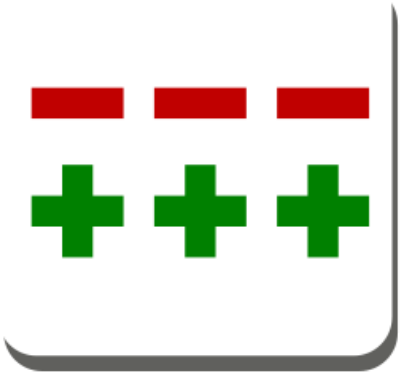
	**Git 1.7.8.3**	
	Versiecontrole Systeem	
	Systeemvereisten: Windows, POSIX, Linux or OS X	
	Ontwikkeld door:	Linus Torvalds, Junio Hamano en vele anderen http://git-scm.com/
Licentie: Gpl v2	**Contactpersoon:**	Koen.Vanderkimpen@smals.be

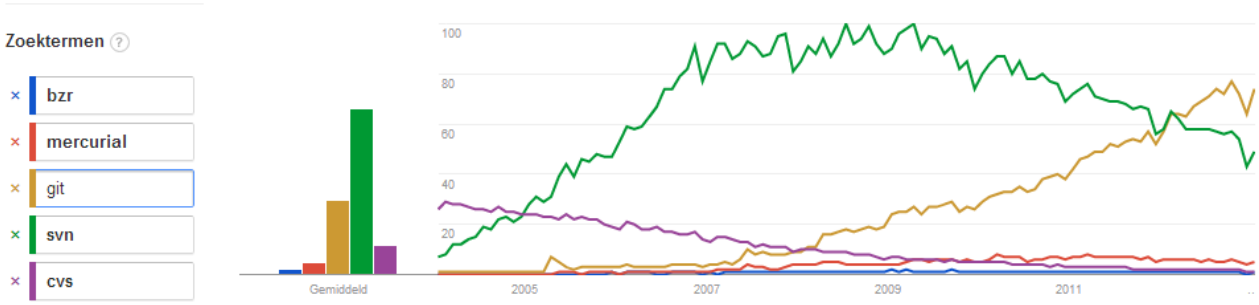
1. Beschrijving product

Git is een versiecontrolesysteem: een systeem dat kan worden gebruikt voor het beheer van verschillende versies van broncode, vooral in de context waar deze door verschillende personen tegelijk moet kunnen worden bewerkt. Versiecontrolesystemen zijn geëvolueerd in de context van software ontwikkeling, maar kunnen in principe gebruikt worden voor het beheer van eender welke documenten, met een belangrijke voorkeur voor tekstuele documenten, waarvan men op een efficiënte manier verschillende versies kan bijhouden d.m.v. het bewaren van een delta.

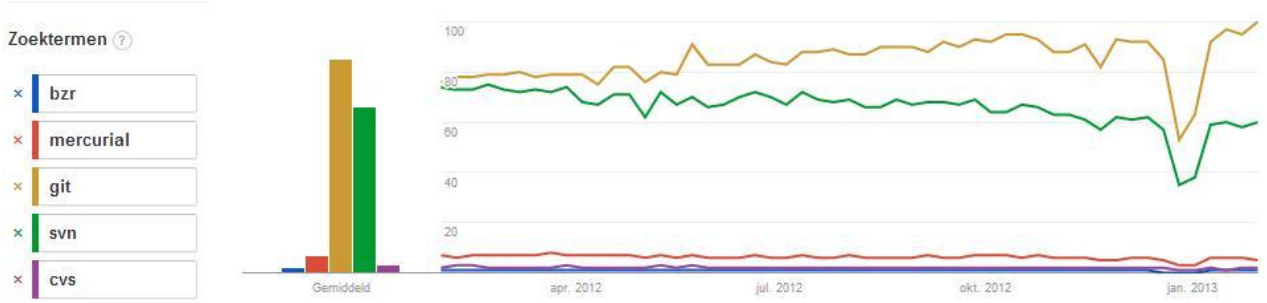
Git is een *gedistribueerd versiecontrolesysteem* (DVCS). Dit wil zeggen dat men, in tegenstelling tot bij een gecentraliseerd VCS (CVCS), geen gebruik hoeft te maken van één centrale versie op een server. Bij een DVCS staat elke kopie – of kloon, in het jargon – van het beheerde project op zich, en kan elke kloon zich ook baseren op meerdere andere klonen. Bovendien kan men eventuele wijzigingen aanbrengen aan een versie (zogenaamd 'committen') zonder te communiceren met de centrale server, en worden deze wijzigingen niet automatisch doorgevoerd bij elke kloon. Dit laat toe dat bij een DVCS verschillende programmeurs of teams op verschillende versies van het project werken (zo kunnen b.v. de zogenaamde feature branches, of de verschillende versie-branches zoals master branch, stable branch en debugging branch, via verschillende klonen worden beheerd). Bovendien kan men ook offline werken, zonder connectiviteit met de server, en dit zonder verlies aan commit-mogelijkheid, waardoor men met kleine

commits kan werken. DVCS systemen laten kortom toe om op een meer diverse en flexibele manier het werk te organiseren dan bij een CVCS.

Git werd in 2010 reeds kort door de sectie Onderzoek behandeld via een intern rapport [1] dat de bedoeling had een vervanger te vinden voor CVS; er werd toen uiteindelijk voor Subversion (SVN [3]) gekozen. Beide genoemde systemen betreffen CVCS. Op dat moment was er weinig interesse voor Git, en er werd dan ook minder aandacht aan besteed. Git was toen teveel op Linux gericht terwijl het systeem goed zou moeten kunnen functioneren onder Windows; bovendien waren DVCS minder ingeburgerd. Op twee jaar tijd zijn, dankzij de evolutie van DVCS, en Git in het bijzonder, die beide beperkingen echter grotendeels van de baan.

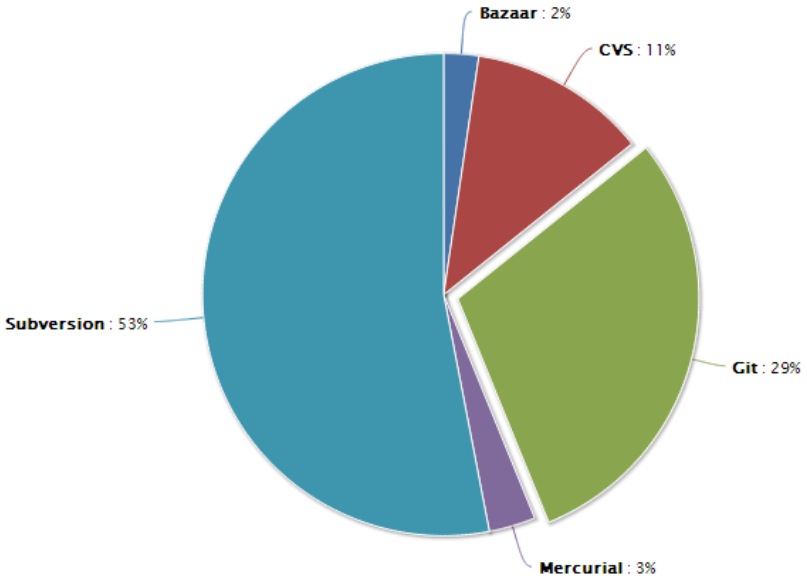


Figuur 1: Google Trends: 5 belangrijkste VCS sinds 2004



Figuur 2: Google Trends: jan 2012 tot jan 2013

In de software-ontwikkelingswereld in het algemeen merken we dat de voorkeur meer en meer naar gedistribueerde systemen zoals Git evolueert. In figuur 1 zien we dat, omstreeks 2010, SVN, als vervanger van het traditioneel meest-gebruikte CVS, nog steeds het populairste systeem was. Nu liggen de kaarten echter anders, en in figuur 2 zien we dat het oude CVS zelfs onderaan de lijst bengelt. Dit wordt echter tegengesproken door figuur 3, waar we merken dat CVS nog wat blijft tegenspartelen door het gebruik in bestaande projecten, ook al is het als zoekterm onderaan het peloton geëindigd. In figuur 1 en 2 is Git duidelijk het populairste, en zelfs in figuur 3 staat het systeem met kop en schouders boven de andere gedistribueerde systemen (Mercurial [4] en Bazaar (bzr)). Het loont dan ook de moeite om Git aan een extensievere beschrijving en testen te onderwerpen.



Figuur 3: Populariteit (jan 2013) van VCS op ohloh

Git werd oorspronkelijk ontwikkeld door Linus Torvalds voor gebruik bij de ontwikkeling van de Linux kernel, en later overgedragen aan Junio Hamano. Het systeem werd geschreven in C en (bourne) shell. De oorspronkelijke opzet was om een VCS te ontwikkelen dat gedistribueerd is, goed beveiligd tegen datacorruptie, zeer performant, en vooral niet gelijkend op CVS, dat volgens eerstgenoemde ontwikkelaar alles was wat een VCS niet mocht zijn. Om hem te citeren: “If you like using CVS, you should be in some kind of mental institution or somewhere else”, en over de opvolger van CVS, die de foute ontwerpbeslissingen moest goedmaken: “Subversion used to say ‘CVS done right’: with that slogan there is nowhere you can go. There is no way to do CVS right”.

In de beginjaren van Git was het een programma vooral bedoeld voor technische gebruikers; slechts later zijn er zaken aan toegevoegd om het gebruiksgemak te verhogen. Gezien de sterke band met Linux, werd Git ook ontwikkeld volgens het Linux-principe “Do one thing and do it well”, wat er voor heeft gezorgd dat het programma sterk gebruik maakt van andere Linux-toepassingen, zoals *diff* en *patch*. Sterker nog: Git heeft een toolkit-achtig ontwerp; het bestaat eigenlijk uit een grote collectie kleine programmaatjes. Deze zaken hebben ervoor gezorgd dat porteren naar andere besturingssystemen lange tijd moeilijk is gebleven. Zoals we in de testen echter zullen zien, heeft men daar tegenwoordig vrij goede oplossingen voor; men kan momenteel bijvoorbeeld zowat alle functionaliteit aanroepen via het programma *git* zelf.

2. Uitgevoerde testen

Bij het testen en vergelijken van functionaliteiten is het onvermijdelijk om terug te vallen op reeds eerdere ervaringen met andere versiecontrolesystemen. Toch is het belangrijk om ruimdenkend te zijn: bij het gebruik van DVCS komt een andere filosofie kijken dan bij de gecentraliseerde systemen die velen van ons al langer kennen. Bij deze bespreking beperken we ons dan ook tot een bespreking van de kwaliteiten van Git in functie van de gedistribueerde versiecontrole methodologie, die zeker voordelen biedt.

Bijgevolg zullen we in dit document, als we al vergelijken met andere systemen, eerder de nadruk leggen op het verschil tussen Git en de tweede populairste DVCS: Mercurial (hg), en niet, b.v. op het verschil met SVN. Bij de testen focussen we ons vooral op de volgende zaken:

- Functionaliteit, met de nadruk op gebruiksvriendelijkheid en tool-integratie in Windows, en ook de mogelijkheden tot conversie van en naar andere VCS-systemen en de ondersteuning voor subrepositories, die het faciliteren om slechts aan een beperkt onderdeel van een project te werken. Uiteraard vergeten we de standaard functionaliteit van een VCS niet.

- Kwaliteit van het product, gebaseerd op ons eigen Smals Open Source Maturity Model [5].

2.1. Standaardfunctionaliteit en Tools

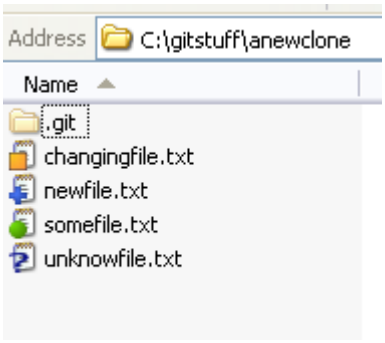
De functionaliteiten van Git werden getest op een Windows-machine. Dit is niet de meest ideale omgeving voor dit systeem, gezien het in de eerste plaats ontwikkeld is in en voor Linux, en zich dus hoofdzakelijk op dit platform richt. Desalniettemin is dit een goede test, gezien collaboratietools, en zeker versiecontrolesystemen, goed moeten kunnen werken over verschillende platformen heen (en in mindere mate omdat Windows het platform bij uitstek is in onze organisatie).

Om vlot met het systeem te kunnen werken, werd TortoiseGit geïnstalleerd, een browser-extensie, die het core programma omvat, en die het gemakkelijk maakt om de gebruikelijke commando's aan het systeem te geven zonder terug te moeten vallen op de command line. De 'Tortoise' groep van softwareproducten omvat zo ook TortoiseCVS, -Svn en –Hg, waarvan de laatste twee ook reeds werden getest in het andere document. Om de basistest te vervolledigen, werd evenwel ook kort de command line interface van het product getest.

Daarnaast werd ook de integratie met Eclipse en Intellij getest; welke twee belangrijke spelers zijn in de IDE-markt. Dit moet uiteraard ook goed werken, gezien dit soort platformen voor het grootste deel van de ontwikkeling worden gebruikt voor het beheer van de broncode, en dus ook voor het versiebeheer.

Resultaten

Zowel TortoiseGit als de Windows-versie van de standaard-Git installeren vlotjes. In Intellij is Git-support ingebouwd, maar wordt het pad gevraagd van een bestaande "git" executable. In Eclipse moesten we de plugin EGit installeren, die achter de schermen gebruik maakt van JGit, de Java-implementatie van Git.



Figuur 4: Een Repository in Explorer met TortoiseGit

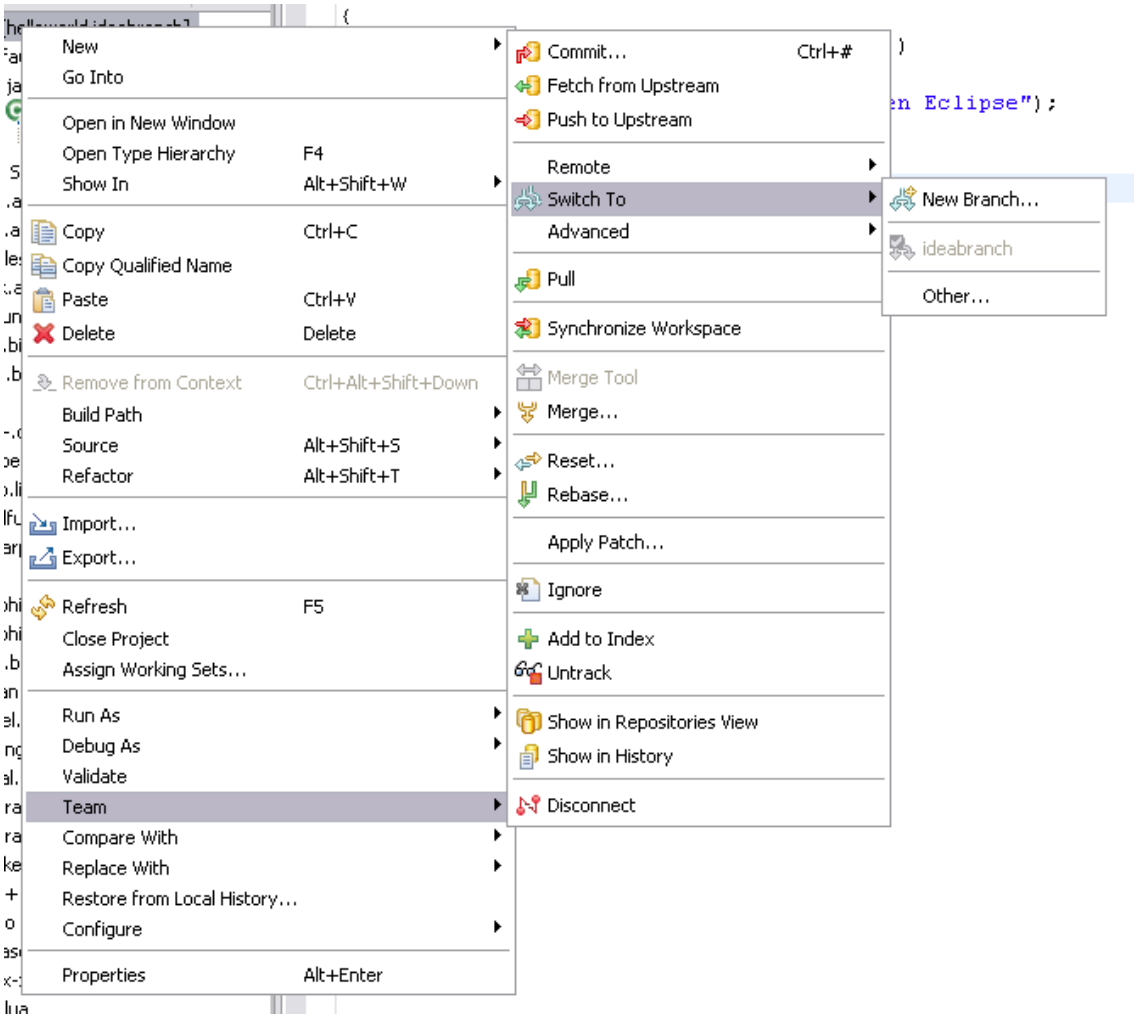
TortoiseGit werkt, zoals men zou verwachten, quasi identiek aan andere Tortoise producten, en is dus even gebruiksvriendelijk. In Figuur 4 zien we een voorbeeld van een repository die beheerd wordt in windows explorer via TortoiseGit. Ook geavanceerde functionaliteit, zoals branchen, rebasen, mergen, enz. is ondersteund. Het enige schoonheidsfoutje dat we kunnen bemerken is dat, af en toe, bestanden een icoontje krijgen dat hun toestand niet correct weergeeft. Zo kregen de bestanden die 'clean' (volledig in orde) zouden moeten zijn, het icoontje voor 'added' (nieuw toegevoegd).

Bij het importeren van Git-beheerde folders in Eclipse en Intellij hadden we ook geen al te grote problemen. Intellij identificeerde onmiddellijk de folders en bood de nodige functionaliteiten aan, bij Eclipse moesten we eerst een kleine instelling aan het project veranderen. Dit laatste kan gezien worden als een klein ongemak, maar is in principe gewoon te zien als het volgen van andere conventies. In beide IDEs waren de meeste Git-functionaliteiten goed bruikbaar. Enkel in Intellij was het iets minder voor de hand liggend (maar wel mogelijk) om een nieuwe branch te creëren. In Figuur 5 zien we het commando-menu met Git-functionaliteiten in Eclipse. Dit laat meteen ook zien welke mogelijkheden Git allemaal biedt.

Wat de command-line betreft, scoort Git opnieuw ongeveer even goed als andere VCS systemen, zoals Mercurial. Voor echte power users is dit nog altijd de snelste en meest uitgebreide manier om het systeem te gebruiken, inclusief alle mogelijke geavanceerde en exotische functies. Bij Git zitten, van bij de installatie, al onmiddellijk een pak meer van deze extra's bij, dit in tegenstelling tot bij sommige andere VCS, waarbij meer via plugins moet worden gewerkt. Dit is enerzijds handig, anderzijds verhoogt het een

beetje de leercurve, maar gezien, alvast onder Windows, de opdrachtprompt normaal gezien voor geavanceerd gebruik is, en secundair aan het werken via Tortoise of de IDE, is dit minder erg. Eén mankement is wel dat bij een commit via de command line, de standaard tekstverwerker die verschijnt voor de commit-boodschap Vim is. Dit komt natuurlijk doordat Git bovenal een Linux-programma is, bovendien is het waarschijnlijk te herconfigureren. In Windows is dit echter toch een klein ongemak, zeker voor iemand die het gebruik van Vim niet machtig is. Zo'n persoon zal op z'n minst schrikken, en zou zelfs kunnen falen in het correct opslaan van de commit-boodschap. Beter was het geweest als de default voor installatie op Windows Note- of Writepad zou aanroepen, zoals dat bij andere VCS systemen het geval is.

Nog een laatste opmerking die we kunnen maken betreffende de standaardfunctionaliteit, is internet-functionaliteit. De firewall van ons bedrijf blokkeert het Git-protocol[1] naar buiten toe, waardoor het merendeel van de testen werd uitgevoerd met lokale repositories (al dan niet met lokale server). Het correct configureren van een proxy lukte na verscheidene pogingen niet, dus deze piste werd begraven. Wel zijn een aantal testen uitgevoerd met een lokale Git-server, en ook vanuit onze van het bedrijfsnetwerk afgezonderde testomgeving (labo onderzoek) ondervonden we geen probleem om met publieke Git-repositories te connecteren.



Figuur 5: Git-commando's in een Eclipse project

[1] Git ondersteunt, naast het eigen en het lokale 'file' protocol ook http(s) en ssh, maar de meeste online repositories waarnaar men kan schrijven gebruiken het git protocol, vandaar het belang om hierover te kunnen testen.

2.2. Submodules

Net als in Mercurial, kan men in Git gebruik maken van subrepositories, in Git heten deze echter submodules. Het gebruik hiervan kan interessant zijn wanneer men allerlei bibliotheken en subprojecten dient te beheren voor gebruik in een bepaald hoofdproject. Een andere mogelijkheid is ontwikkelaars de mogelijkheid te geven een project niet in zijn geheel te klonen wanneer ze slechts aan bepaalde stukken ervan zullen werken. In dat geval heeft men submodules nodig omdat een enkelvoudige Git-repository steeds in z'n geheel moet worden behandeld (en gekopieerd).

Een tip bij het gebruik van dergelijke projectstructuur is om niet het hoofdproject, maar de bouw-omgeving, in hoedanigheid van 'thin shell', als topniveau van de repositorystructuur te gebruiken. Dit wordt verduidelijkt in Figuur 6: In de bovenste versie zit de code van het hoofdproject in de root folder en die van het subproject in 'somelib'. Dit heeft echter een aantal nadelen: door de strikte ouder-kind relatie tussen het hoofdproject en het subproject, kan men de code voor het hoofdproject niet pushen/pullen zonder dat het subproject beschikbaar blijft. Bovendien kunnen recursieve commits hier voor onverwachte problemen zorgen. De onderste versie is een stuk handiger: aan zowel het hoofdproject als het subproject kan apart gewerkt worden; enkel bij de integratie (de 'build') heeft men alles tegelijk nodig.

```
project/      # your main project repository
  somelib/    # your shared library as a nested subrepository
```

```
build/      # thin master repo to manage build environment
  project/  # your main project as a subrepo
  somelib/  # your shared library as a sibling subrepo
```

Figuur 6: Mogelijkheden voor gebruik van geneste repositories

Met gebruik van enkele eenvoudige commando's kan men in Git een submodule inbrengen in een bestaande module. Van dan af aan kan men met deze submodule werken als met een gewone module, maar men moet er steeds op letten dat men eerst wijzigingen in de submodule commit, en dan pas in de bovenliggende module, anders kan de module niet correct worden beheerd door het systeem.

2.3. Compatibiliteit met andere VCS

Om de compatibiliteit met andere VCS systemen te testen, richten we ons op de import/export van en naar een Mercurial repository, en op het converteren van een oude CVS repository. Op die manier is er zowel een conversie van een ander gedistribueerd systeem getest, als van een gecentraliseerd systeem. De (tegenwoordig typische) overstap van een SVN-repository naar Git hebben we niet getest: dit moet in principe gemakkelijker zijn dan een conversie vanuit CVS, dus als deze laatste slaagt, kunnen we er vrij zeker van zijn dat dat ook het geval zal zijn voor de eerste. Bovendien wordt de 'svn2git' migratie reeds uitvoerig besproken op het web.

2.3.1 Mercurial

Op het internet vinden we een aantal verschillende werkwijzen om een bestaande Hg repository om te zetten naar een Git-repository. De belangrijkste twee hiervan zijn hg-fast-export [6], uit het git-milieu, en hg-git, een plugin voor Mercurial [7].

De eenvoudigste manier van werken is de tool hg-fast-export. Met een vijftal commando's konden we eenvoudige repositories al snel migreren.

De Mercurial plugin, daarentegen, is iets complexer, maar werkt wel in twee richtingen. Eens de nodige configuratie gedaan is, kan men via deze plugin een Hg-repository pushen naar een lege Git-repository, en daarna in Git verder werken. Nadien kan men de wijzigingen terug in de Mercurial repository halen via een Hg pull operatie. Het is dus mogelijk de beide systemen synchroon te houden, maar het lijkt enkel te werken vanuit de Mercurial kant. De documentatie van deze tool had daarenboven iets beter gekund: het was niet meteen duidelijk, maar om de conversie goed te kunnen doen, dient men in Hg bookmarks te voorzien, die dan in Git overeenkomen met branch-namen.

Een complexere repository, die gebruik maakt van subrepositories, konden we spijtig genoeg met geen van beide tools converteren: de push-operatie van Mercurial, die normaal gezien eerst alle subrepositories pusht, stopt met werken na het pushen van één subrepository. Bovendien zagen we geen synchronisatie tussen de overeenkomstige hulpbestanden voor subrepositories in de beide hoofdrepositories.

We moeten dus besluiten dat de compatibiliteit beperkt is en de daartoe bestemde tools nog wat evolutie nodig hebben. Een eenmalige conversie lukt echter goed genoeg, zolang men geen complexe repositories gebruikt.

2.3.2 CVS

Voor de conversie van CVS naar Git bestaan er twee mogelijkheden: de eerste is een conversie naar SVN, gebruik makende van de tool cvs2svn, en vervolgens een wel beschreven conversie van SVN naar Git uitvoeren. De tweede, die we hier zullen uittesten, betreft een rechtstreekse conversie, gebruik makende van het commando cvs2git, een recente additie aan de tool cvs2svn.

Als testmateriaal gebruiken we een oude back-up van een CVS-repository van het project “rbo”. Om zeker te zijn dat we de tool goed gebruiken, testen we eerst een conversie naar SVN, zoals we reeds deden in [1]. Spijtig genoeg werkt cvs2svn nog steeds niet rechtstreeks onder Windows: we moesten de testen uitvoeren in Cygwin (een Linux-emulatieomgeving die draait onder Windows). Dit ging echter vrij vlot.

```
cvs2svn Statistics:
------------------
Total CVS Files:              403
Total CVS Revisions:         1577
Total CVS Branches:             0
Total CVS Tags:              4029
Total Unique Tags:             18
Total Unique Branches:          0
CVS Repos Size in KB:        3044
Total SVN Commits:            476
First Revision Date:    Thu May 18 13:17:18 2006
Last Revision Date:     Mon Apr 12 11:28:04 2010
------------------
Timings (seconds):
------------------
203.1   pass1    CollectRevsPass
  0.1   pass2    CleanMetadataPass
  0.0   pass3    CollateSymbolsPass
  0.5   pass4    FilterSymbolsPass
  0.1   pass5    SortRevisionSummaryPass
  0.1   pass6    SortSymbolSummaryPass
  0.5   pass7    InitializeChangesetsPass
  0.3   pass8    BreakRevisionChangesetCyclesPass
  0.3   pass9    RevisionTopologicalSortPass
  0.2   pass10   BreakSymbolChangesetCyclesPass
  0.4   pass11   BreakAllChangesetCyclesPass
  0.4   pass12   TopologicalSortPass
  0.7   pass13   CreateRevsPass
  0.1   pass14   SortSymbolsPass
  0.0   pass15   IndexSymbolsPass
  0.8   pass16   OutputPass
207.5   total
```

Figuur 7: Resultaat van Conversie uit CVS, stap 1

Vervolgens de echte test, met het commando cvs2git. In een eerste fase zal dit commando twee bestanden maken: de zogenaamde ‘blob-’ en ‘dump-files’. In een volgende fase gebruiken we het git-commando fast-import om deze twee bestanden te importeren in een lege Git-repository.

Bij de eerste poging om dit te doen was het initiële werk, de conversie naar de twee bestanden, een pak trager dan een omzetting naar SVN. Het leek echter foutloos te verlopen. De tweede stap, het importeren van de twee databestanden, ging dan weer zeer snel. Het resultaat was een ‘bare’ repository (een repository zonder werk-folder), die we goed konden klonen. Bij het nakijken van deze kloon bleek

dan echter dat alle bestanden 0KB groot waren. We hebben dan een tweede poging ondernomen, waarbij een extra opties-bestand werd gebruikt, waar nog allerlei richtlijnen in staan betreffende de conversie. Deze tweede poging verliep niet alleen sneller, maar gaf ook een bevredigend resultaat. In Figuur 7 is het resultaat van stap 1 te zien: er zijn, bij deze stap, een goeie 200 seconden nodig voor de conversie van een CVS-repository van 4MB. De tweede stap gaat doorgaans enkele factoren sneller. Merk op dat de cvs2git tool achter de schermen nog steeds gebruik maakt van cvs2svn (vandaar deze naam in de figuur).

2.4. Interne werking

Iets waar we tot nu toe geen aandacht aan besteedden, maar dat toch enige aandacht verdient, is hoe intern door het VCS-systeem de bestanden en hun wijzigingen worden opgeslagen.

In principe maakt dit uiteraard niet heel veel uit: zolang de VCS zijn werk doet, kunnen we hem als een 'black box' beschouwen. Vorig jaar kwamen we echter, bij gebruik van Mercurial voor de ontwikkeling van een Proof of Concept, in de problemen, en dit door de manier waarop Mercurial intern bestanden opslaat, hoe dit interageert met het bestandssysteem in Windows (met een beperking van absolute pad namen tot 260 tekens), gecombineerd met een diep geneste projectstructuur (veroorzaakt door intensief gebruik van Maven subprojecten). Drie factoren gecombineerd dus, die leidden tot een bug, waardoor de werking van Mercurial grondig werd verstoord en het systeem de facto onbruikbaar werd.

Dit was uiteraard een samenloop van omstandigheden die niet heel vaak voor zal komen, maar het zorgt ervoor dat we toch moeten letten op de interne werking, wanneer we een VCS uittesten. Er zijn verschillende work-arounds voor dit probleem met Mercurial in Windows, maar daar gaan we in dit document niet dieper op in.

Wat Git betreft, kunnen we voorspellen dat we dit probleem niet zo snel zullen hebben, omdat het systeem een vrij vlakke opslagstructuur gebruikt voor de opslag van zijn bestanden, en dus niet de geneste structuur van de werkfolder overneemt, laat staan dieper maakt. Een bijkomend voordeel van de manier van opslag van Git is efficiëntie qua gebruik van ruimte.

De paden gebruikt voor opslag zijn echter wel van enige (standaard-)diepte, en kunnen dus voor sommige projecten, die zelf een ondiepe folderstructuur hebben, langer uitvallen dan die in de werkfolder. Indien we dan gaan werken met geneste repositories etc., dan kunnen we eventueel wel terug in de problemen komen. We schatten de kans echter een pak kleiner in dan bij Mercurial. Bovendien moeten we hier wel melden dat de eigenlijke bron van deze problematiek toch vooral de schuld is van het besturingssysteem Windows. In deze tijd zou het toch echt mogelijk moeten zijn om paden toe te laten die langer zijn dan 260 tekens, en dit zonder allerlei work-arounds.

Wat we, ten slotte, nog kunnen vermelden betreffende de interne werking, is dat Git bekend staat als het meest performante (qua snelheid) VCS systeem op de markt; voor erg grote projecten een niet te onderschatten pluspunt. Dit hebben we echter niet zelf kunnen uittesten.

3. Evaluatie maturiteit

Aan de hand van het maturiteitsmodel voor open source software [5] van de sectie Onderzoek van Smals werden enkele niet-functionele criteria nagegaan met betrekking tot de software.

Figuur 8 bevat een overzicht van de evaluatie. De gedetailleerde criteria, hun waarden en scores zijn terug te vinden in de bijlagen (paragraaf 7.1). Git bezit een degelijke algemene score (3,75/5). Als gewichtsverdeling werd een custom verdeling gekozen, die we reeds eerder hanteerden om VCS te testen. Git had in vergelijking met de andere systemen waarschijnlijk iets beter gescoord indien performance als factor zou meetellen, maar we vonden het belangrijker de vergelijking te maken op basis van dezelfde criteria.

Evaluated Technology			
Product Name:	**Git**	**Total Score (/5)**	**3,75**
Product Website:	**http://subversion.apache.org/**		
Version :	**1.7.8.3**		
Project size:	**13,6MB**		
Evaluation date:	**5/01/2012**		
Evaluated by:	**Koen Vanderkimpen**		
Product type :	**VCS**		
Usage Setting:	**Custom**	Put your custom weights in the table on the 'Weights' sheet	

Custom is Routine use with support and documentation weights switched, and Professionalism instead of Performance as a factor

Rank	Category	Weight	Unweighted score	Weighted score
1	Installation	23,00%	4,50	1,04
2	Quality	23,00%	2,25	0,52
3	Security	0,00%	0,00	0,00
4	Performance	0,00%	0,00	0,00
5	Scalability	0,00%	2,50	0,00
6	Architecture	0,00%	0,00	0,00
7	Support	5,00%	3,00	0,15
8	Documentation	15,00%	5,00	0,75
9	Adoption	14,00%	4,85	0,68
10	Community	5,00%	4,67	0,23
11	Professionalism	15,00%	2,60	0,39
12	License	0,00%	0,00	0,00

	Total Weight	**Average unweighted score**	**Total weighted score**
	100,00%	**3,84**	**3,75**

Figuur 8: Evaluatie van de maturiteit van Git 1.7.8.3

This spreadsheet contains a number of key elements that should considered as indicators for possible problems. Low scores could indicate serious issues.

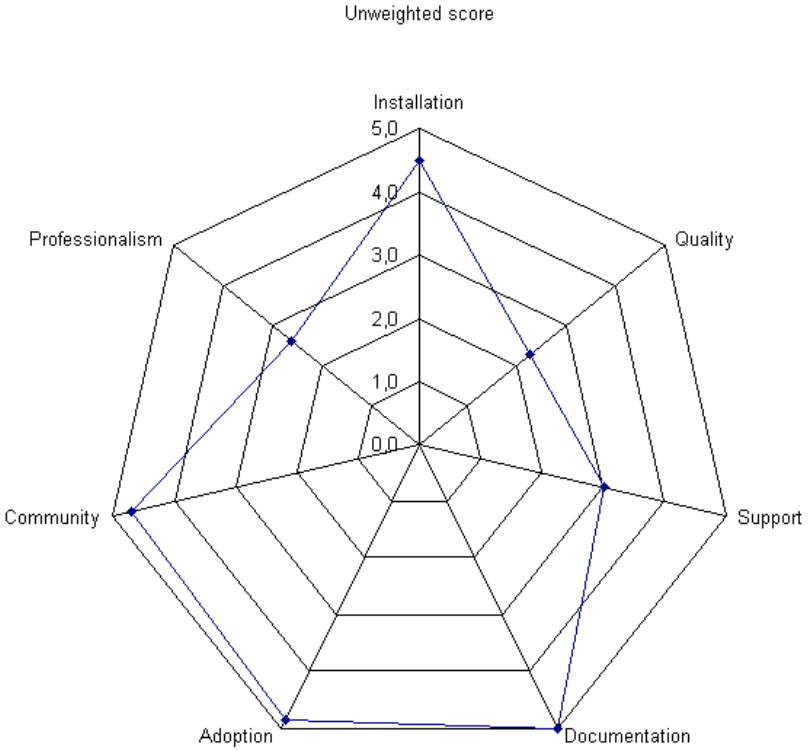
Category	**Unweighted rating (/5)**	**Score**
Number of bugs fixed in last 6 months (compared to # of bugs opened)	0	N/A
Standards	0	
Average volume of general mailing list in the last 6 months	4	[300 – 720) msg per month
Quality of professional support	3	Installation support only
Assessed paid support	1	0-3
Reference deployment	5	Yes, with publication of user's size
Age	5	> 3 years
Status	5	Mature, stable
Possible license issues		
Protection against propietary forks (GPL preferred)	1	Very permissive like BSD or Apache licenses.
Permissiveness preferred	5	Very permissive like BSD or Apache licenses.
Multiple licenses?	5	Only open source license
		No different flavors or flavors offer same functionality.
		Commercial license offers e.g. services, GPL
Limitations of community or free edition	5	protection, …

Figuur 9: Risico Indicatoren

Als we naar de signaalcriteria kijken in Figuur 9, dan valt op dat we het aantal gevonden en herstelde bugs niet konden achterhalen. Dit komt doordat de ontwikkelaars van Git geen expliciet bug tracking systeem hanteren. Er wordt gebruikt gemaakt van een aantal mailing lists voor alles wat met de ontwikkeling van Git te maken heeft. We vermoeden dat deze manier van werken vrij goed functioneert voor het team en beschouwen het dus, op zich, niet als een groot mankement, maar het is wel zo dat het moeilijk wordt om,

gebruik makende van zo'n manier van werken, statistische informatie te achterhalen. Gezien echter de populariteit van Git en de brede support die het krijgt in de open source community, vermoeden we dus dat de score voor kwaliteit in werkelijkheid iets hoger zal liggen dan de 2,25/5 die we bij deze evaluatie hebben kunnen geven door gebrek aan informatie. We kunnen echter wel stellen dat het ontbreken van technologische omkadering, zoals bug tracking systemen, de lage score voor professionaliteit extra kan bevestigen.

Andere rode en oranje vlaggen in Figuur 9 betreffen betaalde/ professionele support. Dit moeten we echter nuanceren: voor VCS systemen wordt zelden professionele support gezocht. Ons bedrijf doet dit bijvoorbeeld niet. De kwaliteit van de documentatie maakt trouwens van in-house ondersteuning de betere en goedkopere oplossing.



Figuur 10: Diagram met de diverse scores voor Git

De laatste rode indicator betreft de licentie, die heel permissief is, waardoor code op basis van Git moeilijker te beschermen wordt. Ook dit is een non-issue: slechts weinig ontwikkelprojecten zullen effectief broncode nodig hebben van een versiecontrolesysteem. Een VCS dient normaliter om te gebruiken bij de ontwikkeling, niet om van te vertrekken.

Tot slot geeft Figuur 10 nog eens de belangrijkste scores weer in grafische vorm. Zoals vermeld geeft het kwaliteitscriterium een vertekend beeld en is het eigenlijk beter dan weergegeven. Het pakket is heel goed gedocumenteerd. Er zijn tutorials en gedetailleerde handleidingen te vinden, zowel op het web als in boekvorm.

We kunnen dus besluiten dat Git, in de categorie van hulpmiddelen bij de ontwikkeling en zeker in de Linux-markt, gezien de kwaliteit van de documentatie en het brede draagvlak, voldoende matuur is, ondanks de semiprofessionele aanpak bij de bugtracking.

4. Besluit & Aanbevelingen

Git is een zeer degelijk gedistribueerd Versiecontrolesysteem, met alle toeters en bellen die dergelijke DVCS systemen doorgaans horen te hebben, 'and then some'. De basisfunctionaliteit werkt robuust en het programma kan projecten aan ter grootte van de Linux kernel en waarschijnlijk ook nog groter. Ook qua gebruiksvriendelijkheid in Windows moet Git niet veel meer onder doen van concurrerende VCS.

Qua geavanceerde functionaliteit werkt de ondersteuning voor subrepositories naar behoren. Git heeft echter een eigen visie hierop, die iets verschilt van de visie bij andere VCS. Het hangt er dus vanaf of dit overeenkomt met de manier waarop men ze wil gebruiken.

Compatibiliteit met andere versiecontrolesystemen werkt correct, maar we mogen geen wonderen verwachten. We raden aan de migratie vanuit andere systemen steeds in een Linux-omgeving uit te voeren en geen verschillende VCS door elkaar te gebruiken met Git als hoofdrepository. Zolang men monolithische projecten migreert, en nadien niet meer terug hoeft te schakelen op de oude repository, is er geen probleem.

Over de interne werking van Git kunnen we zeer tevreden zijn; het is een van de meest stabiele en performante VCS die we tot nog toe tegenkwamen.

Als algemeen besluit kunnen we het volgende meegeven: indien het opportuun is om een gedistribueerd VCS te gebruiken, is Git de geknipte kandidaat. Vooral in een Linux omgeving moet men zeker voor dit systeem opteren; het is dan ook specifiek voor (en door) Linux ontwikkeld. Gaan we naar Windows, dan kunnen we eventueel ook een alternatief als Mercurial kiezen. Er zijn namelijk nog altijd sporen zichtbaar van het feit dat Windows-compatibiliteit achteraf aan Git is toegevoegd. Deze zijn echter minimaal en zullen waarschijnlijk verder worden weggewerkt. Gezien de nog steeds stijgende populariteit van Git in de software development community, is het dus allicht beter om geen alternatief systeem meer te beginnen gebruiken, ook al is het geen must om onmiddellijk over te schakelen.

5. Referenties

[1] Version Control Systems Compared, Koen Vanderkimpen, mei 2010, Intern Rapport
[2] Inventaris Open Source: VersieControle, Koen Vanderkimpen, http://inventarisoss.smals.be/nl/351-DSY/304-DSY/250-RCH.html
[3] Inventaris Open Source: Mercurial, Koen Vanderkimpen, http://inventarisoss.smals.be/nl/251-RCH.html
[4] Inventaris Open Source: Subversion, Koen Vanderkimpen, http://inventarisoss.smals.be/nl/249-RCH.html
[5] Selectiemodel voor Open Source Software, Bob Lannoy, http://inventarisoss.smals.be/nl/160-RCH.html
[6] Converting from Mercurial to Git (using fast-export), Hivelogic, http://hivelogic.com/articles/converting-from-mercurial-to-git/
[7] Hg-Git Mercurial plugin, Augie Fackler, http://hg-git.github.com/

6. Disclaimer

Deze open source review werd reeds geschreven eind 2011, begin 2012. Een aantal zaken die pertinent zijn verbeterd (b.v. de gebruiksvriendelijkheid onder Windows) zijn geüpdatet in januari 2013. Het kan echter zijn dat sommige besproken zaken hierbij over het hoofd werden gezien en, door de snelle evolutie van de betreffende softwaresystemen, misschien niet meer perfect in overeenstemming zijn met de werkelijkheid.

7. Bijlagen

7.1. Open Source Selectiemodel

Zoals besproken in paragraaf 3 werd het maturiteitsmodel voor Open Source Software [5] ingevuld. De onderstaande tabel bevat alle ingevulde criteria.

Evaluated Technology

Git v.1.7.8.3

5/01/2012

	Category Title	Score	Weight	Unweighted Rating	Weighted Rating
	For more information open the outline (+ / -) left from row number	*Select appropriate range or value*			
1	**Installation**		23%	**4,50**	**1,04**
1.1	Time for setup pre-requisites for installing open source software	< 10 minutes	50%	5	2,50
1.2	Time for vanilla installation/configuration	10 - 30 minutes	50%	4	2,00
2	**Quality**		23%	**2,25**	**0,52**
2.1	Number of minor releases in past 12 months	1 or 3	38%	3	1,13
2.2	Number of point/patch releases in past 12 months	0 or > 6	38%	1	0,38
2.3	Number of opened bugs for the last 6 months	100 - 500	25%	3	0,75
2.4	Number of bugs fixed in last 6 months (compared to # of bugs opened)	N/A	0%	0	0,00
2.5	Number of P1/critical bugs opened	N/A	0%	0	0,00
2.6	Average bug age for P1 in last 6 months	N/A	0%	0	0,00
7	**Support**		5%	**3,00**	**0,15**
7.1	Average volume of general mailing list in the last 6 months	[300 – 720) msg per month	50%	4	2,00
7.2	Quality of professional support	Installation support only	25%	3	0,75
7.3	Assessed paid support	0-3	25%	1	0,25
8	**Documentation**		15%	**5,00**	**0,75**
8.1	Existence of various documents.	Install/deploy, user, admin, optimization (tuning), upgrading, devel documentations available in multiple formats (single html, multifile html, pdf)	60%	5	3,00
8.2	User contribution framework	People are allowed to contribute, and it is edited / filtered by experts	40%	5	2,00
9	**Adoption**		14%	**4,85**	**0,68**
9.1	How many books does amazon.com gives in the Books / Advanced Search query: “subject:computer and title:component name”	[6 – 15)	15%	4	0,60
9.2	Reference deployment	Yes, with publication of user’s size	50%	5	2,50

9.3	Total number of downloads	>5000	35%	5	1,75
10	**Community**		5%	**4,67**	**0,23**
10.1	Average volume of general mailing list in the last 6 months	[300 – 720) msg per month	33%	4	1,33
10.2	Number of unique code contributors in the last 6 month	N/A	0%	0	0,00
10.3	Age	> 3 years	27%	5	1,33
10.4	Status	Mature, stable	40%	5	2,00
11	**Professionalism**		15%	**2,60**	**0,39**
11.1	Project Driver	Individuals	30%	1	0,30
11.2	Difficulty to enter the core developer team	Only after being active outside committer for a while	20%	5	1,00
11.3	Leading team	>5	20%	5	1,00
11.4	Roadmap	No published roadmap.	30%	1	0,30

7.2. Testscenario

Als testopstelling werd gebruik gemaakt van:

- Standaardlaptop: Fujitsu Siemens Lifebook E series; 3 GB RAM; Windows XP SP3
- Eclipse IDE 3.7.1
- Intellij CE 10.0.1