# tool view

Albert Mietus'

## toolview splint

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.

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; zowel voor netwerk-beheer, ontwikkelen als debuggen. Ken je ook een handig tool, schrijf dan ook eens een tool(re)view.

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

door & more

## GCC's Wall

*Ook de bekende gnu compiler herkent een veelheid aan fouten. Met optie -Wall worden dergelijke fouten opgemerkt. Toch gaat dit minder ver dan een tool zoals **SPlint**, de opvolger van LcLint, die een vrijwel volledige statische analyse uitvoert.*

*Het kan nog uitgebreider; maar dan moet flink in de buidel getast worden. Met een commercieel tool als Purify kunnen (ook) dynamische analyses uitgevoerd worden.*

*Maar een gratis te gebruiken **SPlint** kan veel fouten voorkomen!*

*Voor meer info zie http://www.splint.org*

**Uit metingen blijkt dat een goede programmeur slechts één regel onjuiste code per scherm schrijft. En dat de meeste fouten triviale verschrijvingen zijn. Spel- of stijlfouten zou men kunnen zeggen. Het verbeteren van zo'n bug is dan secondewerk, tenminste nadat de fout gevonden is. Het probleem is dus niet het oplossen van de fout, maar het - liefst automatisch - vinden van elke fout. Een tool als 'SPlint' kan gezien worden als spellingchecker voor C, waarmee veel fouten automatisch gevonden kunnen worden.**

### Tikfouten

Iedereen maakt fouten; zo bevatte de eerste versie van deze rubriek heel veel (spel) fouten. Door de spellingchecker werd dit gereduceerd tot een acceptabel aantal. Dit is inmiddels heel normaal voor tekstverwerken. Een werkwijze, die net zo goed te gebruiken is voor programma-code! Met ***SPlint*** is er een gratis te gebruiken spel- en stijlcontrole hulpmiddel voor C-programmeurs.

### Gebruik

***SPlint*** wordt gebruikt naast een compiler. Net als de compiler leest het de opgegeven C-file(s), voegt het headerfiles in en kunnen ook macro's verwerkt worden. Maar in plaats van een objectfile te genereren, analyseert het je programmatekst. Bekende en minder bekende fouten worden daarna gerapporteerd. Dit gaat veel verder dan bij een compiler. Immers, zodra de syntax van de code correct is kan de compiler het verwerken. Syntaxcontrole alleen is echter niet voldoende! Neem de bekende enkele '=' in een if-expressie: syntactisch correct maar vaak onjuist. ***SPlint***, en de meeste compilers, geven daarom een melding. De controle van ***SPlint*** gaat echter veel verder. Bijvoorbeeld als een array-index buiten de grenzen van het array valt, is dat syntactisch correct. Veel compilers zien dan ook geen probleem. ***SPlint*** zoekt juist naar dergelijke gevallen. Handig, voor als we weer eens vergeten zijn dat de maximale index 1 kleiner is dan de maat van het array.

### Doortikken

De meeste programmeurs maken niet heel veel fouten, maar herhalen ze frequent. Juist daarom is ***SPlint*** nuttig; het kan zo geconfigureerd worden dat extra aandacht gegeven wordt aan die fouten. Je kunt dan lekker doortikken, in de zekerheid dat die typerende fouten achteraf nog gecorrigeerd (kunnen) worden. In elke schrijfcursus leer je dat je je eerst moet concentreren op de grote lijn om daarna pas de details te verbeteren. Een spellingchecker helpt daarbij, voor documenten. Ook voor programmatuur kan dit. Zorg eerst dat het algoritme correct verwoord is en bekommer je daarna pas om de spreekwoordelijke puntjes op de komma. Laat ***SPlint*** maar zoeken, vooral naar jouw fouten. Dan kun jij je concentreren op nog betere, nog efficiëntere code en op nog minder fouten uitkomen ook!

### Betere code

'Het paart, met een t', is correct als 'Het', het paard zelf is. Deze cryptisch zin is na nalezen wel duidelijk. Ook programmacode is wel eens cryptisch. Soms met opzet, soms omdat het niet anders kan en soms zonder goede reden. ***SPlint*** kan helpen om het laatste te voorkomen. Beter leesbare code, is dan betere code. Soms kan dat niet; zo is leesbare code niet altijd efficiënter. Net als de spellingchecker niet alles herkent, is ook ***SPlint*** geen tovenaar. Maar middels annotations kun je ***SPlint*** wel leren dat hij bepaalde constructies moet negeren. Probeer het maar eens, wedden dan je wel een scherm vol fouten vindt?