Enterprise

De hedendaagse ontwikkelomgevingen stellen ons beter dan voorheen in staat om snel code te ontwikkelen.Daardoor zijn we ook eenvoudiger in staat om ingewikkelde softwaresystemen te bouwen die intensief met andere softwaresystemen samenwerken. Hoewel we dus in staat zijn de software eenvoudiger te ontwikkelen, zorgen we er tegelijkertijd voor dat de omgeving waarin die software draait steeds complexer wordt. Gedragsveranderingen van softwarecomponenten in een softwaresysteem kunnen daardoor onbedoeld een groot effect hebben op de werking van het gehele systeem.

Continue Benchmarks

De vinger aan de pols voor de non-functionals

Naarmate de omvang van een systeem groeit, wordt het ook steeds ingewikkelder en tijdsintensiever om de oorzaak van een probleem snel en goed in kaart te brengen. Het is daarom zaak om de kwaliteit van je softwaresysteem en softwarecomponenten continu te bewaken.
Om een project beheersbaar te houden, is het noodzakelijk dat de geschreven sourcecode regelmatig automatisch wordt gebouwd, gecontroleerd en getest. Pakketten als Visual Studio Team System of Cruise Control .Net bieden je ook allerlei handvatten om periodiek geautomatiseerde kwaliteitscontrole van de software op syntactisch, stilistisch en functioneel niveau uit te voeren. Dit kan door een automatisch buildproces in te richten, waarbij Statische Code Analyse (zoals de static code analisys (SCA) van Visual Studio, voorheen FxCop) en alle geautomatiseerde unit- en integratietesten worden uitgevoerd.
Daarnaast wordt het, zeker bij de complexere systemen, noodzakelijk om ook de niet-functionele kenmerken van de software in het oog te houden. Vaak wordt dat aspect wel in de handmatige testen meegenomen, maar niet in de geautomatiseerde cyclus.
Als je wilt voorkomen dat er na oplevering van de software dagenlange profilesessies nodig zijn om uit te zoeken waarom je code ondanks allerlei inspanningen toch weer trager is geworden dan bij de vorige opleveringen, is het aan te raden om een continue benchmark test zijn intrede in je ontwikkelproces te laten doen.

Benchmarks

Een benchmark, vertaalbaar als 'referentiekader' of 'ijkingskader', is een basis om de prestaties van verschillende systemen, apparaten of organisaties met elkaar te kunnen vergelijken. In de informatietechnologie is een benchmarkstuk een programmacode waarmee verschillende computersystemen, databaseservers, CPU's, harde schijven of andere componenten met elkaar kunnen worden vergeleken. Een dergelijke applicatie beoordeelt de machines op verschillende aspecten (diskspeed, memoryaccess, aantal processor instructies per seconde etcetera) en geeft vervolgens een consensus oordeel over de prestaties en zo kunnen we de verschillende producten ten opzichte van elkaar vergelijken.
Dit kun je met je software doen. Je bepaalt een verschillend aantal aspecten waarop je code wilt beoordelen (functiegroepen, snelheid van bepaalde interface code etcetera), hoe je dat wilt gaan meten (bijvoorbeeld in één tijdseenheid zoveel mogelijk interface calls uitvoeren) en je geeft aan hoe zwaar elk van die punten moet gaan meetellen. Met die gegevens ben je in staat om een consensusoordeel over de verschillende niet-functionele aspecten te geven. Op een punt in de tijd stel je een referentiepunt in en daartegen kun je andere prestaties vergelijken.
Als een benchmark het gangbare gebruik van je applicatie goed benadert, dan kan deze bijvoorbeeld worden gebruikt om te beoordelen op welke doelsystemen je applicatie het best zal gaan werken. Of je kunt ermee bekijken welk effect een bepaalde verandering in de software op de performance van het eindproduct zal hebben. Bij de continue benchmarktest wordt de benchmark gebruikt om verschillende opeenvolgende softwareversies met elkaar te vergelijken. Hierdoor kun je het performancegedrag van de software gedurende de ontwikkeling in te tijd volgen.
We gaan proberen om het doelsysteem zo stabiel

Gert-Jan Messchendorp is managing consultant bij Capgemini en is ICT Architect bij het Command-&Control Support Centre. Email: gert-jan.messchendorp@capgemini.com

mogelijk te houden, dus we gaan idealiter altijd dezelfde machine gebruiken en we zorgen ervoor dat er naast de benchmark zo min mogelijk andere software op draait. Ook de benchmark proberen we stabiel te houden, dus we trachten de criteria zo te maken dat de benchmark nog nauwelijks gewijzigd hoeft te worden. Hierdoor krijg je een meting in de tijd waarbij het enige dat wijzigt de software is. Als je dat doet, dan ben je in staat om de wijzigingen die je in de uitkomsten van je benchmark vindt te relateren aan softwarewijzigingen.

In het ontwikkelproces

Een continue benchmark heeft de meeste kracht als hij heel vaak wordt gedraaid, bij voorkeur bij elke softwarewijziging. Hierdoor kan deze een performanceplaatje opleveren waarin je allereerst trends kunt volgen en trendbreuken kunt gaan waarnemen. Door een korte tijdsspanne tussen de opeenvolgende benchmarktests te hanteren, kun je met deze trends heel gericht naar bepaalde bottlenecks op zoek gaan, omdat elke trendbreuk in de grafiek correspondeert met een beperkt aantal software wijzigingen.
Een continue benchmarkmeting zet je dus vooral in om het performancebewustzijn van de ontwikkelaars/teamleden te verhogen en om performance verslechteringen snel zichtbaar te krijgen en aan te kunnen pakken. Daarmee onderscheidt een continue benchmark zich in zijn doel van andere kwaliteitsmaatregelen die momenteel al vaak in een continu bouwproces zijn opgenomen.
Het buildresultaat en unittest zijn dwingend, deze mogen nooit falen, de mate waarin je SCA laat meewegen hangt van je eigen voorkeur af. Het benchmarkresultaat is informatief, als de benchmark drastisch inzakt, moet een verklaring aanwezig zijn of gezocht worden.

Performance consequenties

Design for performance. Het is erg belangrijk om je altijd bewust te zijn van de performance consequenties van je ontwerp en implementatiebeslissingen. Vaak worden pas performancemaatregelen getroffen op het moment dat blijkt dat een applicatie te traag werkt. Of er wordt wel op performance getest, maar dan helemaal aan het eind van het testtraject.

Wat ga je benchmarken?

Eigenlijk kun je alles wat je maar wilt benchmarken, het hangt er helemaal van af wat je belangrijk vindt en of het goed meetbaar is. Je draait een test en meet dan bijvoorbeeld het aantal keren dat je een bepaalde functie kunt uitvoeren, de hoeveelheid CPU tijd die het proces heeft geconsumeerd, disk access, netwerkverkeer of geheugengebruik.
Het is goed om van tevoren na te denken over de betekenis van je benchmarkresultaat. Ik vind het zelf prettig om te werken met een hoog getal voor 'goed' en een laag getal voor 'slecht'. Denk daarbij bijvoorbeeld aan 'aantal functieaanroepen per tijdseenheid'. Maar ik kan me ook goed voorstellen dat je juist hoog als slecht wilt bestempelen en laag als goed. Dan kom je op de hoeveelheid disk access, of de tijdsduur van een functie.
Het belangrijkste is dat de benchmark zo is opgesteld dat hij je precies aan kan geven hoe hard de performanceveranderingen door je applicatie gebruikers gevoeld zullen gaan worden.

Wanneer en waar invoeren?

De continue benchmark test is een uitbreiding op de set geautomatiseerde kwaliteitscontroles die je in je ontwikkelproces kunt opnemen. Voordat je toe bent aan het inbrengen van iets als een benchmarktest aan je ontwikkelproces moet je al een aantal andere zaken goed geregeld hebben.
Een in mijn ogen goede volgorde voor het invoeren van kwaliteitsverhogende middelen is de volgende:

1. Gebruik een versiebeheerpakket. Voor een goede kwaliteitscontrole is het erg belangrijk om tooling te hebben waarmee kan worden nagegaan welke wijzigingen er zijn geweest en waar, wanneer, door wie en waarom de wijzigingen zijn doorgevoerd.
2. Zorg buiten de geautomatiseerde omgeving om voor een goed ontwikkelproces.
3. Gebruik een geïntegreerde ontwikkelomgeving zoals Visual Studio, zodat vooraf al wordt voor-

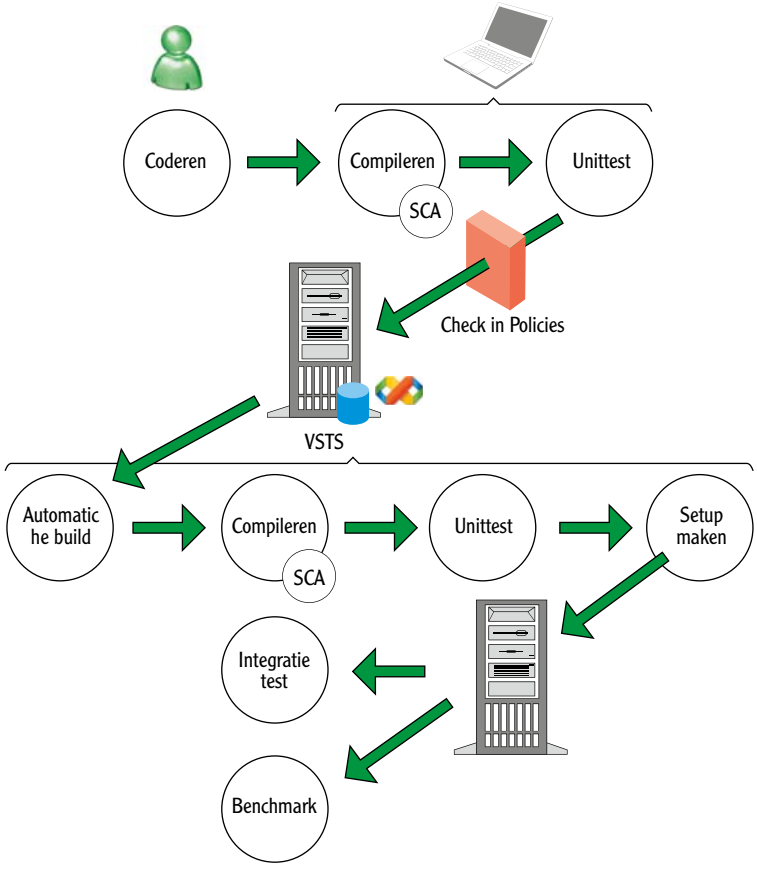
Een continue benchmark is het meest krachtig als hij heel vaak wordt gebruikt

Static Code Analysis	Benchmark Test
SCA doet een niet functionele code-inspectie.	Een benchmark test code op niet functionele aspecten.
SCA beoordeelt de code op uiterlijk en voert de code niet uit.	Bij een Benchmark wordt de code uitgevoerd.
SCA doet uitspraken over aandachtspunten ten aanzien van de performance	Een Benchmark meet de performance.
SCA kan als regressietoetsing in een continu bouwproces worden ingezet.	Een Benchmark kan als regressietoetsing in een continu bouwproces worden ingezet.

Tabel 1: Hoe verhoudt Statische Code Analyse (SCA) zich tot de Benchmark?

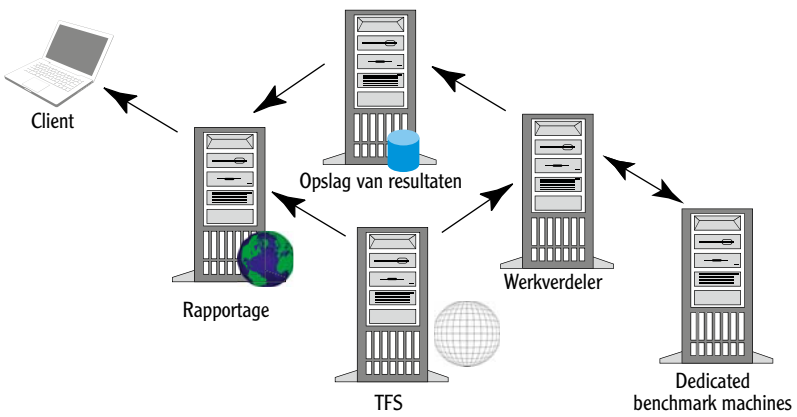
Unittest	Benchmark Test
Unittesten zijn zwart-wit, een unittest doet een absolute uitspraak over de code: het goed of het is fout	Een benchmark is relatief; het is sneller geworden of het is langzamer geworden. Dit hoeft niet goed of fout te zijn, soms neem je bewust een performance verlies om een andere functionele of niet-functionele wens gerealiseerd te krijgen.
Als er harde performance eisen zijn aan een systeem, dan kan je die beter 'hard' aftesten in een test op het doelsysteem. De continue benchmark is er dan om falende 'harde' test voor te zijn.	Bij een Benchmark wordt de code uitgevoerd.
In principe heb je voor elke property of mehode in code een unittest. Er zijn dus veel unittests, terwijl elke test zelf een heel klein stukje code zo volledig mogelijk aftest. Als er nieuwe code wordt ontwikkeld dan komen er per definitie nieuwe unittesten bij. Als je ontwikkelt volgens test-driven development is het zelfs de bedoeling dat je eerst de test maakt voordat je de code realiseert.	Van benchmark testen moet je er niet al te veel hebben, het is de bedoeling dat je per test veel code raakt, waarbij je inzoomt op een performance aspect gerelateerd aan het gangbare gebruik van de code. De kracht ontleen je hier aan de regelmaat waarmee je de test uitvoert; vaker draaien bijvoorbeeld na elke ingecheckte wijziging, betekent gerichter resultaat.
Met het regelmatig uitvoeren van unittest voer je regressie uit over de functionele werking van je applicatie.	Met het regelmatig uitvoeren van benchmark voer je regressie uit over het niet-functionele gedrag van je applicatie.

Tabel 2: Hoe verhoudt een unittest zich tot een benchmark?



Figuur 1: Kwaliteitsverhogende maatregelen in het ontwikkelproces.

komen dat bepaalde fouten worden te gemaakt.
4. Schakel in je projecten static code analysis aan, doe dit op tijd. Als je al veel code hebt, dan is het een lastige en onaantrekkelijke bezigheid om deze achteraf nog SCA-compliant te maken.
5. Maak dagelijks een build. Maak liever nog een build na elke ingecheckte changeset. Hiervoor kun je Visual Studio Team System (VSTS) inzetten. Als je geen VSTS hebt kan je daar bijvoorbeeld Cruisecontrol.Net voor gebruiken.
6. Draai na elke build automatisch een unittest. Zorg ervoor dat de build faalt als een unittest faalt.
7. Maak check-in policies voor je project.
8. Zorg er voor dat je een goede setup hebt om je applicatie mee te installeren en te deinstalleren. Neem de setup ook mee in je testproces.
9. Denk na over geautomatiseerde integratietesten waarin je veel voorkomende scenario's aftest. Draai deze test bij voorkeur op de setup.
10. Als bovenstaande punten, die inmiddels voor elke ontwikkelaar common sense zouden moeten zijn, als een zonnetje draaien, is het nog maar een kleine stap om de benchmark te introduceren. Een benchmarktest kan ook eenvoudig in een bestaande bouwomgeving die niet aan de voorgaande 9 punten voldoet worden ingelast.
11. Zorg ervoor dat het slagen van een build de start van een benchmarktest triggert. Bij mijn huidige project slaan we de automatische build resultaten op een netwerkschijf op en het benchmarkproces houdt deze schijf met een filewatcher in de gaten. Voer de benchmarktest bij voorkeur uit op een lege, onafhankelijke, geïsoleerde, machine en gebruik bij voorkeur de setup om de applicatie te installeren en na de test weer te de-installeren (zie ook het voorbeeld in figuur 2). Sla de testresultaten op in een database, zodat deze naderhand geanalyseerd kunnen worden.
12. Zorg ervoor dat de uitkomst van de benchmark eenvoudig opgevraagd kan worden. Bijvoorbeeld via een website of een agent in een portal. Des te beter je dit integreert, des te sneller worden performance problemen gesignaleerd. Ook periodiek (eens per week) een overzicht mailen is een goede mogelijkheid om iedereen alert te houden.



Figuur 2: Creëer een onafhankelijke benchmark opstelling met zo min mogelijk kans op verstoringen in de test.

Beoordelen en analyseren

Als je regelmatig een benchmarktest draait, dan kun je de resultaten van de test uitzetten tegen de tijd. De grafiek die je dan krijgt toont één of meerdere lijnen die performancemeetwaarden weergeven afhankelijk van de tijd. Die tijd zou niet de tijd van de test moeten zijn, maar liever nog de tijd/identificatie van de laatste changeset die aan de geteste build ten grondslag lag. Over het algemeen zie je in die lijn een trend onstaan. Het is de bedoeling dat die trend aangeeft dat de performance stabiel blijft of geleidelijk sneller wordt.

In de trendlijn probeer je de neerwaardse/opwaartse flanken te vinden. Meestal heb je wat meer vervolgmetingen nodig om te concluderen dat de trend inderdaad op en bepaald punt is toe- of afgenomen, maar soms is het zo drastisch dat je het na één meting al kunt zien of vermoeden.

Deze flanken zijn belangrijk, want markeren een bepaalde gebeurtenis, namelijk het beter of slechter worden van de performance. Bij het plotseling slechter worden van een benchmark is het evident dat je naar een oorzaak op zoek gaat. Maar ook bij het plotseling beter worden van de benchmark moet je je achter de oren gaan krabben. Heeft iemand onbedoeld belangrijke functionaliteit weggegooid die niet door de test wordt geraakt? Of is er onop-

zettelijk een performanceprobleem opgelost en zouden we die kennis elders ook kunnen gebruiken? Bottom line: Als je een flank niet had verwacht, is het in alle gevallen belangrijk om de bijhorende codewijzigingen te inspecteren.

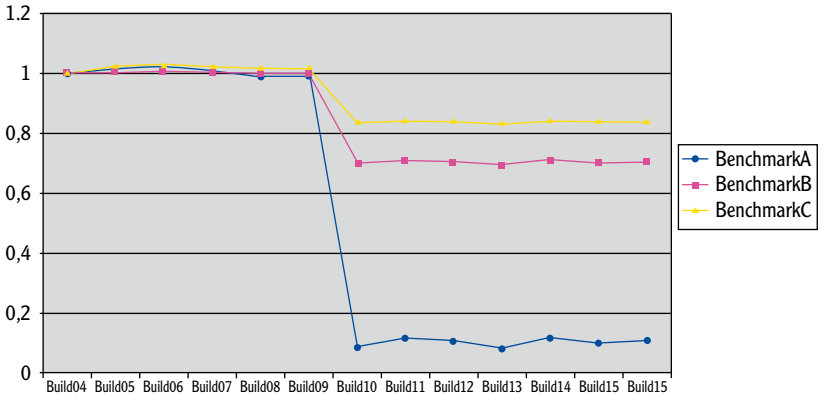
Profijt van goede benchmarks

Als je een continue benchmark gaat invoeren, dan is één van de belangrijkste zaken die je goed geregeld moet hebben de opslag van je meetresultaten. Het volledig automatiseren van je hele testproces en het presenteren van de resultaten kan altijd nog, maar met het verzamelen van de gegevens kun je nooit vroeg genoeg beginnen.
In de tabel op pagina 24 leg ik uit welke informatie je kunt opslaan en waarom je dat doet.

Presentatie van de gegevens

Het presenteren van je gegevens blijft altijd lastig. Wij hebben bij mijn huidige project al een hele tijd een continue benchmarktest lopen, maar we zijn nog steeds aan het stoeien met de presentatie.
Om het continue proces goed in beeld te brengen, ontkom je er niet aan om je gegevens in een grafiek aan te bieden. En tabel met de meetresultaten toont ook alle benodigde informatie, maar daarbij vallen de veranderingen niet zo snel visueel op.
Het blijft een beetje spelen en experimenteren met voorbeeldgrafiekjes. Wat zaken om over na te denken:

- Welke waardes laat je zien in je grafiek? Er zijn verschillende manieren om met de uitkomsten van een meting om te gaan. In mijn projectteam hebben we er bijvoorbeeld voor gekozen om het eerste punt dat in de grafiek zichtbaar is op 1 te stellen. De rest van de grafiek relateren we daaraan. Als de benchmarks een stabiel gedrag vertonen, dan staan alle lijnen van de grafiek over elkaar heen. Mocht voor een bepaalde benchmark de performance inzakken, dan steekt de betreffende lijn meteen onder de rest uit.
- Betekent een stijgende lijn 'goed' of betekent het juist 'fout'. Dus anders gezegd kies je ervoor om het aantal instructies per tijdseenheid te tonen, of kies je ervoor om de snelheid van elke instructie te tonen.
- Ik werk zelf het liefst met 'aantal instructies per seconde', dit is de manier waarop veel benchmarks werken.
- Schaalindeling, lineair versus logaritmisch. Bij een lineaire schaalindeling lukt het je meestal het beste om precies te interpoleren hoeveel procent er bij of af is gegaan. Maar de lineaire presentatie doet de gevoelsperformance tekort. Stel je voor dat iets 10 keer zo snel is geworden en je hebt het originele resultaat op 1 gesteld, in dat geval zorgt dat in je grafiek voor een stijging van 900%. Stel dat je een tweede lijntje tien keer zo langzaam maakt, dan zorgt dat maar voor een daling van 90%. Als je een logaritmische schaal had genomen, had de daling in de grafiek even sterk geweest als de stijging.
- Inzoommogelijkheden. Als je veel benchmarks hebt of veel verschillende producten in een suite hebt waar je de overall performance van bij wilt houden, dan wordt de grafiek niet erg leesbaar als je alle lijnen over elkaar in de grafiek toont. Je kunt er dan over nadenken om een hiërarchie van benchmarks aan te leggen waar je per set gekoppelde benchmarks drill-down/inzoommogelijkheden creëert.
- Directe koppeling versiebeheerpakket: Met behulp van het versiebeheerpakket zijn de bijhorende changesets te vinden. Hiermee kun je meteen pinpointen op de codewijzigingen die de performance verandering hebben veroorzaakt.
- Creëer mogelijkheden om al je benchmarks te tonen. Soms kan het zijn dat een benchmark niet heel stabiel is in het gedrag en veel ruis vertoont. Zo'n benchmark kan dan toch nog interessant zijn. Het gaat bij een continue benchmark om het beeld; als een instabiele benchmark eerst met een afwijking van 50% rond de 100 schommelt en hij schommelt na verloop van tijd met dezelfde afwijking rond de 150, dan is dat nog steeds veelzeggend al kun je dan niet erg nauwkeurig bepalen waar het precies is misgegaan.
- Mailkoppelingen. Het beste is natuurlijk een proces waarbij iedere ontwikkelaar dagelijks in de gaten houdt hoe de continue benchmark ervoor staat. Mocht de benchmnark toch te vaak aan de dagelijkse aandacht ontschieten, dan kun je nadenken over veranderingsniveaus in de benchmark die je als onevenredige bestempelt. Bij een te grote performanceverslechtering zou je meteen een mail kunnen sturen naar de verantwoordelijke ontwikkelaar, teamleider, projectleider of architect.



Figuur 3: Performance terugval is direct zichtbaar in een grafiek.

Configuratiewijzigingen

De continue benchmark valt en staat bij de stabiliteit van de omgeving. Dat wil zeggen; je moet zo goed mogelijk kunnen garanderen dat de enige

De continue benchmark staat of valt bij de stabiliteit van de omgeving

Project identificatie	Dit is nodig als je meerdere projecten wilt monitoren en eventueel ook seperaat of geagregeerd wilt tonen. Bijvoorbeeld de naam van het project.
Build informatie	Elk meetgegeven doet een uitspraak over een bepaalde build. Je wilt later zo nauwkeurig mogelijk kunnen uitzoeken waar en wanneer een bepaald probleem geintroduceerd is. Denk hierbij aan de naam/label/versienummer van de build en de datum dat de build gemaakt is. Je zou er ook aan kunnen denken om de nummers van de changesets in VSTS ten opzichte van de vorige build bij de build informatie op te slaan of een identificerend label uit je versiebeheer pakket. Denk hierbij ook aan informatie over de ontwikkeltak waarop de build is gedaan. Wij werken bijvoorbeeld op verschillende truncs en branches zodat je per team en per productversie controle op de wijzigingen hebt. Het is heel denkbaar dat je de benchmark resultaten per team of per in-onderhoud-zijnde productversie wilt controleren.
Testconfiguratie	Het is aan te raden om ook de configuratie waarop de benchmark is gedraaid identificeerbaar te maken. Een wijziging van je testomgeving heeft namelijk invloed op de resultaten. Om inzicht in die veranderingen te krijgen, of er correcties op door te voeren (zie ook verderop in het verhaal) geef je elke configuratie een unieke versie. Je kunt dat zo ingewikkeld maken als je zelf wilt. In principe is het voldoende om de eerste configuratie waarmee je begint 1 te noemen en dat getal bij elke wijziging op te hogen. Je zou voor extra inzicht bij elke versie een korte verklarende tekst op kunnen slaan.
Benchmark versie identificatie	Bij elke test moet je wel precies kunnen achterhalen welke benchmark getest is. Hoewel het, net als bij de testconfiguratie, aan te raden is om je benchmark zo min mogelijk te wijzigen gaat dat in de loop van de tijd natuurlijk wel een keer gebeuren. Daarom moet je ook je bij de gedraaide benchmarktest ook het versienummer van de benchmark opslaan. Bij elke wijziging van de test zelf gaat dat versienummer omhoog.
Testrun identificatie	Elke testrun is uniek. Het is dus verstandig ook elke testrun een unieke identificatie te geven. Daarnaast kan je als extra informatie bijvoorbeeld opslaan wanneer de test gedraaid is en of er bijzonderheden bij de test waren.
Het testresultaat	Ik ben er een voorstander van om de testresultaten als een ruw gegeven op te slaan. Voor een correcte presentatie van je gegevens zal je zeker allerlei correctiefactoren bij de meting moeten opslaan, maar het resultaat moet je zo ruw en volledig mogelijk houden. Als je dat niet doet dan ontneem je jezelf de mogelijkheid om je presentatie later nog aan te passen of om andere doorsnedes van je meetresultaten te maken.

Referenties

- Visual Studio Team System - http://msdn.microsoft.com/en-us/vsts2008
- Cruise control - http://cruisecontrol.sourceforge.net/
- Resharper - http://www.jetbrains.com/resharper/
- FxCop, SCA - http://code.msdn.microsoft.com/codeanalysis
- Patterns & Practices -Improving .Net Application Performance and Scalability, http://msdn.microsoft.com/en-us/library/ms998530.aspx
- Wikipedia - http://nl.wikipedia.org/wiki/Benchmark
- The Test Automation Manual - http://tam.mark-fink.de/Continuous-Performance/Continuous-Performance.html
- Patterns & Practices: Life Cycle Performance Testing - http://msdn.microsoft.com/en-us/library/bb905531.aspx
- Workshop on Performance and Reliability - http://www.performance-workshop.org

wijziging in de tijd de sourcecode is. Dat is natuurlijk nooit vol te houden.
Het is daarom verstandig om de configuratie ook als een gegeven bij je benchmarks op te nemen. Je krijgt dan de mogelijkheid om verschillende configuraties ten opzichte van elkaar te schalen.
Als de configuratie wijzigt, dan kun je de benchmarktest die het laatst is gedraaid op de oude configuratie nog een keer herhalen op een nieuwe configuratie. In dat geval houd je dus de sourcecode stabiel, terwijl je alleen de configuratie wijzigt. De verhouding tussen de benchmarkuitkomsten van de beide configuraties kun je vervolgens gebruiken als weegfactor waarmee de uitkomsten van oud en nieuwe tests aan elkaar worden gerelateerd. Als formule ziet dat er zo uit:

$$Weegfactor_{Nieuw} = \frac{Weegfactor_{Oud} \cdot \mathrm{Re}\,sultaat_{Oud}}{\mathrm{Re}\,sultaat_{Nieuw}}$$

Deze correctie zou je ook kunnen doen als je benchmark wijzigt.
Ook vanwege dit soort correcties is het verstandig om zo veel mogelijk de ruwe meetresultaten op te slaan met alle kenmerken van de meting en de de definitieve waarde pas in de presentatie te berekenen.
Deze correcties hoeven ook niet persé geautomatiseerd bepaald te worden. Vaak weet je zelf wel wanneer de omgeving of benchmarktest is veranderd. Als je ervoor zorgt dat de testen ook nog handmatig uit te voeren zijn, dan kun je de weegfactor met de hand bepalen.

Benchmarks en Visual Studio Team System

De integratie van geautomatiseerde benchmarks met Visual Studio Team System kan op veel verschillende manieren. De belangrijkste koppelpunten heb ik in onderstaande tabel weergegeven.

Portal	Maak de benchmark rapportage zichtbaar op de teamsite van je team op de TFS Sharepoint Portal. Bij mijn huidige project maken we de testresultaten en benchmark resultaten zicht baar op een Internet Information Server. We hebben bij de ingang van het gebouw een groot scherm hangen waarop de resultaten als beeldkrant worden getoond.
Buildproces	Als je op VSTS een geautomatiseerd bouw proces hebt, dan kan je de benchmark testen door het bouwproces laten triggeren. Wij laten momenteel het buildresultaat naar een netwerkschijf schrijven. Daar hebben we een filewatcher op staan. Zodra er een nieuwe build beschikbaar is wordt een benchmarktest ingepland.
Changesets	Aan de hand van de build informatie kan je precies achterhalen welke changesets in de build zijn meegenomen. Via VSTS Web Access kan je bij elk meetpunt uit je performance grafiek direct doorlinken naar de bijhorende veranderingen.
SCA doet uitspraken over aandachtspunten ten aanzien van de performance	Een Benchmark meet de performance.
SCA kan als regressietoetsing in een continu bouwproces worden ingezet.	Een Benchmark kan als regressietoetsing in een continu bouwproces worden ingezet.

Conclusie

Er zijn veel mogelijkheden om een periodieke performance testen in Visual Studio Team System te integreren en resultaten aan change sets te relateren. Bij een professionele inrichting van het ontwikkelproces hoort geautomatiseerd bouwen en testen. Het invoeren van een geautomatiseerde performance test is essentieel voor het bewaken van de kwaliteit van je systeem. Door deze regelmatig uit te voeren, worden performanceproblemen vroegtijdig gesignaleerd, is het eenvoudig om de oorzaak daarvan te vinden en worden ontwikkelaars veel performance bewuster. «