Albert Mietus
albert@mietus.nl

Hoe programmeer je de multi-mens?

Computers zijn net mensen. Dit geldt zeker bij een gemiddeld software-engineeringproject: sequentieel werk gaat redelijk efficiënt maar parallellisme vraagt om samenwerken, wat lastig is en overheadkosten geeft. Toch is dat vaak nodig. Anders zou een project van honderd mensjaar een eeuwigheid duren. Of we willen of niet, we moeten het leren: werk opdelen in steeds kleinere brokjes en die brokjes razendsnel en efficiënt verdelen. Zowel in onze programma's als bij onze programmeurs.

Er zijn, technisch, vele manieren om parallellisme te organiseren. Elk met hun eigen voor- en nadelen. Zoals de klassieke keuze: multithread versus multiproces. Toen ik nog programmeerde, prefereerde ik processen. Elk stukje functionaliteit in een eigen, stabiel proces met expliciete interfaces in plaats van in threads die ad hoc kunnen verschijnen en verdwijnen. Inmiddels lijken de thread-adepten echter aan de winnende hand. Maar welke vorm van 'multi' we ook hebben, de vraag hoe we het werk efficiënt organiseren, staat centraal.

Mijn uitdaging is inmiddels verschoven van techniek naar projecten. Ik heb echter nog steeds dezelfde vraag. Bij projecten spreken we niet over threads of cores, maar over teams en mensen. Die 'verwerken' features en taken tot een bruikbaar product. Om de time-to-market laag te houden, zijn 'multimensen' nodig. Die moeten efficiënt samenwerken, zowel in als tussen de teams.

Ook hier zien we een verschuiving van zware processen naar lichte, zelfsturende teams. Agile-teams zijn niet alleen goedkoper, telkens weer blijkt dat het hun beter lukt om projecten op tijd af te ronden. Momenteel lijkt de Agile-aanpak om productkenmerken aaneen te rijgen efficiënter dan het klassieke V-proces. Processen lijken dus op hun retour; ze geven al gauw te veel overheadkosten. Als (ex-)programmeur vind ik dat jammer, als projectleider een vooruitgang.

Natuurlijk zijn computers geen mensen. Computers doen nu eenmaal wat je vraagt. En mensen wat je bedoelt. Althans met een beetje geluk. Natuurlijk doen ze ook wel eens iets fout. Daarom zijn testers zo belangrijk in het team. Zij vinden fouten die de programmeurs maken. Slechts door samen te werken, krijgen we een werkend geheel. Dit zit impliciet in hun vakmanschap en het is daarom niet nodig om het expliciet vast te leggen in het proces.

Dit geldt ook technisch. Iedereen is gewend aan web-apps, waarbij samenwerken tussen browser en server essentieel is. Toch werken ze ook onafhankelijk. Doordat de samenwerking impliciet is, is dit allemaal heel logisch. Dit geldt telkens weer: waar er een logische, impliciete manier van samenwerken is, is het niet nodig om dat expliciet vast te leggen. Dat is overbodig en dus te duur.

Maar soms is die impliciete, natuurlijke manier van samenwerken niet voorhanden. Zo leid ik nu een project waar een aanzienlijk stuk code geïsoleerd en verplaatst moet worden. Daarvoor moeten ook twee domeinmodellen geüniformeerd worden. Inclusief testen is dit een kleine honderd mensdagen werk. Voor een project met 25 mensen is dat minder dan één week. Toch zal het niet lukken in een week. Sterker nog: het leek onmogelijk om met meer dan twee of drie mensen parallel te werken, wat zou betekenen dat het niet in één sprint past.

> Processen lijken op hun retour; ze geven al gauw te veel overheadkosten

In dit soort gevallen is het toch handig om het samenwerken expliciet te maken. Voor bovenstaande feature gebruik ik Gantt-charts, die we nog kennen uit het klassieke projectmanagement. Daarmee moduleren we parallelle brokken werk en maken we afhankelijkheden expliciet. Die hebben we vervolgens doorgerekend met algoritmes uit realtime-schedulers, zoals *earliest deadline first*, om inzicht te krijgen in de minimale sprintlengte. Vervolgens gaan we op de Agile-methode verder.

Dit blijkt te werken. Diverse mensen van één team werken nu samen aan deze klus, zonder continu te moeten overleggen met de andere teams. En toch is de sprint kort en kunnen de meeste mensen efficiënt zelfstandig werken. Door de opties voor parallellisme expliciet te maken, leren de programmeurs om ook bij lastige gevallen efficiënt samen te werken.

Hoe we dat ook kunnen leren aan computers en programma's, leest u elders in dit nummer.