20.01.2011 - 15.41 UUR CHRISTIAN

Zolang experts het security lek dat internet heet buiten beschouwing laten zal het nieuwe werken mislukken. De nadruk leggen op de big brother benadering zorgt er juist voor dat het nieuwe werken voor nieuwe lekken in de informatievoorziening gaat zorgen.

Het Nieuwe Werken en beschikbaarheid

IS VIRTUALISATIE ECHT ZO NIEUW?

Is virtualisatie echt zo nieuw? Nee, zeker niet! IBM claimt dat men in de mainframes van decennia geleden al mogelijkheden had ingebouwd voor virtualisatie. En wat is virtualisatie? Iets virtueel? Ook niet, het besturingssysteem is wel degelijk aanwezig. En de hardware ook, alleen is er geen directe verbinding tussen hard- en software. Computable-expert Klaas de Jong vraagt zich af of we dat niet al ergens van kennen.

REACTIE 17.01.2011 - 10.50 UUR

W. DE BOER

'Een beetje een vaag verhaal waarin veel dingen door elkaar gehaald worden. Inderdaad had IBM eind jaren 60 al een systeem dat Virtual Machine heette waarop je meer (verschillende) OS'en naast elkaar kon draaien. Ideaal voor testen van een besturingssysteem. Dus nieuw? Nee absoluut niet. Overigens is de simpelste vorm van virtual machines het aanloggen van verschillende users op hetzelfde operating system. Vaak is dat al meer dan voldoende en virtualisatie domweg overbodig. Daarnaast is de virtualisatie van een Java-machine een vorm van onafhankelijk worden van de instructieset van de onderliggende hardware. Dit is virtualisatie zoals prof. Tanenbaum die uitlegt in zijn boek 'Structured Computer Organisation'. Dit heeft niets met virtual machines te maken die beogen meer OS'en naast elkaar op de zelfde hardware te draaien. Wat wil de schrijver mij hier nu uitleggen?

REACTIE 17.01.2011 - 11.57 UUR

JOHN

**'Lees Tanenbaum er nog een keer op na zou ik zeggen. JVM'en zijn er voor om los van het OS te komen. Een Java app die met dezelfde code draait op meerdere OS'en.
Het OS zelf is er voor om los van de fysieke hardware te komen. Een OS is echter ook vaak gebouwd in bijvoorbeeld C, de compilers voor de diverse hardware zorgen dan voor de vertaling naar de diverse instructiesets. Virtualisatie heeft vaak het doel veel beter gebruik te maken van de hardware.**

REACTIE 17.01.2011 - 13.30 UUR

HENRI KOPPEN

'Virtualisatie en alle evolutie die hierop plaats vindt brengt ons steeds dichter naar de mogelijkheden om it betrouwbaarder, goedkoper, schaalbaarder et cetera te maken. Dat het niet altijd lukt is een essentieel onderdeel in deze evolutie. Denk in mogelijkheden. Vind je dat we al die jaren stil zijn blijven staan? Ik zeker niet!

Agile versus het fenomeen Technology Debt

Klanten hebben steeds meer behoefte aan flexibiliteit

Steeds meer ontwikkelaars gebruiken de iteratieve en incrementele software-ontwikkelmethode Agile. Deze methode biedt de mogelijkheid om vanaf het begin van het ontwikkelproces te sturen op (tussen)resultaten door deze herhaaldelijk af te stemmen met de klant. Maar hoe verhoudt Agile zich eigenlijk tot Technology Debt?

DEVELOPMENT
Dennis Baaten

Technology Debt is het fenomeen dat zogenaamde short-cuts in broncode van software vaak in eerste instantie (tijds)winst lijken op te leveren, maar later problemen kunnen veroorzaken waardoor de 'schuld' inclusief 'rente' alsnog ingelost dient te worden. Interessant is de vraag of het voorkomen van Technology Debt op gespannen voet staat met iteratieve en incrementele software-ontwikkelmethoden zoals Agile. Zulke ontwikkelmethoden lijken het nemen van short-cuts namelijk te stimuleren.

Alvorens verder in te gaan op de vraag of Agile software-ontwikkeling de kans op Technology Debt vergroot door short-cuts te stimuleren, eerst wat meer over Technology Debt. Iedereen kent de uitspraak 'er zijn vele wegen die naar Rome leiden'. Zo is het ook met softwarontwikkeling; het realiseren van een specifieke oplossing kan op vele manieren. Uit eigen ervaring weet ik dat ontwikkelaars wel eens de weg van de minste weerstand kiezen. Bijvoorbeeld om een deadline te halen, of als gevolg van het ontbreken van ervaring. Op het eerste gezicht lijkt er vaak niets aan de hand omdat de software functioneel gezien gewoon werkt, maar er liggen risico's op de loer. Vanuit een technisch perspectief is er meestal sprake van imperfecties en/of omissies. Deze technische zwakheden worden short-cuts genoemd en kunnen serieuze problemen veroorzaken. Ter illustratie een vereenvoudigd voorbeeld om het fenomeen Technology Debt uit te leggen. Op verzoek van een klant voegt een ontwikkelaar nieuwe functionaliteiten toe aan bestaande open source software. Om de deadline te halen voorziet de ontwikkelaar de wijzigingen in de broncode niet van commentaar (bedoeld om de werking van de broncode toe te lichten). De ontwikkelaar realiseert hiermee een besparing van vier uur, maar creëert tegelijkertijd een 'schuld'. Een jaar later vraagt dezelfde klant wederom om uitbreiding van de functionaliteit. Het maken van deze aanpassingen is echter lastig; de ontwikkelaar is inmiddels alweer met een ander project bezig en moet zich opnieuw inleven in dit 'oude' project. Omdat grote delen van de broncode niet zijn voorzien van commentaar, kost het de ontwikkelaar geen acht maar veertien uur om de nodige aanpassingen in de broncode door te voeren. De ontwikkelaar dacht dus vier uur tijd te besparen door bewust een 'schuld' te creëren (het niet gebruiken van commentaar), maar moest hier later zes uur voor terugbetalen (de 'schuld' inclusief de 'rente').

Dit voorbeeld maakt duidelijk dat de 'rente' van tevoren vaak onbekend is. Sterker nog, op het moment dat de ontwikkelaar ervoor koos om de broncode niet van commentaar te voorzien, was hij zich waarschijnlijk niet eens bewust dat hij een 'schuld' creëerde. Het optreden van Technology Debt en de eventuele hoogte van de uiteindelijke 'schuld inclusief rente' is dus lastig te voorspellen. Bovendien is Technology Debt moeilijk meetbaar en/of detecteerbaar. De short-cuts hebben veelal betrekking op niet-functionele aspecten van de broncode zoals bijvoorbeeld de onderhoudbaarheid, beveiliging, en aanpasbaarheid. Het niet opnemen van een specifieke functionaliteit is geen Technology Debt, omdat dit (hopelijk) een bewuste keuze is. Technology Debt heeft een meer verborgen karakter. Dus ook wanneer de functionaliteit in orde is en de applicatie voldoet aan de eisen van de klant, kan er nog steeds sprake zijn van Technology Debt.

Technology Debt is echter anders dan de klassieke financiële schuld. Het kan namelijk gebeuren dat een 'schuld' om wat voor reden dan ook niet terugbetaald hoeft te worden. Bijvoorbeeld omdat de klant nooit meer om wijzigingen vraagt, het project vroegtijdig wordt geannuleerd, of misschien wel omdat een ontwikkelaar denkt dat de 'schuld' doorschuift naar zijn opvolger. En dat is nu juist de factor die Technology Debt aantrekkelijk maakt; een ontwikkelaar weet nooit zeker of short-cuts in de toekomst problemen zullen geven, waardoor er altijd een kans bestaat dat een 'schuld' niet terugbetaald hoeft te worden.

Agile software-ontwikkeling gaat uit van multidisciplinaire teams en korte ontwikkelcycli (weken) met na elke cyclus een testbaar resultaat voor de klant. Feedback en een eventueel veranderde doelstelling worden gebruikt om de prioriteiten voor de volgende ontwikkelcyclus vast te leggen. Op deze manier ontstaat op een snelle manier bruikbare en met de klant afgestemde software. Vaak worden Agile en soortgelijke methoden gezien als tegenhangers van de alom bekende watervalmethode waarin vooraf vastgelegde (proces) stappen in een strikt geplande volgorde worden doorlopen. Dit werkt vaak alleen wanneer de eisen aan het begin van het ontwikkeltraject helder zijn

Verschillende projectformaten breken je op

DEVELOPMENT
Alexander Vermeulen

Regelmatig kom ik in bedrijven tegen dat ze proberen het proces van de it-projecten te uniformeren. Door een eenduidige werkwijze proberen ze het mislukken van projecten te voorkomen. Als iedereen maar hetzelfde werkt, dan gaat er al een stuk minder mis. Nu is dat op zich waar, maar het introduceren van een methodiek is niet de oplossing zelf. Er moet ook eens gekeken worden naar het formaat van de verschillende projecten.

De meeste methodieken schrijven voor dat je eerst een keuze moet maken hoe en welke delen van de methodiek je toepast in je organisatie. Dat wordt dan ook ijverig gedaan, hele boekwerken worden geschreven over de aanpak. Maar toch blijven projecten mislukken. Eén van de redenen is dat in veel organisaties geen keuze gemaakt wordt voor een uniforme omvang van projecten.

Kijk nu eens naar een fabriek; bijvoorbeeld een autofabriek. Zijn productielijn kan, zonder bijstellen, meestal maar één model auto aan. Je kunt niet eerst een SUV, dan een sportwagen en daarna MPV produceren. Kijk nu eens naar wat voor soort it-projecten je door in je organisatie naast elkaar wilt doen en dat met dezelfde aanpak en werkwijze. Wil je echt verbeteren, zorg dan dat je één werkwijze en één soort omvang van je it-projecten hebt. Te grote projecten zijn altijd terug te brengen tot kleinere deelprojecten.

29.01.2011 – 09.36 UUR PETER DANEELS

Vooruitziende it-afdelingen maken vooral gebruik van public cloud-oplossingen. Private clouds worden gepusht door grote, traditionele software vendors die opnieuw hun software willen verkopen om deze private clouds te bouwen.

Met Microsoft in de wolken in 2011

en niet meer veranderen. Tegenwoordig is dit steeds minder het geval. Klanten hebben steeds meer behoefte aan flexibiliteit waardoor ontwikkelaars in toenemende mate iteratieve en incrementele software ontwikkelmethoden zoals Agile gebruiken. Het is echter de vraag of de ontwikkelmethode Agile de kans op Technology Debt groter maakt. Zoals het een goede methodiek betaamt, zegt Agile niks over het ontwikkelen als zodanig. Agile zegt alleen iets over het ontwikkelproces, zoals bijvoorbeeld de interactie tussen betrokkenen. Eén van de twaalf principes van de Agile ontwikkelmethode (zie kader) beschrijft zelfs dat er voortdurende aandacht voor hoge technische kwaliteit en een goed ontwerp dient te zijn. En dan komt het antwoord op de vraag in zicht. Technische uitmuntendheid en een goed ontwerp, of in dit geval correct omgaan met Technology Debt, is mensenwerk. Het enige wat een methode hieraan kan bijdragen is zorgen voor expliciete aandacht gedurende het ontwikkelproces. Daarom is mijn antwoord op de eerder geformuleerde vraag of Agile software ontwikkeling de kans op Technology Debt vergroot door short-cuts te stimuleren 'Nee,' Agile ontwikkeling stimuleert het nemen van short-cuts niet, maar biedt juist mogelijkheden om de kans op Technology Debt te verkleinen door hier expliciet aandacht voor te hebben.

Dat betekent echter niet dat hiermee alle gevaar is geweken. Het gebruiken van Agile is namelijk niet automatisch een garantie voor succes. De effectiviteit van de methode staat of valt namelijk met de wijze waarop ontwikkelaars hier invulling aan geven; een ontwikkelaar dient de Agile-mindset goed tussen de oren te hebben. Als een ontwikkelaar het gevoel heeft dat Agile snelheid prefereert boven kwaliteit, dan heeft deze de principes van Agile niet goed begrepen en gaat het mis. Dat is namelijk niet wat Agile beoogt. Agile zorgt ervoor dat de doorlopen paden niet teveel zijn afgeweken van de (veranderende) beoogde paden. Hierdoor ligt het eindresultaat veel dichter bij de wensen en eisen van de klant. En het heeft tevens tot gevolg dat er een stuk minder vertragingen zijn, wat uiteindelijk resulteert in een korte(re) doorlooptijd. Het verminderen van potentiële vertragingen vind ik echter iets anders dan het versnellen van klassieke software-ontwikkeling (bijvoorbeeld op basis van de watervalmethode) door het project in stukken te hakken en de druk bij de ontwikkelaar op te voeren.

Een veel voorkomend punt van kritiek met betrekking tot Agile is het feit dat deze methodiek alleen werkt bij ervaren software-ontwikkelaars. Vanuit het perspectief van Technology Debt kan ik het met deze kritiek wel eens zijn. Zoals gezegd is het omgaan met Technology Debt mensenwerk, en dan kan ik me prima voorstellen dat een ervaren ontwikkelaar zich eerder bewust is van het nemen van short-cuts, en de implicaties ervan op korte en lange termijn beter kan inschatten. Toch kunnen ook de minder ervaren ontwikkelaars correct omgaan met Technology Debt. In eerste instantie is het belangrijk dat ontwikkelaars het fenomeen Technology Debt goed begrijpen; bewustzijn is namelijk de eerste stap om mogelijk ongewenste short-cuts te herkennen.

Daarnaast is het belangrijk dat Agile op de juiste wijze wordt toegepast, door zoveel als mogelijk vast te houden aan de geformuleerde principes. Zorg er in ieder geval voor dat er ruimte is om zaken als Technology Debt met collega ontwikkelaars, en in sommige gevallen misschien wel met de klant te bespreken. De klant heeft namelijk de meeste kennis van zijn onderneming en kan waarschijnlijk het beste inschatten hoe groot de kans is dat door veranderende omstandigheden zijn eisen zullen veranderen, waardoor aanpassingen in de software noodzakelijk zijn.

■ **Dennis Baaten,** consultant Verdonck, Klooster & Associates

Principes van Agile software-ontwikkeling

1. De hoogste prioriteit is het tevredenstellen van de klant door het vroegtijdig en voortdurend opleveren van waardevolle software.
2. Verwelkom veranderende behoeftes, zelfs laat in het ontwikkelproces. Agile processen benutten verandering tot concurrentievoordeel van de klant.
3. Lever regelmatig software op. Liefst iedere paar weken, hooguit iedere paar maanden.
4. Mensen uit de business en ontwikkelaars moeten dagelijks samenwerken.
5. Bouw projecten rond gemotiveerde individuen. Geef hen de omgeving en ondersteuning die ze nodig hebben en vertrouw erop dat ze de klus klaren.
6. De meest efficiënte en effectieve manier om informatie te delen in en met een ontwikkelteam is door met elkaar te praten.
7. Werkende software is de belangrijkste maat voor voortgang.
8. Agile processen bevorderen constante ontwikkeling. De opdrachtgevers, ontwikkelaars en gebruikers moeten een constant tempo eeuwig kunnen volhouden.
9. Voortdurende aandacht voor een hoge technische kwaliteit en voor een goed ontwerp versterken agility.
10. Eenvoud, de kunst van het maximaliseren van het werk dat niet gedaan wordt, is essentieel.
11. De beste architecturen, eisen en ontwerpen komen voort uit zelfsturende teams.
12. Op vaste tijden, onderzoekt het team hoe het effectiever kan worden en past vervolgens zijn gedrag daarop aan. (bron: http://agilemanifesto.org/)

GENERATIEKLOOF IN CODEREN

Aan het einde van de jaren tachtig werkte softwareontwikkelaar sop MS-Dos pc's. Van Windows 3.1 hadden we nog niet gehoord en op op het bureau van Computable-expert Robert Mansour stonden twee pc's: één om te ontwikkelen en één om te testen. De belangrijkste knop was de resetbutton op je pc; die kennen we nu niet meer. Een leuke en leerzame tijd was het. Hij kijkt er met plezier op terug.

REACTIE 10.01.2011 – 13.28 UUR
RENÉ VAN OEVELEN

Ik verbaas me er ook iedere dag weer over dat het genereren van code geen gemeengoed is geworden. In de tweede helft van de jaren 80 ontwikkelde ik ook op MS-DOS pc's maar gebruikte daar toen de DSA COBOL & Pascal Generatoren voor. Je kon een complete applicatie op deze wijze genereren, eigen code werd door de generator automatisch tussengevoegd, dus het onderhoud bleef op deze wijze mogelijk. Vandaag de dag werk ik met Java. Ik heb nog nooit zoveel code zelf moeten schrijven als nu. Een gemiste kans, jammer!

REACTIE 11.01.2011 – 10.14 UUR
JOAO SCHIM

Code generatie komt goed van pas waar veel sprake is van zogenoemde boilerplate code. Code die geen inhoudelijke functionaliteit toevoegt maar eerder het raamwerk waarin de functionaliteit gehangen kan worden. Kijken we naar (het voorbeeld van René) enterprise Java dan zien we dat heel veel boilerplate code niet meer door een developer uitgevoerd hoeft te worden. Veel wordt achter de schermen al voor de ontwikkelaar uitgevoerd door de applicatie server zelf. Daarnaast zijn er frameworks zoals (jBoss) Seam die er voor zorgen dat je je als ontwikkelaar kunt focussen op louter de business logica van je applicatie. Veel van deze handigheidjes gebeuren veelal door code generatie. Ik vind het dus vreemd om te stellen dat code generatie geen gemeengoed is. De vorm is anders dan de 'oude' code-generatie (geen generator waar je UML-modellen doorheen jast) maar als je goed kijkt ...

REACTIE 18.01.2011 – 08.08 UUR
ALEXANDER VERMEULEN

Ja, ik herken dit ook. Als exprogrammeur hoop je dat je oude vakgebied een bepaalde ontwikkeling doormaakt. Nu zijn er wel degelijk ontwikkelingen, maar die liggen vooral op het vlak van aansluiten op de nieuwste Windows-componenten en userinterface. Dat waren dingen die vroeger niet spannend waren omdat ze redelijk eenvoudig waren.