toolview

Albert Mietus'

GCOV

toolview

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. Daar zijn er oneindig veel van, 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.

Het leren gebruiken van een tool kost tijd. Maar als je de naam en het doel van het tool kent, loont het gebruik al vaak de moeite. Het inleren en inzetten kost vaak minder tijd dan aanmodderen zonder goed tool.

Deze rubriek stelt degelijke, handige tools voor; dit keer een testtool.
Ken je ook een handig tool, schrijf dan ook eens tool(re)view.

Mail me, voor meer info
ALbert dot Mietus at PTS dot nl

door & more

GCC

Een van de bekendste compilers is gcc. Ooit stond dat voor GNU-C-Compiler; maar met GCC kan ook C++, ADA, Fortran, Pascal, Objective-C en zelfs Java gecompileerd worden. Daarom staat GCC tegenwoordig voor de GNU-Compiler-Collection.

GCOV is onderdeel van die collectie. Het is een los tool, maar wel afhankelijk van de gcc compiler. Je moet 2 extra opties aan gcc meegeven (-fprofile-arcs en -ftest-coverage) om de code te instrumenteren.

De beste documentatie van GCC is te vinden in de info-files van gcc. Te bekijken met emacs, op de command-line met 'info'. En natuurlijk op 't web, waar je gcc, met gcov, ook kunt downloaden:
http://www.GNU.org.

Testen, testen, alsmaar weer testen. Veel programmeurs kunnen er niet genoeg van krijgen. Hoewel, als klanten bugs vinden dan wordt gezegd dat er te weinig getest is. Dus wanneer is er voldoende getest? Is daar een test voor? Ja! Zo'n test bestaat! Het tool 'gcov' kan tenminste meten of er al voldoende is getest. Maar er zijn meer mogelijkheden.

Volledigheid

Het maken van een test kost tijd. En het uitvoeren van een test ook. Maar één enkele test zal nooit volledig zijn. Daar zijn vaak meerdere testen voor nodig. Zo'n testset moet dan zo volledig mogelijk zijn en alles testen. Het is echter moeilijk om te bepalen of écht alles getest is. Of er voldoende testen in de set zitten. Soms blijkt, achteraf, dat niet eens alle te testen code geëxecuteerd is. Zelfs de meest domme programmeerfout in dat stuk code kan dan fataal zijn. Een goede testset moet daarom minimaal alle code raken en dat is automatisch te controleren.

Code coverage

Als een programma gecompileerd kan worden met gcc dan is het mogelijk om de code te instrumenteren, zodat het programma zelf bijhoudt welke code hoe vaak doorlopen wordt. Door deze gcc feature wordt extra informatie opgeslagen in twee files. Tijdens het testen – als het programma gebruikt wordt – wordt coverage informatie opgeslagen in een derde file; dit gebeurt cumulatief, zodat alle testen tezamen bekeken kunnen worden. Na het afronden van een testset (of een losse test) kan gcov daaruit een overzicht genereren dat per regel laat zien of en hoe vaak die code gebruikt is.

Op deze manier kan, tijdens het maken van de testset, getracht worden om alle code minimaal één keer te gebruiken. En dat met een zo klein mogelijke testset.

Snelheidsoptimalisatie

Bij parallelle executiepaden kan gcov ook laten zien hoe vaak elke tak gebruikt is. Omdat executiesnelheid vaak belangrijk is, is dat nuttige informatie om het programma te optimaliseren. Immers, het optimaliseren van statements die vaak gebruikt worden heeft het meeste effect.
Natuurlijk moet je dan wel een representatieve testset gebruiken anders optimaliseer je vooral de snelheid van je testen. Als de (eind)gebruiker het programma heel anders gebruikt, andere delen van de code gebruikt, dan is de optimalisatie nog steeds niet optimaal.

Test de test, die test

Ook bij het testen van de coverage geldt iets dergelijks. Ik heb wel eens mensen horen spreken over een perfecte test. Want de coverage was 100%. Dat klinkt goed, maar het zegt niets. Want wellicht was het functionele resultaat van die test wel onjuist. Bovendien wil 100% statement-coverage nog niet zeggen dat alle mogelijke executiepaden doorlopen zijn. Noch dat alle randcondities getest zijn. Toch wordt dat als belangrijk gezien.

Eigenlijk is een coverage test niet anders dan elke andere test. Als de coverage niet (bijna) 100% is, is de test zeker niet goed. Maar een 100% coverage wil niet zeggen dat er goed getest is. Immers, een test kan slechts aantonen dat iets niet goed is; niet dat het werkt. Een goede test probeert fouten aan te tonen. En als dat mislukt, hopen we dat het (altijd) goed is. Klinkt dat complex? Ach, zie het als een test. Om te testen of je verstand hebt van testen ...