvoorbeeld planning en scheduling van robot taken, zoals in een robot fleetmanagement systeem gebeurt, niet willen modelleren. Planning heeft eigen systematiek die buiten de robots plaatsvindt. Het gaat ons om modelleren van gedrag in autonome robots.

Het Cambada Robocup team heeft bijvoorbeeld de bruikbaarheid in het robotvoetbal laten zien door een robot sneller de voetbal te laten onderscheppen (Cunha e.a, 2015). Daardoor hoefde geen complexe regeling te worden gebouwd. Een ander team zoekt een manier om het schieten op het doel met machine learning uit te voeren.

Machine learning is een interessante aanvulling op het modelleren als er sprake is van ongestructureerde situaties. Het maakt het dan mogelijk om in dergelijke situaties een optimaal bruikbare gedragsregel te selecteren waar we met modelleren nog niet voldoende weten om te kunnen beslissen. Reinforcement learning kost over het algemeen ook meer (reken)kracht, er zijn dus ook nadelen aan verbonden: "there is no free lunch". Deze afweging zal gemaakt moeten worden bij keuze tussen modelleren en machine learning. Vanuit het lectoraat werken Eric Dortmans, Peter Lambooij en Suzana Andova aan dit thema via het RIF Robocup-team en het ToLeRo-project.

8.2. MODELLEREN EN LEREN BIJ SENSORNETWERKEN

Het modelleren van sensornetwerken gaat over vragen rondom de data doorvoer en het in stand houden en uitbreiden van netwerken.

KADER I GEDRAGSREGELS

Een goede gedragslijn is optimaal bruikbaar voor de robot. De bruikbaarheid [utility][62] is een belangrijk criterium en optimale robots maxi-maliseren de bruikbaarheid van hun gedragslijnen. De gedragsregel is dan een combinatie van actie [A], de kans [P] op de uitkomsten [O] daarop. De agent berekent de bruikbaarheid [U] hiervan aan de hand van een zogenaamde utility-functie. De robot leert zijn gedragsregels in de praktijk, en haalt zijn data uit de sensoren of maakt daarvoor inschattingen. Uitgaande van de toestand waarin de robot zich bevindt zal de robot acties gaan kiezen. Via de Q-functie bepaalt de robot de optimale actie voor dat moment , aan de hand van geobserveerde toestand [state]. De robot maakt dan gebruik van het Belief Desire Intent [BDI] principe [Burkhard, 1998].

$$EU[A] = \sum_{o \in O} P_A[o]U[o]$$

- A: Actie – dit een optie, een mogelijk te nemen actie
- O: de uitkomsten van de actie
- $P_A[o]$: de [voorwaardelijke] kans [probability] van de uitkomst O.
- U[o]: de bruikbaarheid [utility] van de uitkomst.

62. https://plato.stanford.edu/entries/rationality-normative-utility/

Er zijn dan ook verschillende modellen te bedenken die nuttig kunnen worden toegepast om (aspecten) van sensornetwerken te beschrijven. Een voorbeeld is het berekenen van de verwachte performance van het netwerk. Dit gaat over de snelheid waarmee sensordata door het netwerk worden verspreid. Het model wordt dan bijvoorbeeld gebruikt om middels simuleren de mogelijke bottlenecks in de doorvoer te achterhalen. Andere toepassingen van het model zijn dan bijvoorbeeld om de invloed van een specifiek protocol, bijvoorbeeld MQTT, of het aantal nodes in het netwerk op de data doorvoer te achterhalen.

Modellen voor sensornetwerken zullen vaak probabilistisch van aard zijn omdat er vaak met onzekerheid moet worden omgegaan. De onzekerheid komt door de sensormetingen en de beslissingen die op basis van de sensor data moeten worden genomen. Voor het nemen van beslissingen in onzekere omstandigheden worden vaak Markov-beslismodellen gebruikt (Borst e.a., 2015).

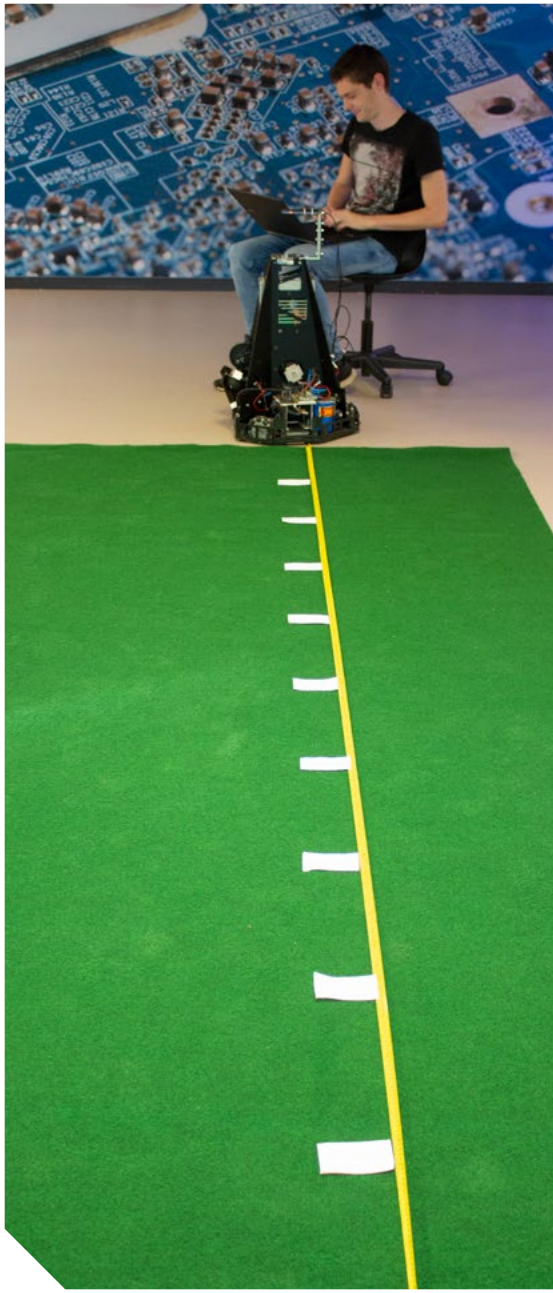
Als we het voorgaande uitwerken en een sensornetwerk als model bekijken, dan kunnen we de nodes en hun verbindingen opvatten als een (wiskundige) graaf[63]. Deze graaf bestaat dan uit N sensornodes die elk data produceren. Omdat de nodes elkaar onderling kunnen verstoren via tegenstelde datapakketjes zullen in het model afspraken gemaakt worden over de onderlinge communicatie tussen nodes, bijvoorbeeld of de nodes gelijktijdig data naar elkaar mogen versturen of niet. Dit wordt vastgelegd in de algoritmes van het netwerk.

Er kunnen ook waarschijnlijkheden aan de nodes worden toegevoegd. Zo heeft het verzenden van data door een node een bepaalde kans, bijvoorbeeld 20%. Deze kansen zijn gekoppeld aan andere gebeurtenissen in het netwerk, bijvoorbeeld dat de buur nodes op dat moment niet actief is. Op deze manier ontstaat een stochastisch netwerk[64] dat bestaat uit nodes die door de tijd afwisselend actief en inactief zijn. Er zijn diverse manieren waarop over kansen kan worden geredeneerd. Per sensornode *i* wordt dan op tijdstip *t* de toestand *Si(t)* aangegeven, of deze inactief (0) of actief (1) is. Zo wordt een Markov keten *S(t) = (S1(t), . . . ,SN(t))* gedefinieerd die de toestand van het netwerk beschrijft.

Vervolgens kan er over het netwerkgedrag worden geredeneerd door eigenschappen aan het netwerk toe te kennen.
In het geval van de eerder genoemde performance van het netwerk willen we de data doorvoer achterhalen. Als bijvoorbeeld uit de analyse van de Markov keten blijkt dat nodes in het midden van het netwerk een lagere doorvoer hebben dan de nodes aan de rand van het netwerk, dan zouden we de nodes in het midden vaker kunnen activeren dan de buitenste nodes om de doorvoer van de data stroom constant te houden (Borst e.a., 2015).

Er is veel wetenschappelijk onderzoek gedaan naar het ontwikkelen van dergelijke stochastische en Markov modellen voor sensornetwerken (Baldwin e.a., 2004), (Borst e.a, 2015), (Kabashkin en Kundler, 2017). Het lectoraat richt zich op het bruikbaar maken van deze inzichten in modellen voor concrete sensornetwerken. Behalve doorvoer van data kunnen modellen dan mogelijk ook inzicht te geven over energieverbruik en beveiliging.

Behalve het hele netwerk van nodes als onderwerp van modellering te nemen kan het ook nuttig zijn modellen op node niveau te maken. Er zijn dan verschillende aspecten, waarvan triggering van de nodes er een is. Immers met energieverbruik in het achterhoofd is het logisch om de nodes uit te zetten. Echter, we willen ook dat nodes op tijd hun metingen uitvoeren. Ze moeten dus op tijd actief worden. Een model op node niveau kan helpen om dergelijk gedrag te modelleren.



63. http://www.hhofstede.nl/modules/grafen.htm
64. https://nl.wikipedia.org/wiki/Stochastisch_proces

9.
SOFTWARE KWALITEIT – KADER, ROBOTS EN SENSORNETWERKEN

> "Software kwaliteit is gratis, maar alleen voor diegenen die er voor willen betalen."
>
> – TOM DE MARCO (MARCO & LISTER, 1987)

In dit hoofdstuk wordt het vierde en laatste aspect van het kader voor systeemdenken uitgewerkt. Software kwaliteit wordt geïllustreerd aan de hand van attributen, waarbij duurzaamheid als (super)eigenschap aan de orde komt. Vervolgens wordt ingegaan op de betekenis van software kwaliteit voor mobiele robots en sensornetwerken.

Het resultaat van ontwerpen aan de hand van architectuur, integratie en test en het modelleren zal een embedded systeem zijn, met daarin embedded software. Dit dient van voldoende kwaliteit te zijn, of meer precies: het heeft een bepaalde kwaliteit, waarbij kwaliteit verbonden is met geld en tijd. Als meer tijd en geld beschikbaar is dan kan hier een product van hogere kwaliteit uit ontstaan. Het goed beschrijven van en het bepalen wat systeem kwaliteit is nog steeds een uitdaging. Kwaliteit wordt vaak immers bepaald door de klant of de stakeholders van het systeem: 'quality is in the eyes of its beholders', het is niet eenduidig. Het wordt daarom door techneuten nogal eens afgedaan als vaag en overgelaten aan project managers.

Aandacht voor software kwaliteit is belangrijk om te kunnen bepalen of embedded systemen doen wat ze moeten doen. Kwaliteit wordt bepaald door de eisen van de klant en wordt vastgelegd door zogenaamde functionele eisen op te stellen. Daarnaast spelen zogenaamde niet-functionele eisen een rol in embedded software.

De ISO 25010 standaard is opgesteld om software kwaliteit te definiëren als een verzameling van 8 karakteristieken: functionele correctheid (geschiktheid), prestatie-efficiëntie, uitwisselbaarheid, bruikbaarheid, betrouwbaarheid, beveiligbaarheid, onderhoudbaarheid en overdraagbaarheid (ISO 25010).

De ISO standaard is een woordenboek voor het beschrijven van software kwaliteit. Om de kwaliteit te bepalen moet deze worden ingeschat bijvoorbeeld via een audit of wordt deze meetbaar gemaakt (Punter e.a., 2004), (Wojcik e.a., 2007). In de wereld van de administratieve en financiële systemen is al veel gedaan aan het meetbaar maken van software kwaliteit. Een voorbeeld is de Maintainability Index die door de Software Improvement Group[65] is opgesteld en organisaties inzicht geeft in de mate waarin hun software onderhoudbaar is. In de verhalen over software ontwikkeling in de High Tech industrie komt beschrijven en vaststellen van software kwaliteit vaak minder aan de orde. Dit betekent niet dat software kwaliteit minder belangrijk is in de High Tech industrie. Integendeel, enkele bedrijven hebben benaderingen om software kwaliteit te meten, zoals Philips Healthcare. Hierbij wordt bijvoorbeeld de complexiteit van de code en de

KADER J
SOFTWARE DOPING

Een speciale invulling van duurzaamheid is software doping omdat het verbonden is met het verbeteren van prestaties, maar soms ook negatief doorwerkt op duurzaamheid. Een software systeem wordt hierbij opgevat als 'gedoped' als de producent verborgen functies heeft ingebouwd die het systeem zo beïnvloeden dat de belangen van maatschappij en/of software gebruiker geschaad worden (Barthe e.a., 2015). Een voorbeeld van software doping zijn printers die bij het gebruik van niet fabrikant cartridges eerder dan nodig melden dat de inkt op is, waarmee printergebruiker 'gedwongen' worden de fabrikant eigen cartridges te gebruiken. Een ander voorbeeld zijn laptops die niet opladen als andere opladers dan die van de fabrikant worden gebruikt. In deze voorbeelden is de extra software functionaliteit in het belang van de producent, maar niet dat van de gebruiker (Barthe e.a., 2015).

Een voorbeeld waar software doping in het belang van de gebruiker is, maar niet dat van de maatschappij is chip tuning in een auto. De software van de electronic control unit wordt dan aangepast om bijvoorbeeld sneller op te kunnen trekken.[73] Dit is immers strijdig met het maatschappelijk belang van beperkte snelheden in het verkeer (Barthe e.a., 2015). Het diesel emissie schandaal van Volkswagen en andere auto leveranciers dat sinds 2015 speelt, is een voorbeeld van software doping waar zeker het belang van de maatschappij, maar ook dat van de auto eigenaar wordt geraakt. Immers. de sjoemel software maakt het eindproduct auto minder duurzaam. Hier heeft de wens van autoproducenten om hun afnemers minder Ad Blue toe te hoeven laten voegen, om de NOx uitstoot te reduceren, er toe geleid dat de software is gemanipuleerd. Dit is zo gedaan dat de NOx uitstoot niet verhoogd werd in bepaalde testopstellingen. Bij het dagelijks gebruik van de auto is de NOx uitstoot weer verhoogd (Barthe e.a., 2015) en bedraagt de uitstoot van schadelijke stoffen soms veertig keer de toegestane hoeveelheid.

Doping verwijst naar de prestatieverbetering in de (duur)sport. Waar van oudsher druivensuiker en vitamines een rol spelen in het dieet van sporters, om beter te kunnen presteren. Dopinggevallen in de wielersport maakten het afgelopen decennium grensoverschrijdend gedrag in de sport duidelijk. Uit de bekentenissen van wielrenners over hun dopinggebruik die de afgelopen jaren naar boven zijn gekomen, weten we ook dat doping breed werd toegepast en dat er een druk onder wielrenners was om doping te gebruiken omdat anderen het ook deden. Dit raakt de ethiek, de discussie over goed en kwaad. De analogie kan ons helpen om over software doping na te denken en er standpunten in te vinden om een ethiek voor software engineers mee op te stellen.

65. https://www.sig.eu/insight/practical-model-measuring-maintainability/

afhankelijkheden tussen code modules vastgesteld[66]. Ook wordt meegenomen of programmeerregels zijn gevolgd en of de kwaliteit van de documentatie goed is.

Het lectoraat richt zich in het onderzoek rondom kwaliteit op het beschrijven en aantonen van software kwaliteit in mobiele robots en sensornetwerken. Hierbij gaat het dan om het opdoen van ervaringen met methoden en tools die vaak in de high tech al worden toegepast, maar die we op het gebied van robots en netwerken nog niet zien. Voorbeelden van tools waar we mee werken:

- Statische en dynamische analyse, zoals: Polyspace[67] en Tics[68].
- Software reviews, zoals Code collaborator[69].
- Modelgebaseerd ontwikkelen, zoals Dezyne[70].
- Simulaties voor analyse systeem gedrag, zoals Software port[71] en Chronix[72].

Verder wil het lectoraat nadenken over en werken aan het realiseren van duurzame software. Omdat software steeds bepalender wordt voor systemen wordt de invloed ervan op mens en maatschappij steeds belangrijker. Duurzame software is software die correct functioneert met een relatief laag energieverbruik (Van Gastel, 2016). Correct functionerende software voldoet aan de hiervoor genoemde kwaliteitseisen en het is daarmee bijvoorbeeld goed te onderhouden en eenvoudig porteerbaar. Van Gastel (2016) geeft aan dat correctheid van software een noodzakelijke voorwaarde voor duurzame software is.

Energieverbruik is een logisch aspect van duurzaamheid. Embedded software draait immers op hardware en naarmate er meer devices komen zal het energieverbruik van de apparaten toenemen. Voorheen werd vooral aan de hardware zijde het energieverbruik verbeterd. Echter door de invloed van software op het gedrag van sensornetwerken is het steeds belangrijker om naar software te kijken om energie te besparen (Van Gastel, 2016).

9.1. KWALITEIT VAN MOBIELE ROBOTS

Het beschrijven van software kwaliteit wordt ook belangrijk voor het maken van software voor robots en sensornetwerken.

Afhankelijk van hoe goed de sensoren en actuatoren werken en hoe goed de taken en functies zijn uitgevoerd in de software zal een robot een bepaalde systeem kwaliteit hebben. Deze kwaliteit of systeem performance (System KPI's) bepaalt hoe goed het systeem is en dat bepaalt vervolgens hoe tevreden de belanghebbenden en klanten over de robot zijn. Als we kijken naar de voetbalrobot dan bepaalt de mate waarin de robots doelpunten maken de systeem kwaliteit. Een indicator van deze kwaliteit is bijvoorbeeld balbezit, net als in het mensen voetbal. Het probleem van indicatoren is vaak sub-optimalisatie: het balbezit is een indicator, maar uiteindelijk moet de robot de bal ook afspelen! Andere robots moeten voldoen aan weer andere systeem kwaliteit. Zo zal een zorgrobot worden afgerekend op de hoeveelheid zorgacties die het ondersteunt.

Systeem kwaliteit wordt daarmee bepaald door hoe goed de functies van de robot het doen. Dit illustreert het belang van het begrijpen van de robot functies, zoals aangegeven in paragraaf 6.1.
Voor robots kijkt het lectoraat vooralsnog naar de volgende kwaliteitsattributen:

- Betrouwbaarheid (Reliability, Robustness) - Robots die functioneren onder diverse omstandigheden en op tijd de benodigde data sturen en acties opstarten. Een voorbeeld is de betrouwbaarheid van de hardware abstractie laag in het robotvoetbal, die sterk het succes van de bovenliggende architectuur bepaalt.
- Veiligheid (Safety) - Hoe kunnen mens en robot goed met elkaar samenwerken? Dit is onderwerp in het Let's Move IT-project waar robots/AGV's in een warehouse rondrijden en niet dusdanig onverwacht gedrag mogen vertonen waarmee ze mensen beschadigen.
- Performance - Hoe bewegen de robots/AGV's zich in een warehouse, productomgeving? En hoe moeten ze daarvoor aangestuurd worden? Het gaat dan om de doorlooptijd (throughput) van de robot op de vloer. Dit is onderwerp van onderzoek in het Let's Move IT-project.
- Beveiliging (Security) - De beveiliging en privacybescherming worden in software robots ingebouwd.

We gaan hier kort in op Beveiliging. Vragen zoals: Wat kunnen ze zien? Kunnen software robots onze gebruiksgeschiedenis beoordelen? Hebben ze toegang tot onze vertrouwelijke informatie, en hoe moet dat worden aangepakt? zijn onderwerpen voor robot security. Robot security is onderwerp van het KIEM Cyber Security van Robots in de Smart Industry-project, dat het lectoraat vanaf september 2018 uitvoert. Het betreft exploratief onderzoek rondom de cyber security bedreigingen van

66. Hierbij wordt de tooling van Tiobe gebruikt. https://www.tiobe.com/
67. https://www.mathworks.com/products/polyspace-code-prover.html
68. https://www.tiobe.com/tics/fact-sheet/
69. https://smartbear.com/product/collaborator/overview/
70. https://www.verum.com/
71. http://www.cordis.nl/
72. http://www.chronix.io/

robotica oplossingen in de Smart Industry. We kijken met de Cyber Security groep van FHICT onder leiding van Casper Schellekens naar de technische aspecten van sensoren, communicatie en het geprogrammeerde gedrag van robot. Ook kijken we naar de keten waarin Smart Industry/robot toepassingen tot stand komen en worden gebruikt. De mobiele robot is een systeem en moet als geheel worden bekeken in plaats van naar de losse componenten wat nu gebeurt als over veiligheid wordt gesproken.

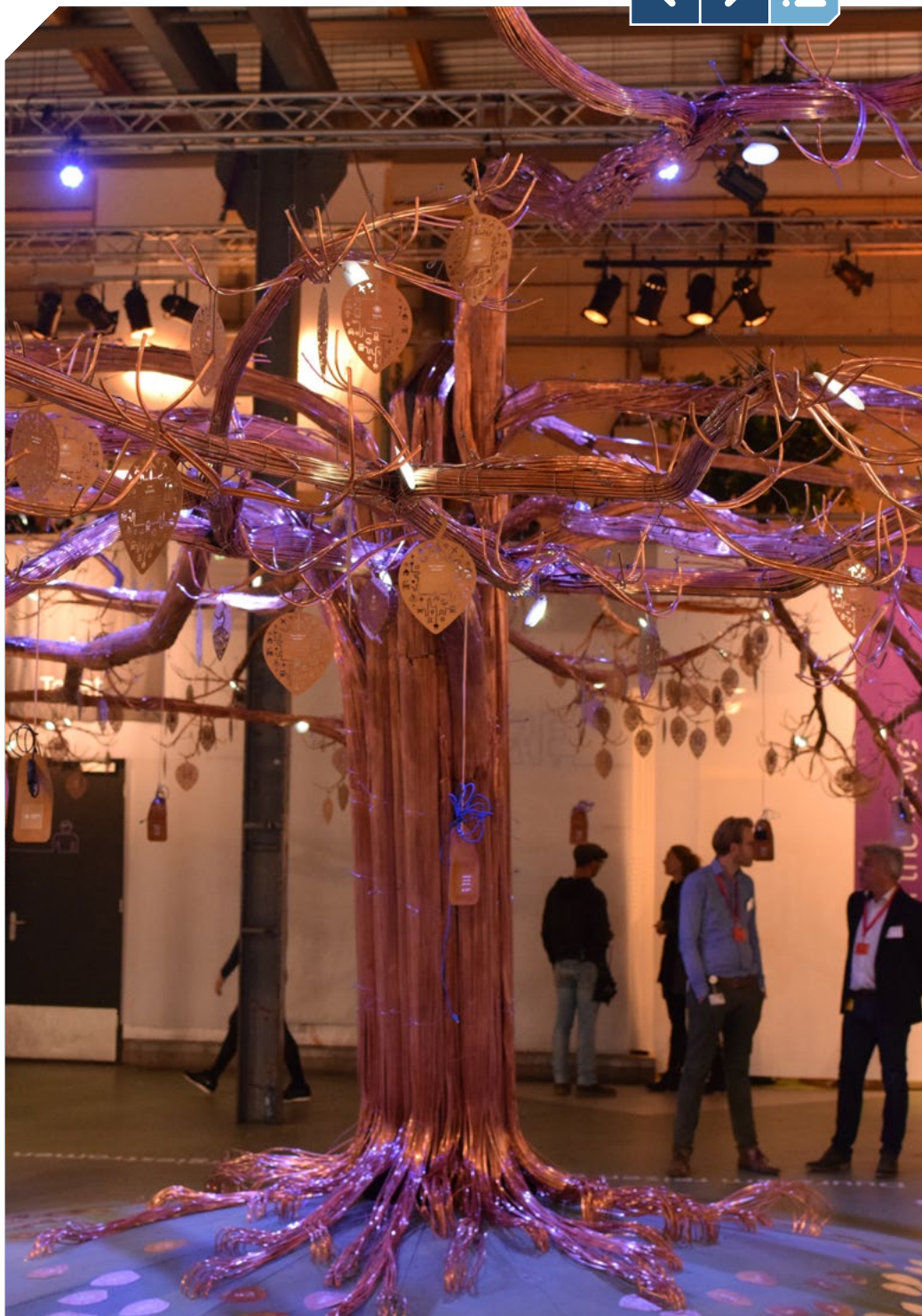
9.2. KWALITEIT VAN SENSORNETWERKEN

Voor sensornetwerken kijkt het lectoraat in eerste instantie naar de volgende kwaliteitsattributen:

- Energieverbruik (Energy efficiency) - het energieverbruik van individuele apparaten, die steeds vaker zonder continue stroomvoorziening zullen functioneren, bepaalt de gebruiksduur van apparaten in het netwerk.
- Betrouwbaarheid (Reliability, Robustness) - netwerken en robots die functioneren onder diverse omstandigheden en op tijd de benodigde data sturen en acties opstarten.
- Koppelbaarheid (Interoperability) - diverse (software) onderdelen in netwerken – diverse sensoren, computers – worden verbonden. Hoe werkt dit naadloos samen?
- Wijzigbaarheid (Modifiability) - binnen bestaande netwerkconfiguraties zullen regelmatig wijzigingen worden doorgevoerd waarbij het netwerk moet blijven werken[74].
- Beveiliging (Security) - de beveiliging en privacybescherming worden in het netwerk ingebouwd.

We gaan hier kort in op energiegebruik.[75] Het energieverbruik bepaalt de gebruiksduur van de nodes in het netwerk: hoe lager het energieverbruik, des te langer werkt de node en daarmee het gehele netwerk. Energieverbruik wordt bepaald door meerdere factoren: zoals de intensiteit van het netwerk verkeer, de omvang van de data en de netwerk topologie. Energy harvesting – waarbij energie wordt onttrokken aan de omgeving zoals wind en zonlicht, of uit beweging en aanwezige elektromagnetische signalen zijn interessante mogelijkheden in sensornetwerken. De software op de nodes moet met al deze fysieke oplossingen kunnen afstemmen.

74. Dit vereist bijvoorbeeld standaards die het mogelijk maken om vanuit apparaten te verbinden, onafhankelijk van merk, type, fabriek. Daarom kijkt het lectoraat naar open standaarden/open toegang. In dit verband kijken we naar Fiware https://www.fiware.org/.
75. https://wetenschapsagenda.nl/



10. ONDERZOEK, ONDERWIJS EN SAMENLEVING

> “Software engineers kunnen onjuiste beslissingen nemen over het toepassen van nieuwe methoden en technieken als zij niet het wetenschappelijke bewijs over de werkzaamheid van deze methoden en technieken kennen. Daarom dienen zij vergelijkbare procedures over het verwerven van kennis te hanteren als bij het ontwikkelen van medicijnen wordt gebruikt.”
>
> – BARBARA KITCHENHAM E.A. (DYBÅ E.A., 2005).

Dit hoofdstuk begint met een overzicht van de activiteiten in de onderzoekslijnen. Vervolgens wordt de betekenis van praktijkgericht onderzoek in het hbo geduid. Daarna gaan we in op de rol van dit onderzoek in onderwijs en samenleving.

10.1. ONDERZOEKSLIJNEN EN ROADMAP

Mobiele robots en sensornetwerken vormen de twee toepassingsgebieden en daarmee ook de onderzoekslijnen van het lectoraat. In de voorgaande hoofdstukken is al aangegeven dat we willen weten hoe we mobiele robots autonoom complexe taken kunnen laten uitvoeren door slimheid in de robots te programmeren en gebruik te maken van sensoren. Ook onderzoeken we hoe robots samenwerken, bijvoorbeeld als ze samen verschillende taken uitvoeren. De onderzoekslijn wordt daarmee aangeduid als ‘Autonome, mobiele robots’.

In de onderzoekslijn sensornetwerken kijken we hoe robuuste sensornetwerken kunnen worden opgezet, zodat deze zonder problemen metingen in de natuur of bebouwde omgeving doorgeven. Ook hier speelt de locatie van de intelligentie een rol. Wordt de intelligentie decentraal op nodes geprogrammeerd of wordt het centraal geregeld? Hier komen dan deelvragen bij kijken, zoals waar vindt de verwerking van gegevens plaats? Deze onderzoekslijn wordt aangeduid als ‘Sensornetwerken’.

We gebruiken de vier perspectieven van het systeemdenken om binnen de twee onderzoekslijnen naar onderzoeksvragen te kijken en deze verder vorm te geven vanuit de bestaande *Body of Knowledge* (Swebok, z.j.), (Sebok, z.j.), (Incose, z.j.) van bestaande methoden en technieken. Een voorbeeld van dit hergebruik in het aspect architectuur is het gebruik van bestaande kennis over het maken van state machines, dat nu in de context van robots wordt toegepast.

Een voorbeeld rondom het aspect modelleren en leren is het doen van een onderzoek naar algoritmes voor reinforcement learning, zodat we een robot een taak aan kunnen leren. De vier aspecten worden ook gebruikt om de kennis aan te vullen. Zo wordt de kennis over state machines aangevuld met een toepassing ervan in een type robot. Het algoritme wordt ontwikkeld en daarover wordt gepubliceerd.

Als we de onderzoekslijnen op deze manier organiseren dan kunnen we nog alle kanten op. Daarom zijn de volgende onderzoeksvragen leidend:

1. Hoe krijgen software engineers een beter begrip van functies en hun relaties van slimme systemen, om deze functies zo beter te kunnen ontwikkelen?
2. Hoe kunnen functies in slimme systemen het beste gesimuleerd en getest worden?
3. Welke modellen zijn nodig om de vragen rondom robots en sensornetwerken te beantwoorden?
4. Wat is de relatie tussen model en leren? Hoe kan robot software worden gemodelleerd zodat robots gedrag kunnen leren, zodat zij samen een opdracht kunnen uitvoeren?

Voor het uitvoeren van het onderzoek gebruikt het lectoraat een roadmap, waarin de vraagstelling van de onderzoekslijnen in de tijd is uitgezet.

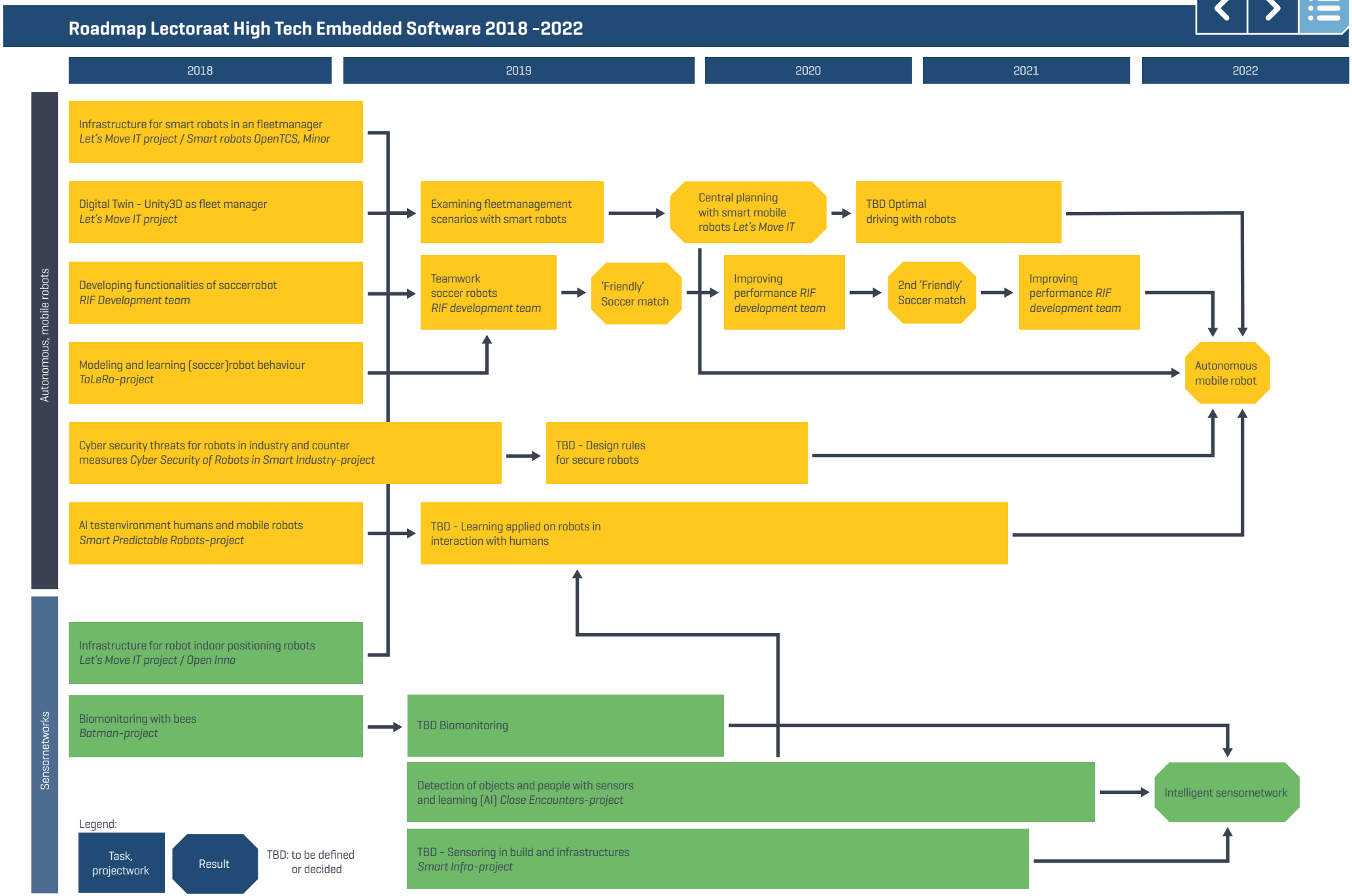
De onderzoeksactiviteiten worden uitgezet in de tijd, met daaraan gekoppeld de benodigde mensen en middelen en geplande onderzoeksresultaten. De roadmap dient om met diverse partijen binnen en buiten Fontys te kunnen overleggen. Het is daarmee vooral een communicatie middel. Belangrijke stakeholders zijn de docenten die hun onderzoek er in terug moeten herkennen. Figuur 13 geeft de lezer een indruk hoe de roadmap er grafisch uitziet. De onderdelen van de roadmap worden hieronder uitgewerkt.

Binnen de onderzoekslijn 'Autonome, mobiele robots' (geel) lopen op dit moment de volgende onderzoeks-activiteiten:
Centrale planning van robots - Aansturen van verschillende typen robots vanuit één fleet manager. Hierbij werken we enerzijds vanuit een bestaand fleet manager dat we zelf kunnen inrichten (OpenTCS). Anderzijds gebruiken we de game engine Unity3D als fleetmanager, waar we de robots in een 'virtuele wereld' simuleren en daarmee robots in de 'echte wereld' aansturen. Hiermee wordt een robot infrastructuur (lab) gemaakt waarmee experimenten met het plannen in het geval van centrale robotaansturing kunnen worden uitgevoerd. Deze activiteiten vinden plaats in het kader van het RAAK MKB Let's Move-project.
Voetbalrobot basisfuncties - Software ontwikkeling voor (basis) acties van een autonome voetbal robot, zoals ball handling, rijden over veld, ball control. Hiervoor wordt o.a. de vision verbeterd. Activiteit uitgevoerd door RIF Robocup MSL team.
Indoor positionering - onderzoek naar manier voor het precies bepalen van locaties van mobiele robots. Activiteit in het kader van het Raak MKB Let's Move IT-project.
Modelleren en leren voetbalrobot - Modelleren van gedrag in voetbal robots door middel van state machines, geïmplementeerd via de Dezyne modelleringstool. En het parallel bepalen welk (deel)gedrag van de robot via reinforcement learning kan worden geïmplementeerd. Activiteit in het kader van het Centre of Expertise HTSM ToLeRo-project en RIF Robocup MSL team.
Exploratief onderzoek naar cyber security bedreigingen van robots als onderdeel van Smart Industry. Hierbij kijken we naar de technische aspecten van sensoren, communicatie en het geprogrammeerde gedrag van robots. Er wordt bepaald wat veiligheidsbedreigingen zijn en welke maatregelen hiertegen genomen kunnen worden. Omdat de robotonderdelen door diverse partijen worden geleverd, wordt ook naar de keten van systeemonderdelen gekeken. Activiteit in kader van het KIEM Cyber Security van Robots in de Smart Industry-project.
Het maken van een testomgeving waarin cobots en mobiele robots met mensen samenwerken om de interactie tussen mens en robot inzichtelijk te maken.

De testomgeving is gericht op het inschatten van situaties en het nemen van de juiste beslissingen door de robot, en maakt daarbij gebruik van machine learning. Aan de menskant wordt gekeken of en hoe machine learning kan worden ingezet om de mensen te helpen met robots te leren omgaan. Activiteit in kader van het Smart Predictable Robots-project dat binnen de HTSM lijn van Fontys' TEC for Society[76] wordt uitgevoerd.

Binnen de onderzoekslijn 'Sensor-netwerken' (groen) wordt op dit moment het onderzoek uitgevoerd naar het *opstellen van een sensornetwerk voor biomonitoring met bijen.* Activiteit in het kader van het Batman-project, gefinancierd door Centre of Expertise HTSM. De verdeling van kleuren in de huidige versie van de roadmap geeft aan dat er veel ontwikkeld is op het gebied van robotica en dat er meer ontwikkeld gaat worden op het gebied van sensornetwerken. De roadmap wordt per halfjaar bijgesteld op basis van nieuwe inzichten.

76. https://www.fontys.nl/nieuws/fontys-kiest-voor-tec-for-society/



FIGUUR 13 - ROADMAP HTES, VERSIE JULI 2018

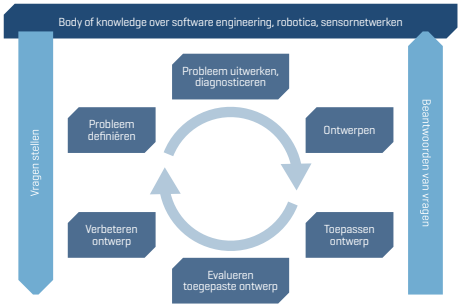
10.2. PRAKTIJKGERICHT ONDERZOEK

De basis van ieder onderzoek is nieuwsgierigheid. Het is 'de honger' naar antwoorden op waarom-, wie-, wat-, wanneer- en hoe- vragen. Het beantwoorden van deze vragen staat centraal in het doen van onderzoek[77]. FHICT duidt dat ook aan als Research & Development.

Praktijkgericht onderzoek in het hbo is ontwerpgericht onderzoek. HBO-onderzoek volgt immers de stadia van de onderzoekscyclus van het ontwerpgerichte onderzoek (Oskam e.a., 2012), (Andriessen, 2007) en (Brouns, 2016). In het onderzoek wordt over het algemeen begonnen met een concrete probleemstelling die afkomstig is uit de beroepspraktijk. Daarvoor zal een diagnose worden opgesteld om tot de kern van het probleem te komen. Er wordt geprobeerd om tot een oplossing van het probleem te komen. Dit gebeurt door een ontwerp te maken en toe te voegen aan de bestaande praktijk. Het ontwerp dat wordt ontwikkeld dient te voldoen aan tal van ontwerpcriteria. Vervolgens wordt het ontwerp toegepast in de praktijk en wordt gekeken of het ontwerp functioneert. Het toegepaste ontwerp wordt geëvalueerd en de waarde ervan wordt bepaald aan de hand van gestelde ontwerpcriteria. Aan de hand van de evaluatie kan het ontwerp worden verbeterd. Figuur 14 geeft de onderzoekscyclus grafisch weer. De verschillende activiteiten in deze cyclus, zoals probleem definiëren, diagnose opstellen, kennis over ontwerp vergaren worden uit de kennis over hoe je software ontwerpt, robots en sensornetwerken maakt, de Body of Knowledge'. Dit heeft in het geval van het lectoraat HTES betrekking op software engineering, robotica en sensornetwerken Ze bestaat onder andere uit de literatuur rondom deze thema's, zoals gepresenteerd in de bronnenlijst van deze rede, en levert richtlijnen op zoals bijvoorbeeld verwoord in (Bahill & Botta, 2008).

Hiervoor is het stellen van vragen van belang. Tijdens en na afloop van het uitvoeren van de onderzoeksactiviteiten wordt er vervolgens nieuwe kennis aan de Body of Knowledge toegevoegd. Dit kan via verschillende activiteiten gebeuren, zoals evalueren ontwerp of juist het verbeteren ervan. Niet elk proces waarmee een probleem wordt opgelost met een ontwerp is praktijkgericht onderzoek. Het herhaald doorlopen van de onderzoekscyclus waarbij producten of artefacten worden gemaakt, zonder nieuwe vragen te stellen en nieuwe toevoegingen op de toegepaste

77. Dit sluit aan op de aanpak '21st Century Skills' die in het onderwijs vaker gehoord wordt http://curriculumvandetoekomst.slo.nl/21e-eeuwse-vaardigheden



FIGUUR 14 - CYCLUS VAN ONTWERPGERICHT ONDERZOEK. AFGELEID VAN PUNTER (2001).

technologie te maken, de vragen te beantwoorden, is geen onderzoek. Het op deze manier doorlopen van de cyclus is advieswerk omdat er geen (nieuwe) kennis wordt gegenereerd. Ook het doen van 'onderzoekende handelingen', zonder dat daarbij kennis aan de voor het vakgebied relevante Body of Knowledge wordt toegevoegd is onvoldoende om deze activiteiten als praktijkgericht onderzoek te bestempelen. Daarentegen, bepaalt het methodisch uitvoeren van de onderzoeksactiviteiten wel of deze activiteiten onderzoek zijn. Dit is onafhankelijk van de personen die de activiteiten uitvoeren. Onderzoek kan dus door lectoren, docenten en studenten worden verricht.

Voor het uitvoeren van praktijkgericht onderzoek is het wel van belang dat valide onderzoeksprocedures worden gevolgd. De onderzoeksmethodologie moet op orde zijn om er voor te zorgen dat kennis goed wordt gegenereerd. Hierbij gaat het enerzijds om het ophalen van de kennis uit de Body of Knowledge, waarbij onderzoeksmatig vragen spelen zoals: zijn de juiste bronnen geraadpleegd voor dit probleem? Is al een keer eerder een aanpak bedacht voor het probleem? Wat zijn de ervaringen met het toepassen van deze aanpak? Anderzijds gaat het om het toevoegen van kennis die is opgedaan bij het doorlopen van de ontwerpcyclus. Onderzoeksprocedures gaan ook over de bewijslast, wanneer moet worden aangetoond dat aan ontwerpcriteria is voldaan. Dergelijke criteria vormen de basis van de argumentatie over de kwaliteit van het geleverde ontwerp. Dit zien we terug als docenten als wij stage- en afstudeeropdrachten beoordelen, en dient ook centraal te staan in onze onderzoeksprojecten. Door het onderzoek procedureel goed uit te voeren kunnen we leren en weten we of we de juiste antwoorden op de gestelde vragen hebben gekregen.

In dit verband moet het belang van het onderzoeksframework worden gezien dat in het onderwijs van FHICT wordt toegepast (Turnhout e.a., 2013). Het lectoraat HTES ziet haar rol daarbij in het verbeteren en faciliteren van de onderzoekscyclus door dit eerst zichtbaar te maken in eigen onderzoek en deze ervaringen via voorbeelden over te dragen

op studenten. Dit doet het lectoraat door onderzoekslijnen uit te zetten, waarbij bedrijfsproblemen worden vertaald naar bredere onderzoeksuitdagingen die ook zijn ingebed in state-of-the-art ontwikkelingen die in de Body of Knowledge rondom software engineering, robotica en sensornetwerken.

Het praktijkgerichte onderzoek binnen de ICT sluit aan op wat wordt aangeduid als Evidence-based Software Engineering (Kitchenham e.a., 2004)[78] Deze beweging van software engineering onderzoekers streeft naar het empirisch toepassen van software methoden en technieken via een sterk methodologisch kader zoals gebruik in case studies, post-mortems en/of surveys (Wohlin e.a., 2000), (Freimut e.a., 2002), (Endres & Rombach, 2003). De nadruk ligt op het onderbouwen van voor- en nadelen van de methoden en technieken. Het lectoraat HTES ziet haar handelen als 'evidence-based'. We werken wetenschappelijk verantwoord door goed bronnenonderzoek te doen, de criteria voor het bepalen van de waardering van technieken en methoden expliciet te maken en onze gevolgtrekkingen navolgbaar te maken. Deze evidence-based denkwijze uit de software engineering gebruikt het lectoraat om richtlijnen op te bouwen voor het verbeteren van software ontwikkeling op het gebied van robotica en sensornetwerken. Naast het genereren van kennis is het praktijkgerichte onderzoek in het hbo ook bezig met het inbedden van de onderzoeksresultaten in de maatschappij. Synoniemen voor inbedden van technologie zijn valorisatie, outreach en technology transfer (Punter e.a., 2009B). Het gaat hier met name om het nemen van verantwoordelijkheid voor het succesvol invoeren van de ingevoerde technieken en methoden tijdens en na het toepassen en het evalueren van het ontwerp.

10.3. ONDERZOEK IN HET ONDERWIJS

Een belangrijk uitgangspunt van Fontys Hogeschool ICT is dat studenten zelfstandig en autonoom leren, binnen een sociale context. Wij geven dus geen lessen waar studenten passief aanzitten, maar opdrachten, die studenten uitnodigen de kennis te verkennen en vragen te stellen. We staan ook midden in de maatschappij en opereren in een wereld waar de ontwikkelingen in de ICT onnavolgbaar snel verlopen (FHICT, 2018). Die studentopdrachten zijn op de praktijk geënt en het lectoraat speelt een rol bij het formuleren van deze opdrachten in overleg met bedrijven waar FHICT en het lectoraat mee samenwerken. Het lectoraat wil samenwerken met studenten en doet dat door haar projecten in het onderwijs uit te voeren. Ook worden de onderzoeksresultaten gebruikt bij het verbeteren en vernieuwen van onderwijsmateriaal. Verder wordt meegewerkt en meegedacht aan het uitwerken van de methodologie voor het praktijk gericht onderzoek in het ICT-onderwijs.

Het lectoraat ziet een eigen rol in het tastbaar maken van het onderzoek. Tastbaarheid gaat over het concreet maken van onderzoek richting de belevingswereld van studenten (Klatter e.a., 2016). Ook helpt het lectoraat bij het vergroten van de nieuwsgierigheid voor bestaande en nieuwe thema's. Nieuwsgierigheid, het stellen van vragen, vormt immers de basis van alle onderzoek. We zien ook dat de complexiteit van systemen maakt dat we studenten moeten helpen om op een hoger plan te kunnen komen met het definiëren en vooral het uitvoeren van hun projecten. De zelfstandigheid en de autonomie van studenten, aangevuld met coaching in de vorm van experts die eindtermen en –doelen helpen definiëren, zien wij als ideale werkvorm om tot kwalitatief goede resultaten te komen.

Het lectoraat richt daarvoor *embedded software labs* op. Het zijn software ontwikkelomgevingen die gericht zijn op de twee toepassingsgebieden van het lectoraat. In een software lab werken meerdere studenten en docenten samen over verschillende opeenvolgende semesters heen. Er wordt domeinkennis rondom het lab thema opgebouwd waardoor het mogelijk wordt om een complexer systeem te maken. Een software lab(oratorium) is daarmee een project dat meer realiseert dan individuele studentgroepen kunnen realiseren. Labs helpen om kennis op het gebied van mobiele robotica en sensornetwerken binnen Fontys op te bouwen. Ook zal in een software lab domeinkennis over het thema worden opgebouwd. In het RIF team zien we dat al met robot functies, zoals calibratie van beeld, motor control, in de Minor Embedded Systems gebeuren.

Labs zijn een vorm van open innovatie (Chesbrough, 2003) door studenten en docenten en waarbij vertegenwoordigers uit bedrijven zijn uitgenodigd om erin mee te doen. Een software lab heeft tot doel om de software ontwikkeling beter zichtbaar voor andere disciplines te maken. Dit helpt bij het communiceren over software ontwikkeling en haar uitdagingen. Het RIF Robocup MSL team van Fontys is een voorbeeld van een software lab. Het lectoraat gaat dit uitbreiden waarbij onder andere gedacht aan software labs voor AGV fleetmanagement en een sensornetwerk voor biomonitoring.

Naast studenten draagt het lectoraat ook bij aan de ontwikkeling van docenten. Het

78. (Wohlin e.a., 2000), (Endres en Rombach, 2003) duiden dit aan als Empirical Software Engineering.

lectoraat kiest daarbij voor een flexibele invulling met docent/onderzoekers, waarbij docenten voor minimaal twee achtereenvolgende semesters bij het lectoraat betrokken zijn. Docenten doen mee in het lectoraat op basis van eigen interesse, ambitie en hun beschikbaarheid om aan onderzoek te werken. Er is afstemming met de lector, zodat lopende onderzoeksprojecten kunnen worden uitgevoerd. Docenten voeren zelf, in samenwerking met lectoraatcollega's en studenten, onderzoek uit. De uitdaging hierbij voor het lectoraat is om over de onderzoeksprojecten heen de resultaten te genereren en onder te brengen in de onderwijsorganisatie. Het lectoraat hecht daarom sterk aan het ontwikkelen van een onderzoekscultuur. Dit vereist voldoende tijd en inzet van de onderzoekers, maar ook voldoende ruimte voor overleg. De roadmap en software labs zijn andere instrumenten waarmee het lectoraat haar onderzoekscultuur wil vormgeven.

10.4. ONDERZOEK IN DE SAMENLEVING

Het lectoraat hecht ook veel waarde aan samenwerking met bedrijven en maatschappelijke organisaties. Het zijn immers deze organisaties die voor het hbo relevante vragen kunnen stellen. Vaak vinden zij hun weg naar Fontys ICT via stage- of afstudeerpopdrachten en/of deelname aan het onderwijs via het Partner In Education (PIE) programma. Het lectoraat HTES ziet haar rol in het ontwikkelen van nieuwe kennis met en voor deze organisaties en het verbeteren van de kennisdeling tussen organisaties. Zij sluit daarbij aan bij het TEC for Society initiatief[79] dat Fontys is gestart om nog meer op de maatschappij aan te sluiten en een bijdrage te leveren aan een duurzame samenleving.

Het lectoraat HTES kan hier in bijdragen door samenwerking met organisaties aan te gaan voor langere periodes, langer dan een semester, waarbij kennis wordt ontwikkeld met of voor bedrijven, of juist periodiek kennis met elkaar wordt gedeeld. In de kennis ontwikkeling mét bedrijven gaat het om nieuwe thema's waarbij beide partijen exploratief onderzoek willen doen om snel tot resultaten te komen. Dit heeft het lectoraat de afgelopen periode gedaan met Robocup MSL robot voetbal, waarbij diverse bedrijven betrokken zijn geweest. De kennisontwikkeling kan ook met één bedrijf worden uitgevoerd. Het bedrijf heeft dan de domein expertise, waarop nieuwe methoden of tools worden toegepast. Het lectoraat brengt de kennis over methoden en technieken in, en experimenteert hiermee in het bedrijf. Het toepassen van de kennis levert het bedrijf een oplossing/meerwaarde voor haar eigen domein, terwijl de kennis via het lectoraat daarna weer in andere organisaties kan worden toegepast. Het lectoraat heeft dan de opgave om zich te verdiepen in de domeinkennis van het bedrijf. Dit heeft het lectoraat bijvoorbeeld gedaan met VDL ETG[80].

Software labs zijn in beide gevallen een instrument om kennis met en voor bedrijven te ontwikkelen. De embedded software labs, die in eerste instantie door docenten en studenten worden ontwikkeld, zijn op termijn ook te gebruiken om bedrijven bij het praktijkgerichte onderzoek te betrekken. We verwachten dat de software labs zogenaamde best practices voor de Body of Knowledge rondom de software toepassingen voor robots en sensornetwerken opleveren. Bedrijven kunnen hier kennis over opdoen en in de showroom meekijken. Bedrijven en andere (onderwijs)instellingen kunnen ook deelnemen in een software lab. Op deze manier kan het onderwijs weer leren van werkwijzen en ervaringen uit het bedrijfsleven.

Bij het verbeteren van de kennisdeling tussen bedrijven ziet het HTES lectoraat voor zichzelf een rol als kennismakelaar en -leverancier. Het zet daarvoor zelf kennisoverleg op, en neemt deel aan kennisoverleg dat wordt georganiseerd door andere bedrijven en (onderwijs) instellingen. Het gaat dan om thema's zoals testen, modelleren en robotica. Dit laatste heeft het lectoraat gedaan door het ICT in Practice 2018, thema robotica, mee te organiseren[81].

79. https://fontys.nl/KennisNetwerk/Ontwikkelingen/TEC-for-Society.htm

80. https://www.regioinbedrijf.nl/projecten/er-zit-muziek-in-deze-samenwerking.6590/
81. https://fontys.nl/Over-Fontys/Fontys-Hogeschool-ICT/Onderzoek/ICT-In-Practice.htm

MEENEEMPUNTEN

Dit essay geeft een systeemvisie op het ontwikkelen van embedded software voor slimme systemen: (mobiele) robots en sensornetwerken. De kernboodschap is hierlangs samengevat:

1. **Embedded software** zit in veel apparaten, de embedded systemen. Voorheen ging het in deze computers om de software die was afgestemd op de hardware. Tegenwoordig werken embedded systemen steeds vaker samen, en is de relatie met de fysieke wereld belangrijk, het worden Cyber Physical Systems. De apparaten worden ook steeds intelligenter ('smarter'): ze kunnen steeds meer dingen voor ons doen. Software is nodig om deze intelligentie te realiseren, en daarmee de slimme systemen te maken. Deze slimme systemen duiden we ook aan als high tech embedded systemen, waarvoor dus high tech embedded software wordt gemaakt.

2. **Het doel van het lectoraat High Tech Embedded Software (HTES)** is het verbeteren van kennis en vaardigheden voor het ontwikkelen van de software voor deze slimme systemen. Het praktijkgerichte onderzoek moet leiden tot het realiseren van autonome, mobiele robots en sensoren die meten in een netwerk.

3. **Het lectoraat hanteert hierbij systeemdenken om met systeemcomplexiteit om te kunnen gaan.** Dit systeemdenken onderkent vier aspecten: architectuur, integratie en test, modelleren en leren en software kwaliteit.

4. **Architectuur** gaat over componenten, de interfaces en de functies waarmee robots en sensornetwerken hun gedrag realiseren. De vraag die het lectoraat hier stelt is: hoe krijgen software engineers een beter begrip van de functies van cyber physical systems, waarvan de ontwikkeling kennis uit diverse disciplines vereist, om deze goed te kunnen ontwikkelen? We werken hiermee aan inzichten rondom centrale versus decentrale aansturing van robots en het verdelen van rekenkracht in een sensornetwerk.

5. **Integratie en test hangt sterk samen met architectuur.** De vraag die het lectoraat hier stelt is: hoe kunnen functies/embedded systemen het beste gesimuleerd en getest worden? We werken hiermee aan inzichten over de samenhang tussen eenvoudige (unit)tests en optreden van emergent behavior in robots, en het implementeren van autonome nodes in een netwerk die zelfdiagnose en zelfherstel kunnen uitvoeren.

6. **Modelleren en leren** is nodig om met complexe systemen om te kunnen gaan. Modellen geven antwoorden tijdens architectuur en integratie en test. De vragen die het lectoraat hier stelt zijn: Welke modellen zijn nodig om de vragen rondom robots en sensornetwerken te beantwoorden? En: Wat is de relatie tussen model en leren? Hoe kan robot software worden gemodelleerd zodat robots gedrag kunnen leren, zodat zij samen een opdracht kunnen uitvoeren? We werken hiermee aan inzichten over het toepassen van reinforcement learning in voetbalrobots. In sensornetwerken willen we met modellen redeneren over netwerken om zo eigenschappen als energieverbruik en veiligheid te begrijpen.

7. **Softwarekwaliteit** geeft aan of het resultaat van architectuur, integratie en test en modelleren in voldoende mate is bereikt. Het lectoraat besteedt aandacht aan diverse kwaliteitseigenschappen, alle onder de noemer van het streven naar duurzame software.

DANKWOORD

Op 1 september 2015 ben ik bij Fontys ICT begonnen als docent ICT&Technology. Al snel werd ik betrokken bij het praktijkgerichte onderzoek van Fontys. Eric Dortmans enthousiasmeerde me voor een ver opgeborgen passie (robots) en Wim Hendriksen betrok me al snel bij de begeleiding van het Robocup team. Vanuit deze praktische invalshoek was het leuk om weer onderzoek te doen naar nieuwe technologie. Na enkele maanden betrokken Rop Pulles en Cees van Tilborg me verder bij het onderzoek en werd ik onderzoeksleider voor ICT&Technology. Ik vond het een geuzentitel omdat het de vrijheid van het onderzoek benadrukt.

Uiteindelijk werd in de zomer van 2016 binnen Fontys ICT besloten om weer een lectoraat met embedded systemen op te richten. De naam van het lectoraat werd al snel High Tech Embedded Software, een titel die mij aansprak omdat ze de nadruk legt op de high tech industrie in de Brainport regio waar Fontys ICT opereert. Het lectoraat HTES is per 1 september 2017 gestart en ik ben per die datum ook als lector benoemd.

In het proces van mijn lectorbenoeming wil ik diverse mensen bedanken. Het College van Bestuur van Fontys bedank ik voor het instellen van het lectoraat HTES en het in mij gestelde vertrouwen om mij daarvoor als lector te benoemen. Mijn dank gaat ook uit naar het Management van FHICT, in het bijzonder Ad Vissers. Ad schept als directeur van Fontys hogeschool ICT een uitdagende omgeving om onderzoek te doen in een onderwijsomgeving. Rop Pulles en Cees van Tilborg bedank ik voor de vele gesprekken om tot een lectoraat te komen.

Wim Hendriksen bedank ik voor de coaching en adviezen over de lectorfunctie. Johan Vlugter ben ik erkentelijk voor het meeschrijven van de lectoraatsaanvraag. Eric Dortmans, Wim Hendriksen, Bart Theelen en Cees van Tilborg hebben bijgedragen aan deze aanvraag met hun commentaar op tussenversies.

Jan Friso Groote en Benno Beuting zijn referenten van de lectoraatsaanvraag. Hen bedank ik voor hun ondersteuning, commentaar en hun bereidheid om, samen met Heico Sandee, deel te nemen in de adviescommissie van het lectoraat.

Voor het schrijven van de voorliggende rede hebben veel mensen en ideeën mij geïnspireerd. Ik bedank met name Peter Lambooij en Eric Dortmans voor de met hen gevoerde gesprekken en hun commentaar op tussenversies. Verder bedank ik Robert Schuwer en Annemarie Diepenbroek voor hun feedback op de rede. Christel van Luttervelt en Yvonne van Hout bedank ik voor de opmaak van deze rede. Marilyn Bailey en Annemarie Diepenbroek bedank ik voor de algemene ondersteuning.

De weg naar het lectorschap heeft zich achteraf gezien als een logische weg aan mij geopenbaard. Op het pad erheen ben ik in de loop de jaren veel mensen tegengekomen die mij, bedoeld en onbedoeld, beïnvloed hebben. Dit betreft vele mensen binnen de Open Universiteit, Technische Universiteit Eindhoven, Fraunhofer IESE, Embedded Systems Institute/TNO en alle bedrijven waar ik heb mogen werken. Ik bedank in het bijzonder: Rini van Solingen, Peter Kaiser, Gerrit Muller en Jac Jamar.

Ik bedank alle collega's van FHICT, en in het bijzonder mijn collega's van de studierichting ICT & Technology en het Centre of Expertise HTSM voor hun ondersteuning van en inzet bij het lectoraat.

De docentonderzoekers die op dit moment direct bij het lectoraat betrokken zijn, bedank ik speciaal voor hun inzet en toewijding om in het verlengde van hun docentschap ook onderzoek te (gaan) doen: René Bakx, Karin Dieleman, Eric Dortmans, Peter Lambooij, Oswald Figaroa, Cees van Tilborg, Suzana Andova, Joeri van Belle, Renáta Frenken, Lake Lakeman, Corné van Dijk en Jan Dobbelsteen. Mijn collega lectoren, binnen en buiten FHICT, bedank ik voor hun openheid en vertrouwen. Ik verheug me op de verdere samenwerking.

Praktijkgericht onderzoek aan het hbo bestaat niet zonder studenten. Het zijn tot nu toe vele en diverse manieren van samenwerking geweest, zoals in het Let's Move IT-project, het RIF Robocup MSL team en in Open Innovatie projecten. Dit smaakt naar meer!

Mijn familie en schoonfamilie bedank ik voor hun belangstelling en gezelligheid. Mijn ouders ben ik dankbaar voor de basis die zij mij hebben meegegeven om diegene te worden die ik ben.

Dieuwke, ik voel jouw liefde en je openheid als de belangrijke ingrediënten voor ons samenzijn. Onze gelijkwaardigheid, jouw gevoel voor ondersteuning en vrijheid maken mijn lectorschap mogelijk. Dank je wel! Karsten & Anne, Wieger & Meike, en Silke, onze kinderen, en hun partners, jullie bedank ik voor de gezelligheid, de dynamiek, de humor en de muziek in ons gezin.

CURRICULUM VITAE

Teade Punter (1965) studeerde Wijsbegeerte van Wetenschap, Technologie en Samenleving aan de Universiteit Twente in de richting Bedrijfskunde en Informatica. Hij begon zijn loopbaan in 1992 als cursusteamleider bij de Open Universiteit in Heerlen.

Voor de studierichting Technische Informatica werkte hij met Karel Lemmen aan een methodologie voor informatiesysteemontwikkeling. Ook ontwikkelde hij een cursus voor data-bases en software management.

In 1996 begon hij aan promotieonderzoek naar software certificatie aan de Technische Universiteit Eindhoven. Dit onderzoek werd ondersteund door KEMA Nederland. In 2001 verdedigde hij hierover het proefschrift ‘Doelgericht beoordelen van software’, onder leiding van professor Theo Bemelmans en professor Aarnout Brombacher.

De opgedane promotiekennis werd vanaf 2000 toegepast en verder uitgewerkt bij het Fraunhofer Institut Experimentelles Software Engineering (IESE) in Kaiserslautern. Teade was daar groepsleider van de Software Quality en Measurement group en werkte aan het verbeteren van software product- en procesbeoordelingen en het meten van software kwaliteit. Ook is er met Danilo Assmann aan diverse beoordelingsmethodes gewerkt, bijvoorbeeld voor het beoordelen van subcontractors. Met Bernd Freimut en Marcus Ciolkowski is een methodologie voor empirical software engineering uitgewerkt.

In 2004 werd Teade consultant bij de Technische Universiteit Eindhoven, faculteit Technische Informatica, Laboratory for Quality Software (Laquso). Hij werkte er aan het toepassen van formele methoden in de high-tech industrie, waarbij de nadruk lag op het de zogenaamde technology transfer. Ook werkte hij met Louis van Gool aan modeltransformatie ten behoeve van het software ontwikkeling bij ASML.

In 2007 stapte Teade over naar het Embedded Systems Institute (ESI) en werkte daar als kennismanager aan de competenties systeem architectuur en integratie en test. Voor de kennisuitwisseling organiseerde hij diverse Special Interest Groups en conferenties. Ook werd met Roelof Hamberg en Hristina Moneva het Design Framework voor het vastleggen van architectuurbeslissingen ontwikkeld. Met de overgang van het ESI naar TNO-Embedded Systems Initiative in 2013 werd Teade research fellow. Hij werkte er met Jac Jamar aan het zichtbaar maken van technische risico’s in complexe systemen en met Evert van de Plassche aan het proces van systeem integratie bij ASML.

In 2015 is Teade als docent ICT & Technology begonnen bij Fontys Hogeschool ICT. Per 1 september 2017 is hij benoemd als lector High Tech Embedded Software bij deze hogeschool.

Teade is geïnteresseerd in kunst, cultuur, natuur, geschiedenis en technologie. Hij is gelukkig getrouwd met Dieuwke de Haan. Zij hebben drie geweldige kinderen: Karsten, Wieger en Silke.

INDEX

INDEX

BRONNEN

A

- Allworth, J. (2013, 30 januari). The 787's problems run deeper than outsourcing. *Harvard Business Review*. Online beschikbaar: http://blogs.hbr.org/cs/2013/01/the_787s_problems_run_deeper_t.html. Geraadpleegd op 26 augustus 2018.
- Andriessen, D. (2007). *Onderzoeksmethodologie in het HBO*. Onderzoek van onderwijs (OVO), 6, 97-99.
- Angelov, S., Meesters, M., Galster, M., (2016), Architects and Scrum - which challenges do they face? In *Proceedings 10th European Conference on Software Architecture (ECSA) (pp. 229-237)*. Copenhagen: Springer.
- Asimov, I. (2017). *Ik robot - heruitgave Nederland leest*. Amsterdam: Meulenhof.
- ASML Falcons (z.j.). ASML Falcons Robocup MSL. Online beschikbaar: http://www.falcons-robocup.nl/. Geraadpleegd op 26 augustus 2018.

B

- Bahill, T., Botta, R. (2008), Fundamental Principles of Good System Design. *Engineering Management Journal*, 20 (4), 9-17. doi: 10.1080/10429247.
- Baldwin, P., S. Kohli, E. Lee (2004), Modeling sensor networks with Ptolemy II, In *Proceedings of Information Processing in Sensor Networks (IPSN)* (pp. 359-368) Berkeley, USA. doi: 10.1145/984622.984675.
- Barthe, G., P. D'Argenio, B. Finkbeiner, H. Hermanns (2016), Facets of Software Doping. In T. Margaria and B. Steffen (Eds.), *ISoLA 2016, LNCS 9953* (601–608). doi: 10.1007/978-3-319-47169-3 46.
- Basten, T. (2011), De computer verdwijnt. Intreerede, Technische Universiteit Eindhoven. Online beschikbaar: https://pure.tue.nl/ws/files/3232668/basten2011.pdf. Geraadpleegd op 20 augustus 2018.
- Beck, K. (2003). Test Driven Development – By example. Boston: Addison-Wesley. ISBN: 0321146530.
- Blair, G., N. Bencomo, R. France (2009), Models @ runtime. *IEEE Computer,* 22-29. Online beschikbaar: http://oro.open.ac.uk/39430/1/modelsSI2012.pdf. Geraadpleegd op 20 augustus 2018.
- Boehm, B., R. Turner (2004). *Balancing Agility and Discipline – A guide for the Perplexed.* Boston: Addison-Wesley. ISBN: 0321186125.

- BOM (2015), *Robotics – Ondersteunen, assisteren en samenwerken – Marktverkenning en mogelijkheden voor Brabant.* Online beschikbaar: https://issuu.com/bombrabant/docs/robotics_marktverkenning_en_mogelij?e=29485023/50619736. Geraadpleegd op 20 augustus 2018.
- Booch, G., J. Rumbauch, I. Jacobson (1999). *The unified modeling language user guide,* Addison-Wesley. ISBN: 0201571684.
- Borst, S., J. van Leeuwaarden, P van de Ven (2015), Stochastische modellen voor random-access-netwerken. *Nieuw Archief voor Wiskunde*, 5(16), Online beschikbaar: http://www.nieuwarchief.nl/serie5/pdf/naw5-2015-16-3-201.pdf. Geraadpleegd op 20 augustus 2018.
- Brockman, J. (2015). *Machines die denken, invloedrijke denkers over de komst van kunstmatige intelligentie.* Amsterdam: Maven Publishing. ISBN 978-9491845826.
- Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal Robotics Automation*, 2, 14-23. doi: 10.1109/JRA.1986.1087032.
- Brouns, M. (2016). Van Agora tot Olympus – een frisse blik op praktijk-gericht onderzoek in onderwijs. *Thema Hoger Onderwijs*, 4, 69 - 74.
- Brown, S. (2014). *Software Architecture for Developers*. Online beschikbaar: https://leanpub.com/software-architecture-for-developers/read#c4. Geraadpleegd op 20 augustus 2018.
- Bruntink, M, A. van Deursen, T. Tourwé (2006). Discovering faults in idiom-based exception handling. In *Proceedings of International Conference on Software Engineering (ICSE)* (pp. 242-251). Sjanghai/New York: ACM. doi: 10.1145/1134285.1134320.
- Bruyninckx, H. (2008), Robotics software: the future should be open. *IEEE Robotics and Automation Magazine,* 15, 9-11.
- Bunse, C., A. Hopfner, E. Mansour, S. Roychoudhury (2009). Exploring the Energy Consumption of Data Sorting Algorithms in Embedded and Mobile Environments. In *Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware (MDM)* (pp. 600-607). Washington DC: IEEE Computer Society.
- Buratti, C., A. Conti, D. Dardari, R. Verdone (2009). An overview on Wireless Sensor Networks Technology and Evolution. *Sensors, 9 (9),* 6869-6896. doi: 10.3390/s90906869.
- Burkhard, H-D, M. Hannebauer, J. Wendler (1998), Belief-Desire-Intention Deliberation in Artificial Soccer. *AI Magazine 19 (3)*, 87-93.
- Buschman, F., K. Henney, D. Schmidt (2007). *Pattern-oriented software architecture – On patterns and pattern languages.* Chichester: John Wiley. ISBN: 978-0-471-48648-0.

C

- Capra, F., P. Luisi (2014). *The Systems view of life: a unified vision*. Cambridge: Cambridge University Press. ISBN: 1107011361.
- Checkland, P.B. (1981). *Systems Thinking, Systems Practice*. Londen: John Wiley & Sons. ISBN: 0 471 27911 0.
- Chesbrough, H. (2003). *Open Innovation: The New Imperative for Creating and Profiting from Technology.* Boston: Harvard Business School Press. ISBN: 1422102831.
- Cianchetti, M., M. Calisti, L. Margheri, M. Kuba, C. Laschi (2015). Bioinspired locomotion and grasping in water: the soft eight-arm Octopus robot. *Bioinspiration and biomimetics, 10 (3)*. doi: 10.1088/1748-3190/10/3/035003
- Cunha, J., R. Serra, N. Lau, L. Lopes, A. Neves (2015). Batch Reinforcement Learning for Robotic Soccer Using the Q-Batch Update-Rule. *Journal Intelligent Robot Systems.* doi: 10.1007/s10846-014-0171-1.

D

- Delicato, F., P. Pires, J. Portocarero, L. Fuentes (2014). Autonomous wireless sensor networks: a systematic literature review. *Journal of Sensors, 3*. doi: 10.1155/2014/782789
- Devloo, J. (2014), *Evaluatie van link-inschattingsalgoritmen voor RPL in dynamische sensornetwerken* (Proefschrift Universiteit Gent). Online beschikbaar: https://lib.ugent.be/fulltxt/RUG01/002/153/639/RUG01-002153639_2014_0001_AC.pdf. Geraadpleegd op 26 augustus 2018.
- Dybå, T., B. Kitchenham, M. Jørgensen (2005), Evidence-Based Software Engineering for Practitioners. *IEEE Software, 22 (1)*, 58-65. doi: 10.1109/MS.2005.6

E

- Endres, A., D. Rombach (2003), *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories.* New York: Addison-Wesley. ISBN: 978-0321154200.
- Est, R. van, L. Kool (Eds.) (2015). *Werken aan de robotsamenleving: visies en inzichten uit de wetenschap over de relatie technologie en werkgelegenheid.*

Den Haag: Rathenau Instituut. Online beschikbaar: https://www.rathenau.nl/sites/default/files/Werken_aan_de_robotsamenleving_-_Rathenau_Instituut.pdf. Geraadpleegd op 21 augustus 2018.

- Eynikel, J. van. *Robot aan het stuur – over de ethiek van de techniek*. Leuven: Lannoo. ISBN: 9789401440998.

F

- FHICT (2018), #2023 Open up - Ambitieplan Fontys Hogeschool ICT. Eindhoven: Fontys Hogeschool ICT.
- FME, Kamer van Koophandel, Ministerie van Economische Zaken, VNO NCW, TNO (2014), Actieagenda Smart Industrie. Online beschikbaar: http://smartindustry.nl/wp-content/uploads/2017/07/smart-industry-actieagenda-lr.pdf. Geraadpleegd op 20 augustus 2018.
- Fontys (2015), Fontys Focus 2020. Online beschikbaar: https://fontys.nl/Over-Fontys/Wie-zijn-wij/Onze-ambities.htm. Geraadpleegd op 20 augustus 2018.
- Freimut, B., T. Punter, S. Biffl, M. Ciolkowski (2002), State-of-the-art in Empirical studies (IESE-Report No. 017.02/E). Online beschikbaar: https://www.researchgate.net/publication/247640253_State-of-the-art_in_empirical_studies. Geraadpleegd op 20 augustus 2018.
- Frey, C., M. Osborne (2013). The Future of Unemployment. How Susceptible Are Jobs to Computerization? (University of Oxford). Online beschikbaar: https://www.oxfordmartin.ox.ac.uk/downloads/academic/The_Future_of_Employment.pdf. Geraadpleegd op 20 augustus 2018.

G

- Gamma, E., R. Helm, R. Johnson, J. Vlissides (1994). *Design Patterns: elements of reusable object-oriented software.* Reading: Addison-Wesley professional computing series. ISBN: 0-201-63361-2.
- Gastel, B. van (2016). *Assessing sustainability of software – analysing correctness, memory and energy consumption* (Proefschrift, Open Universiteit Nederland). Online beschikbaar: https://www.sustainablesoftware.info/download/thesis-met-cover.pdf. Geraadpleegd op 21 augustus 2018.
- Geelen, H., H. van Kranenburg, F. Beenker, W. Leibbrandt (Eds.) (2018). High Tech Systems and Materials (HTSM) Roadmap, Holland High Tech. Online beschikbaar: https://www.hollandhightech.nl/nationaal/innovatie/roadmaps/embedded-systems. Geraadpleegd op 21 augustus 2018.
- Genuchten, M. van (2006, 2 februari). Waarde van embedded software is beperkt. *Automatiseringsgids.* Online beschikbaar: https://www.agconnect.nl/artikel/waarde-van-embedded-software-is-beperkt. Geraadpleegd op 21 augustus 2018.
- Gool, L. van, T. Punter, M. Hamilton, R. van Engelen (2006). Compositional MDA. In O. Nierstrasz e.a., *ACM/IEEE MoDELS - Model Driven Engineering Languages and Systems, LNCS 4199 (pp. 126-139)*. Heidelberg: Springer.
- Graham, D., E. van Veenendaal, I. Evans, R Black (2007). *Foundations of software testing*. London: Thomson. ISBN: 1844809897.
- Grieves, M., J. Vickers (2017). Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In F. Kahlen e.a., *Transdisciplinary Perspectives on Complex Systems*. doi: 10.1007/978-3-319-38756-7_4
- Groote, J.F., M. Mousavi (2014), *Modeling and Analysis of Communicating Systems.* London: MIT-press. ISBN: 9780262027717.

H

- Hendriks D., R. Schiffelers, M. Hüfner, C. Sonntag (2011). A Transformation Framework for the Compositional Interchange Format for Hybrid Systems. In *Proceedings IFAC* (pp. 12509-12514). Elsevier. ISSN: 2405-8963.
- Hendriksen, W., A. Lak, G. Woolderink (2013), Positiebepaling in een Zigbee draadloos sensornetwerk. In: Bits & Chips. Online beschikbaar: http://www.avans.nl/binaries/content/-assets/nextweb/onderzoek/edi/robotica-en-mechatronica/2013-03-01-bc02-13-positiebepaling-in-een-zigbee-draadloos-sensornetwerk.pdf. Geraadpleegd op 21 augustus 2018.
- Hofmeister, C., P. Kruchten, R. Nord, H. Obbink, A. Ran, P. America (2007). A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software, 80*, 106-126.
- Holland Robotics (2018), Kansen voor de Nederlandse robotica (Position paper). Online beschikbaar: https://www.vraagenaanbod.nl/download/Holland%20Robotics%20Position%20Paper.pdf Geraadpleegd op 21 augustus 2018.
- Holzman, G. (2014), Mars Code. *Communications of the ACM, 57 (2)*, 64-73. doi: 10.1145/2560217.2560218

- Horváth, I., B. Gerritsen (2012). Cyber physical systems: concepts, technologies and implementation principles. In I. Horváth e.a., *Proceedings of TMCE 2012* (19-36). Karlsruhe: Germany: TMCE 2012. ISBN 978-90-5155-082-5.
- Huët, B. van, (2017, 22 augustus). Experts: stop de opkomst van killerrobots. Algemeen dagblad. Online beschikbaar: https://www.ad.nl/digitaal/experts-stop-de-opkomst-van-de-killer-robots~a557597a/. Geraadpleegd op 26 augustus 2018.
- Humble, J., D. Farley (2010). *Continuous Delivery.* Upper Sadle River: Addison-Wesley. ISBN: 978-0321601919.
- Hyatt, K. C. Paukert (2018, 29 maart). Self-driving cars: A level-by-level explainer of autonomous vehicles. CNET Roadschow. Online beschikbaar: https://www.cnet.com/roadshow/news/self-driving-car-guide-autonomous-explanation/. Geraadpleegd op 26 augustus 2018.

I

- IERC (z.j.). Clusterbook European Research Cluster on Internet of Things. Online beschikbaar: http://www.internet-of-things-research.eu/cluster_book.htm. Geraadpleegd op 23 augustus 2018.
- Incose (z.j.). *Systems Engineering handbook*. Online beschikbaar: http://sebokwiki.org/wiki/INCOSE_Systems_Engineering_Handbook. Geraadpleegd op 23 augustus 2018.
- In't Veld, R. (1988), *Analyse van organisatieproblemen – een toepassing van denken in systemen en processen.* Leiden: Stenfert Kroese. ISBN: 90 207 1689 1.
- ISO 25010 (z.j.). In Wikipedia. Online beschikbaar: https://nl.wikipedia.org/wiki/ISO_25010. Geraadpleegd op 27 augustus 2018.
- ISO/IEC 42010 (z.j.). In Wikipedia. Online beschikbaar: https://en.wikipedia.org/wiki/ISO/IEC_42010. Geraadpleegd op 26 augustus 2018.

J

- Jamshidi, M. (Ed.) (2009). *Systems of Systems Engineering: principles and applications.* Boca Raton: CRC Press. ISBN: 9781420065886.
- Jordan, M., T. Mitchell (2015). Machine learning – trends, perspectives and prospects. *Science,* 349(6245), 255-260. doi: 10.1126/science.aaa8415.

K

- Kabashkin, I., J. Kundler (2017), Reliability of Sensor Nodes in Wireless Sensor Networks of Cyber Physical Systems. *Procedia Computer Science,* 104, 380 – 384. https://doi.org/10.1016/j.procs.2017.01.149
- Kaner, C. (1997). *The impossibility of complete testing* (tutorial).Online beschikbaar: https://www.researchgate.net/publication/239062039_The_Impossibility_of_Complete_Testing Geraadpleegd op 24 augustus 2018.
- Kang, K., V. Marouas, I. Schizas, F. Bao (2018), Improved distributed particle filters for tracking in a wireless sensor network. *Computational Statistics and Data Analysis, 117,* 90-108. doi: http://dx.doi.org/10.1016/j.csda.2017.07.009
- Kim, K-D., P. Kumar (2012), Cyber-Physical Systems – A perspective at the centennial. IEEE Proceedings, 100, 1287-1308. doi: 10.1109/JPROC.2012.2189792.
- Kirkels, Y., G. Duysters (2010). Brokerage in SME networks. *Research Policy, 39 (3)*, 375-385.
- Kitchenham, B., T. Dybå, M. Jørgensen (2004).Evidence-based Software Engineering. In *Proceedings of 27th International Software Engineering Conference (ICSE)* (pp. 273-281). Washington DC: IEEE Computer Society. ISBN: 0-7695-2163-0.
- Klatter, E, K. Vloet, S. Janssen, G. Kessels e.a. (2016). *Didactiek van het Techniekonderwijs – Tekenend Techniek!* Eindhoven: Pedagogisch Technische Hogeschool (PTH).
- Kniberg, H., (2015), *Scrum and XP from the Trenches - How we do Scrum.* InfoQ. Online beschikbaar: https://www.infoq.com/minibooks/scrum-xp-from-the-trenches-2. Geraadpleegd op 24 augustus 2018.
- Koopman, P. (2014). A Case Study of Toyota Unintended Acceleration and Software Safety. Online beschikbaar: https://users.ece.cmu.edu/~koopman/pubs/-koopman14_toyota-_ua_slides.pdf. Geraadpleegd op 24 augustus 2018.
- Kruchten, P. (1995), Architectural blueprints – the '4+1' view model of software architecture. *IEEE Software, 12*, pp. 42–50.
- Kurtev, I., M. Schuts, J. Hooman. D-J. Swagerman (2017). Integrating Interface Modeling and Analysis in an Industrial Setting. In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)* (pp. 345-352) Porto: Scitepress. ISBN: 978-989-758-210-3.

L

- Laar, P. van de, T. Punter (Eds.) (2010). Views on Evolvability of Embedded Systems. Dordrecht: Springer. ISBN: 978-90-481-9848-1.
- Langendoen, K., A. Baggio, O. Visser (2006). Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In Proceedings of the 20th international conference on Parallel and distributed processing (pp. 174-174).
- Lederer, S., J. Mankoff, A. Dey (2003). Who Wants to Know What When? Privacy Preference Determinants in Ubiquitous Computing. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)* (pp. 724-725) New York: ACM.
- Lee, F., Y. Su, C. Shen (2007). A Comparative Study of Wireless Protocols : Bluetooth, UWB , ZigBee and Wifi. In Proceedings of IEEE Industrial Electronics Society (IECON), (pp. 46-51). Online beschikbaar: http://eee.guc.edu.eg/Announcements/Comparaitive_Wireless_standards.pdf. Geraadpleegd op 24 augustus 2018.
- Lee, E., S. Seshia (2015), I*ntroduction to Embedded Systems – A Cyber-Physical Systems Approach.* Berkeley: LeeSeshia.org. ISBN 978-1-312-42740-2.
- Leeuwen, H. van, R. Tangelder (2012). Wireless Sensortechnologie bij Calamiteiten, Saxion Kenniscentrum Design en Technologie. Online beschikbaar: https://www.saxion.nl/binaries/content/assets/onderzoek/smart-industry/ambient-intelligence/eindpublicatie-wireless-sensortechnology.pdf. Geraadpleegd op 24 augustus 2018.
- Leitner, P, S. Schulte, S. Dustdar, I. Pill, M. Schulz, F. Wotawa (2013), The Dark Side of SOA Testing - Towards Testing Contemporary SOAs Based on Criticality Metrics, In 5th International Workshop on Principles of Engineering Service-Oriented Systems (PESOS). doi: 10.1109/PESOS.2013.6635977
- Lemmen, K., T. Punter, M. Dicker e.a. (1993), *Methodologie van informatie-systeemontwikkeling.* Heerlen: Open Universiteit. ISBN: 90 358 1183 6.
- Lions, L., L. Lbeck, J-L Fauquembergue e.a. (1999), Ariane 5 Flight 501 failure (Report by Inquiry board). Online beschikbaar: http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html. Geraadpleegd op 26 augustus 2016.

M

- Marco, T. de, T. Lister (1987). P*eopleware – Productive projects and teams*. New York: Dorset House Publishing . ISBN: 0-932633-05-6.
- Martin, R. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship.* New York: Pearson. ISBN: 978-0132350884.
- Mataric, M.J. (2001). Learning in behavior-based multi-robot systems: policies, models and other agents. *Journal of Cognitive Systems Research, 2,* 81–93.
- McCandless, D., P. Doughley-While, M. Quick (2015), Code bases. Online beschikbaar:http://www.informationisbeautiful.net/-visualizations/million-lines-of-code/. Geraadpleegd op 24 augustus 2018.
- Mols, B. (2012). *Turings Tango – Waarom de mens de computer de baas blijft*. Amsterdam: Nieuw Amsterdam. ISBN 9789046812372.
- Mols, B. (2017, 6 november) Dit is Sophia, ze heeft een paspoort. *NRC Handelsblad,* p. C4-5.
- Mols, B., N. Vergunst (2017). Hallo robot – de machine als medemens. Amsterdam: Nieuw Amsterdam. ISBN: 978 90 468 2292 0.
- Moneva, H., R. Hamberg, R., T. Punter, J. Vissers (2010). Putting Chaos under Control: on how Modeling should Support Design. In *Proceedings of International Council on Systems Engineering (Incose).* Online beschikbaar: http://www.esi.nl/publications/multiform/0211.pdf. Geraadpleegd op 24 augustus 2018.
- Muller, G. (2012). *Systems Architecting – A business perspective.* Boca Raton: CRC Press. ISBN: 074-1-4398-4752-6.

N

- Nicolai, L.M., G. E. Carichner (2010). *Fundamentals of Aircraft and Airship Design.* Reston, VA: American Institute of Aeronautics and Astronautics. ISBN: 978-1600867514.
- Noort, W. van (2018a, 17 april), Moeten robots een juridische status krijgen? *NRC Handelsblad.* p. E1.
- Noort, W. van (2018b, 16 juni), AI mist nog zoiets als gezond verstand. In: *NRC Handelsblad,*. p. E2-3.

O

- O'Kane, J (2014). *A gentle introduction to ROS.* Columbia, SC: University of South California. ISBN: 978-14-92143-23-9. Online beschikbaar: https://www.cse.sc.edu/~jokane/agitr/. Geraadpleegd op 26 augustus 2018.
- Onderwater, L. (2018, 7 juni). Wat heeft de Marsrover Curiosity ontdekt? *Kijk magazine.* Online beschikbaar: https://www.kijkmagazine.nl/space/wat-heeft-de-marsrover-curiosity-ontdekt/. Geraadpleegd op 22 augustus 2018.
- Osaiweran, A., M. Schuts, J. Hooman, J.F. Groote, B. van Rijnsoever (2016), Evaluating the effect of a lightweight formal technique in industry. *International Journal on Software Tools for Technology Transfer, 18,* 93-108.
- Oskam, I., K. Cowan, L. Hoiting, P. Souren (2012). Ontwerpen van technische innovaties door onderzoek, creatief denken en samenwerken. Groningen: Noordhoff Uitgevers. ISBN: 9789001880590.

P

- Parnas, D.L. (2010). Really rethinking formal methods. IEEE Computer, 43(1), 28-34. doi: 10.1109/MC.2010.22
- Peek, S. (2018, 8 juli), Defensie blokkeert fitness-apps op diensttelefoons na privacyprobleem. *NRC Handelsblad.* Online beschikbaar: https://www.nrc.nl/nieuws/2018/07/08/defensie-blokkeert-fitness-apps-op-diensttelefoons-na-privacyprobleem-a1609288. Geraadpleegd op 26 augustus 2018.
- Pous, I. de (2008, 17 maart), Technologie verandert onze waarden. *Trouw*. Online beschikbaar: https://www.trouw.nl/home/technologie-verandert-onze-waarden~a46a75f7/. Geraadpleegd op 26 augustus 2018.
- Punter, T. (2001)., *Doelgericht beoordelen van software* (Proefschrift Technische Universiteit Eindhoven). ISBN: 90-38600863-2.
- Punter, T., R. Kusters, J. Trienekens, T. Bemelmans, A. Brombacher (2004). The W-Process for Software Product Evaluation: A method for goal-oriented implementation of the ISO 14598 standard. *Software Quality Journal,* 12 (2), 137-158. doi: 10.1023/B:SQJO.0000024060.32026.a2
- Punter, T., J. Voeten, J. Huang (2009a). Quality in Model Driven Engineering, Chapter 2. In J. Rech, C. Bunse (Eds.), *Model-Driven Software Development: Integrating Quality Assurance, Information Science Reference,* pp. 37-56. doi: 10.4018/978-1-60566-006-6.ch002
- Punter, R. Krikhaar, R. Bril (2009b). Software Engineering Technology Innovation: Turning Research Results into an Industrial Success. *Journal of Systems and Software, 82*, 993-1003.

Q

- Quigley, M. B. Gerkey, W.D. Smart (2015). *Programming robots with ROS – a practical introduction to the robot operating system.* Sebastopol, CA: O'Reilly. ISBN: 978-1-4493-2389-9.

R

- Riedmiller, M. (2016). Intelligence scores goals: Machine Learning for autonomous robots (Keynote presentatie). Leipzig:Robocup 2016. Online beschikbaar: http://www.ais.uni-bonn.de/robocup.de/2016/misc/RoboCup_Symposium_2016_Presentation_Martin_Riedmiller.pdf. Geraadpleegd op 24 augustus 2018.
- RIS3 (2013). *Smart Specialisation Strategy RIS3 Zuid – De Kunst van het combineren.* Online beschikbaar: https://www.stimulus.nl/opzuid/wp-content/uploads/sites/4/2015/02/RIS3.pdf. Geraadpleegd op 24 augustus 2018.
- Roos, N. (2015, 26 oktober). Multidisciplinair software ontwikkelen op een virtuele printer. Bits en Chips magazine. Online beschikbaar: https://www.bits-chips.nl/artikel/multidisciplinair-software-ontwikkelen-op-een-virtuele-printer-45358.html. Geraadpleegd op 24 augustus 2018.
- Royakkers, L., F. Daemen, R. van Est (2012). *Overal robots - Automatisering van de liefde tot de dood.* Den Haag: Boom Lemma/Rathenau. Online beschikbaar: https://www.rathenau.nl/sites/default/files/OveralRobots.pdf. Geraadpleegd op 21 augustus 2018.

S

- Schouten, A. (2017, 6 november). Robots zullen nooit beter zijn dan mensen. Robo Sapiens. Online beschikbaar: https://www.vpro.nl/programmas/robo-sapiens/lees/artikelen/2017/robotregels.html. Geraadpleegd op 24 augustus 2018.
- Sebok (z.j.). *Guide to the Systems Engineering Body of Knowledge.* Online beschikbaar: http://sebokwiki.org/wiki/. Geraadpleegd op 24 augustus 2018.
- Selic, B. (2000). A generic framework for modeling resources with UML. *IEEE Computer,* 33(6), 64-69.

- Serebrenik, A., T. Mens (2015). Challenges in software ecosystems research. In I. Crnkovic (Ed.), Proceedings of European Conference on Software Architecture Workshops (ECSAW) (pp. 1-6). New York: ACM.
- Siegwart, R. I.R. Nourbakhsh, D. Scaramuzza (2011). *Introduction to Autonomous Mobile Robots.* Cambridge MA: MIT press. ISBN: 978-0-262-01535-6.
- Solingen, R. van, E. Rustenburg (2010). *De kracht van Scrum.* Amsterdam: Pearson. ISBN: 9789043035095.
- Sommerville, I. (2015). *Software Engineering - 10th Edition*. London: Pearson. ISBN: 9781292096131.
- Sparc (2016), Robotics 2020 Multi Annual Roadmap for Robotics in Europe. European Horizon. Online beschikbaar: https://www.eu-robotics.net/cms/upload/topic_groups/H2020_Robotics_Multi-Annual_Roadmap_ICT-2017B.pdf. Geraadpleegd op 24 augustus 2018.
- Stone, P., R. Brooks, E. Brynjolfsson (2014). *Artificial Intelligence and Life in 2030 - One Hundred Year Study on Artificial Intelligence* (Report of the 2015-2016 Study Panel). Stanford, CA: Stanford University. Online beschikbaar: , http://ai100.stanford.edu/2016-report. Geraadpleegd op 24 augustus 2018.
- Sutton, R., A. Barto (2012). R*einforcement Learning: An Introduction.* Cambridge MA: MIT Press. Online beschikbaar: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf. Geraadpleegd op 24 augustus 2018.
- Swebok (z.j.). *Software Engineering Body of Knowledge.* Online beschikbaar: https://www.computer.org/web/swebok. Geraadpleegd op 24 augustus 2018.

T

- Tech United (z.j.). Tech United Robocup MSL team. Online beschikbaar: http://www.techunited.nl/. Online geraadpleegd op 26 augustus 2018.
- Theelen, B.D., M Geilen, J. Voeten (2011). Performance Model Checking Scenario-Aware Dataflow. In U. Fahrenberg, S. Tripakis (Eds.) *Formal Modeling and Analysis of Timed Systems*. (FORMATS) 2011, LNCS 6919 (pp. 43-59). Berlin: Springer.
- Tretmans, J. (2008). Modelbased testing with labelled transition systems. In R. Hierons e.a. (Eds). *Formal Methods and Testing, LNCS 4949* (pp. 1-38). Berlin: Springer. doi: https://doi.org/10.1007/978-3-540-78917-8_1.
- Turnhout, K. van, S. Craenmehr, R. Holwerda, M. Menijn, J.P. Zwart, R. Bakker (2013). Triangulatie: een basis voor de onderzoeksleerlijn in ICT en Media Onderwijs. In: Proceeding NIOC 2013 (pp. 277-286). doi: 10.13140/2.1.2872.1449.

V

- VDL Robotsports (z.j.). VDL Robotsport Robocup MSL team. Online beschikbaar: http://www.robotsports.nl. Online geraadpleegd op 26 november 2018.

W

- Waibel, M., M. Beetz, J. Civera, R. D'Andrea, J. Elfring, e.a. (2011), RoboEarth. *IEEE Robotics and Automation Magazine, 18* (2), 69-82. doi: 10.1109/MRA.2011.941632
- Webers, W., C. Thörn, K. Sandkuhl (2008). Connecting feature models and Autosar - an approach supporting requirements engineering in automotive industries. In B. Paech, C. Rolland (Eds). *Requirements Engineering: Foundation for Software Quality, Proceedings (REFSQ)* (pp. 95-108). Berlin: Springer.
- Werner-Allen, G., K. Lorincz, M. Welsh, O. Marcillo, e.a. (2006). Deploying a Wireless Sensor Network on an Active Volcano. *IEEE Internet Computing,* 18-25.
- Wohlin, C, Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. (2000), *Experimentation in software engineering.* Dordrecht: Kluwer. ISBN: 978-3-642-29044-2.
- Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., Wood, B., (2007), Attribute Driven Design, Software Engineering Institute (SEI), Carnegie Mellon University.
- Xu, G., W. Shen, X. Wang (2014), Applications of Wireless Sensor Networks in Marine Environment Monitoring: A Survey. *Sensors,* 14, 16932-16954. doi:10.3390/s140916932.

Y

- Yao, W, H. Lu, J. Xiao (2015). A simulation system based on ROS and Gazebo for RoboCup Middle Size League. In *Proceedings IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Zhuhai, China doi: 10.1109/ROBIO.2015.7414623.

Z

- Zestor (2018). *Wat doet een lector?* (Website Werken bij hogescholen). Online beschikbaar: https://www.werkenbijhogescholen.nl/functies/onderzoek-lector/wat-doet-een-lector/. Geraadpleegd op 26 augustus 2018.

LECTORALE REDE

SOFTWARE VOOR SLIMME SYSTEMEN

Een systeemvisie op de softwareontwikkeling van (mobiele) robots en sensornetwerken

DR.IR. TEADE PUNTER | 12.10.2018

Fontys

LECTORAAT
High Tech Embedded Software

Colofon

Fontys Hogeschool ICT
High Tech Embedded Software
Postbus 347
5600 AH Eindhoven

Lector:	dr.ir. Teade Punter
Uitgever:	Fontys Hogescholen
Vormgeving:	Happy Cactus
Versie:	4.5, 25 augustus 2018
Fotografie:	Teade Punter, Fontys ICT Video Crew (p. 8, p. 9) Audrey Kawarmala (p. 22) Stefan Grimminck (p. 53) Bas Gijselhart (p. 104)