Core

Scott W. Ambler is een van de kopstukken uit de agile wereld. Door zijn columns in Dr. Dobb's, zijn boeken en zijn vele lezingen heeft hij grote bekendheid verworven. Hij heeft het Agile Unified Process ontworpen, een vereenvoudigde versie van het Rational Unified Process (RUP) en het Enterprise Unified Process, een uitbreiding op RUP. Ook is hij *Practice Leader Agile Development* bij IBM.

Scott Ambler:

Als je RUP goed doet is het Agile

Dré de Man
Tekst en foto's

Toen Scott Ambler eindelijk verscheen voor het interview was het al happy hour. Toepasselijk, aangezien het in een café plaatsvond. Ambler is één van die mensen die je nauwelijks hoeft te interviewen, één vraag levert meteen een pagina op en met een paar vragen heb je al een publiceerbaar verhaal. Maar Ambler – misschien ook een beetje geholpen door ons eigen Heineken – bleef doorpraten, ook na mijn laatste vraag, dus uiteindelijk zijn het twee interviews geworden. Een deel van dit interview is gepubliceerd in Java Magazine van september. Dit deel gaat meer over het Rational Unified Process en grote projecten. Het gedeelte van het gesprek dat plaatsvond het uur voordat Ambler in de taxi stapte, ging helemaal niet meer over IT en hoewel het zeker amusant was hebben we het maar niet gepubliceerd.

Ik heb RUP wel eens horen beschrijven als een proces om te voorkomen dat je Agile gaat werken. Hoeveel Agile denkt u dat met RUP samen kan gaan?

Ja, er is veel kritiek geweest op RUP op dit punt. Maar RUP is net zo Agile als je het wil maken. Als je kijkt naar de succesvolle RUP-projecten, dan zijn die vaker Agile dan niet. Lang voordat ik bij IBM ging werken, heb ik al het boek Agile Modelling geschreven. Dat was het tweede boek waarin serieus werd gesproken over Agile RUP. Veel mensen brengen het ook in praktijk, er zijn heel vele case studies op het web te vinden. Wanneer je RUP goed doet, dan is het Agile. Helaas is het zo dat veel organisaties bij het adopteren van RUP het advies om het aan te passen in de wind slaan, en het op een veel te massieve manier implementeren. Een vaak voorkomend anti pattern is dat een organisatie op RUP wil overgaan en het proces in handen geeft van engineers die gewend zijn aan een lineair en zeer gedocumenteerd proces. Ze passen het aan om het serieel en zwaar gedocumenteerd te maken en vervolgens zijn ze verbaasd over de uitkomst. Ze hebben dan uiteindelijk dezelfde problemen die ze daarvoor hadden, wel in een iets mindere mate, maar de onderliggende uitdagingen zijn dezelfde. De organisaties die het goed implementeren nemen die aspecten van RUP die goed voor ze zijn en implementeren die. RUP is als een buffet-restaurant. In zo'n restaurant probeer je ook niet alles te eten wat er is, je kiest de dingen uit die je lekker vindt en die doe je op je bord. Met RUP moet je ook de aspecten nemen die je liggen. Het interessante van RUP is dat het ervoor gezorgd heeft dat veel van de dingen die de Agile gemeenschap vanzelfsprekend vindt, geaccepteerd zijn geraakt. Testen door de hele lifecycle heen, bijvoorbeeld. De Agile gemeenschap heeft het test-eerst-idee ingevoerd, maar het is door RUP geadopteerd, iteratieve ontwikkeling hebben we echt populair gemaakt, het afleveren van werkende

RUP heeft de iteratieve ontwikkeling echt populair gemaakt.

software-iteraties, dat concept eveneens. Tien, vijftien jaar geleden waren dat radicale concepten waar veel organisaties mee geworsteld hebben.

De weg geplaveid

We hebben daarmee de weg geplaveid voor de Agile gemeenschap. Iets waar we nooit voor gewaardeerd zijn, is dat RUP veel ideeën bevat voor het schalen van Agile softwareontwikkeling. De Agile gemeenschap probeert het wiel opnieuw uit te vinden en introduceert technieken die in RUP al meer dan tien jaar gemeengoed zijn. Vooraf een beetje architectuur en wat requirements, dat betekent niet dat je van te voren alles heel gedetailleerd gaat vastleggen, maar je moet wel nadenken. Geen mens zal geld vrijmaken voor een project waarvan niemand weet wat de scope is, hoeveel geld het gaat kosten, hoe lang het gaat duren en hoe je het gaat bouwen. Je moet wat dingen van te voren doen, dat begint binnen de Agile gemeenschap ook door te dringen. Dit zit al jarenlang in RUP en heet de inceptie-fase. RUP is risico-gedreven softwareontwikkeling en dus in de inceptiefase, iteratie 0, proberen we de business risico's zoveel mogelijk te beperken. Dat is zeer waarschijnlijk een iteratie, waarbij je geen software schrijft. Je moet je *act* bij elkaar krijgen, je team, je goedkeuringen, je geld. Dat is een reëel gegeven, dus we moeten daarover praten als Agile gemeenschap. Als je de elaboratie fase start, dan kijk je naar technische risico's. Je wil de technisch risicovolle dingen van te voren doen, om een werkend geraamte van het systeem te krijgen. Je wil een bewijs dat de architectuur werkt, om zoveel mogelijk technisch risico uit te sluiten. Sommige mensen worden er zenuwachtig van als je al die technische dingen, frameworks en zo van te voren doet. Maar als je kijkt wat er in de echte wereld gebeurt, de business requirements met hoge prioriteit zijn bijna altijd de technisch riskante requirements, omdat de zeer riskante en zeer waardevolle dingen de tendens hebben samen te gaan. Maar af en toe krijg je een requirement van gemiddelde waarde dat ook technisch riskant is. Je bent dan gemotiveerd om het belangrijker te maken, wat een goede beslissing zou kunnen zijn. Het zou echter ook kunnen betekenen dat je een technisch risico aangaat terwijl dat niet echt nodig is. Je moet het bespreken met de belanghebbenden.

Tiende release

In de constructiefase gaat het om het afleveren van werkende software van hoge kwaliteit in iedere iteratie. Maar we hebben ook een expliciete transitiefase. In ieder redelijk interessant systeem kan het in productie brengen van software best gecompliceerd zijn. Bij een eerste versie minder, maar bij de tiende release kan het echt moeilijk zijn. Je moet letten op dataconversie, training, regulatorische dingen, er is veel waar je advies bij nodig hebt. Het is grappig, een jaar geleden schrijf ik hier een column over en binnen een week na publicatie had ik twee aanbiedingen om een boek te schrijven over hoe een Agile release te doen. En om eerlijk te zijn, kun je ook niet zeggen dat er veel boeken zijn die hier in detail op in gaan. Maar dat zou niet zo mogen zijn, software in productie brengen is vanzelfsprekend belangrijk. Het is iets waar we goed in zouden moeten zijn, en dat zou deel moeten uitmaken van de gehele lifecyle, maar het is niet zo sexy als het zou moeten zijn en we willen er niet over praten. Er zijn dus nog wat uitdagingen in de Agile gemeenschap waar we iets aan moeten doen.

Goed, je kunt dus RUP zodanig gebruiken dat je er Agile mee versterkt. Maar velen gebruiken het om er een waterval mee te doen.

Als je waterval-RUP doet dan heb je het zozeer aangepast dat het geen RUP meer is. Veel bedrijven zullen de inception-fase in de requirements-fase veranderen en elaboration in architecture fase, maar dat is niet waar het om draait. In RUP zou je gedurende het gehele proces requirements en architectuur moeten doen, natuurlijk doe je aan het begin meer dan aan het einde, dus de smaak verandert, maar je kunt nog een paar dagen voordat het in productie komt met het onderzoeken van requirements bezig houden. Wanneer dat gebeurt dan is het heel goed.

Wat vindt u van UML?

Ambler: "Ik ben er helemaal niet tegen, maar je moet je realiseren dat een heel groot deel gewoon op een bord getekend wordt, en of dat nu UML is of niet, dat kan me eigenlijk niet schelen. Je communiceert, je onderzoekt ideeën, daar gaat het om. Nu heb ik veel ervaring met modelleren en met de tools. Veel ontwikkelaars hebben die ervaring niet. Ik zou ze willen aanraden ervoor te zorgen dat ze die wel krijgen. Natuurlijk staat het goed in je CV, maar afgezien daarvan kun je ook je productiviteit enorm verhogen. Ik doe bijvoorbeeld veel database-werk. Ik zou van zijn leven geen DBL schrijven, ik modelleer het liever en druk op een knop. Waarom zou ik Java scaffolding code schrijven? Tools genereren een hoop voor je. Waarom zou ik het schrijven wanneer ik het met het treken van een paar lijntjes kan genereren? Als ik programmeer dan gebruik ik de juiste tool voor het werk, maar daarvoor heb ik wel modelleervaardigheden nodig. Je hebt tijd nodig om te leren modelleren, en om met de tools om te leren gaan en je moet visueel kunnen denken, niet iedereen kan dat.

Modelleervaardigheden

Jammer genoeg zijn er niet zoveel mensen met modelleervaardigheden dan er zouden kunnen zijn. Maar een van de redenen dat mensen zo teleurgesteld zijn in modellen, is dat in de traditionele

De problematiek rond datakwaliteit kost in de V.S. 600 miljard dollar per jaar

wereld er vaak een grote breuk is tussen de modelleerinspanningen en de codeerinspanningen. Er wordt veel te veel tijd gestopt in het seriële modelleren. Wanneer je iteratief modelleert dan zie je dat modelleren een hoop oplevert, maar in serieel werk veel minder. Ik zie steeds voor me hoe heel slimme designers en architecten heel gedetailleerd modelleerwerk doen en het aan de gewone ontwikkelaars geven…"

Die er dan vervolgens niet meer naar kijken!

Ambler: "Ja, want óf de ontwikkelaar heeft niet zo'n hoog niveau en begrijpt er niets van, óf hij begrijpt het wel maar denkt dat hij het beter weet, óf er is eenvoudigweg geen tijd om het nog te implementeren. Wanneer de ontwikkelaars al zijn begonnen met het schrijven van code voordat ze die modellen zien, dan weet je al dat ze niet meer gebruikt worden. Dit gebeurt veel te vaak. Uiteindelijk wordt dan vaak de oplossing gezocht in het opleggen van modellen en dergelijke met een hoop bureaucratie. Je maakt dan een proces dat toch al niet goed verliep alleen nog langzamer.

Je moet echt kijken naar wat er goed en wat niet, waarom die modellen niet gevolgd worden bijvoorbeeld. Dat is iets wat de Agile gemeenschap heel goed gedaan heeft. Zij hebben gekeken naar de werkelijkheid van softwareontwikkeling en de dingen die in de praktijk werken verbeterd.

Silo-systemen

Op het gebied van de architectuurproblematiek zit je nog steeds met het feit dat de neiging bestaat om silo-systemen te ontwerpen die alleen binnen hun eigen wereld goed werken. Maar om eerlijk te zijn moet ik een deel van die problematiek ook aan de data-kant neerleggen. We hebben twee jaar geleden twee onderzoekjes gedaan bij Dr. Dobb's over de stand van zaken in de datamanagement-gemeenschap en in de datakwaliteit-gemeenschap. Daaruit kwamen echt dubieuze dingen naar voren. Veel ontwikkelaars vermijden de data-mensen. Soms weten ze niet dat ze met ze zouden moeten samenwerken, dat is dan een opvoedingsprobleem. Maar tweederde van de ontwikkelaars heeft de tendens om net te doen alsof de datamanagement-collega's niet bestaan. Op korte termijn hoeft dat geen ramp te zijn, maar op langere termijn veroorzaakt het grote schade. Wanneer je vraagt waarom ze die collega's mijden, dan krijg je in 75% van de gevallen te horen dat de datamanagement te traag reageert, geen waarde bijdraagt of dat je moeilijk met ze kunt samenwerken. Die problemen liggen vooral aan de kant van het datamanagement. Zij moeten de manier waarop ze werken veranderen. Dat is ook waar het de Agile data beweging om gaat. Bij agiledata.org praten we over techniek die dataprofessionals mogelijk maken om in een evolutionaire en Agile manier te werken. Dingen als database refactoring, het testen van databases. De problematiek rond datakwaliteit kost in de V.S. 600 miljard dollar per jaar, dat is een niet te verwaarlozen hoeveelheid geld. Toch krijgt het testen van databases weinig aandacht. We horen veel over het werken met metadata, wat prima is, dat heeft ook waarde, maar alle andere mensen in de IT weten dat testen en kwaliteit hand in hand gaan. We hebben dit enorme kwaliteitsprobleem en niemand doet er echt iets aan. Als we datakwaliteitsproblemen hebben, en we weten dat we ze hebben, dan moeten we ze oplossen. Een van de onderliggende aannames in de traditionele datagemeenschap is dat het heel moeilijk is om een bestaand databaseschema te veranderen. Als je dat aanneemt, dan maakt die aanname zichzelf ook waar: 'oh, het is moeilijk, dus vermijd ik het maar'. Dat levert dan ook weer de rechtvaardiging op voor het vooraf modelleren en dat brengt weer allerlei andere risico's met zich mee. Maar klopt die aanname wel?

Vaker doen

Binnen de Agile gemeenschap wordt vaak gezegd dat wanneer iets moeilijk is, dat een indicatie vormt voor het feit dat je het juist vaker moet doen. Dan word je gedwongen om uit te zoeken hoe je het probleem te lijf moet gaan. We hebben het uitgezocht, en daar komt het idee van database refactoring uit voort. We hebben aangenomen dat het gemakkelijk zou moeten zijn om een database te refactoren, want als het niet zo zou zijn, dan zouden we echt een probleem hebben. Het is niet moeilijk, in mijn boek Refactoring Databases staat de volledige source code om productiedatabases te repareren. Dat boek is geschreven vanuit de aanname dat het gaat om een relationele database waar honderd verschillende systemen toegang toe hebben, geschreven in honderd verschillende talen, op honderd verschillende platforms, op honderd verschillende release-cycli, en op negenennegentig van die teams heb ik geen enkele invloed. Zelfs in die complexe situatie is het nog steeds triviaal om een relationeel databaseschema te refactoren. Als je weet hoe het moet. We hebben laten zien hoe gemakkelijk het is. Dit heeft de databasegemeenschap echt voor aap gezet. Jullie aanname is verkeerd en alle praktijken die op die aanname zijn gebaseerd zouden nu in twijfel moeten worden getrokken. We hebben in de Agile gemeenschap databaseontwikkeling echt geavanceerd en de uitdaging is nu de traditionele gemeenschap mee te krijgen: hier is een veel betere manier van werken en het past precies bij moderne softwareontwikkeling. Er is een grote kans dat ze nu mee kunnen werken met de rest van de IT in plaats van een flessenhals te zijn.

Begin er niet over

Ik bezoek grote bedrijven over de hele wereld en ik was een jaar geleden bij een groot financieel instituut. Een van mijn vragen aan het senior management is: 'Hoe gaat het met de datagroep?' De CIO was erbij en rolde met zijn ogen en zei: 'Begin er niet over'. We hebben het hier wel over een groot financieel instituut. Het is een bekend probleem, maar het wordt heel vaak onderschat. In het licht van de grootte van het probleem – 600 miljard dollar – is het duidelijk dat we het beter moeten doen. Het gaat ook beter, in ieder geval in de Agile gemeenschap. Maar er is ook een culturele verandering voor nodig, in de datagemeenschap, in echte verandering in de skill set. Dat laatste is bedreigend. We zeggen: dit is een groot probleem, hier is je nieuwe skill set en o ja, het is compleet anders dan wat je had. We hoeven ons niet te verwonderen wanneer men er tegenin gaat. «

Patches Patches Patches Patches Patches Patches Patches P

Artikelen over onderwerpen als software-ontwikkeling, Java, UML, eXteme Programming en nog veel meer vindt u in het Online Archief van Array Publications. Vaktijdschriften als Software Release, Java Magazine, Database Magazine en ons Oracle vakblad Optimize hebben hun artikelenarchief online gezet. Dankzij de heldere zoekstructuur vindt u snel wat u zoekt op www.release.nl.

PowerBuilder 11.5

Sybase heeft ook een vernieuwde versie van de 4GL ontwikkelomgeving voor Rapid Application Development (RAD) uitgebracht: Sybase PowerBuilder 11.5. Het vernieuwde DataWindow daarin biedt specifieke faciliteiten voor ontwikkelaars die hun applicaties met het .NET framework willen integreren. De belangrijkste nieuwe functionaliteiten:

PowerBuilder voor .NET met regelmatige updates, die in de pas lopen met de nieuwe releases van .Net-technologie door Microsoft, inclusief ondersteuning voor Strong Named Assemblies, toegang tot .Net static, primitive en enumerative classes en ondersteuning van IIS7.

Nieuwe innovaties in het DataWindow - dankzij 3D chart rendering, speciale gradient- en transparantiefunctionaliteiten, de ondersteuning van PNG-bestanden en de nieuwe Rich Text editfaciliteiten voor kolommen, kunnen ontwikkelaars hun applicaties ook visueel verder verfijnen.

Nieuwe native RDBMS-drivers voor SQL Server 2008 en Oracle 11g.

Flexibele applicatieserver plug-in, zodat vanuit PowerBuilder IDE direct gebruikgemaakt kan worden van applicatieservers van derde partijen.

Verbeterde beveiligingsfaciliteiten - met de nieuwe beveiligingsfuncties kunnen ontwikkelaars hun applicaties voorzieningen meegeven op het gebied van beveiliging, compliance en integriteitsborging conform de US Federal Desktop Core Configuration (FDCC) specificaties. PowerBuilder 11.5 biedt ook ondersteuning aan CAS (Code Access Security), waarmee .NET-applicaties in 'partially trusted zones' kunnen draaien.

Meer informatie over de nieuwe versie van PowerBuilder is te vinden op http://www.sybase.com/powerbuilder.

DHPA introduceert code of conduct

De Dutch Hosting Provider Association (DHPA) heeft haar eerder aangekondigde code of conduct en een framework voor service level overeenkomsten geïntroduceerd. De code of conduct en het framework brengen transparantie en de mogelijkheid tot het inhoudelijk vergelijken van aanbieders van managed hosting services. Dit leidt tot een beter begrip van de rol van de sector en de inhoudelijke propositie van hosting providers in het algemeen en van de DHPA leden in het bijzonder. De Dutch Hosting Provider Association (DHPA) streeft naar een verdere professionalisering van de hosting sector. De DHPA en haar doelstellingen worden gesteund door onder andere Cisco, Citrix, F5, IBM, Microsoft, Novell, RedHat, Sun Microsystems, VM-ware en ECP.nl