VAD ÄR AGILE MODELING?

Ove H. Holmberg, Merga Communications AB

Version 1.0

Revision

1.0 – 2002-09-23
Version översatt utan större modifieringar från Engelska till Svenska av Intrasiwan AB

Tack

Originaldokument: Scott W Amber – Ronin International
Översättning: Siw Forsell – Intrasiwan – 0708-655465

Om detta dokument

Detta dokument är en översättning från engelska till svenska av Intrasiwan. Originaltexten är hämtad från www.agilemodeling.com och är översatt med godkännade av upphovsmannen, Scott W Amber. Tanken är att detta dokument skall tjäna som en gemensam plattform för att vidare utveckla Agile och Agile Modeling för svenskt bruk. Vill du bidra till utveckling av detta så kan du modifiera http://agile.rfa.se/am där vi utformar detta vidare.

Innehållsförteckning

Vad är Agile Modeling?

Agile översatt till svenska betyder smidighet, rörlighet och flexibilitet. Agile Modeling (AM) är en användarbaserad metodik för effektiv utformning och dokumentering av mjukvarubaserade system. Enkelt uttryckt är AM en uppsättning **värden, principer och arbetssätt** för modellering av mjukvara, som kan tillämpas på IT-projekt på ett effektivt och lättillgängligt sätt.
Agile modeller är effektivare än traditionella därför att de helt enkelt är, just bra. De behöver inte vara perfekta. Man kan använda AM för behov, analyser, arkitektur och design.

AM är inte någon föreskriven process, dvs. den definierar inte detaljerade tillvägagångssätt för hur man ska skapa en särskild modell. Istället förser den oss med råd hur vi ska bli effektiva modellerare. AM handlar inte om att modellera mindre. I själva verket kommer många att upptäcka att de modellerar mer med AM än de gjorde förut. AM är beröringskänsligt, inte hårt och snabbt - tänk på AM som en konst, inte en vetenskap!

AM fastslår att värdena av Agiles utveckling, som de framställs av Agile Alliance, är nyckeln till din effektivitet. Agile Alliance förespråkar

- **individer och agerande** framför processer o verktyg
- **aktiv mjukvara** framför omfattande dokumentation
- **kundsamarbete** framför kontraktsförhandlingar
- **lyhördhet för förändring** hellre än fullföljande av plan.

Målen för Agile Modeling

- Att definiera och visa hur man **använder en uppsättning värden, principer och arbetssätt** för att uppnå effektiv, lättillgänglig modellering. Hemligheten ligger inte i själva tekniken - som till exempel användar-, klass-, data- eller användarinterface modeller - utan hur den används.

- Att visa hur man ska **använda modelleringstekniker** på mjukvaruprojekt med AM, som t ex eXtreme Programming (XP), Dynamic Systems Development Method (DSDM) eller SCRUM. För att citera Kent Beck ur *eXtreme Programming Explained (XPE):* "Vi kommer ständigt att förbättra utformningen av ett system från ett mycket enkelt utgångsläge. Alla alternativ som inte visar sig användbara tas bort." I motsats till vad några av XPs belackare påstår så investerar du faktiskt tid i modellering med en XP användning, men bara när du inte har något annat alternativ. Ibland är det mer effektivt för en utvecklare att riva upp lite damm för att tänka igenom en idé eller jämföra flera olika sätt att lösa ett problem än att bara börja hacka koder. Läs "*Agile Modeling and XP*" för detaljerad info.

- Att visa hur man **modellerar effektivt** i ett enhetligt projekt (UP), vanliga driftprocesser, som Rational Unified Process (RUP) och Enterprise Unified Process

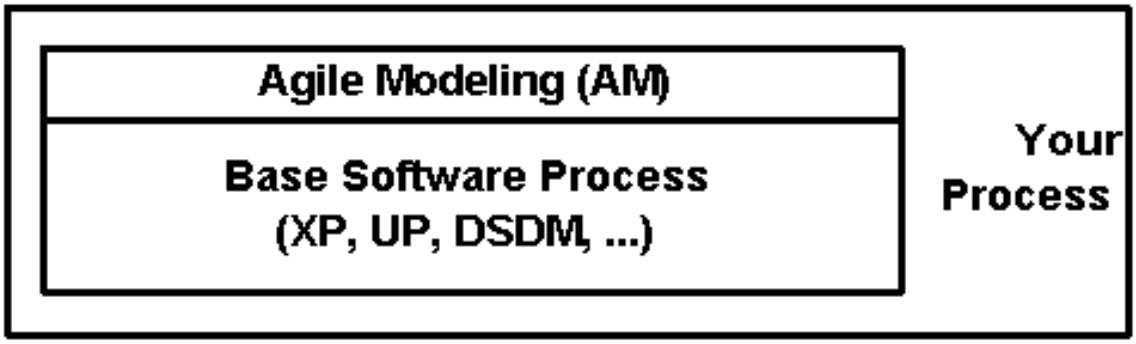
(EUP). EUP omfattar flera modellorienterade arbetsflöden - Behov, Affärs-utformning, Analys & Design, Infrastrukturledning - som kan visa sig bli ganska kostsamma. AM bidrar till att förbättra effektiviteten i dina modelleringsambitioner i ett UP-projekt, eller vilket annat projekt som helst. Läs ”*Agile Modeling and The Unified Process*" för fullständig information.

Omfattningen av Agile Modeling

Det är viktigt att förstå att AM inte är någon komplett mjukvaruprocess. Dess fokus är på effektiv modellering och dokumentering. Inget annat.

AM innefattar inte programmeringsfunktioner - men det rekommenderar att du bekräftar din modellering med kod. Det har inte heller testfunktioner fast du blir upplyst om att överväga testbarheten när du modellerar. AM täcker inte upp projektledning, systemutveckling, systemoperationer eller support, liksom en rad andra funktioner.

Eftersom AM fokuserar på en del av den totala mjukvaruprocessen måste du använda den tillsammans med en annan, fullfjädrad process, typ XP, DSDM, SCRUM eller UP som beskrivs i AM:s Mål.
Figur 1 visar konceptet. Man börjar med en basprocess - XP eller UP - eller kanske även din egen befintliga process. Sen skräddarsyr du den med AM (förhoppningsvis genom att använda all prestanda) och andra lämpliga tekniker för att skapa din egen process som svarar mot just dina behov.



Figur 1. AM förstorar andra mjukvaruprocesser.

Översikt av Värden, Principer och Arbetssätt för Agile Modeling.

AM:s värden, som anammar och utvidgar eXtreme Programmings, är kommunikation, enkelhet, respons, mod och ödmjukhet.

Nyckeln till framgång i modelleringen är att ha effektiv kommunikation mellan alla investerare i projektet, att sträva efter enklast tänkbara lösning som uppfyller alla dina behov,

att få respons på ditt arbete ofta och tidigt, att ha mod att ta beslut och stå fast vid dem och att vara ödmjuk nog att erkänna att du kanske inte vet allt och att andra kan ha värdefulla bidrag till ditt projektarbete.

AM grundar sig på en samling principer, såsom nödvändigheten av att bejaka enkelhet under resans gång eftersom behov faktiskt ändrar sig med tiden. Du bör vara medveten om att tillväxtförändringar i ditt system möjliggör smidighet, och att du ska sträva efter att erhålla snabb respons på ditt arbete för att försäkra dig om att det verkligen återspeglar investerarnas behov. Du måste ha ett syfte när du modellerar. Om du inte vet varför du jobbar med en uppgift ska du inte göra det, du behöver flera modeller i din utvecklingsverktygslåda för att vara effektiv.

En kritisk uppfattning är att modeller inte nödvändigtvis är dokument, en insikt som möjliggör att kasta de flesta modeller när de har uppfyllt sitt syfte. Smidiga modellerare anser att tillfredsställelse är mer än det som syns utåt och att du kan forma samma koncept på många olika sätt och ändå få ett bra resultat. För att bli en effektiv modellerare behöver du känna dina verktyg och modeller och för att bli en effektiv lagkamrat bör du inse att alla har något att lära av varandra.

Du ska jobba med människors instinkt. Öppen och ärlig kommunikation är ofta bästa sättet att försäkra sig om ett effektivt lagarbete. Avslutningsvis är det viktigt att fokusera på kvalitet, eftersom ingen gillar att producera ett hafsverk, och att lokal anpassning av AM är väsentligt för att möta de speciella kraven från omgivningen.

För att modellera på ett smidigt sätt måste du använda AM:s metoder rätt från fall till fall. Fundamentala metoder innefattar skapandet av flera modeller parallellt genom användandet av rätt konstart för situationen och att fortplanta detta till en annan konstform för att fortsätta framåt i stadig takt. Modellering med liten tillväxt och att inte försöka skapa den magiska "allomfattande modellen" från dina höga hästar, är också fundamentalt för din framgång som smidig modellerare.

Eftersom modeller ju bara är abstrakta representanter för mjukvara, så abstrakta att de kanske inte är korrekta, bör du eftersträva att bekräfta det med kod för att visa att dina idéer verkligen fungerar i praktiken och inte bara teoretiskt. Aktivt deltagande från investerarna är ytterst viktigt för framgången av dina modelleringsansträngningar eftersom de vet vad de vill och kan förse dig med den respons du behöver. Det finns två egentliga anledningar till att du skapar modeller. Antingen för att förstå en situation/uppgift (ex. hur du ska formge en del av ditt system) eller för att kommunicera det som ditt team gör eller har gjort.

Principen "förutsätt enkelhet" understöds av arbetssättet "skapa enkelt innehåll" genom att fokusera endast på de aspekter som du behöver för att modellera och inte försöka skapa en mycket detaljerad modell. Skissa bara med enkla noteringar och använd de enklaste verktygen för att skapa dina modeller. Du förenklar genom att kasta tillfälliga modeller och endast i nödfall uppdatera befintliga. Kommunikation sker genom offentlig visning på en vägg (anslagstavla) eller intern hemsida, kollektivt ägande av ditt projekts artefakter, genom att anamma modellstandards samt att skapa tillsammans med andra. Ditt utvecklingsarbete blir

väsentligt större när du inräknar testbarhet, använder prover med försiktighet och återanvänder befintliga artefakter.

Då du ofta behöver integrera med andra system inklusive databaslegat, liksom webbaserade tjänster, kommer du att upptäcka att du behöver upprätta kontraktsmodeller med ägarna till dessa system. Läs *How AM's practices fit together* för bättre förståelse.

En av Agiles upphovsmän - Scott W. Ambler - anser att AM är en smidig väg till modellering, att det i kärnan bara är en samling arbetssätt som reflekterar de principer och värden som delas av många erfarna mjukvaruutvecklare. Hans erfarenhet är att dessa metoder kan appliceras på de flesta mjukvaruutvecklingsprojekt. Man behöver inte jobba med ett projekt som följer en smidig mjukvaruprocess, som t ex XP, för att dra fördel av AM, även om ett av AM:s mål är att förklara hur man ska modellera i enlighet med XP. Ett projektteam behöver inte använda sig av alla de metoder, principer och värden som AM förespråkar för att ha nytta av det. Scott har alltid varit trogen övertygelsen att man ska skräddarsy sin mjukvaruprocess så den motsvarar de specifika behov som omgivningen har, även om det är hans uppfattning att, som med XP, du har större förutsättningar att lyckas om du använder dig av alla AM:s prestanda.

Agile Modelings Värden

AM:s värden inkluderar XP:s kommunikation, enkelhet, respons och mod, med det femte tillägget ödmjukhet.

Kommunikation.

Modeller förespråkar kommunikation mellan ditt team och dina investerare i projektet, liksom mellan utvecklarna i ditt team.

Enkelhet.

Det är viktigt att utvecklare förstår att är ytterst nödvändiga för både mjukvara och mjukvaruprocess - det är mycket enklare att utforska en idé och förbättra den, i takt med att din förståelse växer, genom att rita ett par diagram än att skriva tio eller kanske hundra rader kod.

Respons.

En av Agiles upphovsmän, Kent Beck, säger det bäst i *Extreme Programming Explained*: "Optimism är ett evigt programmerande, respons är behandlingen". Genom att kommunicera dina idéer via diagram får du snabb respons vilket möjliggör agerande på just det momentet.

Mod.

Mod är väsentligt för att du måste fatta viktiga beslut och vara i stånd att ändra riktning genom att antingen förkasta eller återskapa ditt arbete när några av dina beslut visar sig otillräckliga.

Ödmjukhet.

De bästa utvecklarna är ödmjuka nog att erkänna att de inte vet allting, att deras kollegor, kunder och alla investerare också har sina egna områden där de är experter och därför kan tillföra något av värde till ett projekt. Ett bra förhållningssätt är att förutsätta att alla som är inbegripna i ditt projekt har lika värde och därför skall behandlas med respekt.

Agile Modelings principer

AM kan beskrivas som en samling kärn- och tilläggsprinciper vilka, när de används på ett mjukvaruutvecklingsprojekt, utmärker nivån på en samling arbetssätt för modellering. Några av principerna har anammats från eXtreme Programming (XP) och är noggrant beskrivna i Extreme Programming Explained, som i sin tur har tagit dem från vanlig mjukvaruingenjörsteknik.

Kärnprinciper

Förutsätt enkelhet.

Vartefter arbetet fortskrider, förutsätt att den enklaste lösningen är den bästa. Bygg inte ut mjukvaran för mycket. När det gäller AM, skissa inte överflödiga funktioner som du inte behöver idag. Ha modet att inte överarbeta ditt system som du kan modellera baserat på dina nuvarande behov och omforma ditt system i framtiden när behoven ändrats. Värna om enkelheten.

Bejaka förändring.

Behov ändras med tiden. Människors förståelse för dem ändras också. Investerare kan bytas ut vartefter projektet framskrider, nya medarbetare tillkommer och gamla försvinner. Investerare kan även ändra inställning och sålunda eventuellt ändra målen och därmed framgångsfaktorerna för ditt arbete. Följden är att projektets omgivning förändras under arbetets gång och resultatet av dina insatser måste spegla denna verklighet.

Att klara av nästa steg är ditt nästa mål. Projektet kan fortfarande betraktas som ett misslyckande även om ditt team levererar ett fungerade system till användarna. En del av att uppfylla investerarnas krav är att försäkra att systemet är tillräckligt robust för att stå sig över tiden. Som Alistair Cockburn gillar att uttrycka sig: När du spelar mjukvaruspelet är ditt sekundära mål att ställa upp spelpjäserna för nästa omgång.

Följande uppgift kan vara utvecklingen av nästa stora release av ditt system eller kanske helt enkelt funktionerna och supporten av den modell du nu bygger. För att klara detta ska du inte bara utveckla kvalitet i mjukvaran utan också skapa tillräcklig dokumentation och support material så att de som spelar i nästa omgång kan bli effektiva. Faktorer som du måste ta ställning till är om medlemmar av ditt nuvarande team kommer att finnas där i nästa etapp samt prioriteringsgraden av detta för din organisation. I korthet: När du jobbar med ditt system måste du "hålla kollen" på framtiden.

Förändringar i tillväxten.

Viktigt att förstå i modelleringssammanhang är att det inte behöver bli rätt första gången. Det är faktiskt ganska osannolikt att du skulle klara det även om du försökte. Dessutom behöver du inte få med varenda detalj i dina modeller utan bara tillräckligt bra för tillfället. Istället för att envist försöka utveckla en alltigenom perfekt modell från början kan du sätta grundpelarna genom att skapa en liten modell eller kanske en hög-nivå modell och utveckla den med tiden (eller helt enkelt förkasta den när du inte längre behöver den) på tillväxtmanér.

Maximera investerarnas inflytande.

Investerarna av ditt projekt sätter resurser i form av tid, pengar, utrustning mm till förfogande för att få mjukvaran utvecklad efter deras behov. De förtjänar att dessa resurser faller i god jord och inte slösas bort av ditt team. De förtjänar också att få sista ordet när det gäller hur resurserna används eller inte används. Om det var dina pengar, skulle du vilja ha det annorlunda?

Modellera med ett syfte.

Många utvecklare oroar sig för om deras artefakter, källkod eller dokument - är tillräckligt detaljerad eller kanske för detaljerad. Vad de inte frågar sig är varför de skapar produkten över huvud taget och för vem. När det gäller modellering kanske du behöver förstå någon aspekt av din mjukvara bättre, du kanske måste kommunicera hur du går till väga inför företagsledningen för att rättfärdiga ditt projekt eller skapa dokumentation som beskriver systemet för den personal som ska arbeta med och underhålla eller utveckla det i framtiden. Om du inte kan sätta fingret på för vem eller varför du utvecklar en modell - varför då bry sig?

Första steget är att klargöra en verklig anledning att skapa en modell liksom att identifiera mottagarna av modellen ifråga. Baserat på detta, utveckla modellen till den nivå där den är både tillräckligt korrekt och detaljerad. När väl en modell har nått sitt mål är du färdig med den för tillfället och bör gå vidare till något annat, t ex skriva kod för att visa att modellen fungerar. Denna princip gäller även för ändrig av befintlig modell. Om du gör en ändring, kanske genom att använda ett redan känt mönster, ska du ha verklig anledning till det (kanske för att stödja ett nytt önskemål eller för att omforma ditt arbete till någonting renare).

En viktig slutledning av det här är att du måste känna din mottagare, även när den är du själv. Ex. Om du skapar en modell för underhållsutvecklare, vad behöver de egentligen? Ett omfattande dokument på 500 sidor eller räcker en 10 sidors översikt om hur allt fungerar? Vet du inte? Fråga dem och ta reda på det.

Flera modeller.

Du behöver flera modeller för att utveckla mjukvaran därför att varje modell beskriver en enskild aspekt av den. Vilka modeller kan behövas för att bygga toppmoderna affärsanläggningar? Med tanke på komplexiteten av modern mjukvara behöver du ha en bred uppsättning tekniker i din intellektuella verktygslåda för att vara effektiv. (Modeling Artifacts for AM.) En viktig synpunkt är att du inte behöver utveckla alla dessa modeller för ett givet system utan, beroende på mjukvarans exakta egenskaper som du utvecklar, kommer du att behöva minst en extra uppsättning av modellerna. Olika system, olika extrauppsättningar.

Precis som alla fixarjobb hemma inte kräver vartenda verktyg i lådan varje gång, så kommer mängden av arbeten du utför att med tiden kräva alla verktyg förr eller senare. Vissa modelltyper kommer du att använda mer än andra, precis som med verktyg. För mer detaljerad upplysning angående det breda spektrum av artefakter i modelleringen som du kan välja bland, många fler än i UML som påvisas i *Be Realistic About the UML*, hänvisas till *The Object Primer - An Introduction to Techniques for Agile Modeling*.

Kvalitetsarbete.

Ingen gillar hafsverk. De som gör jobbet gillar det inte för att det inte är något att vara stolt över. De som kommer i ett senare led för att omforma av någon anledning gillar inte heller hafsverk för att det är svårare att förstå och uppdatera. Och användarna gillar det inte heller för att det är bräckligt och/eller inte motsvarar deras förväntningar.

Snabb respons.

Tiden mellan en åtgärd och responsen på densamma är kritisk. Genom att arbeta med andra människor på en modell, särskilt om du jobbar med delad modelleringsteknologi (CRC kort eller viktigt modelleringsmaterial som noteslappar) får du nästan en omedelbar respons på dina idéer. Att arbeta nära kunden, förstå hans behov, analysera dem eller utveckla användarinterface som svarar mot behoven, ger möjlighet till snabb respons.

Mjukvara är ditt huvudmål.

Målet för mjukvaruutveckling är att producera mjukvara som motsvarar dina investerares behov på ett effektivt sätt. Det är inte viktigt att producera överflödig dokumentation, managementprodukter eller ens modeller. Varje åtgärd som inte direkt bidrar till att nå detta mål ska ifrågasättas och undvikas om det inte kan rättfärdigas enligt ovan.

Jobba lätt.

Alla artefakter du skapar och bestämmer dig för att behålla kommer du att få dras med i lång tid framöver. Om du behåller sju modeller och en ändring måste göras (uppdatering, ny teknologi, nya behov osv.) måste du överväga betydelsen av den ändringen på alla sju modellerna och sedan handla därefter. Om du bestämmer dig för att behålla bara tre modeller får du självklart mindre arbete att utföra för att verkställa samma förändring, vilket gör dej flexiblare för att du jobbar lättare.

På samma sätt gäller att ju mer komplicerade/detaljerade dina modeller är, desto troligare är det att alla ändringar blir svårare att genomföra (den enskilda modellen är "tyngre" och därför svårare att upprätthålla). Varje gång du bestämmer dig för att behålla en modell backar du på flexibiliteten för bekvämligheten att ha den informationen tillgänglig i ditt team såväl som bland investerarna i projektet. Underskatta aldrig allvaret i detta köpslående. En som korsar öknen har fördel av en karta, en hatt, bra kängor och en dunk vatten. Denne ökenvandrare kommer förmodligen inte att klara det om han belastar sig med hundratals vattenflaskor, komplett överlevnadsutrustning och en samling böcker om öknen. På samma sätt kommer ett utvecklingsteam snart att upptäcka att det spenderar mest tid på att uppdatera dokument istället för att skriva källkod.

Tilläggsprinciper

Innehåll viktigare än representation.

Varje modell kan presenteras på många olika sätt. En modell behöver inte vara ett dokument. Även en omfattande uppsättning diagram som skapats med ett CASE verktyg kanske inte blir del av ett dokument. Istället används diagrammen som ingredienser i andra artefakter, troligen källkod, men aldrig formella officiella dokument. Poängen är att man drar fördel av modelleringen utan att få kostnaderna för skapande och bevarande av dokumentation.

Alla kan lära av varandra.

Du kan aldrig kunna allt till fulländning. Det finns alltid tillfälle att lära något nytt och utvidga dina kunskaper. Ta tillfället att arbeta med och lära av andra, pröva nya infallsvinklar, fundera över vad som verkar fungera och inte. Teknologier ändras snabbt. Java t ex utvecklas med en rasande fart och nya teknologier såsom C# och .NET introduceras regelbundet. Existerande utvecklingstekniker utvecklas långsammare men de utvecklas ändå. Som industri har vi förstått behoven av testning ganska länge, fast vi förbättrar hela tiden vår förståelse genom forskning och praktik. Det vi vill framhäva är att vi arbetar i en industri där förändring är normen och man måste ta varje tillfälle i akt att lära nya vägar att göra saker genom träning, utbildning, mentorskap, läsning och att tillsammans.

Känn dina modeller.

I och med att du har flera modeller måste du känna till deras styrkor och svagheter för att kunna använda dem effektivt.

Känn dina verktyg.

Mjukvara i form av diagram- eller modelleringsverktyg har många användningsområden. Om du ska använda dig av ett modelleringsverktyg måste du känna till hur det fungerar för att veta när och om du inte ska använda det.

Lokal anpassning.

Ditt förhållningssätt till mjukvaruutveckling måste spegla din omgivning och din organisation, dina investerares situation och projektet som sådant. Faktorer som kan påverkas är din modelleringsteknik (dina användare kanske insisterar på konkreta användarinterface istället för ursprungliga skisser eller viktiga prototyper), verktygen du använder (det kanske inte finns någon budget för en digital kamera eller du har redan licenser för ett befintligt CASE verktyg), och mjukvaruprocessen du följer (din organisation insisterar på XP eller RUP på sin egen process).

Du bör anpassa dig både till projektnivån och den individuella nivån. En del utvecklare använder en uppsättning verktyg framför en annan, vissa fokuserar på kod med väldigt lite modellering medan andra föredrar att lägga ned lite extra tid på modellering.

Öppen och ärlig kommunikation.

Människor har behov av frihet och, för att understryka det, att erbjuda förslag. Detta innefattar idéer som hänför sig till en eller flera modeller. Någon kanske har ett nytt sätt att närma sig

delar av designen eller har en ny insikt om ett önskemål, dåliga nyheter som att ligga efter i tidsschemat eller helt enkelt nulägessituationen. Öppen och ärlig kommunikation låter folk fatta bättre beslut därför att grunden till dem är mer tillförlitliga och exakta.

Jobba med folks instinkter.

När någon känner på sig att något inte kommer att fungera, att några saker inte stämmer med varandra eller att något inte "känns rätt", är chanserna goda att det verkligen förhåller sig så. Vartefter din erfarenhet växer i mjukvaruutvecklingen blir dina instinkter tydligare och vad det säger undermedvetet kan ofta bli en viktig inkörsport till ditt modelleringsarbete. Om din instinkt säger att ett behov inte verkar vettigt eller det inte är komplett, ta upp det med användarna. Om du känner på dig att en del av arkitekturen inte kommer att klara dina behov, bygg då en snabb teknisk end-to-end prototyp för att testa din teori. Om din instinkt säger att design A är bättre än alternativ B och det inte finns någon vettig anledning att välja vare sig det ena eller andra, välj då A för tillfället.
Det är viktigt att förstå att värdet av mod innebär att du kan ändra situationen någon gång framöver om det skulle visa sig att dina instinkter var fel.

Agile Modelings Arbetssätt

AM står för en samling kärn- och tilläggsarbetssätt baserade på AM:s principer. Vissa arbetssätt har tagits från XP och är tydligt beskrivna i Extreme Programming Explained. Liksom med AM:s principer är arbetssätten presenterade med fokus på modelleringsarbetet, vilket innebär att material taget från XP kan visas på ett annorlunda sätt.

Kärnarbetssätt

Aktivt deltagande av investerare.

En utvidgning av XPs *On-Site Customer* som beskriver behovet av att ha direkt tillgång till användare som har befogenhet och förmåga att tillföra information till systemet som är under byggnad samt att fatta beslut om behov och prioritering. AM utvidgar XPs *On-Site Customer* metod till att involvera investerare, deras användare, företagsledning, dataoperatörer och support aktivt i projektet.

Använd rätt artefakter

Varje artefakt har sina egna speciella användningsområden. Till exempel är ett UML aktivitetsprogram användbart för att beskriva en affärsprocess medan den statiska strukturen av din databas bättre representeras av en fysisk datamodell. Ofta är ett diagram ett bättre val än källkod. Om en bild är värd tusen ord är en modell värd 1024 rader kod när den används i rätt omständigheter (termen lånad från Karl Wieger´s *Software Requirements*) därför att du ofta kan utforska designmöjligheter mer effektivt genom att rita några diagram på tavlan än genom att sätta dej och utveckla kodmönster. Du behöver alltså veta styrkor och svagheter för varje enskild artefakt för att veta när du ska använda dem och när du inte ska det. Observera att detta kan vara mycket svårt för att flera modeller är tillgängliga. *Artifacts For Agile Modeling: The UML and Beyond* tar upp mer än 30 olika modeller och det är långt ifrån alla.

Kollektivt ägande.

Alla kan jobba med en modell och även en artefakt i ett projekt, om det är nödvändigt.

Överväg testbarhet.

När du modellerar bör du alltid fråga dej: "Hur ska vi testa det här?" För om du inte kan testa din mjukvara ska du inte bygga den. Moderna mjukvaruprocesser inkluderar test- och kvalitetssäkringsfunktioner genom hela projektets livscykel och en del förespråkar även att man skriver testet innan man skriver mjukvaran (XP metod).

Skapa flera parallella modeller.

I och med att varje modell har sina egna styrkor och svagheter är ingen enda modell tillräcklig för dina modelleringsbehov. Till exempel kanske du behöver utveckla några viktiga usecase eller historier, en väsentlig UI prototyp, och några affärsregler medan du utforskar krav. I kombination med arbetssättet att överföra till annan artefakt kommer flexibla modellerare ofta att upptäcka att de är mycket mer effektiva när de arbetar med flera modeller samtidigt än när de bara fokuserar på en i taget.

Skapa enkelt innehåll.

Du bör se till att innehållet i dina modeller; krav, analyser, arkitektur, design är så enkelt som möjligt samtidigt som det fyller investerarnas behov. Innebörden är att du inte ska tillföra något utan att det är rättfärdigat. Ha mod att lita på att du faktiskt kan lägga till en funktion när - och om - den efterfrågas. Detta enligt XPs arbetssätt *Simple Design*.

Skissa enkelt.

När du överväger tänkbara diagram inser du snabbt att du oftast bara behöver en bråkdel av det tillgängliga utbudet. En enkel modell som visar huvudfunktionerna, kanske en klassmodell som visar de primära klasskyldigheterna och deras inbördes relation, kan visa sig räcka.

Visa modellerna offentligt.

Du bör visa dina modeller offentligt, gärna på en vägg. Öppen och ärlig kommunikation innebär att alla i ditt team snabbt kan ta del av utvecklingen i arbetet, liksom även investerarna. Visa att du inte gömmer något för dem. Modelleringsväggar kan vara virtuella, en intern webbsida t ex, som uppdateras med scannade bilder. Ellen Gottesdiener´s Specifying Reuirements With a Wall of Wonder visar detta synsätt.

Överför till annan artefakt.

När du utvecklar en artefakt, såsom ett usecase, CRC kort, sekvensdiagram eller till och med källkod, och upptäcker att du kört fast, byt då och jobba med något annat för stunden. Varje artefakt med sina speciella styrkor och svagheter, lämpar sig för en viss typ av arbete. Om du stöter på problem med en artefakt, kanske ett usecase, och du upptäcker att du kämpar för att beskriva en affärsmetod, då är det tecken på att du ska byta till en annan artefakt. Då lossnar det direkt. Dessutom kommer du ofta underfund med vad det var som var fel från början.

Modellera för liten tillväxt.

Tillväxtutveckling i mindre portioner under flera veckor eller en månad eller två ökar din flexibilitet genom möjligheten att leverera mjukvara snabbare till användarna.

Modellera med andra.

När du modellerar med ett syfte är det för att förstå någonting, att kommunicera dina idéer till andra eller för att försöka utveckla en ordinär bild av ditt projekt. Detta är en gruppaktivitet där du vill ha effektivt samarbete och input. Till exempel för att utveckla strukturen för ditt system behöver du ofta modellera tillsammans med en grupp för att hitta en lösning som alla accepterar och som är så enkel som möjligt. Bästa sättet är att diskutera igenom det med en eller flera personer.

Visa det med kod.

En modell är abstrakt och skall korrekt återge vad du bygger. Men fungerar det? Visa med kod. Skriv testkod, affärskod och kör testerna för att vara säker på att du gjort rätt. Glöm aldrig att med ett upprepande förhållningssätt till mjukvaruutveckling, normen för de flesta projekt, är modellering bara en av många uppgifter som du kommer att utföra. Modellera lite, koda lite, testa lite - bland annat.

Använd de enklaste verktygen.

De flesta modeller kan ritas på tavlan, på papper eller på baksidan av en servett. Om du vill spara något av dessa diagram, ta en bild med digitalkamera. Det fungerar därför att de flesta diagram kastas. Deras sanna värde är att rita dem och tänka igenom innehållet, sen har de inte stort värde längre.

Övriga arbetssätt.

Använd modelleringsstandards.

Grundidén med XPs *Coding Standards* är att utvecklare ska komma överens om att följa en vanlig modelleringsstandard för ett mjukvaruprojekt. Precis som ren kod enligt valda riktlinjer är lättare att förstå och utveckla, har det samma värde att följa vanlig modelleringsstandard. Det finns många att välja bland; The Object Management Group´s Unified Modeling Language (UML), som beskriver beteckningen och semantiken hos vanliga objektsrelaterade modeller, UML ger en bra början men är inte tillräckligt. Som du kan se i *Be Realistic About The UML* innefattas inte alla tänkbara modelleringsartefakter av UML. För övrigt sägs inget om att modellera stilriktlinjer för att skapa tydliga diagram. Vad är skillnaden mellan en stilriktlinje och standard? WRT till källkod, en standard skulle vara att benämna attribut i formatet attribute/Name medan en stilriktlinje ska märka koden inom en kontrollstruktur (en if-when sats, en loop...) med en enhet. WRT till modeller. En standard skulle vara att använda en rektangel för att modellera en klass i ett klassdiagram och en stilriktlinje skulle vara att ha underklasser placerade i diagram nedanför sina överklasser.

Använd mönster varsamt.

Effektiva modellerare lär sig, och använder ändamålsenligt vanliga arkitektur-, design- och analysmönster/mallar i sina modeller. Men, som Martin Fowler poängterar i sin *Is Design Dead*, bör utvecklare överväga att smidigt övergå till att använda mönster på ett varsamt sätt. Detta reflekterar värdet av enkelhet. Med andra ord, om du misstänker att ett mönster passar bör du modellera det så att du implementerar minsta möjliga volym för tillfället, vilket

underlättar omformning senare när det visar sig att ett fullfjädrat mönster kanske är det bästa. Alltså: modellera inte för mycket.

Kasta tillfälliga modeller.

De flesta modeller som du skapar är tillfälliga/arbetsskisser. Modeller blir snabbt avsynkroniserade i förhållande till koden, vilket inte är fel. Då måste du besluta om ny synkronisering tillför något värde till ditt projekt kontra att helt enkelt förkasta dem.

Formalisera kontraktsmodeller.

Kontraktsmodeller behövs ofta när en extern grupp kontrollerar en informationskälla som ditt system behöver, t ex databas, legatanvändning eller informationsservice. En kontraktsmodell är något som båda parter skall komma överens om och gemensamt ändra i framtiden om så erfordras. Exempel på kontraksmodeller är detaljerad dokumentation för ett programmeringsinterface (API), en fil-layout-beskrivning, en XML DTD eller en fysisk datamodell som beskriver en delad databas. Som med ett lagligt kontrakt kräver en kontraktsmodell ofta att man investerar stora resurser för att utveckla och behålla kontraktet och säkra att det är korrekt och tillräckligt detaljerat. Ditt mål är att minimera antalet kontraktsmodeller för ditt system för att följa XP-principen om att jobba lätt. Elektroniska verktyg gäller för framtida bevarande av dessa modeller.

Modellera för att kommunicera.

En annan anledning till att modellera är att kommunicera med personer utanför ditt team eller för att skapa en kontraktsmodell. Eftersom kunderna till en del modeller är externa kan du behöva ta dig tid att ”frisera” dina modeller genom att använda elektroniska verktyg, typ ordbehandlare, ritningsenheter eller till och med sofistikerade CASE verktyg.

Modellera för att förstå.

Den viktigaste användningen av modellering är att utforska ett problemområde, att identifiera och analysera systemets behov eller att jämföra alternativa designtyper för att hitta den enklaste lösningen som ändå uppfyller kraven. Efter dessa riktlinjer utvecklar man ofta små enkla diagram som fokuserar på en aspekt av din mjukvara, t ex livscykeln av en klass eller flödet mellan skärmar, diagram som du oftast kastar när du är klar med dem.

Återanvänd befintliga resurser.

Det finns en uppsjö av information som gagnar modellerare. Själva modellerna är kanske omoderna men med lite efterforskning kan man hitta aktuella versioner.

Uppdatera bara i yttersta nödfall.

Uppdatera bara när det är svårare att inte ha modellen uppdaterad än att göra själva jobbet. Med den här inställningen kommer du att uppdatera ett minde antal modeller än du skulle ha gjort tidigare. Dina modeller behöver inte vara perfekta för att påvisa värde. För mycket tid och pengar läggs ned på att synkronisera modeller och dokument med källkod, en omöjlig uppgift från början. Bättre att lägga de resurserna på att utveckla ny mjukvara.

Verkligt bra idéer.

Följande arbetssätt kompletterar AM men är inte en direkt del av det:

Omforma (Refactoring).

Det här är en kodmetod där man gör små ändringar, s.k. omformningar, till sin kod för att stödja nya behov eller för att hålla sin design så enkel som möjligt. Ur AM-synpunkt innebär det att du säkrar en ren och tydlig design medan du arbetar med din kod. Omformning är en integrerad del av XP.

Testa-Först Design (Test-First design).

Detta är en utvecklingsmetod där du först överväger och sen kodar dina testförsök innan du skriver din affärskod. Ur AM-synpunkt tvingar denna metod dig att tänka igenom din design innan du skriver koden, vilket undanröjer behovet av detaljerad designmodellering. Testa-Först design är en integrerad del av XP.