Van toekomstvast naar toekomstvloeibaar

Wat ik als verkeerskundige geleerd heb van programmeren in iOS

Berend Schotanus
First Flamingo Enterprise B.V.
BS@BSchotanus.nl

Bijdrage aan het Colloquium Vervoersplanologisch Speurwerk
22 en 23 november 2012, Amsterdam

Samenvatting

In deze paper grijpt de auteur het CVS thema “robuuste infrastructuur” aan om zijn eigen zoektocht naar meer betrouwbare systemen te beschrijven.

In de jaren ’90 en ’00 was hij werkzaam als vervoersontwikkelaar voor de spoorwegen en had daar te maken met een systeem dat uitermate kwetsbaar was voor een type incident die je zou kunnen beschrijven als “complexe kettingreacties”. Hij ging op zoek naar systemen en technieken die hier minder gevoelig voor zijn en kwam uit bij Apple.

Het blijkt dat computerprogrammeurs in de afgelopen decennia methoden en technieken hebben ontwikkeld om de complexiteit in hun systemen te beheersen. Een daarvan is de techniek van object-georiënteerd programmeren, dat door Apple uitgebreid wordt toegepast. Kenmerkend voor deze methoden is dat ze zich richten op de relaties tussen componenten van een systeem, niet zo zeer op de essentie van afzonderlijke componenten. De auteur beschrijft deze werkwijze als krachtig maar ook verwarrend.
Hij contrasteert dit met de gangbare Westerse, in de Verlichting gewortelde, denkwijze die waarheid zoekt in de essentie van afzonderlijke componenten. De natuurkunde, die met natuurwetten het gedrag van de materie waaruit componenten zijn opgebouwd beschrijft, is onderdeel van deze denkwijze. Het feit dat deze denkwijze gefundeerd is in de fysieke werkelijkheid geeft houvast aan ons denken. Maar eigenlijk is het niet zo gek dat je bij deze manier van denken ook kwetsbaar bent voor incidenten die worden veroorzaakt door falen in de onderlinge samenhang van componenten.

Op de vraag of robuuste infrastructuur dan mogelijk is verwijst hij naar de materialistische associatie die zowel het woord “robuust” als het woord “infrastructuur” oproepen. Dat is een uitnodiging om juist de oude, kwetsbare, denkwijze te volgen. “Robuuste infrastructuur” zou wel eens niet kunnen zijn wat je denkt dat het is.

1. Inleiding

In de nacht van donderdag 2 op vrijdag 3 februari 2012 viel tien centimeter sneeuw in Nederland. De volgende ochtend liep de treindienst vast.
Als we praten over “robuuste” verkeerssystemen dan is dit waarschijnlijk wat we willen voorkomen: het type incident waarbij je dacht alles goed voor elkaar te hebben en waarbij vervolgens toch, met een complexe kettingreactie, het systeem plat gaat. Niet een beetje, niet wat haperingen, nee, gewoon plat, over uit.

Dit type incident is, zeker voor de spoorwegen, niet nieuw. Vaak wordt gewezen op de rigiditeit van het systeem. Toen ik mij in de jaren ’90 en ’00 bezig hield met het ontwikkelen van toekomstplannen voor de spoorwegen brak ik mijn hoofd over de vraag hoe dit toch mogelijk was en hoe het beter zou kunnen.

Ik ging met steeds meer belangstelling andere sectoren volgen, luchtvaart, ICT. Wat deden zij anders? Wat konden de spoorwegen daar van leren?
In de ICT was de meeste ontwikkeling. Ik ontdekte dat ook binnen de ICT grote verschillen bestonden, dat ook ICT systemen hopeloos kunnen vastlopen. En ik ontdekte het bedrijf dat zich hier schijnbaar aan wist te onttrekken: Apple.
In 2005 kocht ik mijn eerste Mac, wat toen een grote stap was. Het apparaat voelde anders dan een Windows PC: betrouwbaarder, vriendelijker maar ook ondoorgrondelijker, hautain. Anders, op een manier die je niet precies kon thuis brengen.
Ik liet me leiden door mijn nieuwsgierigheid. In 2007 kwam de iPhone uit. In 2008 opende Apple de App Store en promootte expliciet de mogelijkheid voor externe ontwikkelaars om voor de iPhone te programmeren. Het zelf programmeren van zo’n apparaat leek mij een mooie mogelijkheid om te snappen waar ze hun magie vandaan halen. Nou ja... dat is een beetje uit de hand gelopen.

Terwijl ik mij stevig had ondergedompeld in de wereld van Apple, sprak ik af en toe ex-collega’s van de spoorwegen. Tijdens zo’n gesprek kwam de vraag (ik weet niet meer waar over): “Is dat wel toekomstvast?”
Het was zo’n moment dat je ineens in een andere wereld duikelt. “Toekomstvast” - wat een prachtwoord, wat een lading - dat woord had ik in geen jaren meer gehoord. Ik realiseerde me dat dat woord, toen ik nog bij de spoorwegen werkte, ook op het puntje van mijn tong lag. Het feit dat het zo vreemd voor me geworden was impliceerde dat ik een hele ontwikkeling doorgemaakt had.

In deze paper wil ik, vanuit mijn persoonlijke ervaring, een indruk geven van deze twee denkwerelden. Denkwerelden die verschillend zijn maar toch beide geworteld in een uitermate technische context. En ik wil laten zien waarom het, werkend vanuit de denkwereld van de spoorwegen, eigenlijk heel logisch is dat je kwetsbaar bent voor het bovengeschetste type incidenten.

2. De spoorwegtraditie: vast

De spoorwegtraditie is ontstaan in de negentiende eeuw en heeft veel elementen van de toenmalige militaire traditie (zoals discipline en hiërarchie) overgenomen, simpelweg omdat dat toen de enig bekende manier was om een grote organisatie te besturen. De militaire traditie is in hevige mate beïnvloed en veranderd door gebeurtenissen in verschillende oorlogen. De spoorwegtraditie heeft veel meer in zuivere vorm kunnen voortbestaan.

Een ander element in de spoorwegtraditie waarin de negentiende eeuw helder doorklinkt is dat het een mechanistische manier van denken is. De spoorwegen zijn een uitermate fysiek bedrijf: stalen flenzen aan stalen wielen houden tonnen wegende treinen strak op stalen rails.
Het kost niet veel moeite je voor de geest te halen hoe ooit ijzeren drijfstangen ijzeren wielen in beweging brachten, op gang gebracht door stoom. Of om te zien hoe seinen en wisseltongen met staalkabels in beweging gebracht werden om zo het treinverkeer te sturen. Je ruikt de kolen, voelt roet en smeerolie aan je handen kleven. Het is niet gek dat de spoorwegen een mechanistische manier van denken hebben, het bedrijf is een grote verzameling van aan elkaar hangende mechanieken.
En het kost ook niet veel moeite hier het denken van grote natuurkundigen als Newton of Watt in te zien doorschemeren. Het feit dat je eigenschappen van materie kunt reduceren tot simpele mathematische functies, dat je zo als het ware tot een diepere meer fundamentele waarheid kunt doordringen, dat is wat het creëren van deze ijzeren mechanismen überhaupt heeft mogelijk gemaakt.

Het is verleidelijk de spoorwegen te kwalificeren als achterlijk. Maar het is een manier van denken die in de hele maatschappij terug komt, ontstaan is in de negentiende eeuw, op de golven van enorme technologische successen en stevig gefundeerd op het westerse verlichtingsdenken. Het is een intrinsiek mechanistische en materialistische manier van denken, oprecht op zoek naar een fundamentele kern van waarheid.

3. Spoorwegen en ICT: gemeenschappelijke roots

In 2006 kreeg ik de gelegenheid beveiliginsinstallatie van een metro te bezoeken: grote rekken met relais en andere elektronica in een betonnen kelder. Een technische ruimte, TL-buizen, ventilatoren, kaal, weinig mensen zullen hier iets aantrekkelijks in gezien hebben. Maar voor mij was het een bijzondere ervaring.

Ik probeerde mij voor te stellen hoe het hart van een computer werkt, de microprocessor, het onderdeel dat geëtst wordt op een plaatje silicium van een paar vierkante millimeter en dat alleen maar bekeken kan worden door een elektronenmicroscoop. En hier bevond ik mij in een “processor” die groter was dan ikzelf, die je kon zien, horen, ruiken, die in vol bedrijf was bovendien. Het was of ik zelf geslonken was tot minder dan een micrometer en in een microprocessor mocht rondkijken.

De technische ruimte bevond zich direct boven de metrotunnel. Beneden je kon je de metrotreinen horen rijden, vertrouwde geluiden die iedere metroreiziger kent. Je kon je direct een beeld vormen bij het proces dat bestuurd werd.
Je kon zien hoe de kabels uit de tunnel de technische ruimte binnen kwamen en voerden naar een rek die “treindetectie” heette. Telkens als een trein onder je door reed tikten deze relais. Dat was hun rol, als een trein zich op een bepaald spoor bevond moest een stroom op gang gebracht worden of juist onderbroken.
Vanuit de treindetectie gingen kabels naar een paar rekken met heel veel kleine relais. Aan de achterkant van deze rekken was een wirwar van kleine, dunne kabeltjes. Het was een intelligente installatie die zelf seinen en wissels kon bedienen. Het waren deze kabeltjes die voor de intelligentie zorgden. Een fragiel werkje dat ik natuurlijk niet kon overzien maar ik kon me wel precies voorstellen hoe je zo een bepaalde input en output met elkaar kon verbinden.
Computerexperts zouden zeggen: het was geen programmeerbare machine, alle intelligente was in harde bekabeling uitgevoerd zodat maar één “programma” kon worden gedraaid, namelijk het besturen van de metro. Maar juist die beperking maakte dat je het nog net kon bevatten.
Aan de andere kant stond weer een rek met grote relais, hier werden wissels bestuurd. Nog boeiender waren de “ATB-oscillatoren”. De metro’s kregen met toonfrequente signalen een snelheidscode door. Die signalen werden gemaakt in een soort synthesizer. Er zaten geen luidsprekers aan maar door de sterkte van de stromen kon je ze toch zachtjes horen zingen. Iedere keer als een trein voorbij kwam veranderde de seinstand, dus de snelheidscode, en hoorde je andere tonen uit deze synthesizers komen.
Hiermee was de cirkel gesloten: de bewegende treinen, het zware getik van de treindetectie, daar vandaan het veel lichtere geruis van de kleine relais en tenslotte de zingende toon van de synthesizers die weer terug ging naar de trein.

Deze installatie was de perfecte hybride tussen de door mechanieken gedreven spoorwegwereld en de complexe maar onbegrijpelijke computerwereld. Geavanceerd, hij deed dingen die je eigenlijk al niet meer kon bevatten. Maar gebouwd op een menselijke schaal van onderdelen die je stuk voor stuk kon zien, voelen, ruiken, vastpakken, uitproberen.
Ieder onderdeel op zich was volkomen te begrijpen, ze deden niets anders dan de stalen kabels van de wisselwachter ooit gedaan hadden. Geen enkel onderdeel was op zich bijzonder, het was pas in onderlinge samenhang dat een complexe, krachtige, ondoorgrondelijke machine ontstond. De kleine relais kostten een paar kwartjes per stuk maar de man die de bedrading achter die relais goed kon krijgen, daar moesten ze de halve aardbol voor afstruinen.

Officieel is de computer geen afstammeling van de spoorbeveiliging. De eerste computers werden ontwikkeld voor militaire toepassingen en er werd al snel (vanwege de snelheid) gebruik gemaakt van radiobuizen in plaats van relais. Maar de principes waren gebaseerd op mechanieken die al veel langer in gebruik waren in klokken, ponskaartlezers en - inderdaad - spoorwegsystemen.

4. Leren programmeren

Mijn eerste programmeer ervaring was toen ik een jaar of zestien was op de programmeerbare rekenmachine van mijn vader. Vervolgens kon ik meedoen aan het eerste computer practicum dat mijn middelbare school ooit gegeven heeft: met een experimentele programmeertaal op een P2000 van Philips.
Zelf programmeren werd gezien als de logische en enige manier om met computers om te gaan. Het was niet heel anders dan de wiskunde of natuurkunde formules die je leerde, die moest je ook gewoon zelf toepassen. Dat een computer mijn zelf geschreven programma's zou draaien was het ook enige wat ik er van verwachtte, dat je kant en klare programma's zou kunnen draaien of zelfs kopen was een verwarrende gedachte.
Ik heb het altijd leuk gevonden om te programmeren, ik dacht niet dat het veel werk zou zijn om te programmeren, wel dat het mij zou helpen om andere taken sneller en makkelijker uit te voeren.

In feite werd ik afgeschermd voor de complexiteit van de computer, ik kreeg behapbare opgaven in een afgebakende omgeving. Iets wat naderhand in veel sterkere mate zou gebeuren toen er wel kant en klare programma's beschikbaar kwamen. Ineens was het *not done* om zelf te programmeren, je gebruikte een computer om een tekstverwerker of spreadsheet op te draaien. De computer was een geavanceerde typemachine geworden.

Een belangrijk aspect van het programmeren heb ik daardoor nooit meegekregen: dat het echt heel erg moeilijk kan zijn. Dat je, naarmate een programma groter of ouder wordt, tegen een niveau van complexiteit aan kunt lopen waar je helemaal in verstrikt raakt.
Toen ik jaren later het programmeren weer oppakte dacht ik dat ik met mijn jaren '80 programmeer kunsten de essentie toch wel snapte. Dat was wat naïef gedacht. Het was een merkwaardige ervaring om, toen ik ging lezen over de actuele stand van zaken, oplossingen te leren voor problemen waar ik me amper bewust van was.

In de jaren '80 was programmeren vrij letterlijk een verlengstuk van de wiskunde. In de wiskunde is het de kunst je formules zo te herschrijven dat je je probleem kunt reduceren tot een meer simpele, meer essentiële waarheid. Dat herschrijven lukt niet altijd, wiskunde is nu eenmaal moeilijk en de mens is beperkt. Maar intuïtief vertrouwde je er op dat die waarheid achter de schermen wel beschikbaar is, zoals de kerkganger er op vertrouwt dat God zijn ziel onder zijn hoede neemt.
Met een computer kon je je formules een stuk sneller uitrekenen. Het leek dus logisch dat je de achter de horizon glorende waarheid alleen maar sneller zou vinden.

In actuele literatuur over programmeren is een dergelijke notie niet aanwezig. Informatica is een op zichzelf staand vak dat helemaal niet zo sterk met wiskunde is verbonden. Programmeer-technieken hebben een overkoepelend thema: ze ondersteunen de programmeur om complexiteit de baas te blijven. Niks geen achter de horizon glorende waarheid, zorgen dat je niet verdrinkt!

5. De magie van object-oriëntatie

Een van de beroemde mythes van Silicon Valley vertelt hoe Steve Jobs in december 1979 de research laboratoria van XEROX in Palo Alto bezoekt en daar kennis maakt met de grafische gebruikersinterface: het besturen van een computer met een muis, menu's en iconen in plaats van met toetsenbord commando's. De grafische gebruikersinterface zal in 1984 met het uitkomen van de Apple Macintosh een revolutie veroorzaken en draagt uiteindelijk in de jaren '90, door Windows, bij aan het mainstream worden van de PC. Maar tegelijk met de grafische gebruikersinterface maakte hij kennis met nog twee andere technieken die aanvankelijk minder op de voorgrond stonden maar uiteindelijk minstens zo belangrijk zouden zijn: computernetwerken en objectoriëntatie.
Nadat Steve Jobs Apple Computer moest verlaten heeft hij een nieuw bedrijf opgericht: NeXT computer. Dit bedrijf heeft een computer ontwikkeld waarvan het besturingssysteem, NeXTSTEP, gefundeerd is op de principes van objectoriëntatie. NeXT computer zelf is nooit een succes geworden maar is toch om twee redenen van belang. Ten eerste is het de computer die Tim Berners-Lee heeft gebruikt bij het ontwikkelen van het wereldwijde web. In de architectuur van het web zijn vele elementen van objectoriëntatie terug te vinden. Ten tweede is NeXT computer in 1997 door Apple overgenomen waar NeXTSTEP de basis zou vormen voor de nu door Apple gebruikte besturingssystemen OS-X en iOS.

Objectoriëntatie is een van de technieken die helpen om complexiteit te beheersen. In essentie is een "object" een programmamodule die zowel eigenschappen als methodes kan bevatten. Een object kan gebruikt worden om daadwerkelijke objecten in de buitenwereld mee te modelleren. Je zou een hond kunnen modelleren met "vachtkleur" als eigenschap en "blaffen" of "eten" als methodes.
Een object is erg handig omdat je er van elkaar afgescheiden modules mee kunt maken. Als een programma groter wordt, word je er gek van als alles met alles samenhangt: Je lost één probleem op en krijgt er op onverwachtse wijze drie nieuwe voor terug. Het is dus fijn als je stukjes programma kunt opsluiten in afzonderlijke modules die elkaar niet beïnvloeden. Maar die modules moeten vervolgens wel weer met elkaar samenwerken, anders heb je alleen maar een verzameling kleine programmaatjes. Dat is precies wat objecten mogelijk maken. Objecten hebben een interface naar buiten zodat verschillende objecten onderling kunnen interacteren. De interactie tussen objecten vindt in principe op een hoger abstractieniveau plaats dan de problemen die binnen een object worden opgelost.
Als je het goed doet kun je als programmeur complexiteit "opsluiten" in een object. Je voert bijvoorbeeld lastige berekeningen uit binnen het object zonder er de buitenwereld mee lastig te vallen. Probeer je de formules binnen het object goed te krijgen dan hoef je je geen zorgen te maken om de omgeving. En wil je de omgeving goed krijgen dan hoef je niet na te denken over wat er binnen het object gebeurt. Dat lucht echt op!

6. Programmeren in OS-X en iOS: vloeibaar

Programmeren binnen de systemen van Apple is wennen. De objectoriëntatie is op een consequente en diepe manier in de systemen doorgezet. In het begin voelt dat heel indirect. Je bent toch geneigd in waarheid te denken, bijvoorbeeld door aan "het systeem" te vragen

om een bepaald woord in hoofdletters te schrijven. Dat gaat niet werken, dat woord is een object, dus je moet aan dat woord vragen: “Wil jij *jezelf* in hoofdletters schrijven?”
Het idee dat objecten met elkaar kunnen interacteren leek me heel spannend maar dat blijkt in de praktijk ook voor de nodige complicaties en verwarring te zorgen. Zeker als systeemobjecten methodes in door jou geschreven objecten gaan aanroepen. Dat noemen ze bij Apple het Hollywood principe: *“Don’t call us, we call you.”* Na verloop van tijd blijkt het toch heel goed te werken.
Een ander punt is dat Apple soms interfaces maakt op een hoger abstractieniveau dan je verwacht had. Als je een keer door hebt hoe het werkt kan dat je heel veel werk uit handen nemen, bijvoorbeeld bij het beheer van een database. Maar je moet wel navenant meer tijd besteden om te doorgronden wat ze eigenlijk bedoeld hebben.
Wat mij tenslotte verrast heeft is hoe veel tijd en moeite ik moest steken in het georganiseerd houden van de relaties tussen mijn programma-onderdelen. De wis- en natuurkunde formules waar we vroeger zo veel moeite voor deden zijn voetnoten geworden, die zoek je op op Wikipedia als je ze nodig hebt en met een enkele regel programmacode ben je vaak alweer klaar. Het is het relatiebeheer, de onderlinge samenhang van de data, waar het echte werk in zit.

Dit levert een platform op dat “vloeibaar” is, door de modulariteit kunnen onderdelen op elke plek en elk moment gemakkelijk aangepast worden. Dit is niet een theoretische mogelijkheid, Apple maakt hier actief gebruik van en haalt er aantoonbaar voordeel uit. In 2006 schakelde Apple in acht maanden tijd over van PowerPC naar Intel processors, een transitie die tot dan toe voor onmogelijk werd gehouden omdat de instructies op processorniveau compleet verschillend zijn. In 2012 worden de bitraster kaarten van Google vervangen door vector gebaseerde kaarten van TomTom. Er zijn duizenden apps die van deze kaarten gebruik maken, die gaan in een keer allemaal mee.
Apple kan op commercieel of strategisch niveau beslissingen nemen die door de techniek naadloos gevolgd worden.

Als programmeur werk je in een zeer dynamische omgeving. Ieder jaar wordt in juni een developer conferentie gehouden, dan komen er weer honderden nieuwe mogelijkheden bij. Daar moet je best wel even moeite voor doen om dat weer bij te werken.
Als je netjes binnen de richtlijnen geprogrammeerd hebt dan blijven bestaande programma’s bijna altijd goed werken als er een nieuwe versie van het OS uit is. Wel zijn er dan weer talloze methoden “depricated”, ze werken nog maar je wordt vriendelijk verzocht een nieuwe methode te gebruiken. Meestal wil je dat graag doen omdat de nieuwe methode aantoonbare voordelen heeft.

De gebruikelijke termijn tussen aankondiging en uitkomen van een nieuw OS is zes maanden. Dat is de tijd die je als ontwikkelaar krijgt om te zorgen dat jouw programma ook onder de nieuwe versie van het OS goed werkt. En dat is gelijk ook de langste termijn waarop je ontwikkelingen krijgt aangekondigd.

Bij voorkeur kondigt Apple vandaag producten aan die morgen in de winkel liggen. Dat betekent dat letterlijk morgen alles anders kan zijn. Het is in deze context dat ik dat woordje “toekomstvast” niet meer was tegengekomen...

7. De les: van reductionisme naar (de)compositie

De basis van ons denken wordt nog steeds door de Verlichting bepaald, de stroming die ons de vrijheid biedt om zelf over onze omgeving na te denken. Wij hebben onze behoefte gevolgd onze wereld te verklaren vanuit meer simpele, essentiële waarheden. Dat heeft het mogelijk gemaakt al die natuurkundige wetten te ontdekken. Het heeft ons een wereldbeeld opgeleverd waarin we de verschijnselen verklaren vanuit de materiële componenten waaruit ze zijn opgebouwd.
Dat is zo mooi te zien aan de spoorwegen: het vuur van de kolen verklaart het verdampen van het water, verklaart de druk op de ketel, verklaart het bewegen van de cilinders, verklaart de kracht op de drijfstangen, verklaart het draaien van de wielen. De elementen zijn onder controle gebracht. En tegelijk is het reductionisme, reductie van een veel complexer fenomeen - de stoomlocomotief - tot elementaire natuurkrachten.

Met het complexer worden van de systemen blijkt dat we met dat reductionisme iets weggooien. De waarde van systemen ligt niet alleen in de componenten waaruit ze zijn opgebouwd maar ook in de wijze waarop die componenten zijn samengebracht. Denk aan de relaisbeveiliging waar de manier waarop de relais met draadjes verbonden zijn van veel groter belang is dan de relais en de draadjes zelf.
Of denk aan de treindienst die op 3 februari vastliep. De spoorwegen hadden van hun personeelsrooster een breiwerk gemaakt dat minstens zo ingewikkeld was als de bedrading in de beveiliging. Toen dat breiwerk niet meer werkte redeneerden ze dat alle fysieke componenten voor de treindienst - personeel en treinen - toch aanwezig waren en deden ze alsof er in de samenhang van die componenten - het rooster - geen enkele waarde lag.

De klassieke natuurkunde geeft grip op de materiële componenten van een systeem, kan je prima vertellen waarom een relais opkomt als er stroom door loopt. Maar het zegt helemaal niets over de compositie van die elementen tot een beveiligingssysteem.
Een techniek als object-georiënteerd programmeren doet het omgekeerde, het abstraheert weg van de fysieke werkelijkheid en probeert juist grip te krijgen op de waarde die ontstaat door compositie van elementen. Object-georiënteerd programmeren heeft zich bewezen als uiterst krachtige techniek en zal zonder twijfel steeds breder worden toegepast.

Voor ons denken betekent dit, dat we minder houvast zullen ontlenen aan de materiële werkelijkheid en ons meer zullen moeten verlaten op abstracties die complexe samenhang van componenten beschrijven.
Zo kunnen we bereiken dat onze verkeerssystemen minder kwetsbaar worden voor complexe kettingreacties. Ik weet alleen niet of “robuuste infrastructuur” helemaal de goede benaming is voor een dergelijke verbetering. Zowel in het woord “robuust” als in het woord “infrastructuur” klinkt heimwee door naar het materialistische en dat is precies wat de verbetering in de weg staat.

8. Conclusies

- “Robuuste infrastructuur” zou geïnterpreteerd kunnen worden als het streven naar een mindere gevoeligheid voor complexe kettingreacties zoals het uitvallen van de treindienst door sneeuwval.
- Als tegenmaatregel kan gekeken worden naar de manier van denken in sectoren of bij bedrijven die minder kwetsbaar lijken. De auteur heeft dit gedaan door de werkwijze van Apple te onderzoeken.
- Het blijkt dat Apple een uitgebreid instrumentarium heeft ontwikkeld voor het beheer van relaties *tussen* componenten van complexe systemen, bijvoorbeeld in de vorm van object-georiënteerd programmeren.
- Het, in de verlichting gewortelde, westerse denken heeft een voorkeur voor reductionisme: het zoeken naar waarheid op basis van de (materiële) eigenschappen *van* componenten en bijbehorende natuurwetten.
- Eigenlijk is het heel logisch dat je, als je onvoldoende waarde toekent aan de complexe samenhang tussen componenten, kwetsbaar bent voor incidenten veroorzaakt door het falen van deze samenhang.
- Maar een andere aanpak impliceert een denkwijze die minder houvast vindt in de materiële werkelijkheid en zich meer verlaat op abstracties die deze complexe samenhang beschrijven.
- Woorden als “robuust” en “infrastructuur” suggereren een heimwee naar materialisme dat verbetering juist in de weg staat. Het zou maar zo kunnen dat “robuuste infrastructuur” niet is wat het lijkt te zijn.

9. Referenties

Treinreiziger.nl (11 februari 2012): *Analyse van de spoorwegproblemen*
http://www.treinreiziger.nl/kennisnet/analyse_van_de_spoorproblemen

Isaacson, Walter (2011): *Steve Jobs*