



## Outsource nu, bepaal de prijs later

Door kwaliteitsmetingen te koppelen aan productiviteitsmetingen kan de productiviteit tijdens het bouwen of onderhouden van software zeer precies gemeten worden. Op basis van deze metingen is het mogelijk om een prijsmodel af te spreken waardoor de opdracht op een eerlijke manier wordt afgerekend, op basis van wat er feitelijk is geproduceerd. Dit model staat aan de ene kant toe dat de klant zijn eisen en wensen tijdens het project kan bijstellen, zonder dat er opnieuw over de prijs onderhandeld hoeft te worden. En aan de andere kant staat het de leverancier toe om zelfs met wijzigende eisen en wensen een marge te blijven maken op het project, zodat het project niet onder (interne) financiële druk komt te staan.

*door: Tobias Kuipers (Software Improvement Group)*

Het beprijzen van outsourcing van softwareontwikkeling is een heikele zaak. Naast de feitelijke kosten van het totale project is vooral het *model* van groot belang. Kies je het verkeerde model dan loop je het risico dat de leverancier in hoog tempo volkomen onbruikbare software maakt. Kies je een ander verkeerd model, dan maakt de leverancier heel zorgvuldig (lees: langzaam) erg dure software. En kies je weer een ander verkeerd model, dan kan je na oplevering van het project nog jaren ruzie maken met de leverancier over de vraag hoeveel het hele project nou eigenlijk gekost heeft.

Nu de verzamelde Nederlandse industrie een aantal jaar ervaring heeft opgedaan met het uitbesteden van beheer en onderhoud van softwareapplicaties, is het aardig om eens een inventarisatie te maken van de verschillende businessmodellen die in gebruik zijn. Er zijn toch nog steeds organisaties die worstelen met de prijsmodellen van ict-leveranciers. Zeker als een uitbestedingsopdracht verstrekt wordt via een aanbesteding is het niet altijd eenvoudig – of voor de hand liggend – om de verschillende aanbiedingen te vergelijken.

### Vier prijsmodellen

Op dit moment bestaan er ruwweg vier modellen om een prijs voor aanbesteding vast te stellen:

1. Uren maal tarief. De meest gangbare en schijnbaar de meest geaccepteerde. Opdrachtgever werkt x uur, klant betaalt x maal y euro.

2. Fixed price. Van tevoren komen opdrachtgever en opdrachtnemer een hoeveelheid te verrichten werk overeen en wordt er één vast bedrag afgesproken waarvoor de werkzaamheden worden uitgevoerd.
3. Een prijs per functiepunt. Opdrachtgever en opdrachtnemer weten nog niet hoeveel werk er verricht moet worden, maar willen toch alvast een tarief afspreken per geleverde hoeveelheid functionaliteit.
4. Een vastrecht, gecombineerd met een variabel deel. Feitelijk een hybride optie: er is sprake van een fixed price contract voor 'regulier onderhoud' met eventueel meerwerk voor 'niet-regulier onderhoud'.

Uiteraard bestaan er tal van mengvormen waarvan sommige niet direct op één van deze modellen af te beelden is. (Een van de meer complexe voorbeelden is een contract waarin een vast bedrag per eindgebruiker per jaar is afgesproken, van welk bedrag een percentage wordt gereserveerd voor nieuwe ontwikkelingen die dan vervolgens per uur worden afgerekend.)

### Vier eisen waar een model aan moet voldoen

Al deze modellen komen voort uit de wens om een optimaal contract op te stellen. Uit de verschillende modellen en uit gesprekken met betrokkenen is een lijst van aspecten op te stellen die van belang zijn bij het vaststellen van het optimale prijsmodel:

- Meetbaar. De eenheid waarvoor betaald wordt, moet meetbaar zijn. Bij voorkeur objectief en eenvoudig.
- Vergelijkbaar. De prijsmodellen worden doorgaans gebruikt om het aanbod van verschillende leveranciers met elkaar te vergelijken; de eenheden waaruit het model bestaat, moeten dan ook op één of andere manier met elkaar te vergelijken zijn.
- Uitvoer gerelateerd. Uit de modellen blijkt dat de klant wil betalen voor wat er uiteindelijk geleverd wordt en niet voor de hoeveel inspanning die daaraan is besteed.
- Reële prijs. De uiteindelijke prijs moet zo laag mogelijk zijn voor de klant, maar niet te laag. Contracten waarop de leverancier geen winst maakt, leiden zonder uitzondering tot rampzalige projecten.

### De voor- en nadelen op een rij

De voor- en nadelen van de vier modellen ten opzichte van de vier aspecten zijn uitgewerkt in *tabel 1*.

Uit de tabel blijkt dat geen van de modellen alle aspecten van het optimale model afdekken. Er is dus ruimte voor verbetering, voor een model dat wel alle vier de aspecten afdekt.

### Metten van regels code als alternatief

Een alternatieve maat voor het prijzen van uitbestedingsdiensten is het meten van geproduceerde regels code.

	Meetbaar	Vergelijkbaar	Uitvoer gerelateerd	Reële prijs
Uren maal tarief	- De uren zijn eenzijdig meetbaar; alleen de uitvoerende partij kan ze meten.	- Het tarief is vergelijkbaar, maar het aantal uren niet. Daarmee is het totaalbedrag voor een specifieke activiteit onvergelijkbaar.	-- Alleen de invoer, het aantal gewerkte uren, wordt gemeten. Of deze uren een relatie hebben met het geleverde product is onduidelijk.	0 In ieder geval levert dit model een reële prijs op voor de leverancier. De opdrachtgever is afhankelijk van de hoeveelheid gemaakte uren.
Fixed price	+ Heeft als risico dat er onenigheid bestaat over het te behalen doel. Als er een 'werkend systeem' wordt opgeleverd, is dat meestal meetbaar.	+ Gegeven een vaste hoeveelheid werk is dit model goed vergelijkbaar. In de praktijk blijkt juist de hoeveelheid werk aan verandering onderhevig.	++ De opdrachtgever betaalt in principe pas als het afgesproken werk ook daadwerkelijk is opgeleverd.	-- Groot risico van onderschatting door leverancier. Om het contract binnen te halen wordt de prijs te laag gesteld. Dat heeft gedurende het project rampzalige gevolgen, omdat interne bemensing niet gebudgetteerd kan worden.
Prijs per Functiepunt	-- Functiepunten zijn zeer lastig te meten. Twee functiepunttellers komen niet altijd tot dezelfde conclusie. Daarbij is het meten van functiepunten een kostbare aangelegenheid.	++ Is goed te vergelijken tussen verschillende leveranciers.	+ Is een maat voor de geleverde functionaliteit. Functiepunten zijn bij uitstek geschikt voor <i>dataprocessing</i> systemen. De methode is minder geschikt voor rekenintensieve systemen die weinig data processing doen.	+ Er zijn voldoende benchmarkgegevens beschikbaar om vast te stellen hoe ver de geoffreerde prijs afwijkt van het industrie gemiddelde. <sup>1</sup>
Vastrecht + variabel	- Leidt tot discussie over welk gedeelte van het werk tot vast werk en welk gedeelte tot variabel (extra) werk moet worden gerekend. Verder meetbaar als uren maal tarief.	0 Vaste gedeelte goed te vergelijken, variabele gedeelte niet te vergelijken.	0 Waar het specifieke wensen betreft min of meer uitvoer gekoppeld. Het vaste bedrag is dat minder, want afhankelijk wat daar per keer voor geleverd wordt.	+ Het vaste bedrag zorgt voor continuïteit bij de opdrachtnemer, terwijl het variabele gedeelte de opdrachtgever de kans geeft bij te sturen.

Tabel 1: De voor- en nadelen van de vier rekenmodellen

Dit is een historische activiteit, in de zin dat in het verleden (voordat de functiepunt in zwang kwam) er ook al productiviteit werd gemeten op basis van regels code. Kennelijk kleefden er bezwaren aan het meten van regels code die door de functiepunt zijn weggenomen. Allereerst, waarom zou het meten van regels code en het afrekenen per regel code een goed idee zijn.

1. Regels code worden daadwerkelijk geproduceerd, in tegenstelling tot uren en functiepunten. In die zin is afrekenen per regel code afrekenen voor een eenheid die geproduceerd is.
2. Het meten van regels code is relatief eenvoudig. Als er een paar duidelijke afspraken zijn gemaakt, is het meten van regels code een automatiseerbare en herhaalbare activiteit.
3. De hoeveelheid regels code die geproduceerd wordt door een ontwikkelteam is relatief stabiel. Een groep Java-ontwikkelaars bijvoorbeeld zal dit jaar een vergelijkbare hoeveelheid regels code produceren als vorig jaar.
4. De 'overhead' van het programmeren ten opzichte van andere activiteiten in een softwareproject (ontwerp, projectmanagement, accountmanagement, requirements management, testen, deployment) is ook stabiel. De totale projecttijd is gemiddeld twee tot drie keer langer dan de tijd die aan ontwikkeling wordt besteed.

### De nadelen van regels code meten

Tot zover zijn er alleen maar positieve argumenten om per regel code af te rekenen. Waarom is de methode dan toch uit de mode geraakt? Er zijn een paar redenen waarom het afrekenen per regel code een heel slecht idee is.

1. Slechte ontwikkelaars hebben veel meer regels code nodig dan goede ontwikkelaars. Slechte ontwikkelaars krijgen met deze methode dus meer betaald dan goede.
2. Het is erg eenvoudig om veel regels code te produceren zonder functionaliteit of waarde aan het systeem toe te voegen. Met andere woorden, het is erg eenvoudig om meer betaald te krijgen, zonder daar erg veel voor te hoeven doen of te leveren.
3. De ene regel code is de andere niet. Dat is uiteraard in zijn algemeenheid waar, maar geldt vooral voor de vergelijking tussen verschillende programmeertalen. Een systeem in Cobol is typisch tenminste een miljoen regels code groot, terwijl een Java-systeem vele malen kleiner is. Terwijl de systemen hetzelfde doen. De ontwikkelaar die het systeem in Cobol bouwt, zou dan veel meer geld krijgen dan degene die het in Java programmeert.

Met name vanwege die laatste drie argumenten zijn in de jaren zeventig de functiepunten bedacht. Een maat die onafhankelijk van de kwaliteit van de programmeurs en onafhankelijk van de gekozen technologie de hoeveelheid geproduceerde 'functionaliteit' meet. Zoals al eerder vastgesteld, blijkt het echter ingewikkeld die maat te meten. En dat wordt ook steeds ingewikkelder. Functiepunten zijn bedacht om de zogenaamde 'groene scherm' applicaties te beoordelen, die per scherm een *datarecord* inlezen, modifieren of opzoeken. Om dat model op een bruikbare manier af te beelden op een moderne web 2.0 applica-

tie die is opgebouwd uit webservices die portlets vullen middels *Ajax*-technologie blijkt ten eerste nog niet eenvoudig, en ten tweede door elke functiepuntteller anders te worden gedaan.

### De voorgestelde oplossing

Het meten van het aantal regels code, en het afrekenen daarop, is een aantrekkelijk alternatief. Mits de bezwaren die daaraan kleven worden geëlimineerd. Door de technische kwaliteit van een softwaresysteem te meten en vast te leggen, kan het aantal regels code zonder gevaar als maatstaf gehanteerd worden. Primair wordt een aantal dingen gemeten om de technische kwaliteit vast te stellen.

- Dode code. Code die wel in een systeem zit, maar niet wordt gebruikt.
- Geduplicateerde code. Code die dubbel in het systeem zit. Die code is strikt genomen ook niet nodig om het systeem te laten werken.
- Technisch complexe code. Code die niet of nauwelijks te onderhouden is en slechts met veel moeite te testen.
- De lengte van code-eenheden. Elk programma (of procedure) mag maximaal  $x$  regels lang zijn. Dat zorgt ervoor dat het onmogelijk wordt om op een eenvoudige manier extra code te genereren.
- Het totale volume aan code. Een groot systeem is per definitie technisch slechter dan een klein systeem.

Om nu de totale technische kwaliteit van het systeem te meten, worden alle individuele technische metingen geaggregeerd volgens een aggregatiemodel. Een versimpeld voorbeeld van zo'n model is: Als er geen dode code is, geen geduplicateerde code, geen technisch complexe code, geen te lange eenheden en het volume laag, dan is het systeem van maximale technische kwaliteit. Een 5 op een schaal van 1 tot 5. Als er bijvoorbeeld 10 procent dode code is, dan zakt de kwaliteit één punt tot een 4.<sup>2</sup>

Door op deze manier de technische kwaliteit van een systeem te meten worden de bezwaren weggenomen die kleven aan het *droog* meten van functiepunten:

1. Slechte ontwikkelaars hebben meer regels code nodig dan goede. Dat blijft waar, maar door de technische kwaliteit te meten wordt dat wel inzichtelijk. Een prijsafpraak die gemaakt kan worden, is bijvoorbeeld:  $x$  euro per regel code bij een vaste technische kwaliteit van  $y$ .
2. Veel regels code die niets doen. Als er veel regels code in een systeem zitten die niets doen, neemt de technische kwaliteit dramatisch af. Daarnaast zijn de individuele regels die niets doen, aan te wijzen (en dus ook degene die ze heeft ingetypt).
3. De ene regel code is de andere niet. Dat blijft waar. Maar er bestaan intussen voldoende benchmarkcijfers over de productiviteit per regel code per programmeertaal om die vergelijking te maken. Voor projecten waarbij meerdere programmeertalen gebruikt

worden (en dat is tegenwoordig voor ieder project) is het nuttig om voor de verschillende technologieën een uit de benchmark gehaalde omrekenfactor te definiëren.

### De voor- en nadelen van het meten van regels code

In dezelfde tabel die eerder gebruikt is om de andere prijsmodellen te vergelijken ziet dat eruit zoals in tabel 2.

Hoewel het relatief ingewikkeld is de individuele metingen vast te stellen die nodig zijn om de technische kwaliteit vast te stellen, zijn ze bij uitstek automatiseerbaar. Dat betekent dat – als dat proces eenmaal is ingeregeld – het meten van het volume aan code en de technische kwaliteit een volledig automatisch proces is, dat plaatsvindt zonder dat iemand daarvan in zijn dagelijkse werk last heeft. Het geeft opdrachtgever en opdrachtnemer een continu (dagelijks) beeld van zowel de voortgang als de toe- of afnemende kwaliteit van het geleverde werk.

### Het prijsmodel op basis van regels code

Een bruikbaar prijsmodel dat op deze meting gebaseerd kan worden is dan: spreek drie prijzen af per regel code. Een voor hoge kwaliteit code, een voor middel en een voor lage kwaliteit.

Stel vervolgens de technische kwaliteit van het project vast (of laat dit doen). Als het systeem van hoge kwaliteit is (tussen 3,75 en 5, bijvoorbeeld) dan kost een Java-project redelijkerwijs rond de 20 euro per regel code.

Als de technische kwaliteit tussen de 2,25 en 3,75 ligt is het project van middelmatige kwaliteit. Het hele project kost dan bijvoorbeeld 6 euro per regel code.

En als het project tussen de 1 en de 2,25 scoort, dan is het van lage kwaliteit en is 2 euro een redelijke prijs per regel code.

	Meetbaar	Vergelijkbaar	Uitvoer gerelateerd	Reële prijs
Regels code met vaste technische kwaliteit.	++ Zowel de technische kwaliteit als de hoeveelheid regels code zijn onafhankelijk en volledig automatisch te meten.	++ De tarieven per regel code zijn vergelijkbaar. Door de technische kwaliteit vast te stellen is er weinig speelruimte voor meer of minder regels bij gelijke functionaliteit.	+ Er wordt alleen betaald voor code die daadwerkelijk geproduceerd is en ook functioneel iets doet.	+ Het is op basis van benchmarks vrij goed vast te stellen of een leverancier een offerte doet die onder de kostprijs is. Voor de klant is vast te stellen of het te duur is.

Tabel 2: Voor- en nadelen van het meten van regels code

Dit leidt tot een situatie waarin een klant wellicht een goedkoper systeem krijgt als het slechter gebouwd is, maar daar staat tegenover dat het systeem meer kost in onderhoud. Hieronder staat een rekenvoorbeeld om het model inzichtelijk te maken (zie tabel 3). In dit geval wordt er opdracht gegeven een Java-systeem te bouwen van ongeveer 5000 functiepunten. In de praktijk blijkt dat systemen die slechter zijn, ook veel meer regels code nodig hebben voor dezelfde functionaliteit. Bijvoorbeeld door de codeduplicatie en de hoeveelheid dode code. Uitgaand van bovenstaande voorbeeldprijzen krijg je dan de hieronder volgende som. Hierbij is uitgegaan van een gemiddeld onderhoud van 15 procent per jaar, dat wil zeggen, per jaar wordt op een of andere manier 15 procent van het systeem aangepast. Wat opvalt is dat bijvoorbeeld lage kwaliteit code in eerste instantie een winst van 350.000 euro oplevert voor de opdrachtgever, maar omdat er twee man extra nodig zijn voor het onderhoud is die winst binnen twee jaar weer weg. Daarnaast zal de natuurlijke beweging van de leverancier zijn om zo hoog mogelijke kwaliteit code te willen opleveren, omdat hij anders exact die 350.000 euro misloopt.

### Een outsourcingprijsmodel met een redelijke prijs en hoge kwaliteit

Met dit model is het mogelijk om de prijs van outsourcingactiviteiten op een eerlijke, objectieve manier vast te stellen. Het biedt voldoende flexibiliteit om gaandeweg het traject de eisen en wensen van de klant aan te passen. En ook voldoende flexibiliteit om dat te doen zonder dat de winst van de leverancier ernstig onder druk komt te staan, en dat helpt weer in de projectuitvoering. Ten slotte zorgt het model ervoor dat de leverancier er baat bij heeft om kwalitatief goede software te leveren. Dat zal in de onderhoudsfase erg veel geld schelen.

#### Over de auteur:

Dr. Tobias Kuipers is oprichter en technisch directeur van de Software Improvement Group.

#### Voetnoten

1 Software Productivity Research, SPR Programming Language Table; Gartner, Worldwide IT Benchmark; QSM, QSM Function Point Programming Languages Table

2 Een verder uitgewerkt voorbeeld is te vinden in: Heitlager, Kuipers, Visser, A Practical Model for Measuring Maintainability

	Opdracht	Gebouwde regels code	Kosten bouw	Kosten onderhoud, jaarlijks
Hoge kwaliteit	5000 FP	50.000	750.000	1 manjaar
Middel kwaliteit	5000 FP	100.000	600.000	2 manjaar
Lage kwaliteit	5000 FP	200.000	400.000	3 manjaar

Tabel 3: Rekenvoorbeeld