Notitie over programmatuur in de WBSO
Prof.dr.ir. Jan Friso Groote
Prof.dr. Marielle Stoelinga
Juli 2020

1. WBSO en programmatuur

De Wet Bevordering Speur- en Ontwikkelingswerk (WBSO) is een fiscale stimuleringsregeling voor Speur- en Ontwikkelingswerk (S&O) bij bedrijven. De doelstelling van de WBSO is verhoging van de S&O-inspanningen van bedrijven. De WBSO beoogt daarnaast het vestigingsklimaat voor S&O-bedrijvigheid te verbeteren. S&O dat thans kwalificeert voor de WBSO betreft 'technisch wetenschappelijk onderzoek' en de 'ontwikkeling van technisch nieuwe producten, productieprocessen en programmatuur'.

De Wet vermindering afdracht loonbelasting en premie voor de volksverzekeringen definieert programmatuur als volgt:

Het niet-fysieke, logische deelsysteem van een informatiesysteem dat de structuur van de gegevens en van de verwerkingsprocessen bepaalt voor zover dat deelsysteem is vastgelegd in een formele programmeertaal.

In 2018 is de WBSO geëvalueerd. Het evaluatierapport noemt drie belangrijke bezwaren van de huidige definitie van programmatuur zoals die door de WBSO gehanteerd wordt (Sectie 3.6.3, m.n p. 77 - 80)

1. Er wordt te veel nadruk gelegd op het programmeren zelf, en te weinig op de stappen die hieraan vooraf gaan: Feitelijk is programmeren het sluitstuk van het oplossen van een S&O vraagstuk, niet de kern ervan. Het traject voorafgaand aan programmeren (softwarearchitectuur, algoritmiek, etc.) is veel belangrijker in het oplossen van technische knelpunten dan het programmeren zelf.
2. Embedded en besturingssoftware kwalificeert niet voor de WBSO.
3. De definitie van programmatuur strookt niet met de manier waarop tegenwoordig softwareontwikkeling plaatsvindt. De WBSO veronderstelt een lineair ontwikkelingsproces, terwijl software steeds meer iteratief ("agile") wordt ontwikkeld. Hierdoor wordt het moeilijk om vooraf in een WBSO-aanvraag te beschrijven wat het uiteindelijke resultaat van een softwareproject is, wat precies de inbreng van de diverse partijen is, en wanneer het af is.

Deze notitie is een herbezinning op de definitie van programmatuur. Specifieke vragen die in deze notitie beantwoord worden zijn:

I. De WBSO is gericht op het stimuleren van de ontwikkeling van technisch nieuwe programmatuur. Sluit de huidige definitie van programmatuur in de *Wet vermindering afdracht loonbelasting en premie voor de volksverzekeringen* (waarbij is vereist dat deze programmatuur moet zijn vastgelegd in formele programmeertaal) aan op dit uitgangspunt?
II. In de *Regeling S&O-afdrachtvermindering* zijn nadere bepalingen opgenomen die aangeven welke werkzaamheden niet in aanmerking komen als S&O. Sluiten deze bepalingen aan op het uitgangspunt dat de WBSO de ontwikkeling van technisch nieuwe programmatuur stimuleert?

Het ontwikkelen van een nieuwe en betere definitie van het begrip programmatuur kan alleen als we vaststellen aan welke criteria een definitie moet voldoen. Wij hanteren de volgende criteria.

Allereerst moet een nieuwe definitie *effectief* zijn, "fit for the purpose". Dit wil zeggen dat door een nieuwe definitie van programmatuur de hoofddoelstelling van de WBSO zo goed mogelijk gerealiseerd wordt, namelijk het verhogen van S&O-inspanningen door bedrijven.

Daarnaast dient een nieuwe definitie *efficiënt en operationeel* te zijn: Er moet op een eenvoudige manier vast te stellen zijn of een WBSO-aanvraag voor subsidie in aanmerking komt en ook of een gerealiseerd project aan de subsidievoorwaarden voldoet. In het bijzonder betekent dit ook dat de definitie juridisch houdbaar is in rechtszaken.

Tenslotte geldt dat een definitie van programmatuur (en WBSO-subsidiëring in het algemeen) moet aanzetten tot *goede software-engineeringmethoden*. Software is zo belangrijk, zowel economisch als maatschappelijk, en softwarefalen is dusdanig kostbaar dat het zeer wenselijk is om kwalitatief goede software te stimuleren. Het is relatief eenvoudig om een eenvoudig prototype te ontwikkelen ("in elkaar te knutselen") en een werkingsprincipe aan te tonen. Het schrijven van goede, dat wil zeggen kwalitatief hoogwaardige en onderhoudbare software vergt een heel ander soort inspanning.

Deze onderzoeksopdracht beperkt zich tot een technisch- inhoudelijke observaties en laat verdere politieke, beleidsmatige, juridische, budgettaire en uitvoeringstechnische aspecten buiten beschouwing.

2. Trends in programmatuurontwikkeling

Software is een technisch hulpmiddel dat zijn gelijke niet kent. In essentie is software een instructievoorschrift dat wordt uitgevoerd door een computer, in het algemeen in interactie met de omgeving. Door computers onderling aan te sluiten op actuatoren zoals robots is de computer in staat zijn fysieke omgeving zonder supervisie van een mens te beïnvloeden. Door gebruik van communicatienetwerken zijn computers in staat onderling hun gedrag af te stemmen (strikt gestuurd door de programmatuur) en onafhankelijk en snel tot beslissingen te komen.

Wij beschrijven zes tendensen in de software engineering die op dit moment spelen bij de ontwikkeling van software. Gelet op het doel van de WBSO – het stimuleren van innovaties die leiden tot meer winstgevendheid en concurrentiekracht van bedrijven – zal een toekomstbestendige definitie van programmatuur rekening moeten houden met deze trends.

1. Complexiteit

De omvang van programmatuur, de snelheid waarmee ze uitgevoerd wordt en de mogelijkheden om te communiceren zijn ongekend. Dit vindt zijn oorzaak in de extreme miniaturisering van elektronica, waardoor zeer grote en snelle geheugens, verwerkingseenheden en communicatienetwerken kunnen worden gemaakt. In dit alles is programmatuur de menselijke maat inmiddels ver voorbij. Deze ontwikkeling blijkt uitstekend te worden vervat in de wet van Moore uit 1965, die zegt dat het aantal transistoren op een chip iedere 1.2 jaar verdubbelt [1]. Vooralsnog is er geen enkele aanwijzing dat de wet van Moore de komende decennia haar geldigheid zal verliezen. Op dezelfde wijze is vastgesteld dat succesvolle software stabiel met een factor 1.2 groeit per jaar [2]. In beide gevallen is er sprake van exponentiële, dus op termijn zeer dramatische groei.

De feitelijke limiet aan de complexiteit van software wordt al lang niet meer bepaald door de omvang van de geheugens of de snelheid van computers, maar primair door de limiet aan de complexiteit die menselijke programmeur kan doorzien en hanteren. Typisch geldt hier dat er een mensjaar nodig is voor het beheer van 10 tot 50 duizend regels code [3].

Aanbeveling: Het hanteerbaar maken van complexiteit vereist dat programmatuur begrijpelijk is.

2. Software integratie en low coding platforms

Om software hanteerbaar te maken is een enorme software infrastructuur ontstaan. De meeste software die gemaakt wordt is software die wordt geïnterpreteerd door andere software, die vaak zelf ook weer gebouwd is in daartoe geëigende softwareomgevingen. De nieuwe softwarelagen bieden steeds hoger niveau primitieven aan vaak geschikt voor gespecialiseerde domeinen. De specialisten in deze domeinen kunnen op deze manier binnen hun eigen intellectuele vermogens de benodigde functionaliteit realiseren. De hulpmiddelen die zij gebruiken lijken soms wel, maar soms ook in het geheel niet op wat we aan zouden kunnen duiden als programmeertalen. Interfaceontwikkelaars 'programmeren' een interface door middel van 'drag and drop'.

Dit betekent dat de meeste ontwikkeling van software niet bestaat uit het programmeren van een geheel nieuw systeem, maar uit het realiseren van nieuwe toepassingen door aanpassing van bestaande code. Dit houdt in het integreren van software in de eigen context, om daarmee nieuwe functionaliteit te integreren, zoals het opnemen van een diep neuraal netwerk om de kwaliteit van bepaalde dienstverlening te verbeteren.

Aanbeveling: De definitie van programmatuur dient het aanpassen en integreren van software te ondersteunen, ook als dit middels low coding platforms plaatsvindt. Leidend is of daarbij technische knelpunten opgelost worden.

3. Legacy

De menselijke limiet aan het begrijpen van softwarecomplexiteit leidt tot het probleem dat software niet meer wordt begrepen. Dit heet het *legacyprobleem.* Waar een eerste maker van programmatuur nog wel in staat is om een eerste versie te maken voor een toepassing, blijkt het vaak niet mogelijk de programmatuur operationeel te houden als de toepassing of context van aard wijzigt. Activiteiten om de software begrijpelijk te houden zijn essentieel om de software op lange termijn bruikbaar te houden. De levensduur van software is vaak vele malen groter dan de levensduur van de hardware waarop het draait.

Aanverwant aan het probleem van het niet begrijpen van de structuur is het probleem van bestaande software die weliswaar werkt voor een bepaalde toepassing, maar die niet geschikt is als basis voor een uitgebreidere toepassing omdat de kwaliteit ervan onvoldoende is. In dit kader is het zinnig te weten dat het gemiddeld aantal fouten in code in de orde ligt van 1 tot 50 fouten per duizend regels code [3]. De meeste van deze fouten zijn slapende fouten in die zin dat ze onder het vigerende gebruik van de software niet worden geactiveerd.

Aanbeveling: Onderhoud van software is net zo belangrijk als het schrijven van software. Ook dit zou WBSO-subsidiabel moeten zijn, maar natuurlijk alleen wanneer dit tot technische vernieuwing van de software leidt.

4. Agile development

Agile software development is een reactie op de vaststelling dat zeer strikte ontwikkelingsprocessen vaak niet goed werken bij de constructie van software. Door frequent overleg en adaptieve planning wordt tegenmoet gekomen aan de observatie dat er bij veel softwareontwikkeling sprake is van groeiend inzicht en meer dan eens het opdoemen van onvoorziene problemen. Het onbepaalde aspect van agile ontwikkeling lijkt bijzonder op de wijze waarop onderzoek wordt gedaan. Voor de WBSO lijkt het niet erg van belang of een project agile wordt uitgevoerd. De doelstelling van een project en de mate waarin het technisch nieuw is, zijn bepalend.

Aanbeveling: Voor de WBSO lijkt het niet relevant of een project op een 'agile' wijze wordt uitgevoerd. De doelstellingen van een project moeten wel helder zijn en niet agile geformuleerd worden.

5. Model-driven engineering en domein-specifieke talen

Model-driven engineering (MDE) is een methodologie voor softwareontwikkeling waarin domeinmodellen centraal staan, dat wil zegge dat conceptuele modellen de relevantie van alle aspecten van een specifiek toepassingsdomein in kaart brengen. MDE voorziet in abstracte representaties van de kennis en activiteiten binnen het toepassingsdomein, in plaats van de algoritmen en code. MDE wordt algemeen gezien als een effectieve methode om op een efficiënte manier kwalitatief hoogwaardige code te ontwikkelen: Het gebruik van gestandaardiseerde modellen verhoogt de kwaliteit, de compatibiliteit tussen systemen (door hergebruik van gestandaardiseerde modellen) en vereenvoudigt het ontwerpproces (via modellen van terugkerende ontwerppatronen). Waar programmeren meer het sluitstuk is van het oplossen van een S&O vraagstuk, wordt de kern ervan vaak wel vastgelegd in domein-specifieke talen.

Aanbeveling: domein-specifieke talen dienen onder de definitie van programmatuur te vallen. Dit geldt zowel voor bestaande talen, zoals SQL en ASP, als talen die nieuw ontwikkeld worden. Voorbeelden van deze laatste zijn modelleertalen en meta-modellen om communicatie tussen IoT devices te beschrijven.

6. Artificial intelligence en big data

De relevantie van artificiële intelligentie en big data kan nauwelijks worden onderschat. AI, en met name machine learning technieken hebben implicaties voor de definitie van programmatuur, omdat systemen met AI niet langer in de traditionele zin worden geprogrammeerd middels het geven van instructies. In plaats daarvan wordt het gedrag van een systeem geleerd op basis van voorbeelden. Softwaresystemen die (mede) tot stand komen middels machine learning zijn niet geschreven in een formele programmeertaal. Toch zijn hier technische knelpunten op te lossen, bijvoorbeeld door het samenstellen van een geschikte dataset aan trainingsdata, de validatie en cleaning van deze data.

Aanbeveling: Gezien de grote vlucht die AI neemt in allerlei toepassingen, en de enorme economische mogelijkheden, vinden wij dat AI toepassingen, zoals machine learning, WBSO-subsidiabel moeten zijn. Concreet kan gedacht worden aan het vinden van de juiste instructies en instellingen om een neuraal netwerk effectief te laten leren, of het opstellen van een methode om trainingsdata op te schonen.

3. Herformulering van de frase 'Formele programmatuur'

Een professionele beschrijving van software bestaat uit veel meer dan de programmatuur zelf. De programmatuur is veeleer een uiting van de software en veel minder de kern. Door de grote complexiteit van programmatuur is de softwarecode eerder een bedreiging dan een ondersteuning. Het is essentieel dat programmatuur vergezeld gaat van precieze en compacte beschrijvingen. Zulke beschrijvingen kunnen bestaan uit natuurlijke taal en diagrammen, maar die schieten snel tekort. De voorkeur bestaat om een zo wiskundig mogelijke beschrijving te geven. Te denken valt aan pseudo-algoritmen en beschrijvingen in (semi)-formele specificatietalen en architectuurdiagrammen.

Het hanteren van het begrip "formeel" is problematisch omdat het meer moet overdekken dan zij doet. Onder het begrip "formele programmeertaal" wordt verstaan een taal waarvan de semantiek mathematisch precies is vastgelegd. Voor vrijwel alle programmeertalen geldt dat de semantiek inmiddels behoorlijk goed, maar nog steeds in natuurlijke taal is vastgelegd. Daarmee wordt strikt genomen vrijwel alle programmatuur gediskwalificeerd.

Aan de andere kant geldt meer en meer dat programmatuur bestaat uit zeer specifieke input voor domein specifieke talen waaruit de uiteindelijke programmatuur wordt gegenereerd. Machines worden aangestuurd door het opsommen van de uit te voeren taken, samen met hun conflicten, sequentiële afhankelijkheden en verdere constraints. Hieruit wordt door constraint solver de code gegenereerd. Dergelijke beschrijvingen zijn nauwelijks herkenbaar als een programma in een programmeertaal. Toch zal dit op termijn de manier zijn om technisch nieuwe machinerie te maken. Het lijkt de intentie van de WBSO om zulke moderne wijzen van programmeren te stimuleren.

Dit leidt tot de volgende alternatieve formulering van het begrip programmatuur. Deze definitie van programmatuur is ruimer dan enkele bestaande definities, die programmatuur beperken tot programmeercode in een direct executeerbare programmeertaal.

Programmatuur is een precieze beschrijving van waaruit het gedrag van een computergestuurd systeem eenduidig kan worden afgeleid, op zodanige wijze opgesteld dat het door mensen navolgbaar is en waarbij het technisch of mathematisch verifieerbaar is dat het beoogde doel ermee wordt bereikt.

De eisen die worden gesteld aan de beschrijving leiden ertoe dat abstracte representaties in modellen, mathematische bewijsvoeringen en domeinspecifieke talen hieronder vallen. De laatste clausule vereist dat de beschrijving soms zo compact is dat zij door een voldoende getraind expert goed te begrijpen is. Maar meestal betekent het dat de software is voorzien van voldoende en precieze documentatie opdat goed te achterhalen is wat er wordt bedoeld. In deze zin is een programma van 100.000 regels Java geen programmatuur indien de structuur daarvan niet separaat is gedocumenteerd.

Controlemechanismen

Het is gewenst dat de WBSO eenduidig controleerbaar is. Dit is sterk in overeenstemming met de noodzaak dat programmatuur zo wordt opgeleverd dat zij door mensen met de nodige maar niet excessieve competentie kan worden beoordeeld. Dit betekent dat de programmatuur navolgbaar moet worden gedocumenteerd om voor de WBSO in aanmerking te komen.

Concreet moet een aanvraag bestaan uit een helder technisch doel. Dat kan de constructie van een nieuw type programma zijn, maar ook het uitvinden op welke wijze legacy-programmatuur kan worden gereviseerd, alsmede het op hernieuwde wijze documenteren van bestaande software, het opstellen van een nieuwe softwarearchitectuur, of het invoeren van nieuwe test- of verificatietechnieken.

Opgeleverd moet worden een concreet voorschrift waaruit de programmatuur kan worden gegenereerd, op zodanige wijze opgesteld dat het menselijk navolgbaar is en daarmee controleerbaar is. In die gevallen dat er technieken worden toegepast om de kwaliteit van bestaande programmatuur te verbeteren, zal er een beschrijving dienen te komen van de wijze waarop de nieuwe technologie kan worden toegepast, samen met een concreet voorbeeld van gereviseerde programmatuur, die voldoet aan boven gegeven definitie van programmatuur. Samenwerking met het bureau ICT-toetsing om te valideren of uit een gegeven beschrijving het gedrag van een computergestuurd systeem is af te leiden kan worden overwogen.

4. Implicatie voor activiteiten van speur- en ontwikkelingswerk

De huidige definitie van programmatuur lijkt ervan uit te gaan dat de ontwikkeling van programmatuur voornamelijk bestaat uit het schrijven van één programma voor één toepassing. De werkelijkheid is inmiddels veel breder. De focus op het ontwikkelen van formele programmatuur lijkt zelf een stimulans op het ontwikkelen van meer programmatuur, waardoor de kosten door legacy, gebrek aan documentatie en de ontwikkeling van foutvolle programma's oplopen en zelfs tot economische schade kunnen leiden.

Inclusie van werkzaamheden

Wij zijn van mening dat, gezien bovengenoemde argumentatie, de activiteiten te strikt zijn, en daardoor innovatie belemmeren. Het WBSO-programma zou ook ondersteuning moeten leveren aan:

- Herstructurering van legacy-software.
- Documentatie van de programmatuur inclusief precieze specificatie van externe interfaces. Het enumereren van aanroepbare functies van een softwarepakket is niet genoeg en niet vernieuwend. De interfaces moeten eenduidig beschreven zijn, bijvoorbeeld door een toestandsautomaat of een intern conceptueel model.
- Integratie van verschillende softwarepakketten om tegen lage kosten nieuwe functionaliteit beschikbaar te stellen, door gebruik te maken van hoog niveau, domein specifieke beschrijvingen.
- Verbetering van de intrinsieke kwaliteit van software, bijvoorbeeld door een verbeterde testmethodiek of softwareverificatie. Concreet valt te denken aan model-based testing, of model checking.

Van belang blijft dat de methodes nieuw zijn voor de organisatie die haar toepast, en dat ze haar software technisch vernieuwt.

Dit heeft tot gevolg dat de volgende activiteiten wel in aanmerking moeten komen voor subsidiëring. Wij merken op dat bij al deze activiteiten zich serieuze technische knelpunten kunnen voordoen en dus S&O-activiteiten vereisen.

1. Activiteiten ter herstructurering en documentatie van software.
2. Activiteiten om de kwaliteit van bestaande software fundamenteel te verbeteren, bijvoorbeeld door het verwijderen van slapende softwarefouten.
3. Activiteiten om op basis van programmatuur van derden, nieuwe eigen functionaliteit te realiseren.
4. Ontwikkelingstrajecten waarbij een consortium van bedrijven werkt aan nieuwe toepassingen. Er moet daarin ruimte bestaan om gedurende de ontwikkeling verschillende taken dynamisch binnen het consortium te verschuiven.
5. Activiteiten om bestaande software binnen een fundamenteel nieuw executieplatform uit te voeren. Te denken valt aan het omzetten van sequentiële software naar een multiprocessor, gedistribueerd of zelfs een GPU platform. Hierbij kan ook gedacht worden aan het redundant maken door programmatuur resistent te maken tegen de uitval van een of meerdere computers.

In het bijzonder noemt de huidige Regeling S&O-afdrachtvermindering acht punten die uitgesloten zijn als speur- en ontwikkelingswerk. Gezien bovengenoemde argumentatie, stellen wij voor om wel de volgende activiteiten WBSO-subsidiabel te laten zijn:

- Onderhoud van programmatuur, mits dit aantoonbaar in technische zin vernieuwend is.
- Het beschrijven van architectuur.
- Het geschikt maken van bestaande programmatuur voor een ander hardware- of softwareplatform waarbij onder platform wordt verstaan het geheel van hardware en besturingsprogrammatuur waarop informatiesystemen worden ontwikkeld (ontwikkelplatform) of in productie worden genomen (doelplatform);
- Het ontwikkelen van programmatuur die bestaande programmatuur op een voor de S&O-inhoudingsplichtige of S&O-belastingplichtige technisch nieuwe wijze integreert of laat samenwerken, tenzij de bestaande programmatuur hoofdzakelijk binnen de onderneming van de S&O-inhoudingsplichtige, binnen de fiscale eenheid waarvan de S&O-inhoudingsplichtige deel uitmaakt, of binnen de onderneming van de S&O-belastingplichtige, is ontwikkeld en wordt toegepast.

Uitsluiting van bepaalde werkzaamheden

De volgende projecten en activiteiten zouden volgens ons van WBSO-subsidie uitgesloten moeten worden/blijven:

1. Projecten waarvan de doelstellingen onhelder geformuleerd zijn en waarvan de doelstellingen op een agile manier al doende helder moeten worden.
2. Het opstellen of bepalen van functionele eisen en randvoorwaarden, tenzij de formulering zo precies is dat zij volgens de nieuwe definitie als software is aan te merken.
3. Software die ontwikkeld of ingeregeld wordt om de eigen organisatie te laten functioneren.
4. Niet structureel onderhoud of aanpassingen aan programmatuur. Hieronder vallen bijvoorbeeld noodzakelijke aanpassingen voor een nieuwe versie van een operating systeem of aanpassingen noodzakelijk door aangepaste wetgeving.

In het bijzonder kunnen de volgende punten uit de huidige Regeling S&O-afdrachtvermindering uitgesloten blijven van WBSO-subsidiering:

- *Werkzaamheden met betrekking tot het invoeren en aanpassen van aangeschafte of aan te schaffen technologie, producten, processen of programmatuur;*
- *Werkzaamheden, door de S&O-inhoudingsplichtige of S&O-belastingplichtige verricht ten behoeve van door een ander verricht speur- en ontwikkelingswerk, die op zichzelf niet zijn aan te merken als speur- en ontwikkelingswerk;*
- *Het opstellen of bepalen van functionele eisen en randvoorwaarden;*
- *Het ontwerpen of bouwen van een nieuw systeem;*

5. Referenties

[1] Gordon E. Moore. Cramming more components onto integrated circuits. Electronics. Pages 114--117. 1965. Reprinted in the Proceedings of the IEEE 86(1):82-85, 1998.
[2] Genuchten, M. and L. Hatton. Compound Annual Growth Rate for Software. IEEE Software 29(4):19-21.
[3] Jones. C. Software assessments, benchmarks and best practices. Addison-Wesley. 2000.