Mark Blomsma
is werkzaam als software architect bij Omnext.NET bv (www.omnext.net). Hier houdt hij zich bezig met software renovatie, frameworks en Microsoft.NET. Daarnaast is hij actief als sectiehoofd van de C# Usergroup bij de SDGN[1]

Van A naar .NET

Werkelijke kostenbesparing bij migratie of renovatie

De ICT-branche heeft het voorrecht om in één van de meest boeiende periodes van haar bestaan te bivakkeren. Enerzijds gaat het economisch allemaal wat minder, anderzijds gaan technologische ontwikkelingen in de ICT sneller dan ooit. Zakelijk gezien staat kostenbesparing bij de ICT-gebruikers hoog op de agenda. Integreren van ICT met partners en klanten heeft prioriteit, maar mag niet strijdig zijn met het punt van kostenbeheersing. Bedrijven willen Internet steeds meer benutten als channel voor business-to-business en business-to-consumer communicatie, maar onder druk van economisch zwaar weer moeten investeringen op korte termijn al bijdragen in de omzet. Daarnaast is de afgelopen tien jaar zwaar geïnvesteerd in systemen waarvan nu blijkt dat de onderhouds- en licentiekosten de pan uitrijzen. Als klap op de vuurpijl komt Microsoft met .NET. Nieuwe technologie waar de ontwikkelaars van smullen en waarmee de wensen van de organisatie op gebied van e-business ingevuld kunnen worden. Verder lijkt deze nieuwe architectuur in de exploitatie minder duur te zijn dan de 3GL en 4GL legacy-architecturen. Dat zijn dus goede argumenten om van deze legacy-systemen naar .NET over te gaan. De hamvraag is: Hoeveel kost het om van A naar .NET te komen?

Elke bedrijf zal voor zichzelf een keuze moeten maken over hoe de toekomst eruit ziet. Dit heet mooi gezegd 'het ontwikkelen van een visie'. Welke diensten en producten lever ik nu, welke producten wil ik in de toekomst gaan leveren? En natuurlijk voor ons IT-ers het belangrijkst: welke rol speelt ICT in het productieproces en in het eindproduct? Op het moment dat duidelijk is hoe deze toekomst eruit ziet moet een strategie ontwikkeld worden om te komen tot het realiseren van deze toekomst. Technologische ontwikkelingen, en dus .NET, spelen een belangrijke rol in deze toekomst. Blijft alleen de vraag staan: 'Hoe kom ik tot een succesvolle .NET implementatie?'.

Strategie en architectuur

Op het moment dat gekozen wordt voor een strategie voor het implementeren van .NET is het belangrijk om te kiezen voor een passende software-architectuur. De software-architectuur is de blauwdruk op basis waarvan alle nieuwe ontwikkelingen plaats moeten gaan vinden. Het opzetten en hanteren van een duidelijk software architectuur is een hulp voor zaken als[2]

- het krijgen en behouden van overzicht over grote complexe systemen;
- het creëren van meer flexibele, aanpasbare en koppelbare systemen die langer mee kunnen;
- het realiseren van hergebruik c.q. gemeenschappelijk gebruik van componenten tussen systemen, zowel voor versnelling van ontwikkeling als voor het bereiken van consistentie;
- het integreren van bestaande systemen in nieuwe toepassingen;
- het beter beheersen van de kwaliteit van opgeleverde systemen.

Bij het ontwikkelen van de software-architectuur dient niet alleen gekeken te worden naar de technisch meest fraaie oplossing, maar dient vooraf eveneens goed gekeken te worden naar de infrastructuur waarin de applicatie zal komen te draaien. Deze infrastructuur is in belangrijke mate bepalend voor de te kiezen implementatiestrategie. Er is een

1 Software Developers Group Nederland is een vereniging met als doel het ontwikkelen van de C# en VB.NET kennis van haar leden. De SDGN brengt deze kennisontwikkeling tot stand door het organiseren van seminars en conferenties, publiceren van het SDGN Magazine en natuurlijk de inzet van www.sdgn.nl als kennisportaal. Programmeer jij in C# of VB.NET? Kijk dan op de website van SDGN om lid te worden.

2 Software architectuur - een kader voor samenwerking, G. Florijn (SERC), Automatiseringsgids, september 1998

Strategie	Impact	Investering	Kostenbesparing
Parallelle implementatie	Laag	Laag	Laag
Softwaremigratie	Hoog	Hoog	Hoog
Softwarerenovatie	Hoog	Laag	Hoog

Tabel.

drietal strategieën voor het implementeren van nieuwe technologie:

- parallele implementatie;
- softwaremigratie;
- softwarerenovatie.

De implementatiestrategie speelt een rol in het bepalen van de architectuur van de applicatie. Zo zal een .NET-applicatie, die voor data 100% afhankelijk is van een specifieke interface naar een legacy-systeem, bepaalde beperkingen kennen die voor een gerenoveerde applicatie niet van toepassing is. Het is aan te bevelen naar aanleiding van de applicatie-architectuur een library met framework-objecten te realiseren die het implementeren van de gekozen architectuur faciliteren.

We zullen elk van de genoemde strategieën nader beschouwen en beoordelen op een drietal criteria:

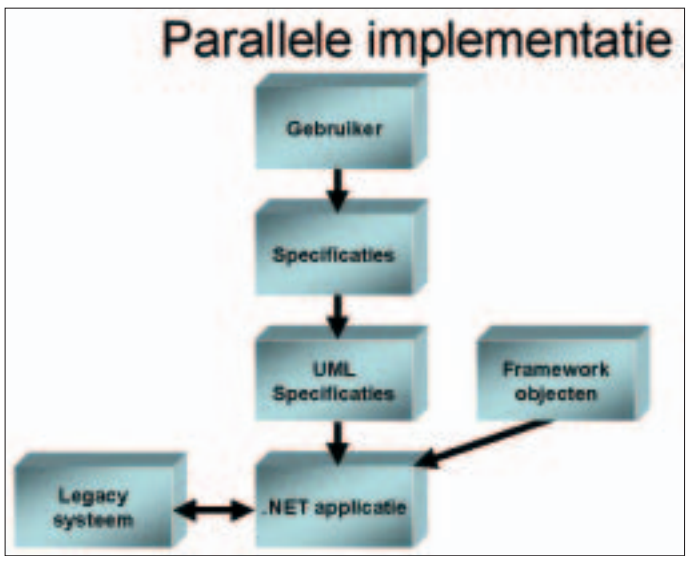
- impact op de (IT-)organisatie;
- investering die gedaan moet worden om de .NET-applicatie te implementeren;
- kostenbesparing op het vlak van onderhoud en licentiekosten die de investering oplevert.

Als we de strategieën uitzetten tegen genoemde punten komen we tot de matrix in de tabel. Merk op dat de kwantificering van de cellen in de matrix relatief is. Per strategie zullen we vervolgens gaan kijken hoe we komen tot genoemde kwantificering.

Parallelle implementatie

Bij parallel implementeren kiest een organisatie ervoor om nieuwe functionaliteit te realiseren met nieuwe technologie - in dit geval .NET - en bestaande legacy-systemen te laten bestaan. Afbeelding 1 geeft aan hoe dit proces verloopt. Qua ontwikkeling is dit eigenlijk het standaardproces van een stuk nieuwe software. Het bijzondere zit hem in de interface naar het legacy-systeem. Deze speelt een dominante rol in het te ontwikkelen systeem. We praten hier over koppelingen die hoogst waarschijnlijk over platformgrenzen gaan en/of de interoperabiliteitsmogelijkheden van .NET beproeven. Soms kan op databaseniveau de data gedeeld worden door applicaties. Dat scheelt, want dan is de interface technisch gezien tenminste redelijk eenvoudig. Op de MSDN website staan diverse whitepapers over interoperability en architectuur.

De gebruiker komt met wensen die het best gerealiseerd kunnen worden met .NET-technologie. Bijvoorbeeld: implementeer een interface waarmee klanten dagelijks via Internet de status van hun order kunnen bekijken. Dit zou kunnen via een webpagina waarop de klant inlogt, of een webservice waarmee de klant deze informatie kan integreren in zijn eigen systeem.



Afbeelding 1.

Hoe ziet het proces eruit? De designer stelt een model op. De meeste moderne tools doen dit op basis van UML. Hierbij houdt de designer rekening met de library van framework-objecten die intern zijn ontwikkeld of zijn aangekocht. Om naast het bestaande systeem te kunnen draaien, maar toch ook te integreren, wordt een interface ontworpen tussen het reeds bestaande systeem en de nieuwe .NET-applicatie. Op het moment dat de applicatie live gaat, wordt via de interface de benodigde data uit het legacy-systeem gehaald. Mogelijkerwijs heeft de nieuwe applicatie ook nog een eigen datastore om bijvoorbeeld data te cachen, of nieuwe applicatiespecifieke data op te slaan.

De impact op de (IT-)organisatie van deze aanpak is laag. Oude systemen en processen draaien gewoon door. Er hoeft slechts een kleine groep mensen te worden opgeleid om te kunnen ontwikkelen met .NET-technologie. Factoren die de investering laag houden zijn:

1. De scope van het project wordt beperkt tot de nieuw te realiseren software.
2. Slechts een (klein) deel van de IT-afdeling hoeft omgeschoold te worden naar .NET.
3. Punt 1 en 2 maken changemanagement rondom het introduceren van .NET in de organisatie behapbaar. Hierdoor is de kans op succes veel groter.
4. Productiviteitswinst door inzet van .NET. De .NET-technologie is uitermate geschikt om Internetapplicaties en webservices te realiseren. Hierdoor zullen de kosten voor het ontwikkelen met .NET een stuk lager liggen dan wanneer gelijke functionaliteit met een andere tool ontwikkeld moet worden.

De kostenbesparing die de parallelle implementatie oplevert is relatief laag. De besparing ligt in het feit dat .NET geen licentiekosten kent. Een Windows 2000 of Windows.NET server is voldoende om een .NET-applicatieserver of webserver te kunnen draaien. Alle kosten voor de oude systemen lopen helaas door. In de operatie van die systemen wordt niet gesneden.

Softwaremigratie

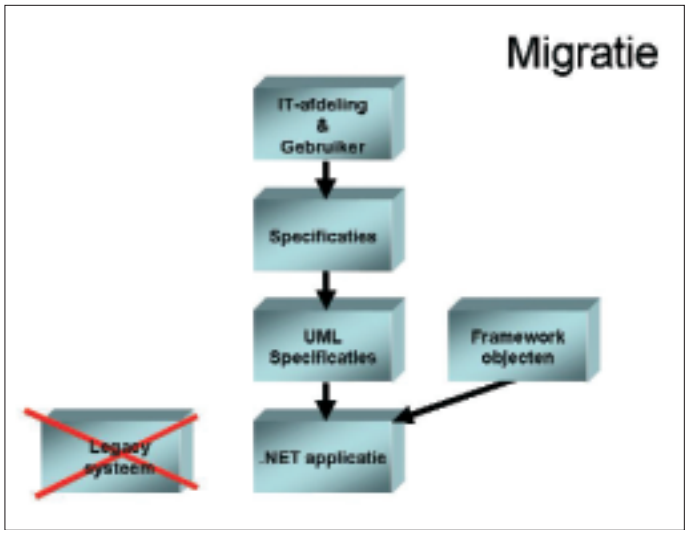
'Migratie bestaat uit het proces en de gehanteerde technieken en hulpmidde-

len om bestaande softwaresystemen te vervangen door (nieuwe) systemen die voldoen aan nieuwe wensen en/of makkelijker onderhoudbaar zijn'[3]. In afbeelding 2 wordt weergegeven hoe dit proces er doorsnee uitziet.

De initiatie van een migratietraject ligt vaak bij de IT-afdeling. Die signaleert dat gebruikerswensen niet gerealiseerd kunnen worden met huidige technologie dan wel dat kosten voor het operationeel houden van de huidige systemen te duur wordt. Men wil graag hoge productiviteit, lage onderhoudskosten, lage integratiekosten en hoge flexibiliteit. Samen met de gebruiker worden specificaties voor een nieuw systeem vastgelegd, waarbij minimaal geïmplementeerd moet worden wat er in het oude systeem zat. Maar openstaande of nieuwe wensen van de gebruiker worden veelal direct meegenomen. De designer maakt van de specificaties een UML-model waarmee, eventueel ondersteund door forward-engineering tools, de nieuwe .NET-applicatie wordt gerealiseerd. Ook hier wordt wederom gebruik gemaakt van beschikbare framework-objecten.

Aandachtspunten voor bedrijven die deze aanpak kiezen:

1. Het is een pré om de ontwikkelaars van de legacy-applicatie ter beschikking te hebben. Indien er ook nog aanpassingen gedaan moeten worden op reeds bestaande systemen is het zelfs een vereiste. De ervaring leert dat documentatie altijd verouderd is, en vaak zijn de gedachtegangen achter bepaalde business rules nooit vastgelegd.
2. Migratietrajecten worden opgezadeld met alle lopende wensen die er zijn vanuit de organisatie. Probeer niet te verdrinken in deze nieuwe functionaliteit. Het doel van de migratie is primair kostenbesparing bij de IT-afdeling. Realiseren van kostenbesparing bij andere afdelingen door procesoptimalisatie is natuurlijk een goede zaak, maar de investering die daarvoor gedaan moet worden hoort eigenlijk niet bij de migratie.
3. Doe een data-analyse op het legacy-systeem. Vaak wordt gekeken naar de huidige business rules van de software. Als een systeem echter al 25 jaar heeft gedraaid, kun je er zeker van zijn dat er inmiddels de nodige datacorruptie heeft plaatsgevonden. Dit zou onder ander kunnen leiden tot aanpassingen in het nieuwe datamodel. Aanvullend leidt het migreren van bijvoorbeeld een netwerkdatabase naar een relationele database, zoals SQL Server, tot nieuwe structuren en hernieuwde inzichten. De mapping, die noodzakelijk is voor de migratie, kan invloed hebben op het nieuwe datamodel.
4. Doe een business-rule-analyse op het legacy-systeem. Uitgangspunt moet in 99% van de gevallen zijn dat, bij verschillen tussen code en documentatie, de code 'gelijk' heeft. Het komt overigens ook voor dat tijdens de migratieslag nieuwe bugs in het legacy-systeem naar voren komen. Er ontstaat dan een beslissingsmoment waarbij gekeken moet worden of het legacy-systeem nog gerepareerd moet worden, of dat de fout kan blijven zitten tot het gerenoveerde systeem in produktie wordt genomen.



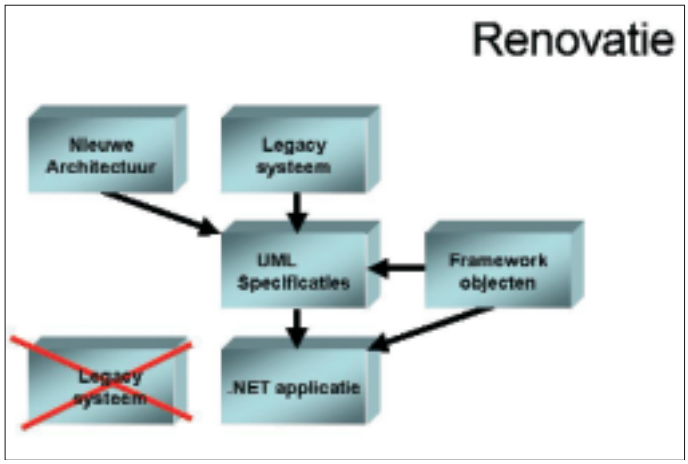
Afbeelding 2.

De impact op de (IT-)organisatie van deze aanpak is hoog. Oude systemen en processen worden uitgezet. De hele IT-afdeling, of in ieder geval het grootste deel hiervan, moet .NET ontwikkelen en .NET-applicaties gaan beheren. De oude systemen en platformen worden immers uitgefaseerd. De investering die gedaan moet worden om de .NET-applicatie te implementeren is hoog. De volledige functionaliteit die werd geboden door het oude systeem, moet ook door het nieuwe systeem geboden worden. Hieraan is vele jaren ontwikkeld. Voordeel is wel dat productiviteit met .NET hoog ligt. Nadeel is dat deze systemen vaak slecht gedocumenteerd zijn en dat rekenregels en business rules moeilijk te achterhalen zijn. Merk op dat door middel van forward engineering tijdwinst geboekt kan worden door vanuit de opgestelde specificaties C#-code te genereren. Hiervoor zijn inmiddels de nodige tools op de markt. Let hierbij wel op dat de gegenereerde code voldoet aan de eerder gestelde architectuureisen. De kostenbesparing die de softwaremigratie oplevert is hoog. .NET kent geen licentiekosten en de kosten voor de oude legacy-systemen gaan terug naar nul aangezien deze systemen de deur uit kunnen. Hierin schuilt overigens wel een projectrisico. Vaak worden migratietrajecten in fasen opgeleverd. Het gebeurt toch regelmatig dat een organisatie er niet aan toekomt om een applicatie 100% uit te faseren. Hierdoor moet een vrij grote applicatie in de lucht gehouden worden omdat een klein deel van de applicatie maar niet gemigreerd raakt.

Softwarerenovatie

'The purpose of renovation is to study the system, by making a specification at a higher abstraction level, adding new functionality to this specification and develop a completely new system on the basis of the original one by

3 Migratie - Het legacy probleem aangepakt, Matthijs Maat en Harald Vogt (SERC), Whitepaper versie 1.1 november 1998



Afbeelding 3.

using forward engineering techniques.'[4] Het doel van softwarerenovatie ligt in het verlengde van softwaremigratie met .NET: komen tot hoge productiviteit, lage onderhoudskosten, lage integratiekosten en hoge flexibiliteit. De aanpak is alleen anders.

Daar waar softwaremigratie eigenlijk begint met het ontwikkelen van een nieuw systeem, pakt softwarerenovatie het legacy-systeem en transformeert deze naar een .NET-applicatie. Dit gebeurt met behulp van Renovation Technology. 'Renovation Technology consist of tools and methods that enable any form of refurbishing of legacy applications, whether it focusses on code refactoring, application integration, platform migration, re-engineering or object-oriented redesign. The tools and methods inherently support effective and efficient delivery of quality software'.[5] Renovation Technology heeft tot doel het extraheren van business rules, workflow, functionaliteit en data-access uit bestaande code. Renovation Technology gaat dus een stuk verder dan code-analyse. De architectuur van het legacy-systeem wordt losgelaten. Er wordt alleen nog maar gekeken naar de nieuwe architectuur. Hierbij wordt de geëxtraheerde informatie geprojecteerd op een nieuwe architectuur om zo te komen tot een nieuw informatiesysteem. Tussentijds wordt een model opgeleverd waarop, indien gewenst, redesign kan plaatsvinden. Uiteindelijk wordt uit het UML-model via codegeneratie omgezet naar C#-code. Tijdens deze forward engineering-slag is het belangrijk om te kunnen genereren naar de gekozen architectuur, aangezien hier de basis wordt gelegd voor toekomstig onderhoud. Merk op dat de gebruiker bij deze aanpak eigenlijk niet of nauwelijks een rol speelt. Zolang de functionaliteit vooraf en achteraf exact gelijk blijft, en dit kan automatisch getest worden, is nauwelijks inzet van de gebruiker nodig. In de praktijk verandert echter wel de user interface en zal de gebruiker hiervoor een acceptatietest moeten doen. De softwarerenovatie heeft niet primair tot doel het implementeren van nieuwe functionaliteit, maar het terugbrengen van de onderhouds- en licentiekosten. Resultaat van de renovatie is een .NET-applicatie die een stuk beter uit te breiden en te integreren is met andere en nieuwe systemen.

De impact op de (IT-)organisatie van deze aanpak is net als bij softwaremigratie hoog. Oude systemen en processen worden wederom uitgezet en verdwijnen uit de organisatie. Ten opzichte van een migratietraject is de investering die gedaan moet worden om de .NET-applicatie te implementeren laag. Door het automatiseren van een groot deel van het werk wordt enorm bespaard op doorlooptijd en in te zetten personeel. Belangrijker nog is dat door het geautomatiseerd laten uitvoeren van de softwarerenovatie het projectrisico laag blijft. Hier geldt dat door het geautomatiseerd omzetten van oude code naar nieuwe code de hoeveelheid nieuwe code zeer nauwkeurig is afgebakend. Het afmaken en controleren van de business rules in de nieuwe code is nog steeds veel werk, maar is in omvang wel exact afgebakend. Merk op dat de gewenste software-architectuur een belangrijk ingrediënt in de renovatie is. Alhoewel de renovatiesoftware de code omzet van oud naar nieuw, wordt de nieuwe code wel volgens een nieuwe architectuur neergezet. Het nadeel van softwaremigratie, dat legacy-systemen vaak slecht gedocumenteerd zijn, wordt door experttooling aangepakt. De kostenbesparing die de softwaremigratie oplevert is hoog. Redenen zijn gelijk aan die van softwaremigratie: .NET kent geen licentiekosten en onderhoud op de legacy-systemen wordt beëindigd.

Migratie of renovatie?

.NET is de technologie om systemen te ontwikkelen met lage onderhoudskosten, lage integratiekosten en hoge flexibiliteit. Op weg van A naar .NET moet een implementatiestrategie ontwikkeld worden. Hierbij kan .NET parallel draaien naast bestaande software, maar echte besparingen ontstaan pas bij migratie of renovatie en het uitfaseren van legacy-systemen. Afhankelijk van omstandigheden en beschikbaarheid van experttooling kan gekozen worden tussen migratie of renovatie.

Nuttige internetadressen

- http://www.software-renovatie.net
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdadotnetarchtopics.asp
- http://www.cwi.nl/htbin/sen1/twiki/bin/view/SEN1/SoftwareRenovationResearch
- http://citeseer.nj.nec.com/vandenbrand97generation.html

4 Reverse Engineering and System Renovation - An Annotated Bibliography, Mark G.J. van den Brand e.a., ACM Software Engineering Notes 22, nr. 1, 1997

5 www.omnext.net/renovatie, Omnext.NET bv, 2002