uit de praktijk

Clemens Schotte en Erwin van der Valk

You show me yours, I'll show you mine (code)

Wat is het verschil tussen goede en slechte code? Elke ontwikkelaar heeft zo zijn eigen ideeën over wat goede of slechte code is. Accolades op dezelfde regel als het statement? Spaties of tabs voor het inspringen van codeblokken? Voorzien van veel commentaar? En ook al kun je over dit soort zaken zeer verhitte discussies met sommige ontwikkelaars voeren, toch zegt het weinig over de kwaliteit van de code. Je kunt alleen een indicatie van de kwaliteit van broncode geven als je de requirements ervan kent. Klein voorbeeldje: is vrijwel onleesbare code en zonder één regel commentaar slecht? Je zou denken van wel. Maar heb je wel eens MSIL-code proberen te lezen? Compactheid en snelheid waren veel belangrijkere requirements voor het team die de C# compiler heeft gemaakt, dan bijvoorbeeld leesbaarheid van de MSIL-code.
De kwaliteit van broncode is dus afhankelijk van de requirements die je eraan stelt. Deze requirements kunnen uiteenlopen van de mate van navolging van standaarden en leesbaarheid tot een scala van '-heden', zoals onderhoudbaarheid, analyseerbaarheid of uitbreidbaarheid.
Aangezien het hanteren van zeer strenge code-kwaliteitsnormen een kostbare zaak kan zijn, is het belangrijk het juiste niveau te vinden. Zowel het hanteren van te strenge kwaliteitsnormen (bijvoorbeeld bij proof of concept) als te weinig kwaliteitsnormen (bijvoorbeeld bij een systeem dat jaren in onderhoud blijft) kan een zeer kostbare zaak zijn.

Juiste balans

Om inzicht te krijgen in de kwaliteit van een stuk code kun je gebruikmaken van code reviews. Dit zijn vaak steekproefsgewijze schouwingen op een verzameling code door een senior-ontwikkelaar of architect. Bij zo'n code review wordt gekeken in hoeverre een stuk code voldoet aan de gestelde requirements. Ook bij het uitvoeren van een code review is het belangrijk de juiste balans te vinden tussen het aantal reviews en de formaliteit ervan. Er zit natuurlijk een heel grijs gebied tussen 'laat mij zo even naar je code kijken' en op gezette tijdstippen een formele review. Daarnaast wordt bij uitbestedingstrajecten vaak een onafhankelijke derde partij (zoals Microsoft Services) gevraagd om een code review uit te voeren. Deze review vindt vaak achteraf plaats en om heel eerlijk te zijn: dan is het kwaad soms al geschied. Wat kun je aan het eind van een project nog doen als blijkt dat de code niet voldoet? Voor code reviews geldt het advies, 'measure early, measure often'. Hoe eerder je problemen constateert, hoe goedkoper het is ze op te lossen.
Een enkele keer komen we bedrijven tegen die geen code reviews uitvoeren. Als reden wordt bijvoorbeeld opgegeven: 'Code reviews kosten (te)veel tijd en we lopen al achter op schema'. Elke ervaren ontwikkelaar of projectleider kan je echter vertellen dat het beknibbelen op kwaliteit vroeg in een project, later voor problemen gaat zorgen. Een andere reden voor het niet uitvoeren van code reviews is het ego van ontwikkelaars. Ze willen niet dat anderen commentaar leveren op hun code, hun kindje, hun kunstwerk. Of schamen ze zich voor de kwaliteit van hun code?
Bij het uitvoeren van een code review is de manier van feedback geven daarom erg belangrijk. Uiteraard geef je feedback op de code en niet op de persoon. Feedback geven houdt niet in dat alleen de negatieve punten worden benadrukt maar ook de positieve.
Pedagogen, psychologen, dierentrainers, vrijwel iedereen is er van overtuigd dat positieve prikkels beter werken bij het veranderen van gedrag dan negatieve prikkels. Hoe vertaalt zich dat naar feedback geven bij code reviews? Het in allerlei kleurrijke termen benadrukken waarom een bepaald stuk code erg slecht is, resulteert er alleen maar in dat de programmeur zich zal afsluiten en dat de feedback niet aankomt. Een betere aanpak is, vragen wat er goed is aan de code. Vraag bij eventuele verbeterpunten, waarom iemand dit zo heeft opgelost. Wellicht bestaat er een goede verklaring voor. Of misschien heeft men wel onder (te) hoge tijdsdruk gewerkt. Geef in ieder geval een ontwikkelaar het gevoel dat hij begrepen en gewaardeerd wordt en dat hij wat geleerd heeft.
Het is erg gaaf om te zien hoe code reviews bij het patterns & practices team van Microsoft zijn verwerkt in het software-ontwikkelproces. Hier geldt de regel: 'Alle codewijzigingen moeten door ten minste twee ontwikkelaars gereviewd worden'. Voor complexere code wordt pair programming toegepast, waarbij twee ontwikkelaars of een ontwikkelaar en een tester, samen achter één toetsenbord aan een oplossing werken. Eenvoudigere wijzigingen voert men individueel door, maar deze worden altijd gereviewed door een andere ontwikkelaar, alvorens ze in te checken. Ten slotte zorgt een tester voor het reviewen en testen van alle codewijzigingen. Deze werkwijze kost in het begin meer tijd, maar levert code van een veel hogere kwaliteit op. Over het hele project gezien, werkt deze aanpak veel efficiënter, omdat bug fixing en stabilisatie veel minder tijd kosten. Daarnaast deelt het team de kennis van de code met elkaar en ontstaat er een gezamenlijk verantwoordelijkheidsgevoel voor de code.

Net als met sporten…

Je kunt veel leren van het lezen en reviewen van andermans code. Zo kom je codeconstructies tegen die je nog niet kende. Of je maakt kennis met nieuwe oplossingen. Het dwingt je om na te denken of je een bepaalde oplossing mooi vindt of niet en hoe jij het anders zou doen. Daarnaast is het lezen van code een belangrijke vaardigheid die je moet blijven oefenen. Tijdens het programmeren ben je veel meer tijd kwijt met het lezen en herlezen van code dan het intikken ervan, dus wat extra hersentraining op dat gebied kan nooit kwaad.
Als je bij jezelf denkt 'eigenlijk zou mijn team meer aan code reviews moeten doen', hoef je niet te wachten tot een architect of projectleider daarmee begint. Spreek eens met een teamgenoot af dat je elke dag, of voor het inchecken, elkaars code bekijkt. Of werk met elkaar aan een stuk code. Net zoals bij het beginnen met sporten, zal je merken dat dit in het begin niet eenvoudig is om vol te houden. Het vereist discipline en doorzettingsvermogen, zeker als de deadlines eraan komen. Net als met sporten duurt het even voor je de verbeteringen gaat zien. Maar daarna wil je nooit meer terug.

'Uit de praktijk' is een column van **Clemens Schotte**, Development Consultant bij Microsoft Services Nederland (http://blogs.msdn.com/cschotte) en **Erwin van der Valk**, Software Development Engineer bij patterns & practices in Redmond (http://blogs.msdn.com/erwinvandervalk).