Viable, adaptive software
Based on the Viable System Model

Pieter Buitenhuis

Abstract

Om levensvatbaar te zijn, moeten systemen (nieuwe) verstoringen uit de (directe) omgeving kunnen reguleren. Beer heeft, in zijn onderzoek naar levensvatbare systemen, het *Viable System Model* ontwikkelt.
In de informatica en informatiekunde wordt er al decennia gewerkt aan de ontwikkeling van kwalitatief goede software. Maar in de huidige tijd, waarin de wereld zo snel verandert, wordt de vraag naar adaptieve ICT steeds groter. Organisaties moeten snel mee veranderen en de ICT moet daarbij geen belemmering zijn. Sterker gesteld: ICT moet hierin juist een positieve bijdrage leveren. Er moet daarom toegewerkt worden naar adaptieve software om zo veel effectiever en efficiënter in te kunnen inspelen op de wensen uit de omgeving. Om dit te bereiken zou software ook *viable* moeten zijn. In dit essay wordt er daarom ingegaan op de vraag in hoeverre het VSM model kan bijdragen bij het realiseren van levensvatbare, adaptieve software.

Sleutelwoorden: Cybernetica, reguleren, levensvatbaarheid, adaptiviteit, VSM, software

1. Inleiding

Wij leven in een samenleving welke continu verandert. Dit heeft een grote invloed op ons dagelijks leven, maar ook op allerlei systemen [Proper] in onze samenleving. Zo stellen consumenten steeds nieuwe wensen en eisen aan producten en levert de technologische vooruitgang steeds nieuwe mogelijkheden waardoor organisaties mee moeten veranderen. Om goed te kunnen inspelen op veranderingen is een juiste inzet van ICT noodzakelijk [Rijsenbrij]. Gesteld moet dan ook worden dat onze huidige, continu veranderende samenleving steeds meer een software afhankelijke samenleving wordt. De laatste decennia is de ICT wereld op zoek naar het juiste antwoord op dit probleem: softwaresystemen welke, zelfstandig, in kunnen spelen op nieuwe wensen en eisen uit de omgeving.

Dit fenomeen, het snel kunnen inspelen op veranderingen, is in de bedrijfswetenschappen al veel langer een belangrijk onderzoeksterrein. Immers, organisaties moeten adaptief zijn om zichzelf staande te houden in deze complexe, veranderende omgeving [Beer]. Adaptiviteit wordt door Beer gedefinieerd als: "*de mate waarin een systeem zich zelfstandig kan aanpassen aan de veranderende omgeving om zodoende levensvatbaar te blijven*". Beer heeft op dit gebied baanbrekend onderzoek verricht, wat heeft geresulteerd in het *Viable System Model* (VSM) [Leonard][1]. Dit model, welke gebaseerd is op de cybernetica [Ashby], pretendeert dat ieder systeem, met coördinatie-, besturings- en vernieuwingsvermogen, adaptief en dus levensvatbaar is. Deze theorie is daarom zeer interessant in het onderzoek naar adaptieve software.

Adaptiviteit, zoals door Beer gedefinieerd, impliceert dat software het eigen gedrag moet kunnen veranderen naar gelang er nieuwe verstoringen, gebruikerswensen en –eisen, plaatsvinden zonder dat hiermee de integriteit van het systeem wordt verwaarloosd. Dit wordt *self-adaptive software* genoemd.
Gesteld moet worden dat software, wil het deze eigenschappen bezitten, moet kunnen leren. Het onderzoek op het vlak van *artificial intelligence* heeft in de laatste jaren echter een grote sprong voorwaarts gemaakt, waardoor *self-adaptiveness* van software geen illusie meer is.

In dit essay wordt er daarom ingegaan op de vraag: *"in hoeverre kan het Viable System Model bijdragen aan het realiseren van viable, adaptive software?"*.
Om deze vraag te beantwoorden zal er in sectie twee en drie ingegaan worden op het *Viable System Model* en de bijbehorende cybernetische grondslagen, waarna er in sectie vier en vijf een beschouwing volgt op de gestelde vraag. In sectie zes volgt de conclusie.

2. Viable System Model

In zijn onderzoek naar *viable systems* heeft Beer onderzocht door welke eigenschappen organismen en organisaties levensvatbaar zijn. Beer beschrijft de levensvatbaarheid van systemen als volgt: "*viability is maintained by engaging in different activities, keeping them from interfering with each other, managing them together, focusing on the future and doing so in the context of an identity within which the interests of the whole over time could be considered"*. Beer heeft deze eigenschappen onderverdeeld in een vijftal subsystemen[2], te weten *primary activities, coordination, control, intelligence* en *policy*. Deze subsystemen en de onderlinge relaties zijn vastgelegd in het *Viable System Model*.
In deze sectie zullen de kenmerken van deze subsystemen worden beschreven.

[1] Leonard beschrijft niet alleen het VSM model van Beer, maar ook de toepassing ervan op het terrein van het kennismanagement.
[2] Systemen bestaan zelf ook uit systemen, een subsysteem is zelf dus ook een systeem [Beer][Proper]

Systeem één: *primary activities*
Beer stelt dat organisaties en organismen een overlevingsdrang hebben. Deze systemen bevatten de capaciteit om te overleven in hun omgeving. Bij organismen komt dit tot uiting door onder andere biologische defensiemechanismen, bij organisaties komt dit naar voren in de capaciteit (activiteiten) om goede producten en/of diensten te leveren aan gewenste afzetmarkt. De missie en visie van een organisatie beschrijven de doelen van de organisatie voor het ontwikkelen van nieuwe producten en diensten.
Het eerste subsysteem bevat de benodigde activiteiten van het *viable system* om deze doelen te verwezenlijken.
Om dit effectief en efficiënt te bewerkstellingen moet dit subsysteem over de kennis bezitten om deze activiteiten uit te voeren. Deze kennis moet betrekking hebben op generieke zaken als marktpositionering en op specifieke aspecten als het koopgedrag van de afzetmarkt. Naast de kennis over de omgeving moet er tevens kennis aanwezig zijn over de doelen van de ondermening, de benodigde uitvoeringsmethoden om deze doelen te bewerkstelligen en de gevolgen van activiteiten. Beer geeft aan dat deze kennis aanwezig moet zijn om zo autonoom mogelijk te kunnen functioneren.
Ten slotte stelt Beer dat het mogelijk is om meerdere instanties van dit systeem te implementeren binnen organisaties om zo efficiënter de gestelde doelen te kunnen verwezenlijken.

Systeem twee: *coordination*
Aangezien er meerdere primaire activiteiten operationeel kunnen zijn binnen een organisatie is het noodzakelijk om de gezamenlijke voorzieningen van deze systemen te reguleren. In informatie intensieve organisaties [Proper] bestaan deze *resources* vooral uit informatiesystemen, werknemers en procedures [Leonard]. Het is de opgave van systeem twee om de variëteit aan resources op een dusdanige manier in te zetten zodat de primaire activiteiten (met een grotere variëteit) doorgang kunnen vinden. Leonard stelt dat dit systeem kennis nodig heeft over de afhankelijkheden tussen en de benodigde resource eisen van de primaire activiteiten.

Systeem drie: *control*
Dit systeem moet de doelen van het gehele systeem omzetten in doelstellingen, strategieën en uitvoerbare plannen voor de primaire activiteiten. Beer stelt dat systeem drie de voorwaarden moet scheppen om deze primaire activiteiten zo goed mogelijk te laten omgaan met de variëteit uit de omgeving. Daarnaast heeft dit systeem tot taak om de primaire activiteiten en het coördinatiesysteem bij te sturen waar nodig en om innovatieplannen van systeem vier te beoordelen.
Ten slotte moet systeem drie ook verschillende kwaliteit en performance audits kunnen uitvoeren op de primaire activiteiten, om zo een beeld te krijgen van de benodigde en geëiste resources. Dit noemt Leonard de "*three star function*". Leonard stelt dat er onder andere kennis nodig is over deze innovatieplannen, de gevolgen hiervan, de benodigde resources hiervoor en het verschil tussen de huidige en nieuwe doelstellingen van het gehele systeem en de primaire activiteiten.

Systeem vier: *intelligence*
Om als levensvatbare systeem te kunnen overleven in een snel veranderende omgeving is *intelligence* onmisbaar. Systeem vier onderzoekt de trends in de omgeving van het systeem en stelt innovatievoorstellen op. Deze innovatievoorstellen zijn het resultaat van risicoanalyses waarin verschillende scenario's worden geanalyseerd. De vernieuwende impulsen zijn zeker in informatie intensieve organisaties belangrijk omdat producten en diensten vaak aan verandering onderhevig moeten zijn om te blijven voldoen aan de wensen en eisen van de omgeving. Dit systeem moet dan ook ervoor zorgen dat het gehele *viable system* adaptief is en blijft. Leonard impliceert daarom dat er kennis aanwezig moet zijn over de ontwikkelingen in de relevante omgeving van de organisatie.

Systeem vijf: *policy*
Tussen systeem drie en vier bevindt zich een bepaald spanningsveld: *Control* moet de organisatie als geheel besturen en dit is onmogelijk door continu te veranderen. *Intelligence* zorgt juist voor innovatie, voor veranderingen. Om dit proces in goede banen te leiden, moet systeem vijf daarom de identiteit van het systeem bewaken en de relevante omgeving identificeren en daar waar nodig bijstellen. Aangezien het *policy* systeem belast is met het reguleren van de relatie tussen *control* en *intelligence*, moet er tevens kennis aanwezig zijn over de normen van deze relatie en de eventuele oorzaken en regelacties om een disbalans in de relatie weer te kunnen stabiliseren. Beer stelt dat deze relatie *homeostatic* moet zijn om als *viable system* goed te kunnen functioneren.

Recursie
Het *Viable System Model* is tevens een recursief model. Dit houdt in dat de subsystemen zelf ook *viable systems* moeten zijn, om zo op het laagste niveau de variëteit aan verstoringen te kunnen reguleren. Beer concludeert dit uit de *the Law of Requisite Variety* [Ashby], welke eist dat de variëteit van de regulator gelijk moet zijn aan die van de verstoringen .

3. Grondslagen cybernetica

Zoals al in de introductie is aangegeven, bevinden de fundamenten van het *Viable System Model* zich in de cybernetica. De cybernetica beschrijft een theoretisch framework waarin het reguleren en coördineren van systemen centraal staat.
Concepten als variëteit, verstoringen, levensvatbaarheid, reguleren, de *Law of Requisite Variety* en de homeostat zijn allen gedefinieerd door Ross Ashby in *"Introduction to Cybernetics"*. Stafford Beer heeft deze concepten vervolgens uitgewerkt op het terrein van de bedrijfswetenschappen. In deze sectie volgt een uiteenzetting van de concepten welke of van belang zijn bij het toelichten van het VSM of waar Beer aanpassingen op heeft verricht.

Proactief reguleren
Een systeem is levensvatbaar wanneer alle verstoringen worden geneutraliseerd. Ashby geeft hiermee de essentie van reguleren aan: het uitvoeren van regelacties om de variëteit van de verstoringen niet door te laten dringen tot de *essentiële variabelen*. Een perfect regelsysteem bevat de capaciteit **proactief** verstoringen waar te nemen en hierop te reageren. Regelsystemen moeten ook kunnen leren, om zo nieuwe verstoringen niet de kans te geven om de levensvatbaarheid aan te tasten. Beer stelt dat dit proactief reguleren alleen mogelijk is wanneer *viable systems* de subsystemen *intelligence, control* en *policy* bevatten.

Law of Requisite Variety
Het tegengaan van verstoringen, het doel van reguleren, kan ook uitgelegd worden als het reduceren van de variëteit van de verstoringen zodat de variëteit van de essentiële variabelen minimaal blijft. Dit concept is door Ashby vastgelegd in de *Law of Requisite Variety.* Hierin stelt Ashby dat als de variëteit van de verstoringen gegeven is en de variëteit van de essentiële variabelen verminderd moet worden, dit alleen mogelijk is door de variëteit van het regelsysteem te vergroten. Beer concludeert dat *viable systems* door recursie, autonomie en de vijf subsystemen de capaciteit van het regelsysteem kunnen vergroten.

Omgaan met variëteit - complexiteitsparadox
De *Law of Requisite Variety* stelt dat het regelsysteem van een *viable system* minimaal dezelfde variëteit moet bezitten dan die van de omgeving. Beer heeft echter vastgesteld dat organisaties toch de capaciteit bezitten om te kunnen overleven in veel complexere omgevingen zonder een, dusdanig, groot regelsysteem. Deze constatering is, samen met Beer's oplossing op deze paradoxale situatie, beschreven in de *complexiteitsparadox*.
Eigenlijk, zo stelt Beer, is de variëteit van de omgeving, waarmee een systeem moet omgaan, helemaal niet zo groot. Doordat een levensvatbaar systeem zichzelf doelen stelt, is een zeer groot gedeelte van de omgevingsverstoringen niet relevant om deze doelen te bewerkstelligen. Daarnaast heeft Beer de begrippen dempen (*attenuation*) en versterken (*amplification*) gedefinieerd. Versterken wordt door Beer beschreven als het uitbreiden van de regelacties. Echter, wat Ashby niet heeft beschreven is het dempen van verstoringen. Door voortijdige maatregelen te nemen om zodoende potentiële verstoringen niet te laten optreden, is het niet nodig om regelacties te implementeren. Zo dempen organisaties verstoringen door een goede organisatiestructuur[3] te realiseren, door kwaliteitsnormen te bewaken of door op creatieve en intuïtieve wijze voorwaarden te implementeren welke verstoringen tegengaan.

4. (Self-)adaptive software

Het VSM schrijft voor dat levensvatbare systemen kunnen anticiperen op veranderingen in de omgeving. Uiteraard moet dit dan ook gelden voor adaptieve software. Laddaga [Laddaga] definieert adaptieve software dan ook als volgt: "*Adaptieve software moet, om te blijven voldoen aan de veranderende omgeving, beschikken over voldoende capaciteit om, op een flexibele manier, het doel te bereiken en moet over voldoende kennis bezitten om runtime effectieve veranderingen door te voeren".*
Om aan deze eisen te voldoen, moeten dus ook softwaresystemen de capaciteit hebben om te leren. Om dit te verwezenlijken zijn er een aantal technieken in ontwikkeling op het gebied van *artificial intelligence,* waaronder de ontwikkeling van *agents* technologie. In dit essay[4] wordt het concept *agent* als volgt gehanteerd: "*a hardware or (more usually) software-based computer system that enjoys the properties of autonomy, social ability, reactivity, pro-activeness*" [Wooldridge, Jennings].
Een *agent* heeft dus de capaciteit om zelfstandig te opereren binnen een omgeving, te communiceren met andere *agents*, de omgeving zelfstandig waar te nemen en op eigen initiatief kunnen handelen (*proactief reguleren*). Een *self-adaptive agent* kan dan ook gedefinieerd worden als een agent met autonome, adaptieve, samenwerkende, flexibele en lerende eigenschappen.

Architectuur van Viable Software
Om *viable software* te ontwikkelen moet, volgens Beer, dit type software over de vijf subsystemen beschikken welke in het *Viable System Model* zijn beschreven. Na het definiëren van het concept *agents*[5] moet geconcludeerd worden dat dit de belangrijkste bouwstenen moeten worden van de softwarearchitectuur van *viable, self-adaptive software*.
Agent drie is belast met de taak om de doelen van het gehele systeem om te zetten naar uitvoerbare plannen voor de primaire activiteiten. Hierbij moet agent drie de mogelijkheid hebben om audits en kwaliteitscontroles uit te voeren op de werkzaamheden van agent één, welke belast is met de operationele uitvoering van de doelen, om zo beter in te kunnen spelen op de uitvoerbaarheid van veranderingen. De uit te voeren activiteiten door agent één moeten uiteraard worden gepland en gecoördineerd door agent twee.
Agent vier is belast met het classificeren van veranderingen in de omgeving als zijnde *threats* of *oppertunities* en moet uit deze analyses innovatieplannen genereren. Bij het onderzoeken van de omgeving moet agent vier een model van de omgeving hanteren welke door agent vijf wordt aangereikt.
Agent vijf bepaald de doelen voor het gehele systeem en moet ervoor zorgen dat agent vier zich alleen op de essentiële omgevingselementen kan richten, door een omgevingsmodel op te stellen welke gebaseerd is op de doelen van het systeem. Om de juiste doelen te bepalen en om innovatievoorstellen te keuren heeft agent vijf ook informatie nodig van agent drie over de al uitgevoerde activiteiten.
Uiteraard volgt uit de recursie van het VSM dat iedere agent zelf ook een *viable system* moet zijn.

[3] De sociotechniek [Sitter] is hier op gebaseerd
[4] Het valt buiten de scope van dit essay om een literatuurstudie te doen naar het concept *agents*
[5] Conventie: "Agent" *x* : agent welke subsysteem *x* vervuld uit het VSM

Implementatievraagstukken
Uit bovenstaande beschrijving blijkt dat het mogelijk is om *agents* te gebruiken bij het ontwikkelen van *viable software* op basis van het *Viable System Model*. Echter, er zijn een aantal belangrijke implementatievraagstukken te benoemen.

Complexiteitsparadox
Zoals gesteld is in de *complexiteitsparadox*, moet de software, om levensvatbaar te zijn, zelfstandig (nieuwe) doelen kunnen stellen, op deze doelen kunnen inspelen en nieuwe demp- en versterkacties kunnen ontwikkelen. De vraag is echter hoe agent vijf deze (levensvatbare) doelen en de, daarbij horende, relevante omgeving moet bepalen. In hoeverre mag de agent zelf aanpassingen doen aan deze doelen en omgevingsgrenzen? Hoe kan agent vijf de slagingskans van een innovatievoorstel bepalen?
Zijn agents in staat om op creatieve en intuïtieve wijze demp- en regelacties te kunnen ontwerpen? Luca Consoli geeft aan dat software al wel de deductie- en inductieprocessen kan simuleren, maar (nog) geen intuïtie heeft.

Waarnemen en interpreteren van relevante omgeving
Agent vier moet veranderingen in de omgeving waarnemen en deze interpreteren. Hoe bepaalt een agent wat zijn relevante omgeving is? Is een verandering in deze omgeving een *opportunity* of een *threat* en hoe kan agent vier deze onderscheiden?
Wij mensen hebben de vaardigheid om met behulp van selectie, abstractie en intuïtie hiermee om te kunnen gaan, agents kennen geen intuïtie [Consoli]. Hoe moet agent vier een geheel nieuw type verstoring identificeren, maatregelen (dempen/versterken) hierop ontwerpen en implementeren?

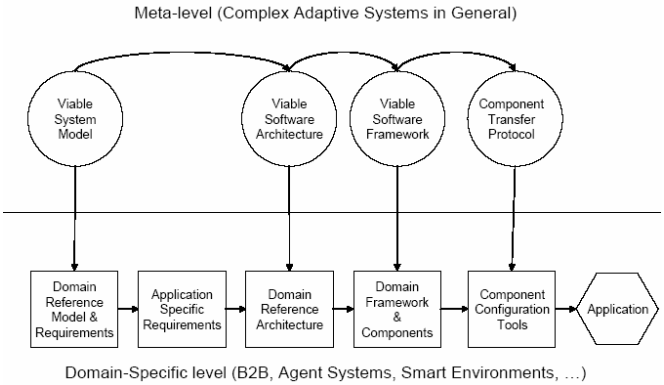
Zelfstandig veranderen
Agent drie moet ervoor zorgen dat de centrale doelstelling (de *essentiële variabelen*) door instanties van systeem één worden opgevolgd. Hoe ontwikkelt agent drie nieuwe uitvoeringsplannen om deze nieuwe doelen te kunnen bewerkstelligen?
Beer stelt dat *viable systems* zelfstandig moeten kunnen veranderen om in een veranderende omgeving te kunnen overleven. Om software de capaciteit te geven om te veranderen, moeten de agents zelfstandig nieuwe modellen kunnen ontwerpen en executeerbare code kunnen schrijven en implementeren. Dit is nu nog niet mogelijk.

Uit de hierboven gestelde vragen en een korte literatuurstudie naar het bestaan van adaptieve software, moet geconcludeerd worden dat het op dit moment nog niet mogelijk is om *viable* softwaresystemen te ontwikkelen.
Echter, zoals er reeds is aangegeven, is er op dit moment ook behoefte aan adaptieve ICT. Aangezien de implementatievraagstukken zich vooral toespitsen op de gebieden van *policy*, *intelligence* en *control* en mensen deze benodigde capaciteiten wel bezitten, is het een logische stap om te zoeken naar een architectuur waarin mensen deze rollen (tijdelijk) vervullen.
In de volgende sectie zal hier kort op in worden gegaan, naar aanleiding van Charles Herring's [Herring] onderzoek.

5. Viable Software Approach

Herring hanteert de volgende definitie: "*Software Viability is the quality a software system has if it can be adapted over time by humans (design-time adaptable) toward becoming an adaptive (intelligent supervisory-adaptive-control) system (run-time a-daptive)"*. Herring richt zich dus meer op de architectuur en ontwerpprocessen van informatiesystemen[6] dan op het daadwerkelijk adaptief zijn van de software zelf. Als basis heeft Herring ook het *Viable System Model* gehanteerd aangezien ook Herring een *viable system* wil ontwikkelen.
Herring heeft, tijdens zijn onderzoek, een *Viable Software Approach* ontwikkeld welke de voorwaarden moet scheppen om , in de toekomst, complexe adaptieve software te ontwikkelen. Deze aanpak biedt inzichten voor het definiëren van de requirements, de architectuur, de frameworks, componenten en configuratie tools om een *viable* applicatie te ontwikkelen en te onderhouden. Verder heeft Herring communicatiepatterns in XML ontworpen om gestandaardiseerde communicatie op gang te brengen tussen de verschillende componenten.

Herring gebruikt het VSM om de softwareprocessen en de ontwikkelorganisatie te stroomlijnen. Het *domain framework* is de brug tussen architectuur en ontwerp. De *domain architecture* bevat de eigenschappen van het *Viable System Model* en is grotendeels gelijk aan de architectuur in sectie vier, afgezien van het feit dat de, componentgebaseerde, architectuur[7] dusdanig is opgezet zodat de subsystemen drie, vier en vijf ook door menselijke actoren uitgevoerd kunnen worden.
Herring heeft binnen de *Viable Software Approach* vijf verschillende "*viability*" niveau's gedefinieerd waarin een informatiesysteem zich kan bevinden. Dit zijn respectievelijk *control*, *regulation*, *auditing*, *adaption* en *supervision*, verwijzend naar het *Viable System Model* en de cybernetica.



[6] Een informatiesysteem bestaat uit mensen, procedures, computers, programma's en bestanden [Rijsenbrij]
[7] Deze architectuur bestaat uit componenten welke zelf ook viable systems zijn (recursie)

Met deze aanpak zijn er reeds een aantal case studies[8] gedaan, onder andere voor een e-commerce oplossing [Herring]. Helaas is het in deze case studies nog niet mogelijk gebleken om de softwaresystemen nieuwe *use-cases* te laten ontwikkelen, aangezien menselijke actoren de *intelligence, control* en *policy* rollen vervulden.
Deze e-commerce oplossing moet standaard geleverd worden met een overvloed aan functionaliteit aangezien er zeer veel verschillende type actoren het systeem moeten kunnen gebruiken. Zo heeft een topmanager behoefte aan andere functionaliteit dan een potentiële koper of een medewerker van de logistieke afdeling. Verder moet het systeem 'passen als een prothese' [Rijsenbrij], waardoor gebruikers de eigen persoonlijke voorkeuren moeten kunnen instellen.
Een bekend probleem bij deze grote systemen is het, op maat, inrichten van de interfaces, aangezien er veel tijd overheen gaat voordat de interfaces volledig geaccepteerd zijn door alle verschillende gebruikers.
Dit systeem, gebaseerd op Herring's *viable architecture*, bepaald zelfstandig welke use-cases een gebruiker te zien krijgt en hoe deze in de interface worden gepresenteerd. De gebruiker is hierin de bron van *verstoringen* (het frequent gebruiken van opties, historie) en de ontwikkelaars geven hierbij de *policy* aan. De *intelligence* en *control* rollen worden vervuld door mensen uit de organisatie en het ontwikkelteam en kunnen, wanneer gewenst, innovatieplannen gemakkelijk ontwikkelen en implementeren. Hoewel dit niet het summum van adaptiviteit betreft, is het toch al een goede stap in de richting aangezien een organisatie nu een groot, uitgebreid, systeem kan implementeren en de interfacevraagstukken aan de gebruiker en het systeem zelf kan overlaten.
Door de gekozen architectuur is tevens mogelijk om componenten in het systeem te vervangen, zonder dat het systeem *offline* hoeft.

6. Conclusie

In dit essay is er ingegaan op de vraag in hoeverre het *Viable System Model* (VSM) als basis kan dienen voor het realiseren van *viable software*. Na een beschrijving van het VSM en de cybernetische grondslagen, is er in sectie vier, na het definiëren van *agents*, een architectuur beschreven welke alle elementen van het VSM bevat. Zodoende moet geconcludeerd worden dat het in theorie mogelijk is om *viable software* te ontwikkelen. Immers, Beer stelt dat wanneer een systeem deze vijf subsystemen bezit een dergelijk systeem levensvatbaar is. Hierin is een softwaresysteem gelijk aan een organisme, organisatie of mens.
In sectie vier is echter tevens geconcludeerd dat de *agents* technologie op dit moment niet ver genoeg doorontwikkeld is om de *control*, *intelligence* en *policy* systemen succesvol te implementeren.
Het werk van Henning biedt echter een oplossing voor deze problemen. Herring heeft een architectuur ontwikkeld waarin de systemen uit het *Viable System Model* zijn verwerkt, waarbij deze systemen ook door een menselijke actor uitgevoerd kunnen worden. In de loop van de tijd kunnen, wanneer de technische vraagstukken zijn opgelost, deze menselijke actoren dan vervangen worden door software actoren (*agents*).

Alles samenvattend moet geconcludeerd worden dat het VSM als basis moet dienen voor *viable, self-adaptive software*, aangezien dit type software moet kunnen functioneren als alle andere *viable systems*. Er zijn echter nog een aantal belangrijke implementatievraagstukken op te lossen. Doordat de vraag naar adaptieve ICT nu al bestaat, moet er gezocht worden naar een tussenoplossing. Charles Herring heeft deze tussenoplossing al vorm gegeven in zijn *Viable Software Approach*, maar ook het werk op het terrein van digitale architectuur van onder andere Daan Rijsenbrij belooft veel goeds voor *adaptable*[9] ICT .

Literatuur

Ashby, W.R.	An introduction to Cybernetics. Chapman & Hall, London, 1957.
Beer, S.	Heart of Enterprise, John Wiley & Sons, Chichester, 1979.
	Diagnosing the System for Organizations, John Wiley & Sons, Chichester, 1985.
Consoli, L.	Lecture-notes ICT & Samenleving, Radboud Universiteit, Nijmegen, 2005.
Proper, H.A.	Information Intensive Organisations. Radboud Universiteit, Nijmegen, 2005.
Herring, C.E.	Viable Software, University of Queensland, Brisbane, 2002.
Laddaga, R.	Self Adaptive Software: a position paper, Massachusetts institute of technology, 2000/2001
Leonard, A.	A Viable System Model: Consideration of Knowledge management, American Society of Cybernetics, 1999.
Rijsenbrij, D.	Architectuur in de Digitale Wereld: versie nulpuntdrie, Radboud Universiteit, Nijmegen, 2005.
Sitter, de	Op weg naar nieuwe fabrieken en kantoren, Kluwer, Deventer, 1981
Wooldridge, Jennings	Intelligent Agents: Theory and Practice, gepubliceerd in Knowledge Engineering Review, 1995.

[8] Herring beschrijft in zijn thesis rapport nog twee case studies
[9] Let wel: adaptable is niet hetzelfde als adaptive.