toolview

Albert Mietus'

Check

Elke ingenieur gebruikt tools en iedereen heeft zijn eigen voorkeur; vaak gebaseerd op wat er toevallig als eerste beschikbaar was. Zo ook de ICT-ingenieur. Ook hij gebruikt tools, software-tools wel te verstaan. Daarvan zijn er oneindig veel, waardoor kiezen erg moeilijk is. Veel handige tools worden niet gebruikt, eenvoudigweg omdat niet bekend was dat er een tool is, specifiek voor dat probleem.

Deze rubriek stelt degelijke, handige tools voor. Ken je ook een handig tool, neem dan contact op; dan schrijf ik er over! Mail me, voor meer info: ALbert dot Mietus at PTS dot nl

Bij systematisch software ontwikkelen denkt iedereen meteen aan het V-model. Met testen als sluitpost; althans dat is waar testexperts bang voor zijn. Met 'Early testing' kan dat voorkomen worden. Maar kunnen we testen voordat we coderen? Ja, dat kan! Een tool als 'Check' maakt dat we een test kunnen programmeren voordat we de module implementeren.

Testen specificeren

Wanneer werkt software? Als het aan de specificaties voldoet, of als het goed getest is? En wanneer is een software-module (of unit) goed? Als hij doet wat hij moet doen? Of moet je ook de bruikbaarheid van de interface testen?
Niet iedereen denkt zo over testen. Sommige mensen willen gewoon 'even' proberen of een stukje software werkt naar verwachting. Maar is dat anders? Dit 'even' valt vaak tegen. Het kost gewoonlijk veel tijd: tijd om 'weer' een stukje code te schrijven, tijd om de test uit te voeren en tijd om die testcode te debuggen...
Zou het niet handig zijn als dat we de testcode kunnen hergebruiken?

OhOh, Java en ... C

Sommige zogenaamde 'moderne' talen hebben handige testfaciliteiten ingebouwd. Bijvoorbeeld een 'object-inspector', om een object te bekijken. En voor Java zijn er heel mooie unittest frameworks. Met diverse creatieve macro's en een beetje goede wil zijn deze faciliteiten ook mogelijk in plain, old (ANSI-)C. *Check* is zo'n framework. Het is geschreven in en voor C, maar maakt hevig gebruik van de precompiler. Kennelijk is dat nodig; hoewel het de testen minder leesbaar maakt.
Het concept van *Check* is eenvoudig: maak een programma dat de module aanroept die getest moet worden. Het ontwikkelen van de module en van de testcode moet bij voorkeur incrementeel en door dezelfde programmeur gebeuren. Dan is de (te testen) interface (API) bekend. Maar ook de condities waarbij een test 'goed' of 'fout' als resultaat moet geven. De programmeur moet dat uitschrijven.
Het framework zorgt dat alle testen stuk voor stuk worden uitgevoerd, de deelresultaten worden gecontroleerd en er komt uiteindelijk één 'success' of 'failure' uit de test. Het is hierdoor niet nodig om (handmatig) alle output van de module te inspecteren.

Ervaring

Tijdens een test kan er veel fout gaan. Zo is een 'core-dump' niet uit te sluiten. Als het (test)programma dan acuut stopt, zal het resultaat niet worden geprint. Maar wat als dat gebeurt tijdens een geautomatiseerde test? De kans is dan groot dat niemand deze 'failure' zal zien! Met *Check* zal dat niet gebeuren.
Er is veel ervaring in het framework verwerkt die deze bijzonderheid - en heel veel andere - voorkomt. Bijvoorbeeld door de te testen code in een apart proces te executeren en de status van dat proces te monitoren. Een onverwacht einde van de module zal zo altijd resulteren in een 'failure'; die netjes geprint wordt.

V-, C- of www-model?

Met *Check* testen is handig en effectief. We kunnen snel beginnen met testen. Strikt genomen verlaten we daarmee het 'V-model' en gaan we cyclisch werken. Het C-model? Nou nee, het is meer een opeenvolging van een aantal kleine v-tjes. Mijn voorstel: 6! Met 6 kleine v'tjes hebben we een moderne, goedklinkende opvolger van het V-model, ook in C.

door & more

Er zijn meerdere soortgelijke testframeworks. De bekendste is JUnit, voor Java. Zowel JUnit als Check leunen op de ideeën van eXtreme programming. De cyclische aanpak staat daar centraal.

Deze manier van unit-testen wordt ook wel 'xUnit(testen)' genoemd. Voor zover ik weet is er geen universeel framework voor meerdere talen, maar zijn er meerdere implementaties van dit idee.

Naast Check, zijn er ook andere C-implementaties. Het voordeel van Check is de 'fork' optie, plus dat er geen GUI nodig is. Nadeel is de unix aanpak.
De alternatieven zijn (o.a.):
- cUnit: grafisch, geen fork
- AutoUnit: grafisch, fork-optie
- CUnit: ook voor Windows
- TUTTI: een beetje van alles wat

Keus zat dus. Nadeel van alle tools is, dat er geen onderscheid gemaakt wordt tussen white- en black-box testen, waardoor de kans ontstaat dat er verkeerd getest wordt. Een doordachte aanpak blijft dus noodzakelijk.

Kijk op http://check.sourceforge.net/