

# De weg der geleidelijkheid

## Echte ontwikkeling begint pas na eerste release

Ontwikkeltrajecten die slecht verlopen kunnen voor oplevering van het product al een grote renovatie-slag nodig hebben. Als ontwikkelaars OO-technieken bijvoorbeeld al te enthousiast gebruiken, kunnen er zomaar meer dan 13 lagen in de software ontstaan, zodat niemand meer snapt wat er precies gebeurt. Dan is renovatie voor release een must.

Maar ook de zogeheten *pre-delivery maintenance* is een voorbeeld van onderhoud en beheer voordat een systeem operationeel wordt. Hierin spelen onderhoudsexperts een rol bij het tegengaan van de natuurlijke neiging van ontwikkelaars om er zo snel mogelijk een punt achter te zetten, om de deadline te halen. In grote trajecten wordt wel een jaar voor oplevering tegenwicht geboden door beheerexperts om te zorgen dat als het systeem operationeel wordt, dat het dan ook onderhoudbaar is, en blijft. Uit onderzoek blijkt dat dit van grote invloed is op de prijs/prestatie verhouding van het systeem.

Maar meestal wordt een systeem zonder preventieve maatregelen over de schutting gegooid, wat tot gevolg heeft dat er hoge kosten aan het onderhoud zijn verbonden. Dit

Het belang van onderhoud en beheer kan niet genoeg benadrukt worden. Onderhoud en beheer omvat alles dat geen zuivere ontwikkeling is. Dat is meestal al het werk aan informatietechnologie nadat het operationeel is geworden, maar het kan ook al tijdens de ontwikkelingsfase beginnen.

Chris Verhoef

is geen klein incident, maar wordt al 30 jaar lang keer op keer gemeten en bevestigd. Van de totale levenscycluskosten van een systeem blijkt 50 tot 80 procent van de kosten in de onderhoudsfeer te zitten. In feite kun je een opgeleverd it-systeem dus zien als een eerste investering. Daarmee ga je nog een heel traject in van hoge extra kosten, voor vele jaren. Dan moet de businesscase voor zo'n systeem wel bikkelhard zijn, zou je denken. Vaak is het voor de besluitvormers echter niet duidelijk wat de 'gevolgschade' is van een beslissing tot het inslaan van een nieuwe softwareweg.

### Catastrofes

Besluiten over software spelen zich vaak af in twee extremen: de zogeheten complexiteitscatastrofe of de zogenaamde errorcatastrofe. Bij de complexiteitscatastrofe is het nauwelijks meer mogelijk om kosteneffectief een verandering aan te brengen in een van de vele verouderde

systemen. Op het moment dat er op een plek iets verandert, gaat er op een andere plek iets stuk. Bovendien komen er steeds meer witte vlekken in de it-portfolio waarvan niemand meer weet wat het precies doet, en hoe het werkt. Toch kun je die systemen absoluut niet uitzetten omdat de organisatie in een paar uur plat ligt en – als het maar lang genoeg duurt – de boel op de fles gaat. Als voorbeeld: bij de eerste aanslag op het WTC in New York in 1993 bleek dat 150 bedrijven failliet waren gegaan omdat ze meer dan 72 uur uit de lucht waren met hun it-systemen. Zo bedrijfskritisch is de continuïteit van it dus. De andere situatie is de errorcatastrofe. Dat is een situatie waarin je geraakt als je zo ziek bent geworden van de complexiteitscatastrofe dat je in één keer helemaal overnieuw begint. Dat zie je wel vaker. Nederlandse voorbeelden die publiek zijn gemaakt zijn: de nieuwe studentenadministratie van de IB-groep, het Hoger Beroep Systeem van het

Ministerie van Justitie, en er is meer.

### **Schone lei**

Wat al dit soort grote projecten kenschetst, is dat men met een schone lei wil beginnen. Meestal met de bedoeling de uitwassen van de huidige situatie eens en voor altijd de baas te zijn. Maar wat blijkt er vaak te gebeuren? Het lukt totaal niet om ook maar bij benadering de systemen op te leveren met de vereiste bedrijfslogica erin. Die projecten falen dan ook jammerlijk. Bij de IB-groep kostte dat honderden miljoenen, en bij Justitie liep het verlies in de tientallen miljoenen.

Een mogelijke oorzaak van dit falen is, dat er in de loop der jaren dusdanig veel business rules in de verouderende systemen is geaccumuleerd, dat het onmogelijk is dit in een nieuwe poging ineens in één keer goed te krijgen. Het is als het – met het geluid aan – in één uur doorspoelen van een bandje met 30 jaar muziek erop, met de verwachting dat je daarna de muziek zonder hulp in het normale tempo kunt reproduceren.

### **Geleidelijkheid**

In de praktijk is er eigenlijk maar één mogelijkheid om adequaat om te gaan met grote bedrijfskritische softwaresystemen. En dat is de weg der geleidelijkheid. Het is de balans zoeken. Aan de ene kant moet je uit de complexiteitscatastrofe zien te blijven, en aan de andere kant moet je niet te veel, te snel, in te korte tijd willen vernieuwen. In het Engels heet dat wel: 'Exploit the old, while exploring the new'. Maar hoe doe je dat?

Met het gebruik van methodes, tools en technieken die de moderne onderhoud- en beheerwereld te bieden hebben, kun je tussen de twee uitersten laveren. Er zijn namelijk allerlei redenen waarom

systemen verouderen, niet meer up-to-date zijn, of aangepast moeten worden. Zowel de wereld om de systemen heen als de technologie waarmee ze ooit gemaakt zijn veranderen snel. Sneller dan je kunt bijhouden in een groot it-systeem. Denk maar eens aan een telefooncentrale. Iedereen die tot tien kan tellen kan de hele wereld bellen. Vrijwel niemand staat echter stil bij het feit dat die centrales vaak tientallen jaren oud zijn. Ze hebben dus tientallen jaren van technologische revoluties overleefd. Denk eens aan isdn, mobiele telefoons, sms, adsl, 112, voicemail, en er is nog veel meer. Al die technologische vernieuwing is zonder uitzondering op een evolutionaire manier in operationele telefooncentrales geïmplementeerd. Dat kan ook niet anders: in sommige landen mag de downtime van zulke systemen niet meer dan twee minuten per jaar bedragen, en dat is inclusief software updates.

Niet voor niets concludeerde Ivar Jacobsson, de vader van de use case, na vijftientig jaar ervaring in de telecommunicatiesector dat de echte ontwikkeling pas begint na de eerste release van de software. De eerste release is net als het halen van je rijbewijs, daarna moet je de echte softwarekilometers gaan maken. En niet zomaar rondjes rijden, maar op avontuur met de systemen, en er continu aan verbeteren en vernieuwen om weer nieuwe horizonten te kunnen zien.

### **Geautomatiseerd**

Een aantal gevallen kun je niet af met handmatige onderhoudsprojecten. Daarvoor is de impact op het systeem of de hele it-portfolio veel te groot. Een mooi voorbeeld was het Jaar 2000-probleem, en iets later de euroconversie. Maar er zijn dagelijks van dat soort issues: bijvoorbeeld toen Wang failliet ging in 1992, moest er ineens een hoop

Wang-Cobol worden omgezet naar een nieuwe variant die nog wel ondersteund werd. En hoewel dit eenvoudig klinkt, is het heel lastig. Nog 10 jaar na een dergelijke operatie kun je er in de operationele code last van hebben. Gelukkig biedt geautomatiseerde taalconversietechnologie dan enig soelaas. In dit themanummer zien we hier voorbeelden van. Maar voordat je bijvoorbeeld taalconversie gaat toepassen moet je eerst de rotzooi schoonmaken. Zo kan het aantal ongestructureerde stukken code door de jaren heen dusdanig toenemen dat deze situatie niet meer zonder systematisch en geautomatiseerd ingrijpen is terug te draaien. Dan is men dus in de complexiteitscatastrofe beland. Hier bieden refactoringtools uitkomst. Met als doel de leesbaarheid en veranderbaarheid van de software te verhogen, passen deze tools in hoge mate geautomatiseerd de code stelselmatig aan. Daarmee ontstaat nette code die verder 'gerecycled' kan worden, met de hand verwerkt, of wellicht met een tool geconverteerd naar een andere doeltaal, andere functionaliteit, of een hoger abstractieniveau. In sommige gevallen wil je namelijk dieper inzicht hebben in een systeem, op een hoger abstractieniveau. Dan zijn tools en technieken die dit soort logica boven tafel halen een uitkomst.

Maar met het opleuken van de it alleen ben je er niet. De meeste systemen beheren ook nog eens grote hoeveelheden relevante data. Die data raken door de jaren heen vervuild. Een succesvolle schoonmaak is niet compleet zonder dat de bezem door de data gaat. Dat is ook een prachtbasis voor een eventuele overgang naar een gerenoveerd it-systeem. En er is meer. Er is altijd meer, in dit o zo relevante vakgebied waarin nog heel veel te ontdekken valt, zowel voor de praktijk als voor de wetenschappers.



Zo treft u allerlei verhalen in dit thema aan. Alle hebben uiteindelijk tot doel uit de complexiteitscatastrofe te geraken, of er niet in te komen, onderwijl de val van de errorcatastrofe vermijdend. Evolutie in plaats van revolutie.

**Prof.dr. Chris Verhoef**

werkt bij het instituut voor Informatie Management en Software Engineering aan de Faculteit der Exacte Wetenschappen van de Vrije Universiteit. E-mail: [x@cs.vu.nl](mailto:x@cs.vu.nl).

**Conferentie onderhoud en beheer**

In de week van 22 september van dit jaar komt de grootste conferentie over onderhoud en beheer naar Amsterdam. Deze International Conference on Software Maintenance (ICSM) startte in 1983 en is daarna vrijwel elk jaar gehouden. Deze conferentie, die onder auspiciën van de IEEE Computer Society staat, heeft ook dit jaar vele toonaangevende sprekers uit binnen en buitenland, aanpalende workshops, tutorials en meer. Dit thema is een mooi voorproefje van wat er in Nederland en België op dit gebied aan hoogwaardige technologie ontwikkeld wordt.

[www.cs.vu.nl/icsm200](http://www.cs.vu.nl/icsm200)