AKADEMOS Informatieblad van de Vrije Universiteit Brussel
jaargang 4 (2001) nummer 1

SOFTWARE-ARCHITECTUUR ALS MIDDEL TEGEN SOFTWARE-EROSIE

Computerprogramma s kunnen vele duizenden, zelfs miljoenen lijnen code bevatten. Als je bedenkt dat dit allemaal instructies zijn die correct moeten samenwerken, dan kom je snel tot de conclusie dat computerprogramma s tot de meest complexe bouwsels behoren die de mens ooit heeft verwezenlijkt. Het beheersen van de complexiteit van computerprogramma s is g n eenvoudig probleem. Alhoewel de meeste hedendaagse programmeertalen toelaten te programmeren op een zeer hoog niveau van abstractie, blijft het vaak moeilijk een globaal overzicht te krijgen van een programma. Software-architectuur kan hier een oplossing bieden. Naar analogie met architectuur van gebouwen is software-architectuur een manier om de globale structuur van een complex en grootschalig computerprogramma samen te vatten in een summier en duidelijk overzicht met slechts een beperkt aantal elementen. Dr. Kim Mens, VUB-laboratorium voor programmeerkunde, deed zijn doctoraatsonderzoek naar software-architectuur in het algemeen en naar *conformiteitscontrole* ter bestrijding van *software-erosie* in het bijzonder.

Software-architectuur? Wat moet een informatica-leek zich daar bij voorstellen?

Dr. Kim Mens: Net zoals een plan van een gebouw uit verschillende gezichtspunten kan bestaan (vooraanzicht, zijaanzicht, grondplan, elektrische bedrading,) kan het ook nuttig zijn een computerprogramma vanuit verschillende *architecturale gezichtspunten* te documenteren. Vanuit n gezichtspunt kunnen we een programma bekijken als een hoop samenwerkende processen en zijn we ge nteresseerd in welke processen weer andere processen oproepen. Een ander gezichtspunt kan beschrijven hoe de software is opgedeeld in verschillende onafhankelijke modules. Nog een gezichtspunt kan zich dan concentreren op de gegevens die door het programma worden verwerkt. Of we kunnen ge nteresseerd zijn in een meer abstract gezichtspunt dat beschrijft hoe en waar de belangrijke concepten uit het probleemdomein terug te vinden zijn in het programma."

Welke voordelen biedt dergelijke software-architectuur?

Dr. Kim Mens: "Het grote voordeel van al deze architecturale gezichtspunten is dat ze toelaten het programma te bekijken vanuit verschillende perspectieven. Afhankelijk van over welk aspect van het programma we willen redeneren - bijvoorbeeld kunnen we een bepaalde functionaliteit toevoegen? Is het programma snel genoeg? Welke gegevens worden het meest gebruikt? - kan het ene perspectief al interessanter zijn dan een ander. In elk geval vormen deze architecturale gezichtspunten een ideale documentatie en communicatiemiddel tussen de verschillende ontwikkelaars van het programma. Architectuur is echter m r dan alleen documentatie. Een programma met een duidelijke architectuur is veel gemakkelijker te begrijpen, uit te breiden, aan te passen en te onderhouden dan een programma zonder weldoordachte architectuur."

Maar .

Dr. Kim Mens: "Software-architectuur staat voor n belangrijk probleem en dat is dat software snel de neiging heeft te eroderen . Met andere woorden, zelfs voor programma s die aanvankelijk een goed gedocumenteerde architectuur hebben, gebeurt het vaak dat onder tijdsdruk enkel de programmacode wordt gewijzigd om bijvoorbeeld fouten te verbeteren of een functionaliteit toe te voegen, zonder dat de architectuur mee wordt aangepast. Het duurt dan meestal ook niet lang voor we met een volledig achterhaalde architectuur zitten waar we nog weinig aan hebben, behalve dan een momentopname van hoe het programma ooit gestructureerd was. Dit probleem van *software-erosie* is n van de voornaamste redenen waarom programmeurs vandaag maar zeer sporadisch de architectuur van hun programma s documenteren. Waarom zouden ze ook: enerzijds kruipt er vrij veel tijd in, en anderzijds is de gedocumenteerde architectuur toch binnen de kortste keren achterhaald in vergelijking met de eigenlijke programmacode."

Vandaar dat er nood is aan technieken om dit probleem tegen te gaan.

Dr. Kim Mens: "Inderdaad. Op zijn minst hebben we een techniek nodig om na te kunnen gaan of de programmacode nog steeds consistent is met een gegeven architectuur en, indien niet, informatie te geven over waar er inconsistenties tussen de programmacode en de architectuur bestaan. *Conformiteitscontrole* is zo n techniek. Ik zal niet in detail treden, maar het komt erop neer dat we een afbeelding tussen een architectuur en de programmacode defini ren en op basis hiervan de consistentie controleren. Als er n of andere relatie bestaat tussen elementen op niveau van een architectuur, dan verwachten we deze relatie ook terug te vinden in de programmacode. Dit ligt allemaal nogal voor de hand, maar de moeilijkheid ligt vooral in het defini ren van een goede afbeelding tussen een architectuur en de programmacode.

Hoe ziet zo n afbeelding van een architectuur naar een programma er dan uit?

Dr. Kim Mens: Het volstaat om voor elk element op architectuurniveau te vermelden met welke programma-entiteiten het overeenstemt. Een probleem is echter dat het om heel veel programma-entiteiten kan gaan, en dat die kunnen verspreid zitten over het gehele programma. Het is dus in de praktijk bijna onmogelijk om zo n afbeelding met de hand op te stellen. Daarom beschrijven we deze afbeelding door middel van een meer compacte abstracte beshrijving. Deze aanpak biedt tal van voordelen. Zo is er *minder geheugen* nodig om de abstracte beschrijving te stockeren dan om de honderden elementen die aan de beschrijving voldoen op te slaan. Bovendien is de abstracte beschrijving zeer *leesbaar* en *eenvoudig te begrijpen*. Ze legt duidelijk het gemeenschappelijke kenmerk vast van de programma-entiteiten die met een bepaald architectuurelement overeenstemmen. En, last but not least, de beschrijving is *robuust t.o.v. veranderingen*. Wanneer het programma verandert levert de abstracte beschrijving na berekening een aangepaste verzameling van programma-entiteiten op, dit in tegenstelling tot wanneer de entiteiten expliciet opgesomd zouden zijn. Deze manier om de architectuur van een programma voor te stellen en de conformiteit van de code ten opzichte van deze architectuur na te gaan, is dus zeer eenvoudig en intu tief. Het lijkt een veelbelovende eerste stap om het probleem van software-erosie op te lossen."

SVM