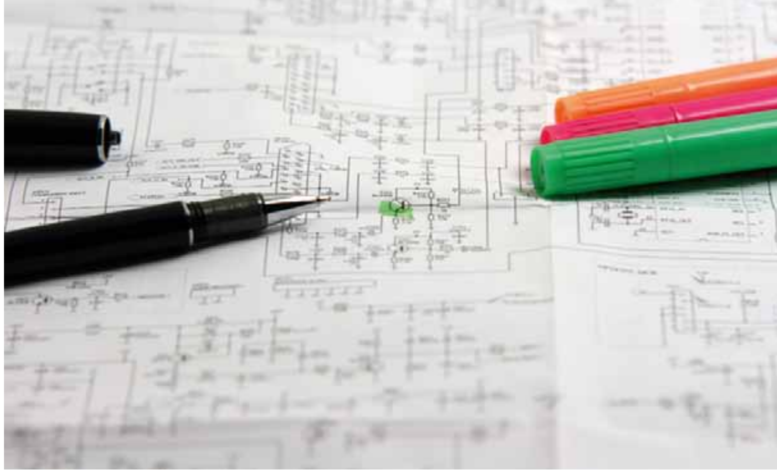


Grip op de kwaliteit van software

Dr. Jan Amoraal, dr. Giovanni Lanzani, drs. Peter Kuiters en drs. Joost Koedijk CISA CISM

Software rukt (nog) steeds verder op in samenleving en bedrijfsleven. Softwarefouten kunnen daarbij grote impact hebben, voor de reputatie en het budget van de onderneming of overheid. Het op goede wijze ontwikkelen van software is een keuze die niet alleen om inzet en volwassenheid vraagt van de softwareontwikkelaars, maar ook van hun leidinggevenden en het management! Er zijn goede en efficiënte hulpmiddelen beschikbaar om de softwarekwaliteit te meten en te beïnvloeden. Dat leidt tot effectievere software die voor lagere kosten kan worden gerealiseerd en gebruikt!

Dr. J.M. Amoraal
is adviseur bij KPMG Advisory.
amoraal.jan@kpmg.nl

Dr. G. Lanzani
is adviseur bij KPMG Advisory.
lanzani.giovanni@kpmg.nl

Drs. P. Kuiters
is manager bij KPMG Advisory.
kuiters.peter@kpmg.nl

Drs. J.M.A. Koedijk CISA CISM
is partner bij KPMG Advisory.
koedijk.joost@kpmg.nl

Inleiding

De noodzaak voor software van goede kwaliteit is in het digitale tijdperk van vandaag groter dan ooit. Vooral omdat kwalitatief slechte software kan leiden tot kapitaalverlies, gegevensverlies en imagoschade. Zoals bijvoorbeeld blijkt uit een incident bij de Knight Capital Group in de zomer van 2012: door een mislukte upgrade van hun trading software verloor men gedurende 44 minuten 10 miljoen dollar per minuut. Het verhaal, onder andere door de New York Times gerapporteerd ([TNYT12]), heeft de managers van de hele wereld wakker geschud over een onderwerp dat normaal weinig media-aandacht krijgt: softwarekwaliteit.

Softwarekwaliteit heeft vele aspecten; één daarvan die vaak het nieuws haalt is beveiliging en betrouwbaarheid van gegevens. Een voorbeeld van een niet-betrouwbare applicatie troffen wij aan bij een ziekenhuis dat tien jaar probleemloos draaide met de software totdat opeens drie dagen aan patiëntengegevens zoek waren. De ontwikkelaars van de software vonden één niet correct afgehandelde foutsituatie in de software en hebben dit probleem verholpen. Achteraf bleek dat een duct tape- of ‘houtje-touwtje’-oplossing. Onderzoek van de auteurs toonde daarna aan dat er nog meer mogelijkheden aanwezig waren die konden leiden tot ongewenst gegevensverlies.

Een ander belangrijk aspect van softwarekwaliteit is de performance. Ter illustratie, voor iedere 100 milliseconden dat een klant minder hoeft te wachten bij het laden van Amazon.com stijgt de omzet met 1%! ([King08]).

Door het uitstellen van het inlossen van de technische schuld lopen de kosten op

Met het goed uitvoeren van een software-implementatie kan veel geld worden bespaard; een kortere doorlooptijd leidt immers direct tot lagere projectkosten. Veel vooruitgang is er daarom geboekt met gestandaardiseerde projectaanpakken zoals Prince2 en MSP, maar in de praktijk blijkt een te sterke sturing op tijd en budget de overhand te hebben, het softwarekwaliteitsaspect blijft vaak onderbelicht. Uiteraard is kwaliteit een onderdeel van deze methoden, maar vooral vanuit een procesmatig oogpunt, de feitelijke invulling van de kwaliteitstesten wordt niet verder uitgewerkt.

Veel projecten beperken zich tot een serie van testen aan het einde van het project en dan alleen op die kwaliteitsaspecten van software die aan de buitenkant waarneembaar zijn, zoals vooral functionaliteit, en in beperkte mate integratie en performance.

Door deze in de praktijk gegroeide aanpak ontstaan twee risico's. Het eerste is een projectrisico: door te testen aan het einde van het project worden fouten vaak te laat in het proces gevonden, waardoor de impact op de einddatum groot is. Het tweede risico is subtieler en wordt 'technische schuld' genoemd. De voorbeelden aan het begin van deze inleiding zijn het gevolg van technische schuld. Dit betekent dat de software ogenschijnlijk goed werkt, maar onder de motorkap niet op een optimale manier is gerealiseerd. Vaak op ongewenste momenten leidt dat eens tot problemen.

Men noemt het schuld omdat er vaak op korte termijn een voordeel wordt behaald door inzet van suboptimale, tijdelijke, oplossingen met het plan om dit later te corrigeren. Het daadwerkelijk corrigeren van deze tijdelijke oplossingen (aflossen van de schuld) laat vaak lang op zich wachten. Naast deze bewuste technische schulden zijn onbewuste technische schulden, ofwel defecten – het systeem doet niet wat ervan verwacht wordt – ook een bron van geldverspilling. In een onderzoek van Jones en Bonsignour ([Jone12]) is aangetoond dat in elke fase van softwareontwikkeling, te beginnen bij het uitwerken van een idee, defecten worden geïntroduceerd en dat hoe later in de tijd de problemen worden opgelost, hoe hoger de kosten zijn. Door het uitstellen van het inlossen van de technische schuld lopen dus ook de kosten verder op. Deze kosten zijn in zekere zin te vergelijken met intrest op een schuld, alleen zijn de percentages fors hoger dan in de huidige financiële markten gebruikelijk is.

Het is duidelijk van groot belang om inzicht te hebben in en sturing te geven aan de kwaliteit van software, maar zoals aangegeven verloopt dit vaak nogal ad hoc. In dit artikel presenteren we een raamwerk voor softwarekwaliteit en de mogelijkheden om kwaliteitsaspecten eenvoudig te meten en, door herhaling, mogelijk verbetering aan te tonen. Uit praktijkvoorbeelden zal blijken hoe nuttig de toepassing van het raamwerk kan zijn.

Softwarekwaliteitsraamwerk

Wat is softwarekwaliteit? Hoe wordt de kwaliteit van software getoetst? Is software van goede kwaliteit als het bedrijfsprocessen ondersteunt? Of wanneer gegevens op een veilige en betrouwbare wijze zijn verwerkt? Is de software voor de eindgebruiker makkelijk te gebruiken? De behoefte om alle relevante vragen in kaart te brengen en meetbaar te maken heeft geleid tot een kwaliteitsstandaard voor software, namelijk de ISO-standaard 25010 ([ISO]), die de opvolger is van de ISO 9126-standaard. De ISO 25010-standaard is verdeeld in acht aspecten die op hun beurt onderverdeeld zijn in aantal subaspecten (zie figuur 1). De ISO 25010-standaard geeft een raamwerk om de kwaliteit van een stuk software in kaart te brengen. Net als voor de meeste raamwerken geldt dat voor een individueel systeem niet alle aspecten even belangrijk zijn. Bijvoorbeeld de beveiligingseisen voor een publieke internetapplicatie zoals internetbankieren zijn natuurlijk strikter dan voor een lokaal draaiend administratiepakket. Wel moeten beide applicaties in een hoge mate betrouwbaar zijn als het gaat om de kwaliteit van gegevens.

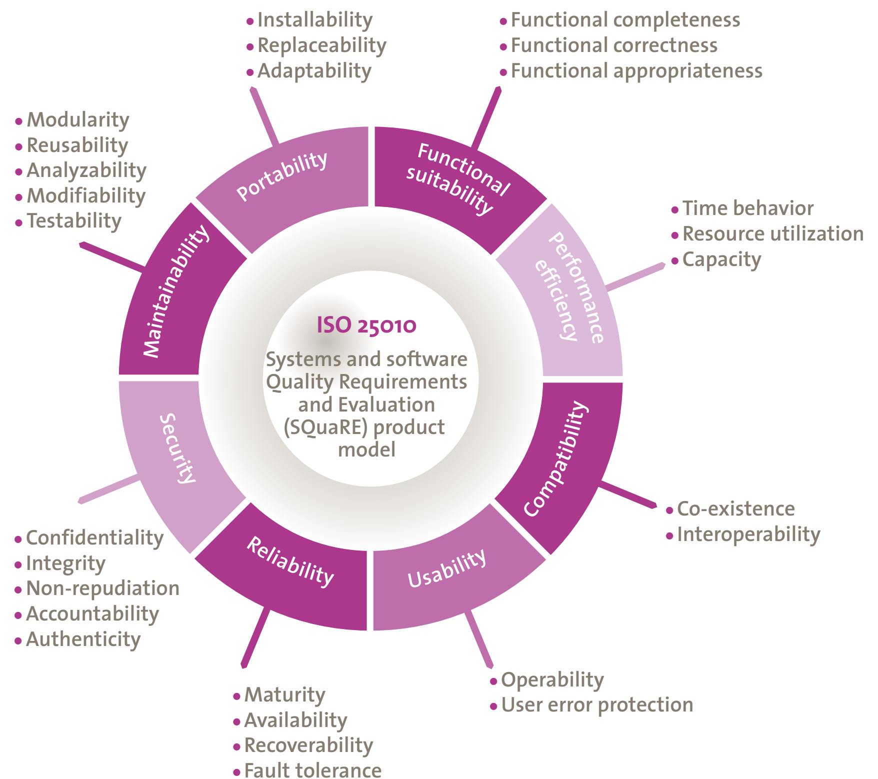
De ISO 25010-standaard is zeer uitgebreid gedocumenteerd en bevat ook methoden om veel van de gedefinieerde kwaliteitsaspecten te meten. Wat ontbreekt is een normering die aangeeft wanneer een score op een aspect als 'goed' of 'slecht' mag worden beschouwd. Dit is niet heel verwonderlijk, het is namelijk lastig, zo niet onmogelijk, om normen vooraf te bepalen omdat deze zeer van de context afhankelijk zijn. Wel zijn er benchmarks beschikbaar voor een aantal aspecten, zoals bijvoorbeeld *onderhoudbaarheid*. Maar ook mét deze benchmarks moet een beoordeling van de kwaliteit met zorg worden gemaakt; een vergelijking tussen systemen met een andere context is vragen om verkeerde gevolgtrekkingen. Om hieraan tegemoet te komen is er een aantal best practices gedefinieerd voor verschillende applicaties en platformen die binnen de context van de ISO 25010-standaard vallen.

In elk gesprek over softwarekwaliteit hoort ook de discussie over prijs-kwaliteitverhoudingen een rol te spelen. Wat zijn de kosten van een defect? Wat zijn de kosten om de kwaliteit echt te verbeteren? Wat zijn de (onderhouds) kosten als we vanuit de huidige situatie doormodderen? Er bestaat een groot verschil bij de analyse wanneer deze vragen worden beantwoord voor bijvoorbeeld vliegtuigsoftware waarbij mensenlevens op het spel staan, of spelsoftware voor een flight simulator.

Maar elke discussie over softwarekwaliteit moet uiteindelijk vooral over de software zelf gaan. Want alleen met kennis van de software zelf zijn zinnige afwegingen over de kwaliteit te maken. Dat sluit aan bij de praktijk dat vaklui, die 'software engineering' als ambacht zien, in staat zijn om herhaalbaar op een succesvolle wijze software te maken. Deze vaklui accepteren dan ook alleen kwaliteitstoetsen die de techniek en context van het product in beschouwing nemen.

Naar de bron

Meten is weten, maar wat bedoelen we precies wanneer we spreken over het meten van software? Wanneer ontwikkelaars spreken over software bedoelen ze altijd de broncode, zie het kader voor een voorbeeld. Deze broncode is het meest bepalend voor de kwaliteit van de software. De code beïnvloedt direct vijf van de acht kwaliteitsattributen ('Functional Suitability', 'Performance Efficiency', 'Reliability', 'Security', 'Maintainability'). Zeker de onderhoudbaarheid ('Maintainability') is gebaat bij goede code. Met onderhoudbaarheid bedoelen wij de inspanning die nodig is om onderhoud en uitbreidingen op de software uit te voeren. Voor kosteneffectief onderhoud is het noodzakelijk dat de structuur en architectuur van de software inzichtelijk zijn en de broncode goed leesbaar is en begrijpelijk is gedocumenteerd. Anders gezegd, er is sprake van weinig of geen technische schuld.



Figuur 1. **De aspecten en subaspecten van de ISO 25010-standaard.**

De broncode is het meest bepalend voor de kwaliteit van de software

Wat is broncode?

Simpel samengevat is een applicatie niks anders dan een set instructies in een voor een machine begrijpelijke taal die door de computer wordt uitgevoerd. Deze machinetaal is afhankelijk van de specifieke computerhardware en dusdanig complex dat deze onbegrijpelijk is voor mensen. Om het schrijven en ontwikkelen van een applicatie te vergemakkelijken, te versnellen en onafhankelijk te maken van de hardwarekeuzes, zijn er over de jaren heen verscheidene programmeertalen ontwikkeld die meer op mensen gericht zijn. Elk van deze talen heeft een vergelijkbare uitdrukkingskracht, maar heeft zijn eigen woordenboek, grammatica en bijzonderheden; vergelijk dit met natuurlijke talen zoals Engels ten opzichte van Nederlands. Deze programmeertalen zijn ook continu aan het doorontwikkelen; vergelijk het toevoegen van woorden als *ontvrienden* aan de Nederlandse taal.

De bestanden met tekst, geschreven in één van deze programmeertalen, noemen we broncode. Een simpel voorbeeld van broncode is de volgende regel tekst geschreven in de programmeertaal Python, een voorbeeld van een mensgerichte taal.

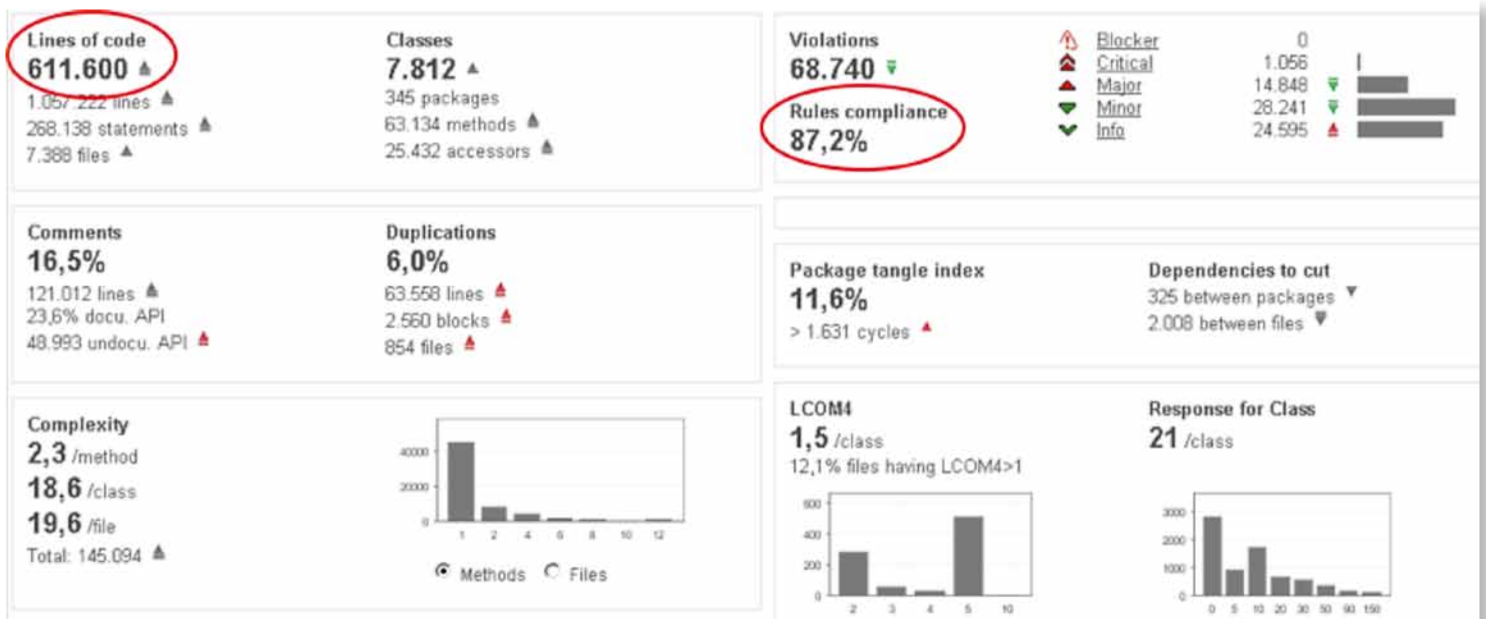
Deze regel zorgt dat de tekst 'Hello World' op het scherm wordt weergegeven, ofwel wordt ge*print* op het scherm.

```
print 'Hello World'
```

Om een indruk te geven van de complexiteit van de machinetaal die hieraan ten grondslag ligt volgt hieronder de machinetaal die nodig is om hetzelfde resultaat te bereiken wanneer dit programma wordt uitgevoerd op een Intel machine (x86-architectuur).

```
section .data
str:     db 'Hello world!',
str_len: equ $ - str

section .text
global _start
_start:
      mov    eax, 4
      mov    ebx, 1
      mov    ecx, str
      mov    edx, str_len
      int    80h
      mov    eax, 1
      mov    ebx, 0
      int    80h
```



Figuur 2. **Samenvattende uitkomsten van geautomatiseerde hulpmiddelen voor een broncodeonderzoek.**

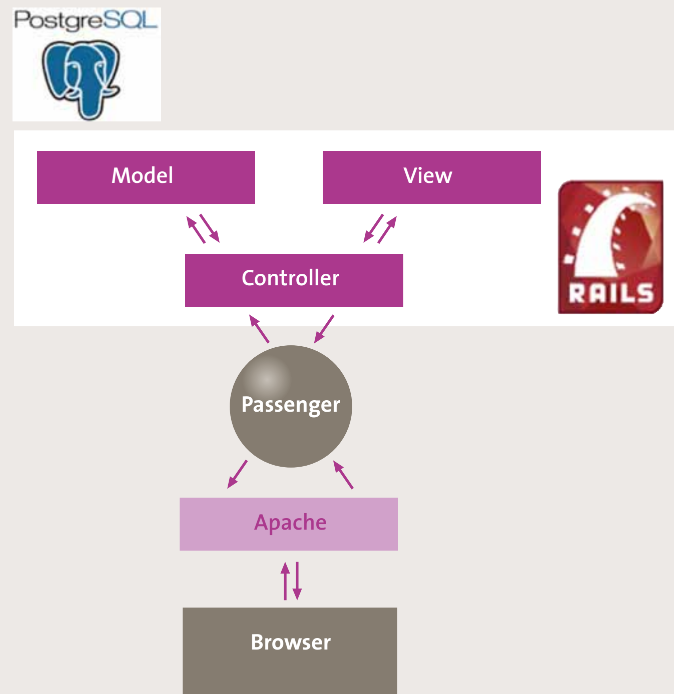
De kwaliteit van de broncode bepaalt dus in belangrijke mate de kwaliteit van de software. Om de kwaliteit van de broncode te toetsen zijn er veel geautomatiseerde hulpmiddelen voor vrijwel elk platform beschikbaar waarmee veel inzicht, zeker rond het aspect onderhoudbaarheid, is te verkrijgen. Deze hulpmiddelen zijn vaak nagenoeg gratis en geven veel nuttige en objectieve informatie over de kwaliteit van de broncode. Deze hulpmiddelen lezen automatisch de broncode, verzamelen kengetallen en toetsen de broncode tegen relevante 'good practices' ten aanzien van spelling, grammatica en schrijfstijl. In figuur 2 geven de omcirkelde stukken bijvoorbeeld aan dat er sprake is van een groot stuk software dat bestaat uit ruim 600.000 regels broncode en deze voldoen voor 87,2% aan de voor deze taal van toepassing zijnde grammatica- en stijlregels.

We komen in onze praktijk weinig klanten tegen waar het (project)management dit soort (objectief) inzicht in de kwaliteit van de broncode heeft. Vaak zien wij dat enkel de betrokken programmeur haar/zijn code leest en beoordeelt. In onze praktijk voeren we dus als startpunt altijd een eenvoudige broncodescan uit met behulp van de relevante hulpmiddelen, om in korte tijd al veel inzicht in de kwaliteit van de broncode te krijgen. Dit inzicht wordt daarna door ons geïnterpreteerd, beschreven en voorzien van concrete aanbevelingen om de kwaliteit van de software te verbeteren. Het is onze ervaring dat de objectiviteit en meetbaarheid van deze aanpak veel inzicht en begrip geeft bij het (project)management, de gebruikersorganisatie en de softwareontwikkelaars.

Systeemarchitectuur – patronen en structuur

Goed werkende software komt niet alleen door goede broncode. Een applicatie bestaat vaak uit een samenspel van meerdere standaardcomponenten zoals een database, een webserver, enz. De keuze voor de specifieke componenten en de inrichting daarvan zijn uiteindelijk ook heel belangrijk voor een correcte werking van de applicatie. Het geheel van componenten en hun samenhang in relatie tot de applicatie wordt ook systeemarchitectuur genoemd. Bij de keuze van de systeemarchitectuur spelen aspecten zoals beveiliging, audit-trails, performance en gegevensuitwisseling met anders systemen een grote rol. Het moge duidelijk zijn dat bij het opstellen van de systeemarchitectuur keuzes en afwegingen moeten worden gemaakt ten aanzien van deze aspecten. Een goede systeemarchitectuur zorgt ervoor dat het voor de programmeur eenvoudig is om haar/zijn taken uit te voeren. Hoe eenvoudiger het programmeerwerk wordt gemaakt, hoe groter de kans dat het werk van goede kwaliteit is.

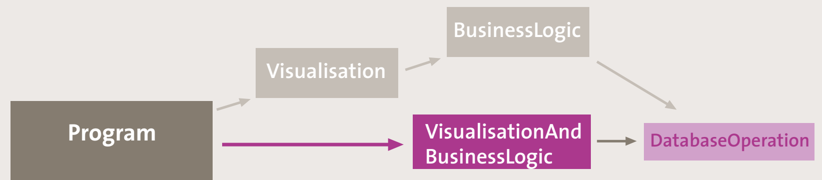
Voorbeeld van een systeemarchitectuur en dependencydiagram



Figuur 3. **Typische systeemarchitectuur van een webapplicatie met de programmeertaal Ruby.**

Software van enige omvang dient gestructureerd te zijn opgezet; de gekozen structuur wordt software- of systeemarchitectuur genoemd. Door het systeem in verschillende componenten op te delen blijft de broncode in de componenten, door de scheiding van taken ('separation of concerns'), eenvoudig. Dit bevordert de onderhoudbaarheid van het systeem! De weergave van de systeemarchitectuur maakt het verder mogelijk om met belanghebbenden over ontwerpbeslissingen te spreken.

Uit de broncode blijkt of de ontwikkelaars zich aan de systeemarchitectuurafspraken houden. Veelvuldig komen, zoals in het in figuur 4 weergegeven vereenvoudigd 'dependencydiagram', overtredingen aan het licht. Deze overtredingen hinderen altijd de onderhoudbaarheid en kunnen ook de betrouwbaarheid, performance en schaalbaarheid in gevaar brengen!



Figuur 4. **Een dependencydiagram geeft de afhankelijkheden van de softwarecomponenten weer. In het geïmplementeerde (MVC-)patroon horen visualisatie en businesslogica niet in één component te zijn opgenomen!**

Maar als een systeemarchitectuur goed kan zijn, dan komt het ook voor dat de systeemarchitectuur niet effectief is ingericht. Een eenvoudig voorbeeld kwamen we tegen bij de review van het maatwerk op een standaard-CRM-pakket. Dit webgebaseerde standaardpakket had een horizontale schaalbaarheidsstrategie waarbij bij een toenemend aantal gebruikers eenvoudig additionele hardware kan worden ingezet voor de applicatiecomponent om de extra vraag op te vangen. Het aangebouwde maatwerk volgde echter een andere schaalbaarheidsstrategie door de applicatielogica binnen de database te programmeren. Door deze keuze zal bij een toenemend aantal gebruikers ook additionele hardware voor de databasecomponenten moeten worden ingezet. De gebruikskosten nemen hierdoor extra toe!

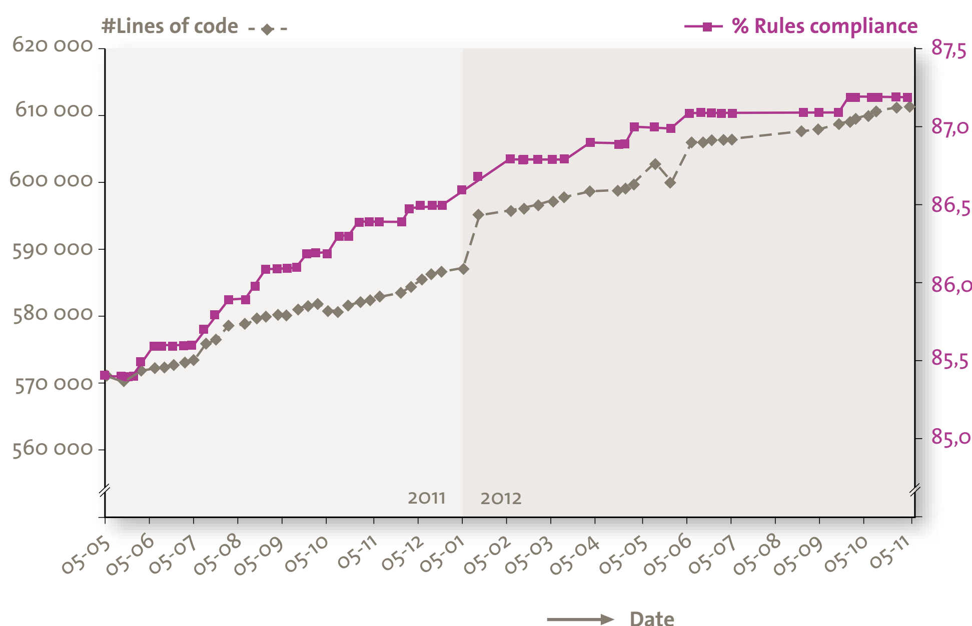
Genoemd voorbeeld geeft wederom aan waarom de afwijkingen van kwaliteitsstandaarden en architectuur van een systeem tegenwoordig vaak technische schuld worden genoemd. De 'technische schuld' an sich verhindert vaak de werking van een systeem niet. Wel dienen additionele kosten te worden gemaakt om de werking voor langere tijd te borgen. In dit voorbeeld wanneer op termijn het aantal gebruikers toeneemt.

Grip?

Kwaliteit inbedden in een project betekent vroeg defecten vinden en technische schuld aflossen, ofwel eerder in het project meten, testen en oplossen. Gevolg is een kwalitatief beter resultaat, kortere doorlooptijd en lagere kosten. Een additioneel voordeel van weinig defecten en technische schuld is dat men frequenter en met kortere doorlooptijden aanpassingen kan doorvoeren (en met minder desastreuze effecten als bij Knight Capital Group).

Een werkwijze van frequenter en geautomatiseerd testen, die ook goed aansluit bij de thans veelgebruikte Agile ontwikkelmethoden, is de DevOps-methode. In deze methode wordt er gestreefd naar goede samenwerking en informatie-uitwisseling tussen de ontwikkel- en de operationsorganisatie. Deze samenwerking, waar ook de naam DevOps symbool voor staat, wordt vormgegeven door gezamenlijke tooling, procesautomatisering en goede afspraken. Kenmerkend in deze aanpak is ook dat er veelvuldig kleine stukken software in gebruik worden genomen (en dus niet maar een of twee releases per jaar zoals in veel organisaties nog gebruikelijk is). Dit leidt tot een systeem met de gewenste lenigheid om snel fouten te corrigeren en aan nieuwe wensen te voldoen.

Let wel, deze aanpak garandeert geen foutloze software. Foutloze software is wellicht ook een utopie; zeker in een zakelijke omgeving waar het gevoel van urgentie voor softwarekwaliteit vaak te laat ontstaat om de daarvoor noodzakelijke resources en inspanning tijdig te mobiliseren. Maar door in staat te zijn (snel) ontwikkelde software in gebruik te nemen worden investeringen (eerder) te gelde gemaakt. En daar de gezamenlijke tooling de technische schuld inzichtelijk maakt worden hoge beheerlasten voorkomen!



Figuur 5. **Ontwikkeling van het systeemvolume (in effectieve broncoderegels, eLoc) en een samenvattende kwaliteitsmetriek. Uit deze gegevens kan worden geconcludeerd dat, vanaf de start van de metingen, de kwaliteit van de broncode beter wordt.**

Ook als de aandacht voor softwarekwaliteit niet vanaf het begin aanwezig is heeft het zin om de code te gaan onderzoeken en de kwaliteit in kaart te brengen. We troffen in een project dat al zes jaar liep een stuk software aan waarvan, zoals zo vaak, niemand wist hoe groot het was. Het bleek een omvangrijk stuk software met een grote 'technische schuld' – onder meer slecht leesbare en complexe code, veel afwijkingen van de architectuur en gebruik van (zeer) verouderde standaardsoftware. Het project had op dat moment geen tijd voor grote verbeterslagen, maar aandacht voor softwarekwaliteit voorkwam dat het probleem groter werd. Sterker nog – in relatieve mate werd de code door de tijd beter. Dit had ook een effect op de 'oplostijd' van de gevonden defecten. In figuur 5 is de kwaliteitsverbetering te zien (de software is hier hetzelfde als die in figuur 2). Het aantal regels code neemt toe van 570.000 naar 610.000; tegelijkertijd neemt de mate waarin de code voldoet aan de regels ook toe van 85,5% naar 87,2%. Deze verbetering wordt uitsluitend veroorzaakt doordat (vrijwel) alle toegevoegde regels aan de codeerstandaarden voldoen!

Als iets belangrijk is dan moet daar vroegtijdig, en vanaf voldoende hoog (project)managementniveau, aandacht en urgentie aan worden gegeven. Voor veel zaken is kwaliteitsmanagement in organisaties vanzelfsprekend. Dit zou ook moeten gelden voor softwareontwikkeling en onderhoud. De praktijk wijst ook uit dat als de aandacht ontbreekt de kwaliteit vaak tegenvalt. In dit artikel is aangetoond dat het goed mogelijk is om op verschillende aspecten de kwaliteit transparant in kaart te brengen. In veel succesvolle trajecten gebeurt dat ook. Uiteindelijk helpt inzicht om succesvol de eindstreep te passeren met een goede blik op de 'technische schuld'; een schuld die hoe dan ook gedurende het gebruik moet worden afgelost.

Conclusie

Problemen rond het maken en onderhouden van specifieke software zullen zeker het komend decennium niet verdwijnen. Een aantal software-incidenten haalt de publiciteit waarbij vaak de kosten van het incident zijn in te schatten. Matige softwarekwaliteit zorgt echter ook voor projecten die langer lopen en hogere onderhoudskosten. We voeren daarom in dit verband de term technische schuld in om duidelijk te maken dat softwarekwaliteit een zakelijk perspectief heeft.

Organisaties doen er daarom verstandig aan niet van kwaliteitsproblemen weg te kijken. Inzicht in de kwaliteit van software geeft de mogelijkheden om verbeteringen sneller te realiseren, lagere (onderhouds)kosten en betere houdbaarheid van de software. Het zo vroeg mogelijk adresseren van softwarekwaliteitsproblemen blijkt het meest efficiënt. Omdat ook inzicht in de softwarekwaliteit relatief eenvoudig, met geautomatiseerde hulpmiddelen, is te verkrijgen kan op feiten gebaseerd kwaliteitsmanagement worden ingericht.

Dit is alleen te bereiken met voldoende aandacht door (project)management voor softwarekwaliteit. De uitdaging is om het inzicht te krijgen dat software van goede kwaliteit aantoonbare waarde heeft voor de onderneming!

Bronnen

[ISO] *ISO/IEC 25010:2011: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE).*

[Jone12] Capers Jones & Olivier Bonsignour, *The Economics of Software Quality*, Addison-Wesley 2012, ISBN 978-0-13-258220-9.

[King08] Andrew B. King, *Website Optimization*, O'Reilly Media, Inc., 2008, ISBN 978-0-596-515-08-9.

[TNYT12] The New York Times, *Knight Capital Says Trading Glitch Cost It $440 Million*, 2 August 2012, http://nyti.ms/NnZL8r.

Over de auteurs

Dr. J.M. Amoraal is adviseur bij KPMG Advisory. Hij is gepromoveerd bij het LHCb-experiment op CERN en heeft daarbij veelvuldig zeer complexe data-analyses uitgevoerd, waarvoor hij ook het nodige aan software heeft ontwikkeld. Zijn werkzaamheden richten zich momenteel voornamelijk op Software Quality en dan met name broncodeanalyse wat betreft de aspecten beveiliging en onderhoudbaarheid.

Dr. G. Lanzani is adviseur bij KPMG Advisory. Hij heeft zijn promotieonderzoek in theoretische natuurkunde (Universiteit Leiden) gedaan waarbij hij veel ervaring heeft verzameld in het kader van software en testen hierop. Bij KPMG richt hij zich op Software Quality en Data Analyse&Modellering.

Drs. P. Kuiters is manager bij KPMG Advisory en vijftien jaar werkzaam in de IT, met het accent op standaardpakketsoftware en integratie. Hij heeft verschillende rollen vervuld van ontwikkelaar tot architect en van teamleider tot reviewer. De laatste jaren is hij zich gaan specialiseren in IT-architectuur om structuur en samenhang aan te brengen in complexe en ondoorzichtige IT-omgevingen. Voorbeelden waarin zijn expertise tot uiting kwam zijn applicatieconsolidatie en/of integratieprojecten en IT-strategietrajecten.

Drs. J.M.A. Koedijk CISA CISM is partner bij KPMG Advisory en al jaren actief op het gebied van software engineering en internetstandaarden. Hij heeft veel ervaring rond review, ontwerp, ontwikkeling en implementatie van complexe systemen en geeft thans leiding aan de Software Quality-praktijk van KPMG. Zijn expertisegebieden omvatten softwarekwaliteit, webapplicaties, gegevensverwerking, reliable messaging en Service Oriented Architecturen.