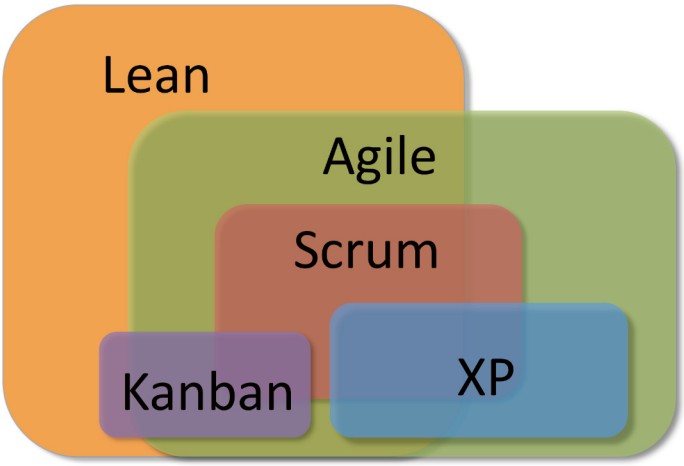
PRIORITERA
FOKUSERA
LEVERERA

DIN SNABBGUIDE TILL
LEAN, AGILE, SCRUM, KANBAN OCH XP

TOMAS BJÖRKHOLM & HANS BRATTBERG



Version 1.7 Beta

Det här är boken för dig som behöver en snabb introduktion till Scrum — eller som har ett behov av att ge bort en sådan till din chef eller uppdragsgivare. Bokens författare Hasse och Tomas berättar hur bokidén kom till:

Hasse: Jag har en kund som har så mycket att göra att VD inte hinner lyssna på utvecklingsavdelningens förslag och tankar om Scrum. Men jag tror att VD skulle läsa en artikel eller kortare skrift om ämnet när han är på resande fot. Känner någon till en sådan artikel eller bok?

Tomas: Festligt att du nämner det. Min fru frågade i går samma sak. Hennes företag ska börja med Scrum och hon sitter på en position som kommer beröras men inte tillräckligt nära för att hon ska få någon utbildning.

Hasse: OK, vi ser ett behov men ingen bok, varför?

Tomas: Tja, för att ingen kommit på idén att skriva den. Ska vi göra det?

Hasse: Vi behöver se till mer än Scrum. Utan XP finns risk att kvaliteten på projekt blir för låg.

Tomas: Bra idé. Jag vill även få med Lean och Agile så läsaren förstår tankarna bakom Scrum. Kanban är också bra att kunna, det ger bra flöde.

Hasse: Om vi riktar oss till alla de berörda men inte direkt involverade bör vi hålla oss ifrån teknobabbel och specialtermer för redan invigda...

Tomas: ... och även beskriva varför Scrum fungerar, inte bara hur.

Hasse: Okej, vi har en bokidé. Då kör vi!

Om författarna

Vi är Tomas Björkholm och Hans Brattberg. Vi arbetar båda på Crisp som Lean, Agile, Scrum, Kanban och XP-coacher. Vi brinner för att hjälpa företag lyckas med utveckling av produkter och tjänster.

– Vårt mål är att hjälpa er bli så bra på det att ni inte behöver oss längre.

Tack till

Henrik Kniberg för hans bilder, kunnande och pedagogiska förklaringar, och för att han visat att det går att skriva en bok.

Mats Henricson och Anna Sandell för deras ändlösa tålamod att putsa på texterna. Jan Grape för hans bidrag till avsnitten om parprogrammering och SAFe. Mattias Skarin för hans insiktsfulla pedagogiska förklaringar om Lean.

Jimmy Janlén för att vi fått låna bilder av honom.

Tack också till alla som bidragit med granskning och insiktsfulla kommentarer, i stort som smått: Jannika Björkholm, Per Brattberg, Claes Burlin, Jan Grape, Mats Henricson, Viktor Nordling, Minal Parekh Nygårds, Staffan Nöteberg, Per Lundholm, David Barnholdt, Oscar Lantz, Stefan Östergaard och Anneli Olsen.

Tack också till Google som tagit fram Google docs. Utan det skulle det varit svårt för oss att skriva samtidigt utan att krocka med varandra.

Till sist ett tack till Robin Dymond som inspirerade till bilden på omslaget.

© Författarna Tomas Björkholm och Hans Brattberg

ISBN: 978-91-978630-5-6

Illustrationer: Jimmy Janlén, Henrik Kniberg samt författarna

Senast uppdaterad: Feb 2021

Om oss

Vi är Tomas Björkholm och Hans Brattberg. Vi arbetar båda på Crisp som Lean-, Agile-, Scrum- och XP-coacher. Vi brinner för att hjälpa företag lyckas med mjukvara — vårt mål är att hjälpa er bli så bra på det att ni inte behöver oss längre. Om du medan du läser denna bok känner att det är saker som du skulle vilja ha utförligare förklarade, tveka inte att kontakta oss på tomas.bjorkholm@crisp.se och hans.brattberg@crisp.se för stora och små frågor!

Tomas Björkholm
har jobbat i projekt med Agila synsätt sedan 1996. Var IT- och utvecklingschef på Resfeber från starten 1997 och var kvar till 2007. Började med Scrum 2004 och Kanban 2008.

hans Brattberg
hade den första öppna XP-utbildningen i Sverige och har varit med om att införa Agila metoder baserat på Scrum, Lean och eXtreme Programming i flera projekt och organisationer.

6 Förord
7 Om boken
8 Introduktion till Agile
8 Historia
10 Introduktion till Lean
10 Historia
12 Introduktion till Scrum
12 Historia
14 Introduktion till Kanban
14 Historia
14 Kanban vs. Scrum
16 Introduktion till eXtreme Programming (XP)
16 Historia
18 Introduktion till Lean Startup
18 Historia
20 Design Thinking
21 Business Agility
22 Översikt av Lean, Agile, Scrum, Kanban och XP ihop
24 Fördelarna med Agila metoder
25 Detta får ett mjukvaruprojekt att lyckas
26 Nyttan med Agila metoder
27 Hur påverkas organisationen när vi inför Lean, Agile, Scrum och XP?
27 Förändring för organisationen
28 Affärssidans involvering i projektet
29 Smärta på nya ställen i organisationen på grund av effektivisering
30 Transparens triggar diskussioner
30 Motivera, prioritera, fokusera, leverera
31 När bör Agila metoder inte användas
32 Agila huset
32 Agil strategi - pelarna
33 Agila målet - taket
33 Agila kärnvärderingar - grunden
34 Kopplingen mellan Lean och Agile
35 Cynefin
36 Agile – Gemensamt för Lean, Scrum och XP
37 Prioritera och fokusera – för att maximera Return On Investment, ROI
39 Transparens – för att synliggöra problem och bygga förtroende
39 Hög kvalitet – för att fel är kostsamma
40 Iterativ och inkrementell utveckling möjliggör tidiga intäkter
41 Få feedback tidigt för att lära och förbättra
50 Lean för mjukvaruutveckling
51 Principer för Lean
59 Några verktyg från Lean
62 Effektivt flöde

63 Låt flaskhalsarna bestämma tempot
65 Sammanfattning av strategin
66 Minska ledtiden
67 Value stream mapping – ett verktyg för att minska ledtiden
68 Kanban (fördjupning)
68 Receptet för succé
74 Scrum (fördjupning)
75 Kort repetition av Scrum
75 Ansvarsområden (tidigare kallat för roller)
76 Produktägare (Product Owner)
76 Utvecklarna
79 Scrum-mästaren
79 Produktbacklog
84 Scrumprocessen
85 Starta sprinten
86 Sprintens gång
88 Sprintavslutning – tid för reflektion och anpassning (inspect and adapt)
90 Kontrakt med fast pris
91 Distribuerat Scrum - till exempel för off shoring
94 Scrum för stora företag
94 Gemensamt för LeSS, SAFe och Scrum@Scale
94 LeSS (Large Scaled Scrum)
97 LeSS Hugh
98 SAFe
101 Scrum@Scale
103 SAFe, LeSS eller Scrum@Scale eller något eget?
104 Scrum Pattern
104 Bakgrund
105 Välbefinnandeindex (Happiness metric)
105 Svärma (Swarm)
108 XP – eXtreme Programming
111 Fördelarna med eXtreme Programming
112 Ett par avslutande ord
113 Litteraturlista

Förord

Förord av Henrik Kniberg, författare till böckerna ”Scrum and XP from the Trenches”, ”Kanban and Scrum - making the most of both” och ”Lean from the Trenches”

Äntligen, en bok som förklarar hur dessa viktiga begrepp hänger ihop! En bok som inte enbart riktar sig till utvecklingsteamen, utan även de som är utanför. En bok som ger en helhetsbild snarare än att fokusera på en specifik metod som XP. En kort, lättläst bok. Och inte minst – en bok på svenska!

Det är lite överraskande att det finns så få böcker om Lean och Agile på svenska, med tanke på att Sverige anses vara en av de länder som kommit
längst inom området.

Prioritera, Fokusera, Leverera är ett resultat av åratals praktiskt arbete med att hjälpa företag att förbättra sin utvecklingsprocess med utgångspunkt från Lean och Agile. Boken är i praktiken en manual i hur man lyckas med IT projekt. Den visar hur teori hänger ihop med praktik och hur man egentligen gör för att få ut fungerande mjukvara till hög kvalitet inom rimlig tid.

Trevlig läsning!
Henrik Kniberg

Om boken

Vi vill med den här boken visa hur din process kan förbättras genom att använda Scrum, Lean, Agile och XP (eXtreme Programming). Stora delar kommer handla om det som metoderna har gemensamt. Eftersom vi tycker det är jobbigt att i var och varannan mening skriva Scrum, Kanban,

Lean, Agile och XP, kommer vi att skriva Agile när vi talar om gemensamma saker för alla metoderna. Inte helt korrekt, men tillräckligt, vilket du kommer att se.

Med boken kommer du att:

1. Snabbt få en överblick över Lean, Agile, Scrum, Kanban, XP och andra ord som ofta nämns i samma sammanhang.
2. Få reda på hur bra metoderna fungerar i verkliga livet och på vilket sätt du kan ha nytta av dem.
3. Få reda på vad Lean, Agile, Scrum, Kanban och XP har gemensamt, vilka grundläggande principer de bygger på.
4. Få en beskrivning av Lean, Agile, Scrum, Kanban och XP, var och en för sig mer i detalj.
5. Förstå varför Lean, Agile, Scrum, Kanban och XP fungerar.
6. Kunna använda boken att slå i för att snabbt kunna få en överblick över någon del.

Introduktion till Agile

De gemensamma värderingarna för Agile-metoderna är:

Värdera individer och interaktion högre än processer och verktyg
Värdera fungerande programvara högre än omfattande dokumentation
Värdera samarbete med kunden högre än att förhandla om kontrakt
Värdera att anpassa sig till förändringar högre än att följa en uppgjord plan

Även om det finns värde i de saker som står till höger värdesätter Agile-rörelsen sakerna till vänster mer. Till exempel tycker man att processer och verktyg är viktiga men de får aldrig stå i vägen för ett möte mellan individer. Om så skulle vara fallet får helt enkelt processen ändras.

Historia

Kring år 2000 började RUP (Rational Unified Process) vinna popularitet. RUP blev en tung process med flera tusen sidor beskrivning och förslag på många roller. Samtidigt förespråkades på olika håll en mer lättviktig process. För att kunna utgöra ett attraktivt alternativ till tungrodda processmodeller, bland annat RUP, behövde anhängarna av de lättviktiga processerna ena sig.

I februari 2001 bjöd Robert C. Martin in ledarna för några populära lättviktsprocesser till ett möte. Sjutton personer anslöt och träffades på Snowbird ski resort i Wasatch-bergen i Utah för att åka skidor, prata och försöka hitta gemensamma nämnare. Trots olika bakgrunder kunde deltagarna enas om fyra gemensamma värderingar och tolv principer. Resultatet publicerades som “The Agile Manifesto” (http://agilemanifesto.org/).

Agile är de gemensamma värderingar som ligger till grund för olika metoder att utveckla mjukvara. Metoderna har namn som Scrum, eXtreme Programming, FDD, DSDM och Crystal Clear.

Agile är ett svåröversatt ord men betyder bland annat snabb, flexibel, lättrörlig och vig.

För att förtydliga de fyra värdegrunderna innehåller Agile tolv principer. Vi kommer i kapitlet om Agile att djupdyka i några av dem.

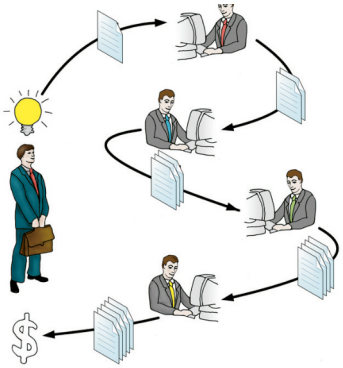
Här följer nio punkter som sammanfattar vad vi tycker karaktäriserar ett Agilt arbetssätt:

- **Prioritera och fokusera** – för att maximera Return On Investment (ROI).
- **Transparens** – för att synliggöra problem och bygga förtroende.
- **Iterativ och inkrementell utveckling** – för att lära, få feedback och minska risken att vi lägger tid på saker som senare visar sig ha lågt värde.
- **Samarbete** – för att minska missförstånden och klyftorna och säkerställa de affärsmässiga målen.
- **Uppmuntra och välkomna förändring** – eftersom verkligheten ändrar sig.
- **Enkla verktyg** – för att kunna anpassa processen efter verksamheten.
- **Målstyrning/decentralisering** – för att kunna arbeta effektivt.
- **Ständig förbättring** – för att bli effektivare.
- **Hög kvalitet** – för att fel är kostsamma.

En mycket förenklad beskrivning av ett Agila projekt skulle kunna vara: Sätt den betalande kunden (eller representant för denna) i samma rum som tre till nio utvecklare som tillsammans har den samlade kompetens som behövs för att ta en idé och omvandla den till en färdig produkt. Fyll rummet med whiteboards, datorer och annat de behöver. Se till att de levererar fungerande, demonstrerbar mjukvara regelbundet med några få veckors mellanrum. Säkra återkoppling från slutanvändarna. Ge dem information om problemet och kontexten men låt dem själva komma på lösningarna. Låt ingenting störa dem.

Introduktion till Lean

En av Leans principer handlar om att minimera tiden från att vi har en idé/ beställning tills vi har producerat det som någon är beredd att betala för. "Concept to cash" eller "Time to market".



Historia

Begreppet Lean Software Development är hämtat från en bok med samma namn av Mary och Tom Poppendieck. Där överförde de Toyotas produktionssystem och Leans principer till mjukvaruutveckling.

Toyotas produktionssystem har sin vagga i familjen Toyodas väveri under första halvan av 1900-talet och i en patenterad uppfinning att med automatik stoppa vävstolen om en tråd brast. Patentet såldes och för pengarna valde Toyoda att starta en bilfabrik, Toyota. Att konkurrera med den högeffektiva bilindustrin i Chicago blev en utmaning, men efter studier hittades en brist i processen som japanerna kunde dra fördel av. Den amerikanska industrin var optimerad så att varje maskin gick för högtryck, vilket medförde att stora lager byggdes för att säkerställa materialtillgången. Toyotas möjlighet var att optimera på helheten och ledtiden, det vill säga den tid det tar från att en bil börjar byggas till dess att den är klar att säljas. Lagren minimerades, vilket tog bort bl.a. lagerkostnaderna.

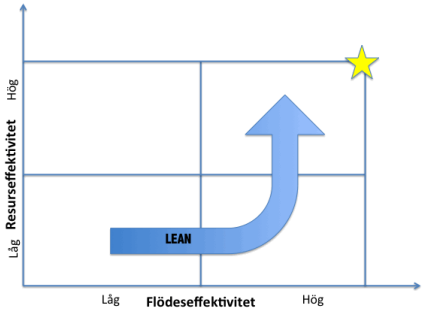
Lean handlar om att utifrån 14 värderingar och principer ta fram bästa möjliga processen som i sin tur leder fram till den bästa produkten.

Kort sammanfattat handlar principerna om att optimera mer på flödet (just-in-time) än på resursutnyttjande och att ständigt förbättra processen genom att se helheten (Jidoka). ”Jidoka handlar om att skapa en organisation som är så synlig och visuell att om något hämmar, hindrar eller stör flödet identifieras det direkt.” *

Ett exempel är de olika momenten som krävs för att komma ombord på ett flygplan. Typiska resurser som passeras är incheckning, baggageinlämning, säkerhetskontroll, passkontroll, biljettkontroll och till sist entrén till planet. Varje steg är oftast maximerat utifrån att just det steget ska vara så effektivt som möjligt. Men tänk om hela processen skulle optimeras på flöde? Då kanske inchekning, baggagehantering, säkerhetskontroll och passkontroll sker på höghastighetståget ut till flygplatsen. Väl framme går passageraren med det redan kontrollerade handbagaget direkt till gaten där biljetten kontrolleras. Då skulle tiden på flygplatsen minska från 1 timme till kanske 15 minuter.

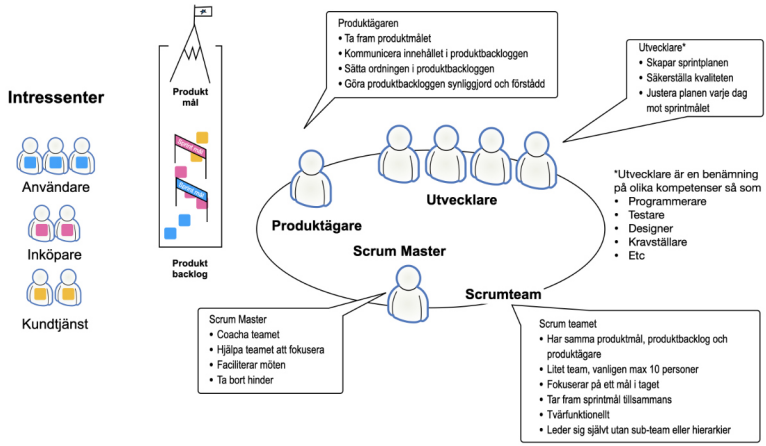
Nästa steg skulle kunna vara att minska tiden ytterligare genom att använda fler ingångar till planet och därmed förkorta köerna. Alla som gått igenom de olika stegen för att komma med ett plan har säkert en mängd idéer om hur flödet kan förbättras. Det är just det som är Lean, att ständiga förbättra flödet.

När väl flödet är bra (vad nu “bra” innebär) är det dags att även optimera resursutnyttjandet samtidigt som flödeseffektiviteten förblir minst densamma eller helst ännu bättre.



*) Utdrag från ”Detta är Lean” av Pär Åhlström och Niklas Modig

Introduktion till Scrum



Ett scrumteam består av en produktägare, en scrum master och flera utvecklare. Produktägaren sätter visionen för produkten, rangordnar listan över det som ska göras (product backlog) och stämmer av med omvärlden (stakeholders). Utvecklare bygger produkten och i möjligaste mån talar om hur lång tid det tar. Scrum mastern alla inblandade alla att förstå processen och hjälper teamet att bli effektivare.

Historia

Under 80- och 90-talet sökte teknikchefen Jeff Sutherland efter mönstret för hyperproduktiva team. Han besökte olika team och försökte finna gemensamma mönster bland de team som var effektivare än andra. Han hittade ett mönster som blev grunden till Scrum. Ordet Scrum kommer från en artikel av Hirotaka Takeuchi och Ikujiro Nonaka i Harward Business Review 1986. Där beskrev de hur modern utveckling mer liknar rugby än ett stafettlopp, vilket vattenfallsmodellen1 kan liknas vid. Jeff tyckte om liknelsen med rugby och hämtade namnet på sin metodik från en rugby-term (Scrum). Jeff tog hjälp av Ken Schwaber för att formalisera Scrum och tillsammans presenterade de processen 1995.

1. Vattenfallsmodellen togs fram av Winston W. Royce 1970 och delar upp utvecklingsprocessen i isolerade steg. Tanken är att varje steg ska vara helt klart innan man går vidare i nästa steg.

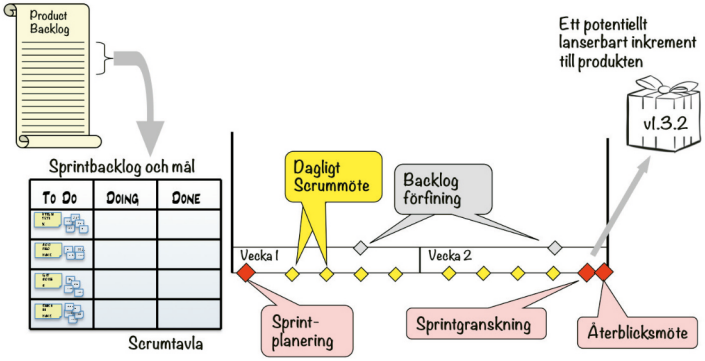
Scrum är det mest kända och använda Agila processramverket (d.v.s. utgångspunkten för att skapa en, för kontexten skräddarsydd, process).

Scrum gör det väldigt tydligt vem som prioriterar och bestämmer vad som ska göras (Produktägaren), vilka som tidsuppskattar och bestämmer hur det ska göras (utvecklarna) samt vem som driver processen framåt (Scrummästaren). Produktägaren samlar det som ska göras i en prioriterad att-göra-lista (Produktbacklog).

Scrum bedriver allt arbete i cykler som kallas sprintar. En sprint är en fast, på förhand bestämd, tidsperiod på vanligtvis en till fyra veckor. Under dessa veckor är två parallella processer igång.

Den ena processen fokuserar på att vidareutveckla produkten. Den börjar med ett sprintplaneringsmöte där man kommer överens om ett sprintmål och planerar vilka funktioner som ska utvecklas under sprinten och hur man lägger upp arbetet för att nå dit. Varje dag justerar man planen för att nå sprintmålet. Sprinten slutar med ett granskningsmöte där scrumteamet presenterar arbetsresultaten för intressenterna varpå framtida stegen mot produktmålet diskuteras.

Den andra processen handlar om hur man förbättrar sitt sätt att arbeta. Utgångspunkten för denna process är de regelbundna återblicksmöten (Retrospective) man har i slutet på varje sprint. Scrumteamet granskar senaste Sprinten med avseende på medarbetare, samverkan, processer och verktyg, Scrumteamet identifierar de mest betydelsefulla förändringarna för att öka sin effektivitet. Förbättringarna följs upp kontinuerligt under nästkommande sprint.



Introduktion till Kanban

Historia

David J Anderson utvecklade Kanban för kunskapsarbete/ mjukvaruutveckling 2005. Han kombinerade läror från Edwards Deming, Eliyaho Goldratt, Peter Drucker och Taiichi Ohno. Kanban inkluderar koncept så som köteori, flödesoptimering och system där alla tar på sig jobb istället för att bli tilldelad jobb (pull system). Under flera år jobbade han med att sammanfatta metoden till en antal principer och tillvägagångssätt. De har sedan evolverat fram i samarbete med den bredare skaran av anhängare till Kanban för att komma fram till den Kanban-metod som vi beskriver här.

Kanban vs. Scrum

Eftersom Scrum beskriver hur arbetsuppgifter löper genom företaget medan Kanban beskriver en förbättringsprocess, finns egentligen inte någon anledning att välja Scrum eller Kanban utan de fungerar bra tillsammans.

Många ser dock på Kanban som Scrum fast utan sprintar. Man låter då jobbet löpa genom systemet i ett ständigt flöde, så som tillverkningen av bilar på Toyota, till skillnad från Scrum där jobbet tas in i omgångar (eng: batches) och betas av i iterationer. Det finns dock inget i Kanban som förbjuder att jobba i iterationer men inte heller något som förordar det.

Kanban är inte en färdig arbetsprocess som beskriver hur arbetet ska utföras. Det är istället en beskrivning av hur förbättringsarbete kan genomföras på ett effektivt sätt. Förhoppningen är att Kanban ska leda fram till en process som ger hög effektivitet i varje enskild kontext.

Kanban består av ett par grundläggande principer:

1. Utgå från ditt nuvarande arbetssätt.
2. Kom överens om att göra evolutionära förändringar.
3. Uppmuntra alla att ta en ledarroll i förändringsarbetet.
4. Förstå och fokusera på kundernas behov och förväntningar.
5. Managera arbetsuppgifterna och låt personerna självorganisera sig för att göra jobbet.
6. Låt policy:er hantera leveransen av tjänster och produkter.

Tillsammans med principerna finns också sex tillvägagångssätt (eng. practices)

1. Visualisera alla arbetsuppgifter (process och aktuell status).
2. Begränsa hur mycket som görs parallellt.
3. Mät och förbättra flödet.
4. Tydliggör de policy-beslut som styr processen.
5. Inför feedbackloopar.
6. Förbättra tillsammans, evolvera genom att experimentera (använd modeller och vetenskapliga metoder)

Källa: LeanKanban University https://kanban.university/#aboutkanban

Introduktion till eXtreme Programming (XP)

Historia

Kent Beck skrev en bok med titeln ”eXtreme Programming explained” 1999 där han utgick från en metod som han tillsammans med några andra tagit fram i ett projekt för Chrysler i mitten av 90-talet. Denna bok ligger till grund för metoden.

Den från början mest kända Agila metoden heter eXtreme Programming, och brukar förkortas XP. Den fick tidigt stöd tack vare konkreta tips om hur programmering kunde förbättras, sett som hantverk, och hur man effektivt arbetar tillsammans i team. Dessutom såg många XP som en välbehövlig protest mot de tunga processer, exempelvis RUP, som påstod sig vara iterativa men som oftast havererade till smärtsamma vattenfallsprocesser.

Tillspetsat kan man säga att den utgår från bra saker och drar dem till sin extrem. Exempelvis:

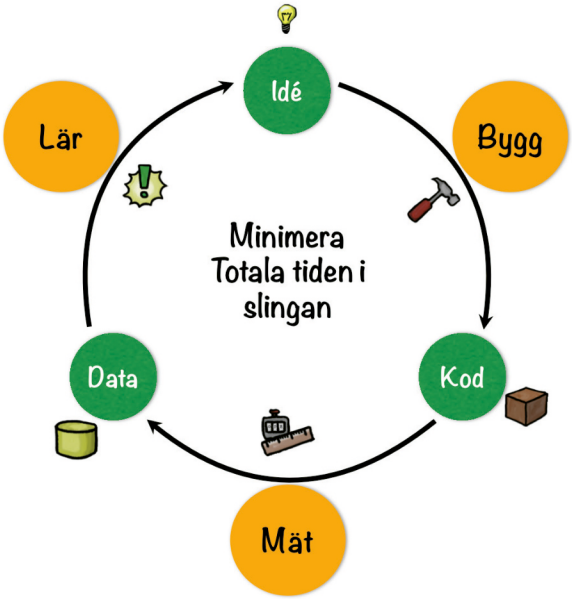
- Eftersom kodgranskning är bra låt oss göra det hela tiden. Det kallas par-programmering.
- Eftersom automatiserade tester är bra, låt oss skriva dem först och designa för testbarhet. Det kallas Test Driven Development.
- Eftersom det är bra om fler än den ursprungliga programmeraren förstår koden, låt oss ha gemensamt ansvar för den. Det kallas Collective Code Ownership.
- Och så vidare.

Till skillnad från Scrum talar XP om hur arbetsflödet inom ett team bör se ut. Detta arbetsflöde har ett starkt fokus på kvalitet och samarbete kring kod. Mer om hur och varför detta fungerar i kapitlet om eXtreme Programming.

Introduktion till Lean Startup

Historia

Eric Ries kom 2011 ut med boken ”The Lean Startup: How Today’s Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses”. I den beskriver han sin erfarenhet av kommersiellt mer eller mindre lyckade projekt och vad som behövs för att lyckas.



Du har en idé om en tjänst eller produkt. Hur kan du snabbast och enklast verifiera att någon vill använda den? Det är vad Lean Start-up handlar om.

Den bärande tanken i Lean Startup är att i en snabbt föränderlig värld kan vi inte planera fram en ny tjänst, utan måste via experiment och feedback från verkliga användare verifiera att den är värd att ta fram.

Metoden har följande steg:

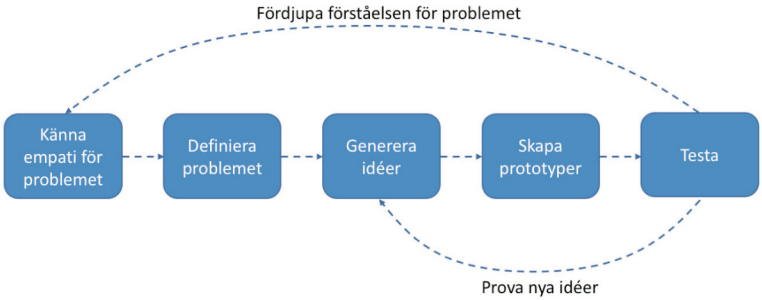
Steg	Exempel
Idé – Formulera hypotes(er)	Grannar har behov av att låna av varandra. De saknar forum för att låna.
Bygg – Det minsta du kan tänka dig för att validera dina hypotes(er)*	Sätt upp två listor i din trappuppgång: “Låna”, och “Låna ut”, skriv upp dig själv med några prylar under låna ut.
Mät – Verkligt användande	Antal som skriver upp sig under en helg. Antal som kontaktar dig under en helg för att låna.
Lär – Utifrån insamlad data	Håller idén? Ska vi göra ett nytt försök i större skala? Eller måste vi omformulera hypoteserna och göra ett nytt experiment?

De olika stegen brukar illustreras med bilden till vänster. Trots ordet startup i namnet är detta en metod som lämpar sig så fort du har en idé som du är osäker på ifall marknaden och användarna vill ha eller ej. Det vill säga tämligen ofta. Det kan vara för en ny produkt eller för en ny funktion.

* I sammanhanget brukar man prata om MVP:er = Minimal Viable Product = Det minsta man kan bygga för att validera sin hypotes.

Design Thinking

Design thinking är, liksom tankarna bakom Lean Startup, en form av problemlösning som passar för komplexa problem (se avsnittet om Cynefin-modellen) där bästa vägen fram är att iterativt testa en prototyp och studera resultat. Vi kommer nedan beskriva de fem steg som tagits fram av Hasso-Plattner Institute of Design at Stanford.



1 Känna empati för problemet (eng. Empathise)

Det första steget i design thinking-processen är att få en djup personlig förståelse av problemet du försöker lösa. Det kan göras genom att observera eller engagera sig med berörda människor för att förstå deras motivation och erfarenheter. Målet är att utveckla bästa möjliga förståelse för användarna, deras behov och de problem som ligger till grund för utvecklingen av den specifika produkten

2 Definiera problemet (eng. Define)

Det är här du kommer att analysera dina observationer och sammanställa dem för att definiera de kärnproblem som du och ditt team har identifierat. Försök definiera problemet ur ett människocentrerat perspektiv. Istället för att säga att ”vi behöver öka vår försäljning” säg att ”vi behöver får fler att köpa våra produkter”.

3 Generera idéer (eng. Ideate)

När problemet är väl definierat är det dags att vara kreativ och komma på möjliga lösningar. Här finns det många tekniker så som Brain Storming och Brainwriting. Det är viktigt att få så många idéer eller problemlösningar som möjligt i början av idégenereringsfasen. Du bör välja några andra tekniker i slutet av fasen, för att hjälpa dig undersöka och testa dina idéer så att du kan hitta det bästa sättet att antingen lösa ett problem eller tillhandahålla de element som krävs för att kringgå det.

4 Skapa prototyper (eng. Prototype)

Nästa steg är att konstruera ”billiga” versioner av produkten eller tjänsten och testa dem på en vänlig grupp. Det kan vara inom företaget men helst hos vänligt inställda personer utanför. Fasen innebär mycket experimenterande och har som syfte att bättre förstå produktens begränsningar, användarnas reaktioner och beteenden.

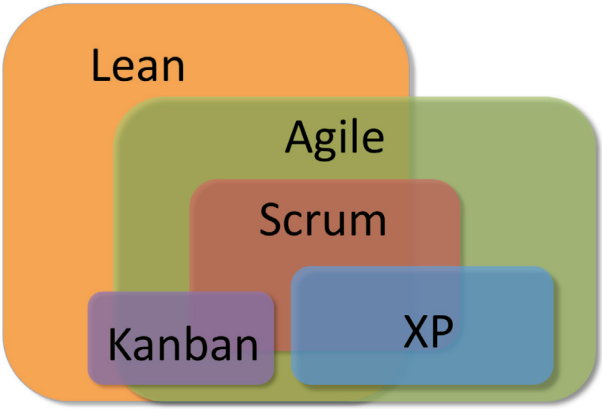
5 Testa (eng. Test)

Nu är det dags att testa produkten eller tjänsten för riktiga användare. Förslagsvis görs utrullningen i små steg för att minska skadan av ett misslyckande. Här gäller det att vara vaksam, lyssna på feedback och snabbt reagera samt korrigera när ny kunskap inhämtas. Utifrån studierna är det möjligt att vi inser att vi antingen behöver ändra på prototypen, behöver prova andra idéer eller omdefiniera problemet.

Business Agility

Business Agility är ett begrepp för företag där hela organisationen anammar en Agil kultur för att snabbt kunna reagera på möjligheter och hot. Det innebär att alla utgår från ett kundperspektiv och tar på sig att utforska och genomföra möjligheter till förbättring av både produkt, tjänst och process. En lärande kultur där misslyckanden är en oundviklig del av framgången. Organisera bygger på att alla som behövs för att uppfylla kundens behov sitter nära varandra oberoende kompetenser. Ofta innebär det tvärfunktionella team där kompetenser så som sälj, marknadsföring, ekonomi, kundservice, juridik, produktutveckling och IT samlas.

Översikt av Lean, Agile, Scrum, Kanban och XP ihop



Lean är ett synsätt eller en kultur som ska genomsyra allt arbete. Fokus ligger på att så resurseffektivt som möjligt uppnå maximalt kundvärde genom att ständigt ifrågasätta varje steg i processen.

Agile är en samling värderingar där förmåga till anpassning och träffsäkerhet är främsta ledorden. Det kräver i sin tur ett nära samarbete.

Scrum är en processramverk som på ett enkelt sätt lägger grunden för att kunna jobba lättrörligt och odla kulturen av Agile.

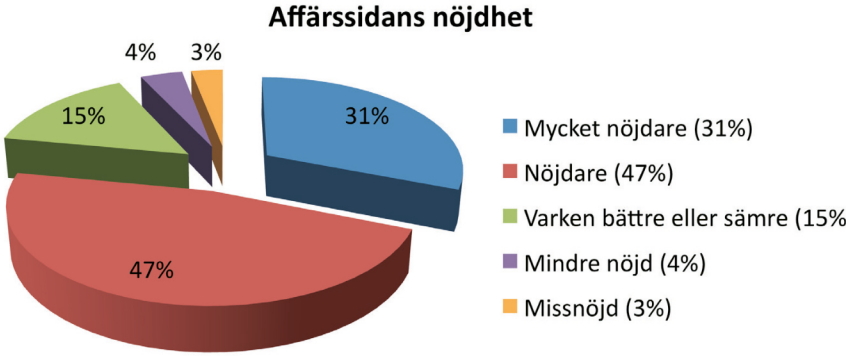
Kanban är ett par principer och tillvägagångssätt som hjälper användarna att ständigt förbättra sitt flöde.

XP är en utvecklingsmetod och en samling tekniker som får ett mjukvaruteam att jobba effektivt ihop för att producera kod med hög kvalitet.

Gemensamt för Lean, Agile, Scrum, Kanban och XP är att alla är framtagna utifrån erfarenheter snarare än teoretiska skrivbordsprodukter, genom att studera vad som är typiskt för grupper som fungerar bra och de som fungerar mindre bra.

Metoderna går utmärkt att kombinera (med visst överlapp). Till exempel kan man analysera hela orderkedjan med hjälp av Lean och optimera flödet med hjälp av Kanban, använda Scrumteam för själva utvecklingen, och inom dessa team använda eXtreme Programming för att leverera med hög kvalitet.

Fördelarna med Agila metoder



Tidningen Dr. Dobb's Journal frågade affärsutvecklare ifall de var mer eller mindre nöjda med Agila metoder jämfört med den metod de använde innan införandet. 78% svarade att de var mer nöjda, medan 7% hellre föredrog den tidigare metoden. 15% tycker de var lika bra.

Detta får ett mjukvaruprojekt att lyckas

Enligt Standish Group var andelen lyckade mjukvaruprojekt 1994 15%. Tio år senare är siffran 34%, vilket är en enorm förbättring men fortfarande uselt. Förbättringen mellan mätningarna förklarades i undersökningen med att projekten har blivit mindre och att man i större grad använder sig av iterativa Agila processer istället för vattenfallsmodellen[2].

Enligt Alan MacCormack (professor på Harvard Business School)[3] karaktäriseras ett lyckat projekt av följande:

- Tidig release så kunden kan utvärdera och ge feedback.
- Dagliga byggen för att få snabb feedback från integrationstester.
- Ett team och/eller en ledare som har instinkt och erfarenhet att fatta rätt beslut.
- En modulär arkitektur som underlättar förändringar.
- Beslut tas så sent som möjligt eftersom det då finns mest information och kunskap. Det är bättre än att gissa allt från början.

En liknande men något annorlunda lista får man från Standish Group:

1. Involvering av användare av systemet.
2. Stöd hos högsta ledningen för projektet.
3. Tydliga affärsmål.
4. Klar och tydlig omfattning av projektet.
5. Agila processer.

2. http://www.softwaremag.com/L.cfm?Doc=newsletter/2004-01-15/Standish
3. Alan MacCormack, "Product-Development Practices That Work: How Internet Companies Build Software" MIT Sloan Management Review, Winter 2001, Vol. 40 number 2

Enligt undersökningen Agile Adoption Rate Survey4 lyckas 83% av de Agila projekten om teamet är samlat på en och samma geografisk plats. De främsta fördelarna man ser med Agila metoder är:

- Ökad produktivitet
- Snabbare time to market
- Färre defekter
- Lägre kostnad
- Större flexibilitet
- Närmre samarbete med kunden

Nyttan med Agila metoder

Time to market: Samtliga metoder vi beskriver i boken hjälper dig att korta ned tiden från att någon i din organisation eller hos din kund vet vad de vill ha, tills de får det. Ofta kan kunden få en första version demonstrerad för sig redan efter några veckor.

Nöjd personal: En Agil metod betonar vikten av samarbete och engagemang och har som mål att personalen ska kunna ta egna beslut. Det handlar om en fungerande målstyrning istället för detaljstyrning.

Effektivitet: Agila metoder gör att du kan fokusera på att utveckla rätt saker i rätt tid. Stenhård fokusering och prioritering. I ett genomsnittligt icke Agilt projekt är 80% av den funktionalitet som utvecklas inte efterfrågat av vare sig kunden eller användarna!

Flexibilitet: Agila metoder uppmuntrar förändring och räknar med att omvärlden förändras. Det är ingen som helst motsättning i att ena veckan vilja ha en sak utvecklad, och att några veckor senare vilja ha en helt annan sak utvecklad, eftersom du insett att du då kan hinna före en konkurrent. Det är bara att ändra sig. Inga halvårsplaner, utan istället omedelbar respons. Men även processen själv ska förändras fortlöpande, vilket görs genom ständiga förbättringar och anpassningar till just din organisation.

4. http://www.ambysoft.com/surveys/agileFebruary2008.html

Hur påverkas organisationen när vi inför Lean, Agile, Scrum och XP?

Alla verktyg kan användas fel.

Att införa en ny process innebär och kräver större eller mindre förändringar. Här ska vi belysa några förändringar vi ofta ser hos företag vi coachar.

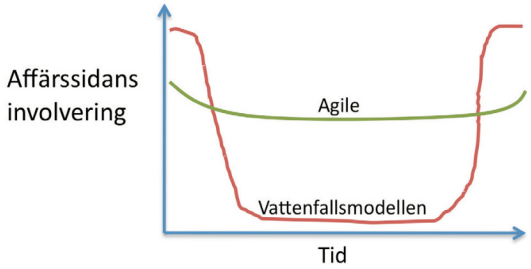
Förändring för organisationen

De personer som sitter på roller som inte längre finns kvar har naturligtvis svårt att direkt se positivt på förändringarna. Liksom alltid för organisationsförändringar gäller för chefer att säkerställa att alla känner sig informerade och involverade samt att de har möjlighet att få kontroll över sin vardag. Förändringen påverkar även chefer. Agilt tänkande innebär att ansvar och befogenheter delegeras nedåt, närmare där jobbet utför. De som har stort behov av kontroll kan känna att de tappar kontroll.

De behöver fråga sig själva vilken information de minst behöver för att göra sina uppdragsgivare nöjda och fokusera på den och inga andra. En chef i ett Agilt företag behöver hjälpa till att ta bort hinder för medarbetarna och främja samarbete mellan olika grupper inom företaget. Det kan även innebära att via sina kontakter se till att medarbetarna får åtkomst till rätt personer även om de är svåra att få tag på.

Affärssidans involvering i projektet

Beställarorganisationer som är vana vid en process som är uppdelad i små faser kan bli förvånade över en Agil process. Tidigare har de endast haft en nära kontakt med leverantören under faserna för kravinsamling och test. Nu blir de involverade under hela projektet. De måste avsätta tid mer

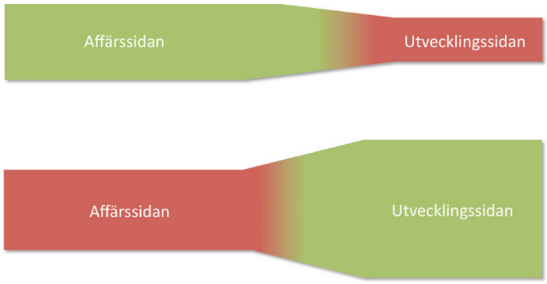


I traditionella vattenfallsprojekt är affärssidan och kunden involverade i början för att specificera vad kunden vill ha. Därefter involveras kunden igen i slutet av projekten när de testar leveransen för att avgöra om de fått det de förväntade sig. I Agila projekt blir det en jämnare involvering, lite hela tiden, men också mindre risk för överraskningar på slutet.

De måste vara beredda på att de första releaserna inte är den färdiga produkten. Har de bara förståelse för Agila processer ser de fördelarna med att tidigt vara involverad och kunna påverka produktens utformning.

Smärta på nya ställen i organisationen på grund av effektivisering

I Agila projekt blir mer saker synliga. Bland annat synliggörs flaskhalsar i
organisationen vilket i sig kan vara jobbigt för en del. Förutom att flaskhalsar bli synliga, har de dessutom en tendens att dyka upp på nya ställen där man inte är van att se dem. En kund till oss sade, när vi frågade honom vilka skillnader det var före och efter att vi introducerade Scrum, att ”vi har svårt att mata utvecklingsavdelningen med tillräckligt att göra”.
Anledningen var dels att utvecklingsavdelningen jobbade effektivare och dels för att affärssidan var mer involverad vilket naturligtvis tar resurser i anspråk.



Genom att vissa delar av organisationen blir effektivare flyttas flaskhalsen (och smärtan) till nya delar som tidigare inte varit en flaskhals.

Transparens triggar diskussioner

Agile sätter högt värde på transparens, det vill säga att synliggöra och gärna visualisera vad som ska utvecklas, vad som utvecklas just nu och när det beräknas bli klart. När information är lätt tillgänglig uppstår diskussioner. Så länge det uppstår hälsosamma diskussioner är allt gott och väl, men det händer att någon är obekväm med den information som blir synlig av olika anledningar, vilket motverkar viljan till synlighet. När allt
fungerar som det ska föder transparens tillit som göder mer transparens vilket blir en god spiral.

Det är svårt att kräva information. Information är något mottagaren måste göra sig förtjänt av. Den som får dåliga nyheter kan istället för att skälla ut budbäraren vara glad att informationen kom nu och inte senare. Förhoppningsvis kan problemen lösas i god tid för att minimera skadan.

Motivera, prioritera, fokusera, leverera

Effektivitet kommer utifrån att människor mår bra och får jobba ostört. Det som motverkar effektiviteten brukar på engelska kallas ”thrashing”, till exempel när man har så mycket att göra att man inte får något gjort. Några vanliga orsaker till thrashing är:

- att inte kunna hålla prioriteringen för de saker som påbörjats
- att ständigt störa personer med sidopuckar
- att sätta mål som måste uppnås, utan att medarbetarna fått påverka
- att trycka på och stressa människor att jobba hårdare
- att långvarigt jobba övertid
- att inte ta sig tid att reflektera och förbättra

Thrashing gör att effektiviteten minskar och att kvaliteten blir sämre.

När bör Agila metoder inte användas

Det finns naturligtvis situationer då det kan vara svårt eller till och med olämpligt att inför Agila metoder. Några exempel är:

- Om acceptansen att förändra organisation är så låg att det är omöjligt att införa någon ny metod.
- Om man har en organisation där det inte finns en önskan om samarbete mellan individer eller mellan avdelningar.
- Om nyckelpersoner i en grupp föredrar att jobba individuellt istället för tillsammans.
- Om uppgiftens art eller dess genomförande inte ger något utrymme för lärande och omvärdering.

Agile innebär inte bara att personer inom samma avdelning samarbetar utan också att det är ett gott samarbete mellan avdelningar.

Det måste finnas en vilja att dela med sig, både av kunskap och av information. Företag som tycker att ”need to know-basis”[5] är lagom kommer ha svårt att få Agile att fungera.

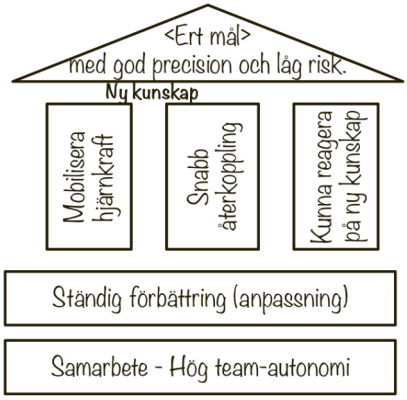
Till sist måste alla lita på varandra och tro att var och en gör så gott den kan. De som bara vill ha högpresterare och avskedar de andra riskerar att få en organisation som drivs av rädsla. För att uppnå ett bra resultat behövs experimenterande, och då måste det också vara tillåtet att misslyckas.

5. Need to know-basis är ett management-uttryck som innebär att medarbetaren bara får veta det han eller hon absolut behöver veta för att genomföra sitt dagliga arbete.

Agila huset

Agile är en kultur som definieras av Agila manifestets 4 värderingar och 12 principer. För de allra flesta är det svårt att, utifrån manifestet, förstå vad Agilt verkligen är. Det blir då lätt diffusa förklaringar som att vara flexibel och jobba effektivt. Den Agila kulturen är så mycket mer än så och därför kan det vara bra att ta fram en enklare modell så att det finns något konkret att hålla sig till när Agilt ska förklaras, implementeras och förfinas. En sådan modell är Agila huset. Det är fritt upp till var och en att göra sin modell men fördelen med det Agila huset är att den förklarar liknelsen med både Lean och Cynefin-modellen. Mer om den kopplingen senare.

Det Agila huset består, precis som Lean-huset, av tre delar: grunden (kärnvärdena), pelarna (strategierna) och taket (målet).



Det Agila huset har tagits fram genom att fokusera på den önskade effekten av att följa Agils värderingar och principer.

Agil strategi - pelarna

Mobilisera hjärnkraft – För att hantera den situation med hård konkurrens och snabbt föränderliga kundbehov som många företag idag har, krävs många hjärnor som försöker komma på företagets nästa succé. Det kan både vara en produkt eller en tjänst. Det är omöjligt att på förhand förutse vem som kommer vara framtidens hjälte, så det är bäst att ställa upp med så många som möjligt i loppet. För att lyckas krävs att

så många som möjligt inkluderas i problemlösningen och att de får en bred förståelse för företagets och även branschens utmaningar. Det gör även att de känner mer ansvar för lösningen än om de bara blir ombedda att implementera enligt ett redan framtaget lösningsförslag.

Snabb återkoppling – Även om många hjärnor är inbjudna i problemlösningen vet vi inte om vi hittat rätt innan riktiga kunder fått testa. Därför är det viktigt att lägga upp projektet så att det så snabbt som möjligt går att få svar på om vi hittat rätt eller behöver tänka ut en annan lösning.

Ny kunskap – Genom att ta små steg och sätta så många hjärnor som möjligt på att analysera resultatet genererar vi en massa kunskap om våra kunder och hur de reagerar på våra produkter och tjänster.

Kunna reagera på ny kunskap – För att inte skapa frustration måste den nya kunskapen kunna omsättas i nya eller i alla fall korrigerade lösningsförslag och planer. Det gäller att, om så behövs, kunna svänga på en femöring men utan att det kostar så mycket mer än fem öre. Det här ställer krav både på tekniken men också på vår egen inställning till att ändra riktning.

Agila målet - taket

Eftersom den nya planen, precis som den gamla, är en hypotes, behöver vi återigen optimera för att så snabbt som möjligt få feedback, som vi ska mobilisera hjärnkraft för att analysera. Kopplingen gör att vi får ett system som, beroende på vilka parametrar vi använder när vi utvärderar feedback, kan leda oss till det mål vi satt upp med bra precision och relativt låg risk.

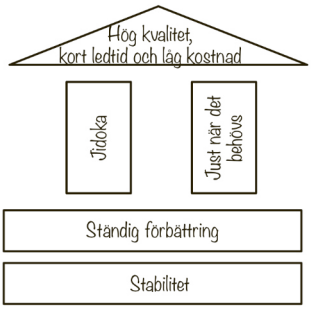
Agila kärnvärderingar - grunden

Samarbete - hög Team-autonomi - Mycket i det Agila manifestet bygger på samarbete men också på självstyrande. För att främja samarbete införs ofta standarder vilka kan gå tvärs emot tanken om självstyret. Det här är en balans företaget måste hitta vilket är speciellt svårt för stora företag. En balans som så mycket annat i vårt sätt att jobba är svår att finna på förhand. Därför är det viktigt med den andra kärnvärderingen: den ständiga förbättringen.

Ständig förbättring (anpassning) - Så länge vi ständigt förbättrar kommer vi förr eller senare bli bättre än konkurrenterna även om de kan tyckas ha ett försprång från början. Det här gör implicit att Agile inte är en plats utan en riktning och implementationen av Agile tar aldrig slut. Agila huset kan vara ett sätt att hjälpa till med den ständiga förbättringen av processen. Kan vi få ännu kortare feedback? Kan vi mobilisera hjärnkraften bättre? Kan vi byta riktning enklare? Osv…

Kopplingen mellan Lean och Agile

Utifrån Leans 14 principer och Agiles 12 principer och 4 värden är det svårt att hitta liknelser. Samtidigt nämns de ofta i samma mening. På många företag sägs det att de satsar på Lean och Agile, inte antingen eller, så var är kopplingen egentligen?



Lean-huset så som mål, pelare och fundament beskrivs av Jeffrey Liker i boken ”TheToyota Way”

Kopplingen är lätt att se om Agila huset ställs bredvid Lean-huset. Låt oss börja med Jidoka. I boken ”Detta är Lean” av Modigh och Åhlström beskrivs att Jidoka handlar om att ”skapa en organisation som är så synlig
och visuell att om något hämmar, hindrar eller stör flödet identifieras det direkt”. Låter som samma tankar som ligger bakom mobilisera hjärnkraft.

Den andra pelaren i Lean-huset är just-in-time vilket av Mary Poppendieck beskrivs som ”Defer Commitment, Deliver fast” dvs. Håll möjligheten att ändra öppen så länge som möjligt och leverera så snabbt det bara går. Låter som reagera på förändring och snabb feedback. Strategierna är alltså desamma.

Vad det gäller kärnvärderingarna så har båda husen med den ständiga förbättringen. Det som skiljer är att Agila huset har balansen mellan samarbete och självstyrande team medan Lean har stabilitet. Just där är kopplingen inte självklar men i övrigt ger liknelsen en god förklaring till

varför Lean och Agile ofta nämns i samma mening.

Cynefin

Dave Snowden har genom sin Cynefin-modell ([kʌn□vɪn]) givit oss ett verktyg för att välja rätt sätt att jobba beroende på karaktären på den situation vi befinner oss i. Han delar upp situationerna i självklar, komplicerad, komplex och kaotisk.

En självklar situation är där det finns ett tydligt samband mellan orsak och verkan och där de allra flesta klarar av att lösa uppgiften. Ett exempel skulle kunna vara att måla en planka.

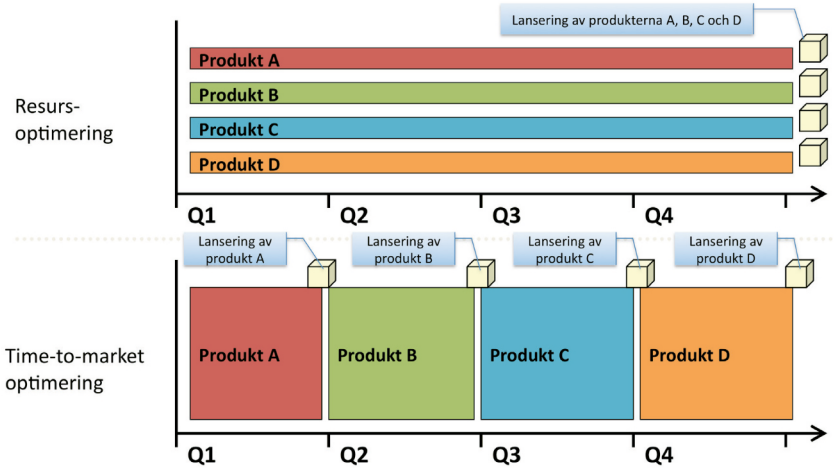
En komplicerad situation är där kopplingen mellan orsak och verkan finns men den är lite svårare att se. Här behövs det en expert som analyserar läget, tar fram ett lösningsförslag och genomför detsamma. Ett exempel skulle kunna vara att måla Golden Gate-bron.

I en komplex situation är sambandet mellan orsak och verkan synlig först i efterhand och bästa sättet att hitta rätt är att testa sig fram. Ett exempel skulle kunna vara att vinna en konstnärstävling. I komplexa situationer rekommenderar Dave Snowden oss att minimera kostnaden för att misslyckas genom att ta små steg, inspektera och anpassa samt maximera lärandet genom att blanda in så många hjärnor som möjligt.

Känns det igen? Just det, det är den Agila strategin enligt Agila huset. Verktyget för att lösa komplexa situationer är alltså Agile medan komplicerade kan hanteras med metoder som t.ex. vattenfallsmetoden som bygger på kontrollstationer (toll gates) och experter som var och en gör sin bit.

En kaotisk situation är det viktigt att snabbt komma ut från eftersom det inte finns någon möjlighet till lärande. Tipset är att så snabbt som möjligt rädda situationen med första möjliga lösning. Snyggt utförd Scrum hanterar komplexa initiativ genom att planera i sprintar som är komplicerade och jobba i vardagen som ska vara självklar.

Agile – Gemensamt för Lean, Scrum och XP



I det övre exemplet (resursoptimerade) jobbar varje person på sitt eget projekt och alla blir klara samtidigt efter Q4. I det nedre jobbar de fyra personerna tillsammans med ett projekt i taget. Projekt A lanseras efter Q1, projekt B efter Q2 och så vidare. Genom att fokusera på ett projekt i taget får man tidigare betalt för det levererade projektet.

Som du kanske minns från inledningen består Agile av fyra grundläggande värderingar och 12 principer. Vi ska nu i detalj förklara de typiska karaktärer, av ett Agilt arbetssätt, vi tycker är viktigast.

Prioritera och fokusera – för att maximera Return On Investment, ROI

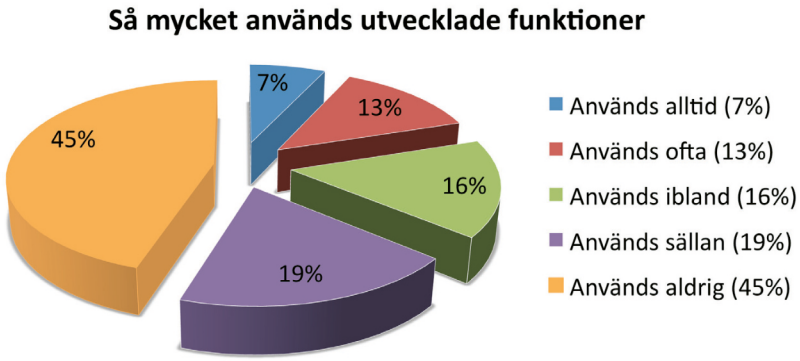
I Agila metoder såsom Scrum och XP används strikt prioritering. Det betyder att det aldrig finns två saker som är lika viktiga. Antingen är A viktigare än B, eller tvärt om. Vi kan naturligtvis ändra oss, och vi bör ändra oss när verkligheten förändras, eller när vi lärt oss något nytt som påverkar prioriteringen. Men vid varje ögonblick ska det alltid vara en tydlig prioritering.

Många gånger har vi stött på utvecklingsgrupper som visar upp en prioriteringslista där allt har högsta prioritet. Eftersom resurser inte är oändliga måste någon bestämma vad som ska göras först. Genom en otydlig prioritering läggs beslutet i händerna på utvecklarna som kanske inte alltid har bäst kunskap om vad som ger mest förtjänst för företaget. Vill företaget ha kontroll på sin ROI måste de också ha kontroll på sin prioritering. Det innebär att det bara finns en sak som är viktigast, en sak som är näst viktigast, och så vidare. Allt annat är bara ett sätt att slippa ta beslut och därmed skjuta över ansvaret till någon annan. Om prioriteringen är oklar riskerar vi att utvecklarna först gör det som är lättbegripligt, enkelt och roligt, inte nödvändigtvis det som är viktigt.

Ett annat vanligt fel är att göra flera saker samtidigt. Många tror att ju fler saker som påbörjas, desto fler saker avslutas. Tyvärr är det så att alla påbörjade projekt kostar pengar medan bara de avslutade projekten genererar pengar. Många samtidiga projekt gör att utvecklare får svårare att fokusera och arbeta effektivt. Dessutom ökar administrationen för exempelvis mellanchefer när listan över pågående projekt och aktiviteter blir lång.

Anledningen att man ändå påbörjar nya aktiviteter innan påbörjade är avslutade är att vi både som individer och som grupp vill vara vår omgivning till lags. Det är lättare att säga ja, vi fixar detta, än att säga nej, vi har redan fullt kapacitetsutnyttjande. Nu behöver man inte säga nej, utan istället, ja, vad ska vi ta bort istället?

Agile säger att det bästa sättet att få en bra ROI är att minimera tiden mellan att projekt kostar pengar tills det genererar pengar. Detta görs bäst genom att lägga mesta möjliga resurserna på de påbörjade projekten så att de så snabbt som möjligt blir klara.



I ett genomsnittligt mjukvaruprojekt används 7% av funktionerna hela tiden, och 13% ofta. 45% används aldrig. De funktioner man borde utveckla är endast 20% av vad som normalt utvecklas. Uppemot 80% borde kanske inte ha utvecklats alls. (Källa: Standish Group)

Genom att prioritera med fokus på kundvärde, och kontinuerligt låta kunden testa, kan många projekt uppnå 80% av kundvärdet med bara 20% av funktionerna. Därmed kan du kapa utvecklingstiden och kostnaderna till en femtedel! Dessutom ökar underhållet med ökad mängd kod — det blir en hävstångseffekt. De 80% som inte behövs kostar mycket mer än 80% av underhållet.

Metoder som definierar alla krav i början av projektet tenderar att få för många krav. Beställaren lägger gärna in extra krav ”just in case” bara för att det senare är dyrt att ändra sig. Eftersom Agila processer gör det billigt att ändra sig bestämmer sig beställaren bara för de viktigaste kraven. Mindre viktiga funktioner fylls på allt eftersom just när utvecklarna behöver veta dem. På så sätt tillkommer endast funktioner som användarna har behov för. Snabb feedback gör att vi tidigt upptäcker missförstånd. Tidig lansering säkerställer att funktionerna fortfarande är aktuella när de lanseras.

Transparens – för att synliggöra problem och bygga förtroende

Har du någonsin hört rykten om att ingen vet vad utvecklingsavdelningen egentligen håller på med? När du vill bringa klarhet i ryktena får du svar som du inte förstår. Vi lägger stor vikt vid att ta fram en transparens som tar bort alla rykten och tydligt visar vad som pågår. Därför pratar vi om kundvärde och funktionalitet kunden upplever istället för tekniska saker som måste göras för att få till funktionerna.

Transparens kan också användas för att belysa problem. Det är svårt att få igenom en förändring bara utifrån en känsla men om du istället kommer på hur problemet blir synligt blir det uppenbart för fler att en förändring behövs. ”In och lys med ficklampan” är ett begrepp som Agila coacher ofta använder.

Hög kvalitet – för att fel är kostsamma

I ett stressat läge är det lätt att tumma på kvaliteten, men det är ofta en kortsiktig vinst som blir dyr att betala tillbaka. Om vi ständigt slarvar bygger vi upp en skuld som tynger oss, och sakta men säkert minskar vår produktivitet tills vi till slut inte får mycket gjort. Något som definitivt minskar våra chanser att svara på marknadens förändringar.

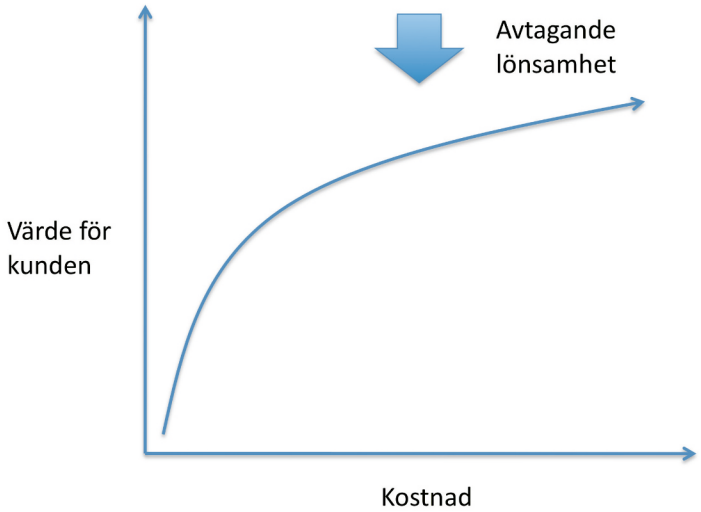
Tyvärr är det inte alltid reversibelt att i första läget strunta i kvalitet och senare försöka ta ikapp förlorad kvalitet. Få personer tycker om att rensa upp det andra stökat till och att skriva snyggt i annars ful kod känns som att kasta pärlor för svin. Om du som utvecklare ser kod som någon annan struntat i, ”ful-snabb-hackat” ihop, ska det mycket till för att du ska förbättra den. Snarare fulhackar du in något själv, och snart har vi ”kodslum”.

När personerna bakom det Agila manifestet pratar om ”fungerande mjukvara” menar de att mjukvaran inte bara ska kunna demonstreras. Den ska även vara av hög kvalitet med städad, lättförståelig och väl testad kod. Först då kan den anses vara ”fungerande”.

”Continuous attention to technical excellence and good design enhances agility.” AGILE MANIFESTO

Iterativ och inkrementell utveckling möjliggör tidiga intäkter

Agile förespråkar att bygga system inkrementellt. Det betyder att systemet växer fram funktion för funktion. Det kan liknas vid att bygga ett småhusområde, hus för hus istället för att gjuta 100 bottenplattor för att sedan sätta upp väggarna på 100 hus för att sedan bygga tak på 100 hus och så vidare. Genom att bygga ett hus åt gången kan boende flytta in så snart ett hus är klart, vilket ger byggherren inkomster tidigt, vilket i sin tur kan finansiera fortsatt byggande. Om en lågkonjunktur kommer under byggandet kan bygget tillfälligt stoppas utan att något större värde går förlorat. De byggda husen har sina boende vilket genererar pengar. Byggherren kan dessutom maximera ROI genom att bygga de mest attraktiva och därmed dyraste och värdefullaste husen först.



Genom att först prioritera funktioner med högst värde till minst kostnad ges tidigt ett högt ROI. Under projektets gång minskar värdet per nedlagd tid och till sist är det inte ekonomiskt motiverbart att fortsätta.

Få feedback tidigt för att lära och förbättra

Traditionella sätt att bygga mjukvara utgår felaktigt från analogin med att bygga ett (1) hus. För att bygga ett bra hus börjar en arkitekt först att rita ett väldigt bra hus. Därefter ger arkitekten ritningarna till en byggherre som anlitar alla som behövs och så gjuter man grunden följd av stommen, taket (fest!), ytterväggarna, innerväggarna etc.

Denna analogi med husbygge har lett oss fel i årtionden. Den har gett oss metoder som säger att vi först ska specificera allt (arkitektens roll), därefter designa allt, följt av programmera allt, för att till sist testa allt och slutligen leverera.

Sällan är resultatet som man hade hoppats på. Varför inte? Jo för att vi oftast bygger en typ av program för allra första gången. Hus har vi byggt i flera tusen år och långsamt förfinat för varje gång. Att ändra saker i mjukvara är billigt jämfört med att ändra saker i hus.

Vi liknar hellre att bygga mjukvara med att man som arkitekt sitter vid sin dator, och när man ritat lite lagom mycket trycker man bara på byggknappen på sin dator, och vips står huset i trädgården, färdigt att gå ut och känna och smaka på.

Om en arkitekt hade ett sådant verktyg, hur skulle hennes process för att bygga då förändras?

Antagligen skulle hon rita några streck, ungefärlig storlek på huset, trycka på bygg-knappen, gå ut i trädgården, kika på huset och tänka ”det blev lagom stort, men det behövs dörrar så att jag kan gå in i huset”. Tillbaka vid datorn, rita in några dörrar, trycka på bygg-knappen, gå ut i trädgården och in i huset, och tänka ”vad mörkt här är, det behövs nog några fönster”.

Detta sätt att arbeta kallas inkrementellt (bygga funktion efter funktion) och iterativt (upprepa och få feedback, för att vi lär oss under gång).

Lagom mycket förändring

Det är inte effektivt att ständigt ändra sig; då riskerar man att inget bli färdigställt. Samtidigt har vi satt ett värde på att kunna ändra oss. Hur får vi det att gå ihop; kan vi både vara snabbt föränderliga och effektiva? Det fungerar om vi jobbar i iterationer där planen för vad som ska göras i en iteration spikas vid iterationens början och förblir oförändrad fram till iterationens slut. Inför nästa iteration kan den övergripande planen

ändras totalt. På så sätt får utvecklarna möjlighet att fokusera och färdigställa funktioner på ett effektivt sätt samtidigt som affärssidan får möjligheten att följa marknadens svängningar. Att jobba lättrörligt betyder alltså inte att beställaren kan ändra sig varje dag, i alla fall inte utan att veta konsekvenserna och kostnaderna av förändringen.

”Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.” AGILE MANIFESTO

Samarbete – för att minska missförstånden och säkerställa de affärsmässiga målen

Ett lyckat projekt är ett projekt som genererar hög kundnytta snarare än ett projekt som levererar exakt det som stod i specifikationen. Det kan vara samma sak, men det viktiga är vad som är huvudmålet. För att lyckas med projektet krävs att systemet när det lanseras är precis det marknaden efterfrågar just då. Den informationen sitter oftast inte i en utvecklares huvud. Utvecklarna har å andra sidan bäst kunskap om vilka möjligheter systemet erbjuder. Vi behöver samarbete under utvecklingens gång så att ändringar i kunders önskemål också förändrar systemets utveckling. Än viktigare är att någon som vet vad kunden efterfrågar ofta får använda systemet medan det utvecklas och snabbt kan korrigera eventuella missuppfattningar. Dessutom kan det tidiga användandet ge idéer som visar sig leda till en ännu bättre produkt. De flesta företag har en strategi med sina produkter och den måste också avspeglas i all utveckling. Bästa sättet att nå kundernas önskemål på kort och lång sikt är just när personer med olika bakgrund och kunskaper samarbetar effektivt med varandra. Dessutom minskar vi risken för vi-och-de-känslor.

”Business people and developers must work together daily throughout the project.” AGILE MANIFESTO

Effektiv kommunikation

Alla har vi någon gång hamnat i en lång mejldiskussion där missförstånd leder till nya missförstånd. Ibland är det helt enkelt bättre att lyfta

telefonluren eller ännu hellre lyfta på rumpan och gå över till personen vi behöver kommunicera med. När det gäller att överföra information har undersökningar visat att två personer vid en whiteboard är det absolut effektivaste sättet. Där kan en diskussion uppstå som består av talade och
skrivna ord blandat med ritningar och kroppsspråk. Den som vill berätta något ser omedelbart om han lyckats förklara och ifall den andre håller med på kroppsspråk och hållning, sådant som försvinner i till exempel en mejlkonversation.

Det som tar fem minuter att förklara ansikte mot ansikte tar oftast timmar att förklara via mejl, efter oändliga turer fram och tillbaka.

Agile strävar efter att använda den mest effektiva kommunikationsformen, 2 personer vid en whiteboard, före mindre effektiva former. (Källa: forskning gjord av McCarthy och Monk, 1994)

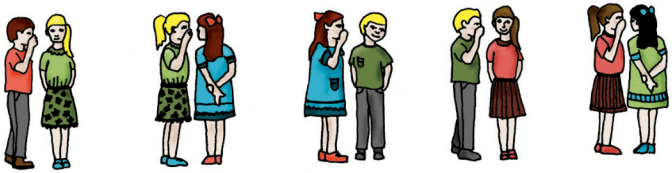
Att skriva bra dokument och kravspecifikationer är en konst få av oss behärskar. Och tyvärr, även om vi är duktiga lider dokument av en mycket besvärande egenskap: de kommunicerar bara i en riktning, utan möjlighet att ställa frågor. Därmed inte sagt att det inte finns fördelar: den här boken är till exempel en envägskommunikation, men det är en kommunikation från oss (två) till många (några tusen). När det bara är frågan om att överföra information mellan ett fåtal individer kostar det skrivna ordet för mycket.

Den vanligaste kommunikationsformen är oftast att en person behöver ge en annan person information här och nu. Mer sällan än vad vi tror behöver informationen sparas för framtiden. När man levererar snabbt och ofta så minskar också behovet av att spara en massa mellanresultat. Därför vill vi inte ha en massa rapporter som skickas omkring utan hellre korta effektiva möten där deltagarna möts ansikte mot ansikte.

”The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.”
AGILE MANIFESTO

Minimera antal informationsled — Viskleken

Har du någon gång lekt viskleken som barn eller med dina barn? Att man viskar ett ord i örat på den första i en kedja av personer, och sen hör man vad meningen blev tillslut?



Redan efter några få led så har vi tappat mycket information.

Låt oss anta att vi lyckas föra över 50% av informationen i ett led. Redan efter två led är vi nere på 25% korrekt information för att efter fem led vara nere på 3%. Hur många led har ni mellan kund och någon som är med och levererar?

Effektivaste kommunikation har man mellan två personer utan mellanled.

Små korta effektiva möten utan mellanled. För att detta inte ska bli för dyrt måste möten vara korta, gärna tidsbegränsade.

De Agila metoderna försöker minska antal mellanled. Dels genom att ha kunden eller kundrepresentanten på plats, så nära utvecklingsteamet som möjligt och så tillgänglig som möjligt för utvecklingsteamet, dels genom att ha några få effektiva tidsbegränsade möten.

Gemensamt språk

Även om vi träffas ofta framför en whiteboard, kan vi missförstå varandra om vi inte pratar samma språk. För att råda bot på det problemet kan utvecklare och affärsmänniskor komma överens om vilka uttryck som ska användas. Det kallas att definiera domänen och görs enklast genom att rita upp ett så kallat domändiagram, en skiss med rutor för substantiv som är relevanta. Mellan rutorna dras streck som anger hur de relaterar till varandra. Orden som bestäms ska genomsyra krav, kod och databasfält, allt för att öka förståelsen för affären och minska missförstånden.

Planera effektivt — planera när planen behövs

Visst är det bra med planer och visst är det härligt med planer som stämmer, men det får aldrig vara ett självändamål att planerna följs. Det är som att bygga om verkligheten för att kartan ska stämma. Dyrt och onödigt. Nu får du inte tolka det som att du ska lägga mer tid på planeringen för att säkerställa att den inte behöver ändras. Tänk istället på att omvärlden förändras och ny kunskap gör att vårt mål med rätta förändras. Alltså kan en tung planering i början vara bortkastad. Ett bättre angreppssätt är att göra en grov övergripande plan för framtiden (road map) och en mer detaljerad plan för den närmaste tiden.

Uppmuntra och välkomna förändring

Om en kund vill göra förändringar kommer kunden inte bli nöjd om du sätter dig i vägen. Se förändringar som något bra! Det innebär att kunden får något bättre än vad han eller hon hade tänkt sig från början. Dessutom är förändringar viktiga för att säkerställa att projektet resulterar i ett modernt och användbart system. Eftersom omvärlden inte står stilla, krävs förändringar under projektets gång. En liknelse är planerandet av en bilresa där stora resurser läggs på att ta fram en detaljerad

vägbeskrivning och milstolpar som avstämningspunkter. Vad är den planen värd om vägen av någon anledning spärras av och vi tvingas hitta en alternativ väg? Då är det bättre att införskaffa en GPS som ger dig information om vägval just när du behöver dem, och snabbt anpassar vägval efter rådande situation. Lean kallar det just-in-time-beslut, Agile och Scrum talar om en föränderlig backlog mellan iterationerna.

”Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.” AGILE MANIFESTO

Enkla verktyg – för att kunna anpassa processen efter verksamheten

En bra process är en process som fokuserar på att ständigt förbättra sig. Då får inte verktygen komma i vägen. Vi kan inte vänta med förbättringen tills verktygsföretaget kommer med en uppdaterad version och hoppas på att de nya funktionerna stödjer den förbättring vi önskar göra. Handskrivna post-it-lappar på en whiteboard kan ändras som vi önskar och kommer inte vara i vägen för våra förbättringar. Enkla verktyg är lättare att anpassa efter den situation just ditt företag befinner sig i.

”Individuals and interactions over processes and tools ” AGILE MANIFESTO

Decentralisering och målstyrning – för att kunna arbeta effektivt

Har du någon gång tänkt på hur lite du med tanken styr kroppen och hur mycket kroppen löser själv? Till exempel tänker du bara att du ska lyfta en tekopp och hur handen ska röra sig grovt sett. Att det sedan är en massa muskler som är inblandade behöver du inte tänka på. Om koppen är för varm släpper handen reflexmässigt utan att hjärnan först måste få informationen för att fatta beslutet att släppa koppen. Kroppens decentraliserade styrning gör att vi blir snabba, effektiva och undviker att skada oss. Kan vi implementera kroppens effektiva process i en organisation?

För att kunna fatta snabba och bra beslut behöver man decentralisera beslutsfattandet. Där den mest detaljerade informationen och

nödvändiga kunskapen finns; där kan man gå fort från beslut till handling. Den centrala funktionens uppgift blir att sätta mål och koordinera och sprida information mellan avdelningar, så att beslut som fattas decentraliserat ändå har den centrala och övergripande synen. Den chef som inte respekterar och litar på medarbetarnas kompetens går miste om en stor potential. Men det är viktigt att även medarbetarna ser sin roll i företagets framgång och tar på sig ansvaret tillsammans med befogenheterna. Respekt och förtroende är inget som kommer gratis – det är något som förtjänas. Medarbetarna måste vara tydliga i det som görs, själva vara sina tidsstudiemän och visa på förbättringsåtgärder. Kort sagt, medarbetarna blir sina egna chefer och styr sig själva med samma syn och mål som deras egentliga chefer har.

”Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.” AGILE MANIFESTO

Pull – Åta sig istället för att bli tilldelad, anpassa åtagande efter kapaciteten

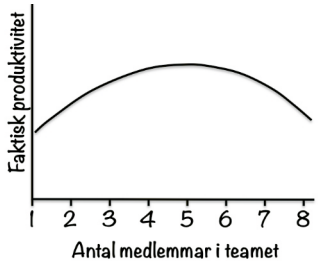
Undersökningar har visat att positiv stress är nyttig för hjärnan medan negativ stress inte är det. Men vad är skillnaden? Positiv stress är när en person själv väljer att ta på sig lite mer än normalt. Negativ stress är när någon annan tilldelar för mycket. Om negativ stress får fortgå utan möjlighet till tillräcklig återhämtning kan det leda till utbrändhet.

Frågan är hur vi får fram positiv stress utan att riskera att den blir negativ. Ett bra sätt är att säkerställa att intaget anpassas efter kapaciteten. Till exempel när ingen talar om för teamen hur mycket de ska göra varje sprint. Det är istället teamen själva som tar på sig (eng. pull). På så sätt finns en bra drivkraft för att också färdigställa det som teamet tagit på sig och uppnå sprint-målet.

En annan aspekt är att åtagande till skillnad från tilldelning minskar risken för att team påbörjar nya saker innan de tidigare är avslutade.

Fungerande team

Det som skiljer ett team från en grupp är att de jobbar mot ett gemensamt mål och håller varandra ansvariga. Inom gruppsykologi har man kommit fram till att bästa effektiviteten fås när man har 3–9 personer i ett team.



Produktivitet beroende på gruppstorlek enligt psykologen Ivan Steiner.

Många av de Agila värderingarna och principerna är hämtade från psykologin och har som syfte att hitta vad vi kan kalla för överväxeln eller flytet. Förutom gruppstorleken är motivationen en viktig faktor, att teamen själva får både befogenheter och ansvar. De ska se sig själva vid rodret och tillsammans finna bästa vägen fram, men de blir också ansvariga för resultatet. De bör alltså vara väl medvetna om vad som förväntas av dem. Att pressa ett team för att få det att jobba hårdare resulterar snarare i sämre kvalitet och inte i högre produktivitet. Istället bör team få en miljö där de kan jobba ostört och fokuserat. Det bör vara få förändringar i teamet så att en bra gruppkänsla kan byggas. Teamet ska kunna ta uppgiften i mål på egen hand med få externa beroenden och framför allt tydligt kunna titta tillbaka och se att man åstadkommit något bra.

Ständig förbättring (Kaizen) – för att bli effektivare

Hur bra en process än är finns det alltid utrymme för förbättringar. Ständig förbättring eller Kaizen, som det heter på japanska, är en princip som går ut på att alltid sträva efter att nästa vecka ska bli bättre än denna. Egentligen skulle det räcka med denna princip, för så länge alla i en organisation vill förbättra sin process kommer den till sist bli effektiv.De övriga principerna kan ses som tips från företag som redan

lyckats bli mycket effektivare, något alla andra kan lära sig av för att snabbare nå en märkbar förbättring. För att förbättra sig räcker det inte med att bara finslipa den existerande processen. Det behövs även en hel del kreativitet
och mod att prova något helt nytt. Det ger oftast större utväxling men innebär också större risk.

En annan aspekt av kaizen är att ta tag i problem och lösa dem istället för att acceptera att de finns där och hitta omvägar runt dem.

”At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.” AGILE MANIFESTO

Ett Agilt projekt i ett nötskal

Starten av ett typiskt Agilt projekt skulle kunna se ut så här: Kunden sitter tillsammans med utvecklarna och gör en grov plan över vad som ska utvecklas. Utvecklarna tar fram det minsta möjliga system (s.k. ”Walking skeleton”) som kan säkerställa att det tekniska valet och arkitekturen kommer att hålla. Detta visas för kunden för att få bekräftelse på att det lilla som hittills tagits fram ligger i linje med kundens önskemål. Produkten byggs sedan inkrementellt, funktion för funktion och kan när som helst med kort varsel anses lanseringsklar med de funktioner som är klara. För att säkerställa att resultatet ger det högsta möjliga värdet kommunicerar utvecklarna kontinuerligt med kunden, dels för att få svar på frågor och dels för att visa upp resultatet. Om kunden anser att den ursprungliga planen inte längre är aktuell, ändras den inför nästa iteration. För att ständigt vara beredd på lansering skjuts inte en massa fixande på framtiden. Koden ändras kontinuerligt för att ständigt vara av högsta kvalitet. Det gör att vi inte bygger upp en tung ryggsäck med det-spar-vi-till-senare-saker, som till slut gör oss så baktunga av gamla synder att vi har svårt att röra oss framåt. Hellre röra oss framåt i en lägre hastighet som vi kan hålla under lång tid än att hasta fram ett tidigt resultat som senare blir en dyr och jobbig belastning. Regelbundet hålls möten för att komma på saker som kan förbättras. Chefers fokus flyttas från att kommendera vem som ska göra vad, till att ordna en miljö där varje person kan jobba effektivt. Beslut fattas av dem som har mest kunskap.

Lean för mjukvaruutveckling

Lean anser att framgång ligger i att göra komplexa saker enkla snarare än att göra enkla saker billigt. De företag som bara jagar skalfördelar kommer antagligen inte se fördelarna med Lean.

Principer för Lean

Precis som Agile består Lean Software Development av principer. Mary och Tom Poppendieck har översatt Toyotas fjorton principer till sju som passar för mjukvaruutveckling. Vi har lagt till Toyotas princip nummer 1 eftersom vi tycker den är för viktig för att glömmas bort.

Principerna är:

1. Långsiktighet
2. Ta bort ojämnheter, överbelastning och slöseri
3. Satsa på kvalitet
4. Fokusera på lärandet
5. Vänta så länge som möjligt med beslut
6. Respektera andras arbete och kunnande
7. Leverera snabbt
8. Optimera på helheten

Princip 1. Långsiktighet

Basera besluten på långsiktigt tänkande, även då det sker på bekostnad av kortsiktiga ekonomiska mål.

Princip 2. Ta bort ojämnheter, överbelastning och slöseri – Eliminera Mura, Muri och Muda

Ojämnt flöde (mura) leder till överbelastning och stress (muri) vilket lätt resulterar i en dålig process med många onödiga moment (muda).

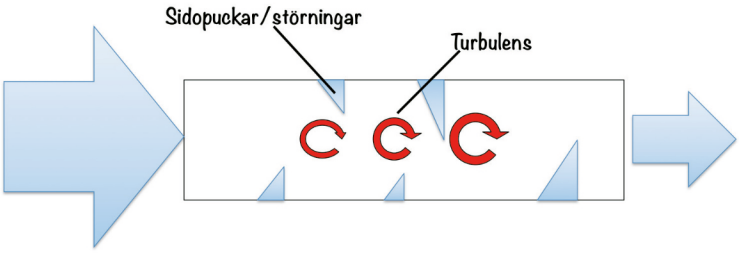
Mura – ojämnt flöde (thrashing)

För att uppnå maximal effektivitet måste vi se till att flödet inom processen går jämnt och smidigt. Vi måste få bort störningsmoment såsom påtryckning, sidopuckar och ändrade krav. Störningsmomenten påverkar fokus negativt och minskar effektiviteten. Det blir turbulens.

Här är några tips för att jämna ut flödet och minimera ledtiden:

- Anpassa inflödet efter flaskhalsarna för att eliminera köer.
- Vidga flaskhalsarna istället för att öka trycket.
- Prioritera att bli klar med påbörjade saker istället för att påbörja nya.
- Samarbeta istället för att jobba med en sak var.

Med korta ledtider blir behovet för sidopuckarna färre eftersom de kan läggas efter påbörjat jobb. Om detaljerna definieras när de behövs (just-in-time) är de aktuella när de implementeras. Om dessutom genomförandetiden är kort blir behovet för förändringar färre. Ett jämt flöde innebär också stressnivån ska vara på samma nivå under hela projekttiden. Alltså bör stress i slutet av ett projekt undvikas.



Sidopuckar och påtryckningar ger turbulens som ökar friktionen vilket minskar flödet.

Muri – överbelastning

Överlasta aldrig ett system, en person eller en process. En väg som är fylld till 100% ger dålig genomströmning, bilar fastnar i köer. Samma sak gäller för personer. De som är överlastade med jobb tar ofta fel beslut och får aldrig tid för förbättringar. De springer med cykeln på ryggen istället för att ta sig tid att stanna upp, ställa ner cykeln, hoppa upp på den och cykla iväg i högre fart. Tänk långsiktigt.

Muda – Slöseri och onödiga moment (waste)

Det är svårt att hitta en bra översättning på det engelska ordet ”waste”. Några kandidater är tidstjuvar, resursätare, slöseri, svinn eller spill. Det är allt som ur kundens synvinkel inte tillför värde.

Slöseri ska identifieras och elimineras för att få bästa effektivitet. Det kan vara onödiga möten, krångliga rutiner eller svåråtkomlig information. Mer om vanliga källor till slöseri kommer senare i boken.

Princip 3. Satsa på kvalitet (Build quality in)

I USA ville man undersöka vad som gjorde att ett område, på bara några få månader, kunde gå från trevligt område till slum. Det visade sig att det alltid började med ett trasigt fönster. Ett trasigt fönster som ingen lagade omedelbart, och som i sin tur gav upphov till att någon förstörde ett till i samma byggnad. Därefter blev en dörr och sedan en trappuppgång förstörd. Fenomenet brukar kallas för ”A broken window”. Kod har lite samma karaktär. När en utvecklare går in och modifierar befintlig kod är det väldigt svårt att höja kvaliteten på den. I bästa fall lämnar man den i ungefär samma skick som den var tidigare. Är genvägar tagna så måste man bygga vidare på dem – det kostar för mycket just då att fixa det hela. Har man väl börjat släppa på kvaliteten är det svårt att vända det förloppet.

Säkerställ kvaliteten redan från början och lita inte på att manuella tester fångar kvalitetsbrister på slutet. Lämpligen görs det genom att redan från start skriva automatiska tester som dagligen kontrollerar systemets kvalitet. Skjut inte funna fel på framtiden genom att samla dem i ett defekthanteringssystem utan fixa istället felen på en gång. Då går det snabbare eftersom utvecklarna har koden i gott minne.

Princip 4. Fokusera på lärandet (Create knowledge)

Att lära är en kostnad men samtidigt en viktig investering. För att få ett effektivt lärande är det viktigt att slippa återlärning inom organisationen. Den som behöver kunskap måste söka upp den som har den. Den som sitter inne med kunskap ansvarar även för att sprida den till dem som behöver den. Överlämningar i form av dokumentation i tydliga steg är inte den effektivaste metoden. Betydligt bättre är att jobba tillsammans och i det dagliga jobbet överföra kunskap. Det tjänar hela företaget på.

En annan aspekt på lärandet är den ständiga förbättringen, kaizen på japanska. Se tidigare kapitel. Dels handlar det om att ständigt fila på processen och produkten, men också om att utmana sig själv och se synliga problem som möjligheter till förbättring. Gå inte runt problemen utan lös dem. Nästan alla framgångsrika bolag bygger på en idé som löser ett problem.

Princip 5 . Vänta med beslut (Defer Commitment)
Det är bra att planera eftersom det ger bra träning, men att följa planerna har inget egenvärde. Istället för att planera och ta beslut tidigt är det bättre att vänta med beslutet så länge det är möjligt. Då finns det mest information och kunskap, och då är det mest sannolikt att rätt beslut fattas. Ett exempel på att senarelägga beslut är tekniken ”set-based-design”. Det innebär att ett företag istället för att ta ett tidigt beslut, baserat på för lite information, jobbar på flera spår parallellt och lär sig mera under tiden. När tillräcklig kunskap och information inhämtats kan ett bra beslut tas om vilket spår som ska arbetas vidare på och vilka som ska läggas ner. Taktiken kan med fördel användas för att inte riskera att göra ett felaktigt kritiskt val, till exempel av teknik. Tänk de företag som enbart satsade på BetaMax-spåret när kriget om video-format-standarden rasade på 70- och 80-talet. Hur kul var det när VHS vann?

Set-based-design kan även vara att jobba på en snabb men ful lösning och samtidigt med en snygg och långvarig lösning. Den snabba lösningen är bortkastad i det långa loppet men ger tillräckligt andrum eller marknadsfönster tills den långsiktiga lösningen finns på plats.

Inom Scrum ser vi just-in-time beslut på sprintplaneringsmötet. Just då vet vi som bäst vad som ger mest värde och det är också just då beslutet tas.

Princip 6. Respektera medarbetarnas kunskap, engagemang och tid (Respect people)
Chefer styr bäst genom att sätta ramar och förväntningar och förutsätta att alla gör sitt jobb. Det handlar om att respektera människor för deras kunskap både om vad som ska göras och hur det görs bäst, samt respektera det resultat de åstadkommer. Processförbättringar ska göras av dem som jobbar i processen. Ge dem tid och guidning för att lösa problemen, ett i taget, det viktigaste först. Ha en process som säkerställer att det är det största problemet som elimineras först och inte det som är tråkigast.

Princip 7. Leverera snabbt (Deliver fast)
Så snart man börjat arbeta på en funktion, även om det bara är

tankearbete, börjar kostnaden att ticka. Inkomsterna kommer först när funktionen är levererad och börjar generera pengar. För att få hög lönsamhet bör tiden mellan kostnader och inkomster minimeras. Det finns även andra skäl att minimera ledtiden. Snabb leverans innebär att kunderna inte hinner ändra sig och vill ha något annat än det ni börjat utveckla/tillverka. Se bara till att hastigheten inte medför en baksida i form av ökade kostnader för underhåll. En annan aspekt är att snabb leverans ger snabb återkoppling, vilket öppnar möjligheten att experimentera för att hitta den bästa lösningen, så kallat ”learning by doing”.

Princip 8. Optimera helheten (Optimize the whole)

Suboptimera inte utan lyft blicken och titta på helheten. Här kommer fyra exempel på olämpliga suboptimeringar:

- Att bara utvecklingsavdelningen jobbar med processförbättringar. Hela företaget bör ta till sig Lean för att vinsten ska bli optimal.
- Att ta in nya funktioner ”igår” för att glädja kunden. Detta förstör ofta kodbasen och gör processen långsammare på längre sikt. Dessutom minskar möjligheterna till snabba ändringar i framtiden, vilket leder till missnöjda kunder.
- Att låta en person bli experter på ett område och sedan bara låta den personen göra de sakerna. Kortsiktigt går det fortare om alla bara gör det de är bäst på men det minskar flexibiliteten och ökar risken att personen blir en flaskhals.
- Att optimera testandet genom att lägga det sist i projektet, när koden är ”stabil”. Problemet är att det då ofta är stressigt, och de fel testarna hittar resulterar i snabbfixar som genererar fler fel som snabbfixas med ännu fler fel som följd. Testande sent ger ofta mer jobb för testarna trots att det på pappret känns som en optimering.

Hitta balansen mellan principerna

För att lyckas med Lean gäller det att hitta den rätta balansen mellan de olika principerna. Till exempel krockar tanken om jämnt flöde, som kan behöva buffertar/köer, med eliminerandet av köer. Det är här de bästa företagen utmärker sig, de som lyckas hitta den rätta balansen mellan

minsta möjliga köer som ändå ger ett tillräckligt jämnt flöde. Ett annat exempel är att vara effektiv men ändå ha tid att leverera hög kvalitet samt höja sin kunskap och förbättra sin effektivitet.

Tio exempel på vanliga källor till slöseri

För att du ska känna igen dig från engelskspråkig litteratur har vi även skrivit med det engelska namnet, inom parentes, i rubriken. Slöseri 8 är vårt eget bidrag och har därför ingen engelsk översättning.

Slöseri 1. Onödiga funktioner(Extra feature)

Som vi nämnt tidigare har Standish group genom undersökningar fått fram att närmare två tredjedelar av alla funktioner som utvecklas sällan eller aldrig används. Det är inte bara tiden för att utveckla funktionerna som är onödig. Koden blir onödigt stor vilket ger mångdubbelt ökad komplexitet och därmed ökad kostnad i form av underhåll och vidareutveckling. Sedan blir applikationen onödigt krånglig eftersom de ofta använda funktionerna ska samsas med dem som inte används.

Enligt den undersökning av Standish Groups, som vi tidigare refererat till, är det bara 20% av funktionerna som används ofta. Hela 45% av funktionerna används aldrig utan kostar bara pengar i utveckling och underhåll. Den bästa effektivitetsvinsten du kan göra är att aldrig utveckla fler funktioner än vad som behövs och se till att utveckla de viktigaste först. Avsluta när de nya funktionerna inte tillför tillräcklig kundnytta.

En vanlig källa till att funktioner som sedan inte används kommer in i projektet är den process som används. Processer som använder milstenar eller toll gates, kräver att all funktionalitet är bestämd innan projektet får börja. Användarna ges då en och endast en möjlighet att bestämma vad som ska vara med. Risken är stor att många funktioner läggs in bara för att de ska vara säkra på att inte missa något.

Slöseri 2. Ej färdiggjort arbete (Partially done work)

All kod är kostnad, en del har även ett värde. Något som definitivt är en kostnad utan värde är de påbörjade saker som inte gjorts så klara att de kan levereras till kund. Exempel på saker som är påbörjade men inte avklarade är ej uppdaterad dokumentation, osynkroniserad kod, icke testad kod, odokumenterad kod och icke lanserad kod.

Slöseri 3. Återlärande (Relearning)
Om en kunskap finns i en organisation ska den underhållas och överföras. Slöseri är kunskap som måste läras från grunden eftersom den antingen glömts bort eller för att personer med rätt kunskap inte involveras. ”Jag lärde mig genom att läsa boken, det kan du också göra” är inget effektivt sätt att överföra kunskap. Inga fler RTFM (Read the F*cking Manual) heller.

Slöseri 4. Överlämningar (Handoffs)
Besläktat med återlärande är överlämningar. När du lämnar över uppgifter till en annan person måste du se till att lära upp den personen på ett effektivt sätt. Ett bra sätt är att inte ha tydliga brytpunkter eller överlämningar utan successivt fasa över information genom att jobba tillsammans och involvera varandra. Till exempel bör test- och supportpersoner vara med tidigt i utvecklingsprojekt. Ett dåligt sätt är att bara skriva ett dokument eller lämna en manual.

Slöseri 5 . Växla mellan uppgifter(Task switching)
Att hoppa mellan olika uppgifter gör dels att ställtiden (switching-time) måste adderas eftersom det tar tid att sätta sig in i den nya uppgiften. Du känner säkert igen dig i att du får väldigt lite gjort när du ständigt blir avbruten. Dessutom innebär det att projekt blir klara senare och därmed genererar värde senare och ibland med missnöjda kunder som följd. Vi har sett organisationer som bara vibrerar och inte får något gjort för att de hoppar mellan många uppgifter. De ägnar mer tid åt brandsläckning och att ursäkta sig inför missnöjda kunder än att producera och leverera. Att jobba med en sak i taget innebär alltså inte bara att ledtiden minskar utan även att effektiviteten och kundnöjdheten ökar.

För att minska växlandet mellan arbetsuppgifter men samtidigt kunna underhålla produkter kan ni göra på något av följande sätt:

- Två personer (roterande) tar hand om underhållet medan de övriga koncentrerar sig på utveckling.
- Alla spenderar två timmar om dagen för underhåll.
- Kritiska fel hanteras genast medan övriga samlas till en period per vecka.
- Ha samma kodbas för alla kunder så att fixar kommer alla till godo med nästa lansering.

Slöseri 6. Väntetider(Delays)

Mycket tid går åt till att vänta på svar eller beslut. Sitter personen med svaret långt borta eller är svår att nå måste utvecklaren stanna upp och leta upp svaret själv, göra något lägre prioriterat i väntan på svaret eller helt enkelt gissa. Bättre är att personen med svaret finns enkelt åtkomligt i samma rum eller i alla fall alltid är tillgänglig via telefon eller chatt. Ännu bättre är om utvecklarna har god förståelse och kunskap om kundens behov och världsbild och kan fatta rätt beslut utan att behöva fråga någon. Andra uppenbara väntetider är överlämningar till andra avdelningar där mottagaren inte är redo.

Slöseri 7. Defekter

Defekter kostar tid och pengar att hitta och rätta. Och hur mycket kostar inte defekter i minskat förtroende från kunder?

Slöseri 8. Frustration

Energi går förlorad när medarbetare är frustrerade eller gruppen inte fungerar optimalt. Det kan vara när cheferna inte lyssnar, eller omorganisationer där stor osäkerhet råder och genomförande dröjer. För att inte tala om politik, oärlighet, snack bakom ryggen eller till och med ren mobbning. Vi skulle kunna lägga till avsaknad av information men enligt Leans principer är det varje medarbetares ansvar att skaffa sig den information han eller hon anser sig behöva.

Slöseri 9 . Onödigt arbete (Extra efforts)
Det som inte direkt eller indirekt tillför värde för kunden är kostnader utan värde. Frågor du kan ställa dig är till vilken nytta du tidsrapporterar eller skriver dokumentation ingen läser. Finns inget värde är ansträngningen onödig och borde kanske inte ens göras.

Slöseri 10. Outnyttjad kreativitet (Unused employee creativity)
För allt fler företag, speciellt inom mjukvaruutveckling, är kreativ problemlösning en viktig del av det dagliga arbetet. För att nå bästa möjliga resultat behöver alla delta i problemlösningen och inte bara de som fått det i sin rollfördelning. Om bara projektledare och chefer är problemlösare, planläggare och visionärer medan alla andra bara ska utföra sina uppgift går mycket kapacitet förlorad.

Några verktyg från Lean

En grundtanke från Lean är att leverera ett beslut eller en produkt just när det behövs. Just då finns mesta möjliga information för ett bra beslut, och produkten har just då sitt högsta värde och sin lägsta kostnad. För att uppnå önskad effekt måste köer eller lager minimeras. Det görs genom att varje station anpassar sin kapacitet efter vad nästkommande station klarar av. Det är tvärtom vad som traditionellt anses rätt: att varje station ska anpassa sin kapacitet efter hur mycket som finns att göra, det vill säga, vad stationen före levererar. Risken är att det bildas köer eller lager när en station producerar mer än vad nästa klarar av. Lager kostar pengar och transporter till och från lager kostar pengar, samtidigt som varor på lager föråldras och förlorar i värde. Att leverera just när något behövs kallas på Lean-språk för Just-in-Time.

Andra grunden i Lean är att inte tillåta några som helst brister i kvaliteten. På Toyota finns ett snöre för alla att dra i för att stoppa produktionen så snart ett fel upptäcks. Kulturen kallas stop-the-line-culture.

För att finna grundorsaken till felet startas en utredning. Toyota har en metod för detta som de kallar ”the five why”. Metoden går ut på att ställa frågan ”varför” fem gånger. Svaren ska leda oss till grundorsaken, som löses, varefter produktionen kan återupptas.

Exempel:
På ett dagligt synkroniseringsmöte säger någon:
Jag har inte blivit klar med det jag tog på mig igår
Varför?
För att jag inte haft möjlighet att koncentrera mig på uppgiften.
Varför?
För att jag tar supportärenden och de har varit så många.
Varför?
För att vi levererade med för många buggar.
Varför?
För att vi inte hann testa ordentligt.
Varför?
För att vi stressades att leverera.

Aha, grundorsaken verkar vara att teamet stressats till att leverera snabbt vilket nu gör att de är långsammare. Om stressen fortsätter blir den tekniska skulden (buggarna, kodduplicering, läsbarheten) till sist så stor att det blir svårt att över huvud taget komma framåt. Hade vi bara ställt frågan ”varför” en eller två gånger hade åtgärden troligen varit att isolera personen, låta någon annan ta supporten eller rent av minska möjligheten för användare att få hjälp.

För att lyckas med Lean gäller det att ha öga för detaljer och ständigt leta efter små och stora möjligheter till förbättring. På japanska kallas det kaizen. Kaizen innebär även att utmana hinder genom att synliggöra och ta bort problem istället för att gå runt dem. Ett bra exempel har vi hittat på Toyota.

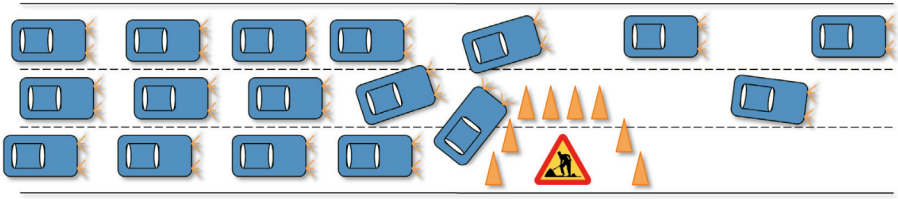
För att kunna tillverka små serier är det viktigt att tiden det tar att ställa om en maskin, till att tillverka något nytt, är så kort som möjligt. På 50-talet tog det mellan 4 dagar och flera veckor för montörerna på Toyotas fabrik att ställa om en maskin från att tillverka delar till en bilmodell till en annan. Den långa ställtiden gjorde det olönsamt att jobba med små serier, vilket var ledningens önskan. Ett mål sattes att korta ner ställtiden till 9 minuter, vilket då ansågs omöjligt. Problemen synliggjordes genom att fråga vad som hindrade att tiden kunde kortas ytterligare ett par minuter. Istället för att hindras av problemen, löstes dem ett efter ett. Idag, 50 år senare, är ställtiden 9 minuter.

Ett närbesläktat ord till kaizen är kaikaku som istället för små ständiga förändringar betyder paradigmskifte. Att gå från en vattenfallsmetod till en Agil metod görs lämpligen med kaikaku eftersom det skulle ta väldigt lång tid med kaizen.

Effektivt flöde

Lean värdesätter det som levereras och inte det som påbörjas: att få saker ur händerna (optimera ledtiden) istället för att jobba hårt.

För att kunna optimera ledtiden kikar vi bland annat på vilka flaskhalsar och köer som finns i arbetsflödena i en organisation. Allt som hindrar eller försvårar detta flöde är slöseri (waste). Ett enkelt sätt att hitta de flaskhalsar vi har är att kika på var vi har saker på hög, var är våra köer? Efter en kö finns alltid en flaskhals.



Efter en kö finns alltid en flaskhals.

Var någonstans ligger halvfärdiga saker på hög? Ligger de i bugghanteringssystem, orderhanteringssystem, otestad mjukvara eller i kravspecifikationer? Vilken flaskhals är det som kommer direkt efter denna hög?

Lean handlar om att eliminera flaskhalsar och skapa ett jämnt och effektivt flöde. Ett flöde som därigenom på kortast möjliga tid och med minsta möjliga resursåtgång levererar det kunden vill ha. För att kunna göra detta har Lean ett antal verktyg. Mer om verktygen och de grundläggande principerna i kapitlet om Lean.

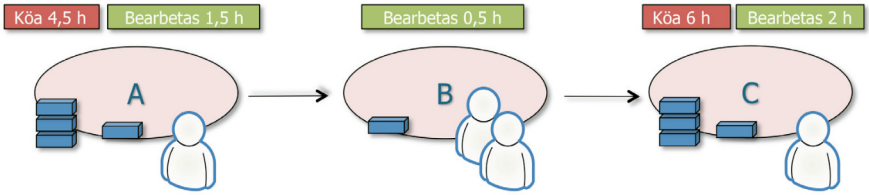
Låt flaskhalsarna bestämma tempot

Här ska vi visa ett intressant om än inte så intuitivt exempel på hur en process kan förbättras utan att öka resurserna. Det kommer att krävas lite tankemöda från din sida men vi lovar att det är värt det.

Många företag arbetar enligt filosofin att hela företaget går bra om alla delar arbetar effektivt och hårt. Lean optimerar istället på helheten och mäter ledtiden, ”from concept to cash”, vilket inte nödvändigtvis är samma sak. Låt oss ta ett exempel.

uppställning 1. utgångsläge

Här visas en genomsnittlig situation av ett arbetsflöde:



Tre stationer med fyra personer som arbetar i ett gemensamt flöde.

Vi har tre stationer. Vid varje station arbetar man optimalt och hårt. Trots det ligger saker i kö. Vi har:

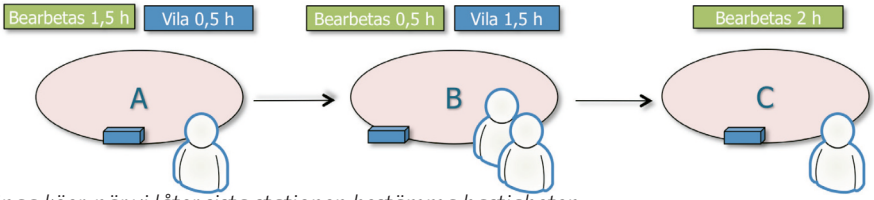
1. Station A där det i genomsnitt tar 1,5 timmar att genomföra en sak, och där det i genomsnitt ligger 3 saker i kö och väntar.
2. Station B där det är två som arbetar och har överkapacitet. Därför har de inte heller någon kö. Där tar det i genomsnitt 0,5 timmar att göra en sak.
3. Station C där det tar i snitt 2 timmar att göra en sak, och det brukar ligga 3 saker i kö.

Låt oss säga att alla arbetar heltid. Då har vi en arbetstid på 160 timmar i veckan (4 personer á 40 h). Vi slutför 20 saker i veckan eftersom sista stationen C levererar en sak varannan timme. I genomsnitt tar det 14,5h från start till mål att utföra en sak (summan av hur länge en sak ligger i kö + utförandetid).

Tid genom systemet	14,5 timmar
Levererade saker i veckan	20 st

uppställning 2. ta bort köerna

Låt oss nu begränsa mängden som vi arbetar med genom att anpassa oss efter flaskhalsen, station C. Vi tillåter helt enkelt inte att A och B producerar så länge det finns en kö i C utan tvingar dem att vila.



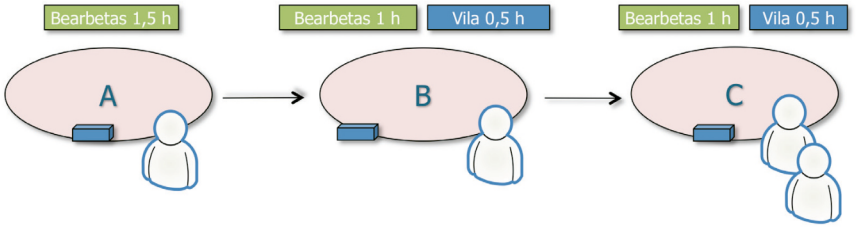
Inga köer, när vi låter sista stationen bestämma hastigheten.

Vi har nu den totala arbetstiden fortfarande 160 timmar, men utav den vilar vi i stationerna A och B totalt 70 timmar (10 + 2*30) per vecka och vi producerar fortfarande 20 saker i veckan. Tiden det tar från att vi börjar med något tills det är klart för leverans har sjunkit från 14,5 timmar till 6.

	Med köer	Utan köer
Tid genom systemet	14,5 timmar	6 timmar
Levererade saker / vecka	20 st	20 st

uppställning 3. vidga flaskhalsen

Vi har två personer på station B som inte behöver vara fler än en. Vi flyttar en av dem till flaskhalsen C och ser vad vi får för resultat.



Tider när vi bygger bort flaskhalsen (och flyttar den delvis till första ledet).

Vi har fortfarande 160 timmars arbetstid, men nu går A för fullt, så på station B och C vilar de en del av arbetstiden. A är alltså den nya flaskhalsen som bestämmer hur mycket som produceras. Då den blir klar med en sak på 1,5 timmar så blir antal levererade saker per vecka 40/1,5 = 27. Tid genom systemet är nu 4,5 timmar.

	Med köer	Utan köer	Optimerat
Tid genom systemet	14,5 timmar	6 timmar	4,5 timmar
Levererade saker / vecka	20	20	27

Efter att ha identifierat flaskhalsen och åtgärdat den har vi alltså ökat det vi levererar med 35%, och jämfört med vårt utgångsläge levererar vi nu tre gånger så fort.

Sammanfattning av strategin

För att sammanfatta strategin ovan

1. Begränsa arbetet som var och en gör så att ingen producerar mer saker än vad som behövs i det totala flödet (limit work to capacity).
2. Identifiera flaskhalsen.
3. Åtgärda flaskhalsen

Exemplet ovan är taget från typiskt fabriksarbete, men om du vill se ett sådant flöde i praktiken, gå in på ett snabbmatsställe i rusning, till exempel Subway. I ett mjukvaruutvecklande företag så motsvaras köer av specifikationer som ännu inte är kodade, kod som väntar på att bli testad, kod som väntar på att bli dokumenterad och levererad.

Några avslutande kommentarer

Vart tog köerna vägen när vi förbjöd dem? De hamnade först, innan vi tog vid. Så vad var vitsen? Jo, att köerna blev synliga för omgivningen, som därigenom enklare kunde ta beslut om de skulle vänta eller försöka med någon annan lösning. Dessutom blir det billigare eftersom inget har investerats i påbörjade produkter.

En vinst som vi inte nämnt ovan är att vi fått färre bollar i luften. I första fallet, när vi har köer, har vi nio saker igång samtidigt. När vi tar bort köerna är det endast tre saker igång samtidigt. Detta underlättar enormt för de som arbetar med det hela, att inte behöva hålla ordning på så många saker.

Minska ledtiden

Minska ledtiden genom att minska köerna

I en process som har flera steg bildas lätt köer. Det kan vara en produkt som är färdigutvecklad men eftersom testavdelningen är underdimensionerad blir produkten liggande i väntan på att testas. Tid spenderas men ingen nytta tillförs. Marknadsfönster riskerar att stängas under väntetiden. När väl en produkt börjar utvecklas ska den färdigställas så fort som möjligt, utan väntetider i köer.

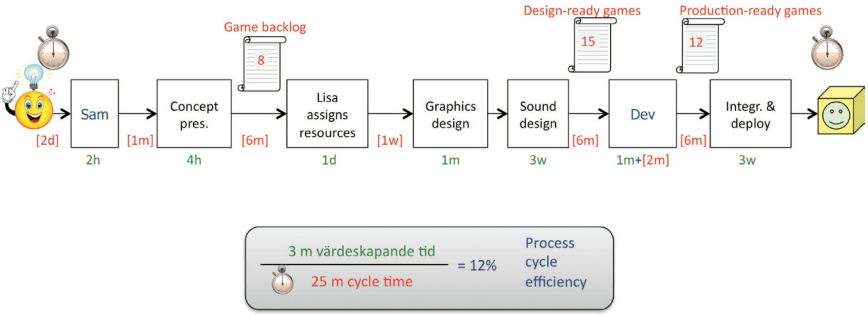
Även köer före produktionsstart är olämpliga. Många sparar alla ändringsförslag på en hög som ständigt växer, av ren lättja eller av kon-fliktsrädsla. Ta istället bort de förslag som inte anses värdefulla nog att implementeras inom den närmaste tiden, förslagsvis tre månader. Att ha en lista med önskemål som skulle ta flera år att utveckla ger bara falska förhoppningar med frustration som följd. Dessutom är en lång lista svår att överblicka.

Minska ledtiden genom att fokusera på en (1) sak

Visst är det lockande att påbörja nya projekt, och ibland kräver kunderna det. Men med Lean i bakhuvudet bör du istället fokusera på att avsluta projekt. Har du ett team med fyra utvecklare som jobbar med fyra projekt tar det betydligt längre tid för produkten att nå marknaden jämfört med om alla utvecklarna jobbade på samma projekt. Lika illa blir det om de fyra utvecklarna samarbetar men startar nya projekt innan de påbörjade projekten avslutas. Visst är det svårt att stå emot när kunderna kräver att ni ska börja jobba på just deras projekt, men det går ofta att övertyga kunden om att det är klara projekt som genererar pengar, inte påbörjade.

Value stream mapping – ett verktyg för att minska ledtiden

Value stream mapping är ett sätt att identifiera flaskhalsar och köer. Det innebär att du ritar upp din process från kund till leverans. Börja med att någon får en idé, vem tar idén vidare, vad händer sedan? Sluta med att produkten lanseras till kund. Ange för varje steg hur lång tid som gått och hur mycket av den tiden som tillförde värde. Nu ser du var tid spills och hur din process kan förbättras för att få en kortare ledtid.



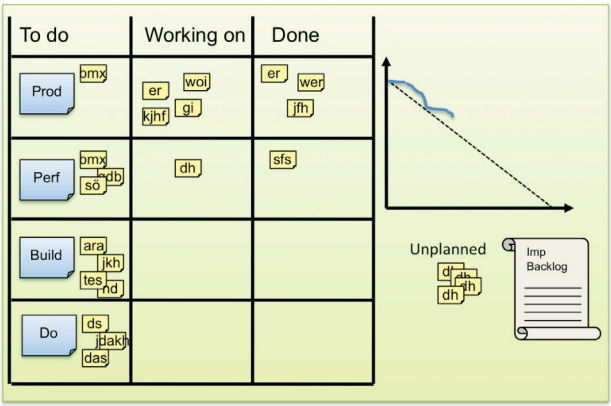
Ett verkligt exempel på processen för ett företag som tillverkar spel. Siffror inom hakparanteser markerar tid som går utan att något värde skapas medan siffror utom hakparanteser är den tid som skapar värde.
h = timmar, d = dagar, w = veckor och m = månader.

Kanban (fördjupning)

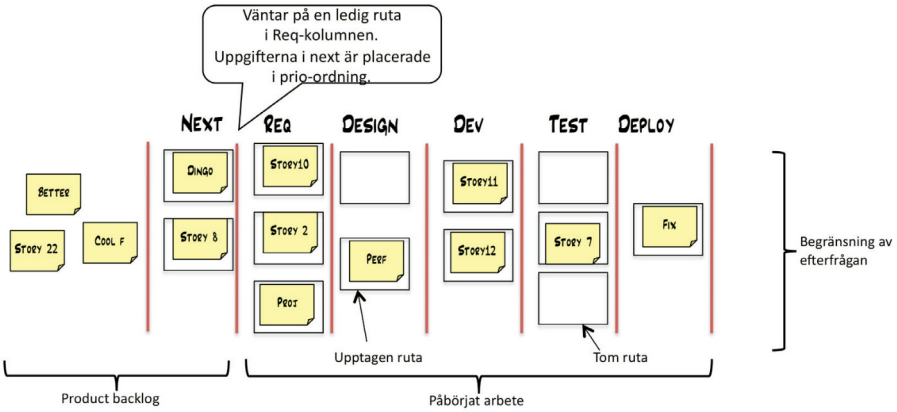
Receptet för succé

David J. Anderson, en pionjär inom Lean för mjukvaruutveckling i allmänhet och metoden Kanban i synnerhet, har tagit fram ett recept för succé. Hans ”Recipe for success” består av fyra enkla regler: Fokusera på kvalitet, reducera antalet samtidiga arbetsuppgifter, balansera efterfrågan efter kapaciteten och till sist, prioritera. Till sin hjälp för att få till de enkla reglerna har David haft Donald (Don) Reinertsen, fattare till böckerna ”Principles of Product Development Flow”, ”Developing Products in Half the Time” och ”Managing the Design Factory”.

Kanban är en metod som är ett alternativ eller komplettering till Scrum. Liksom Scrum uppfyller den tre av David Andersons faktorerna för succé: fokusera, prioritera och anpassa arbetsbördan efter kapaciteten. Till skillnad från Scrum, som delar upp tiden i iterationer, ser Kanban utvecklingen som ett ständigt flöde av arbetsuppgifter, precis som Lean. Medan Scrum begränsar arbetsbelastningen genom att begränsa hur mycket jobb som görs i en iteration har Kanban sina begränsningar i hur många saker som max får arbetas med parallellt.



I Scrum begränsas arbetet efter hur mycket som teamet tror sig hinna med under en iteration



I Kanban begränsas arbetet efter en förbestämd gräns av samtidigt pågående arbete

Kanban är japanska och betyder signalkort och används av Toyota för att få ett kontrollerat flöde av komponenter till deras fabriker. Kanban-korten är mekanismen som får just-in-time att fungera. Korten används för att beställa nya delar och följer med delarna till fabriken. När delen sätts fast på bilen skickas kortet tillbaka till tillverkaren för att meddela att det är dags att tillverka och leverera en ny del. Genom att inga nya kort förs in i systemet, bortsett från att ersätta förstörda, säkerställs att bildelarna kommer med en säker men minimal ström, just när de behövs.

Liksom Kanban begränsar antalet bildelar i omlopp hos Toyota, begränsar den antalet uppgifter utvecklare jobbar med samtidigt. En utvecklingsavdelning kan bestämma att de bara jobbar på två saker samtidigt. Inget nytt får påbörjas innan något pågående är avklarat. Det blir ett ständigt flöde av uppgifter som betas av.

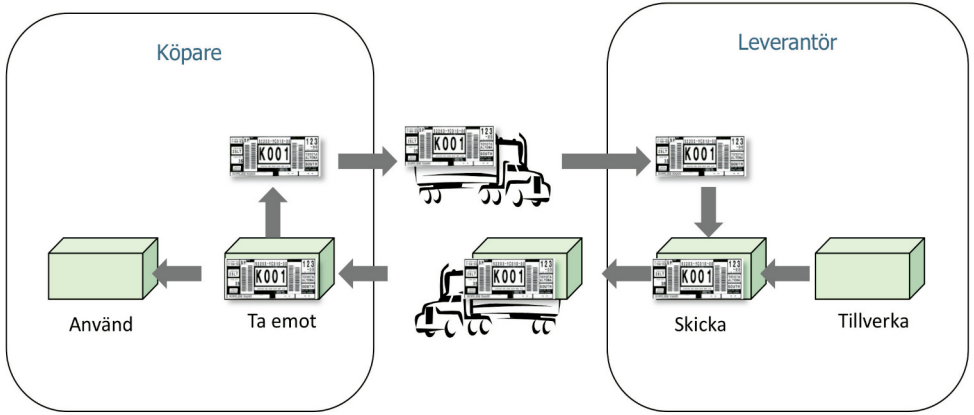
Som tidigare nämnts får vi en liknande begränsande effekt med Scrum, genom de åtaganden teamen gör i början av en sprint. Så vad tillför Kanban som vi inte får av Scrum?

Det finns omständigheter då Scrums sprintuppdelning med åtagande inte fungerar så bra. Det gäller till exempel för avdelningar som har svårt att förutse hur mycket de kan ta på sig för en sprint. Typiskt gäller det för driftavdelningar som parallellt med driften ska göra förbättringar i form av utvecklingsarbete. Har de stora variationer i hur mycket tid de kan lägga på utvecklingsarbetet fungerar inte Scrum så bra eftersom sprintprognosen inte har något värde. Med hjälp av Kanban-tavlan får driftavdelningen fokus på vad som ska göras så snart bränder (urgent) är släckta och de dagliga kontrollerna (daily duty) är utförda.

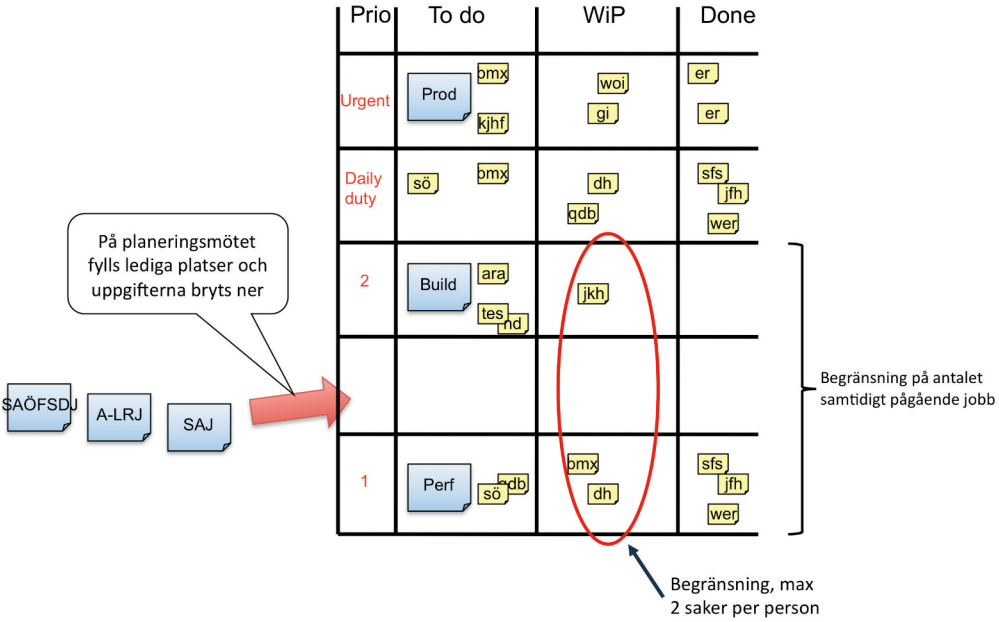
看板

Kan Ban

”Visuellt kort”



Kanban är ett kort som säkerställer att bildelar beställs i just den mängd som behövs för att hålla tillverkningen igång utan att bygga upp lager.

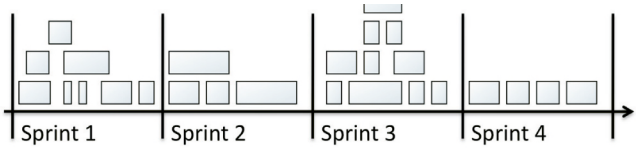


En Kanban-tavla för drift. Högst prioriterat är "Urgent", därefter "Daily duties" och först därefter jobbar teamet på projekt enligt den ordning prio-kolumnen anger. Genom att alltid ha en tydlig bild på vad nästa uppgift är får teamet bra fokus och får saker gjorda som förhoppnings minskar sakerna i "Urgent". För att inte tappa fokus begränsas i exemplet antalet stories till tre och antalet uppgifter till två per person.

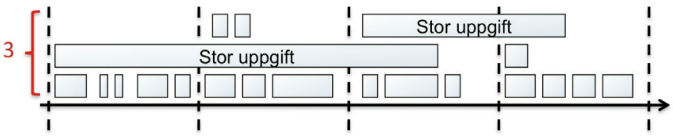
En annan omständighet som kan göra att Scrum inte är optimalt är då kultur eller arbetsuppgifter förhindrar team med blandad kunskap (s.k. tvärfunktionella team). Med hjälp av Kanban-tavlan visualiseras flödet vilket hjälper flera komponentteam att samarbeta samtidigt som stopp i flödet snabbt upptäcks. Det dagliga synkroniseringsmötet (motsvarande Daily Scrum) hanterar frågor runt flödet.

Eftersom Kanban inte jobbar i iterationer finns heller ingen riktig begränsning i storleken på arbetsuppgifterna även om många små oftast är enklare att jobba med än några få stora.

Scrum



Kanban med begränsningen 3



I Scrum måste uppgifterna vara så väl nedbrutna att de med god marginal får plats i en sprint. Om marginalen är för liten riskerar teamet att avsluta sprinten utan att något är klart och får då "velocity" lika med noll. I Kanban finns inte sprintar men istället en begränsning i antalet samtidigt pågående jobb. Även om det är bra att bryta ner uppgifter till mindre delar för effektivitetens och flexibilitetens skull, finns det inget i Kanban som tvingar fram en nedbrytning — vilket är både en fördel och en nackdel.

I övrigt kan processen låna det mesta från Scrum. Det går bra att ha en produktägare som prioriterar och en scrum-mästere som håller koll på processen och rensar bort hinder. Det finns en att-göra-lista för framtida uppgifter och en Kanban-tavla, som liksom Scrum-tavlan underhålls för att tydligt visa vad som jobbas på just nu.

Planeringsmöten hålls men de är enbart för att överföra information från produktägaren till utvecklingsgruppen. Tidsuppskattning kan göras om produktägaren behöver det för prioriteringen eller projektuppföljning men är annars onödigt. Reflektionsmöte, för att förbättra processen, hålls regelbundet.

Scrum (fördjupning)

Scrum är en minimalistisk metod som belyser problem organisationen har.

> ”Scrum does not provide the answers
> to how to build quality software faster.
>
> Scrum is a tool, a framework, you can use to find out
> what you need to do to build quality software faster.”
>
> KEN SCHWABER

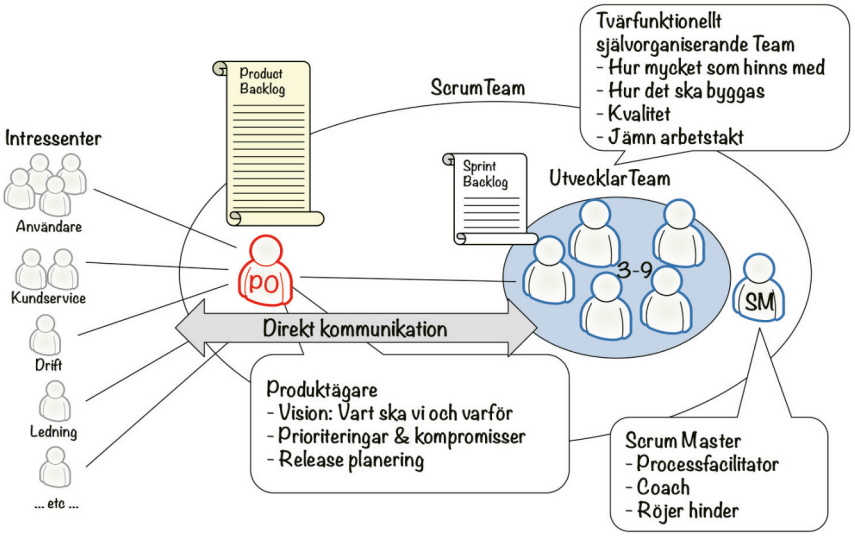
För att lösa problemen krävs en hälsosam inställning till mjukvaruutveckling och team-byggande. Här kommer idéer hämtade från Lean och Agile väl till pass, men också tankar från böcker som ”Good to Great” av Jim Collins och ”Five dysfunctions of a team” av Patric Lencioni.

Kort repetition av Scrum

I Scrum jobbar ett team bestående av olika kompetenser tillsammans för att uppnå ett mål. Målet bestäms för en tidsperiod om en till fyra veckor (kallas sprint). Prioritering sköts av en person som kallas produktägare. Hur mycket som ingår i sprintmålet bestäms av utvecklarna. För att hålla koll på att processen följs och förbättras utses en person som får titeln scrum-master.

Ansvarsområden (tidigare kallat för roller)

Som vi nämnde i inledningen har Scrum tre ansvarsområden: produktägaren, utvecklare och scrum-mästaren. Personer som är utanför detta men ändå har intressen kallas intressenter (stakeholders).



Ansvarsområden i Scrum: intressenter, produktägare, utvecklare och scrum-master.

Intressenter är alltså alla som produktägaren väljer att lyssna på när hen tar sina prioriteringsbeslut och när hen hämtar in information som ligger till grund för de saker som behöver göras.

Utvecklarna uppskattar kostnader och scrum-mästaren är ansvarig för att hålla Scrum-processen på plats och säkra dess ständiga förbättring, facilitera möten, röja hinder samt coacha teamet samt intressenter.

Produktägare (Product Owner)

Produktägaren ansvarar för att produktutvecklingen genererar så mycket kundvärde som möjligt. Det görs genom att prioritera bland önskade förbättringar och göra produktbackloggen (att-göra-listan) så tydlig som möjligt. Produktägaren måste ha god koll på alla intressenter (kunder, företagsledning, marknadsförare, support, drift med flera) och deras önskemål. Ett nära samarbete med utvecklarna är bra och det underlättas av att produktägaren sitter tillsammans eller i alla fall nära dem.

Produktägaren måste inte kunna alla krav utan och innan, bara så mycket att denne klarar av att prioritera mellan dem och vet vilken intressent som vet mer. Dem kan produktägaren hänvisa till när teamet vill ställa frågor kring olika mer eller mindre oklara önskemål.

Det är tungt att vara produktägare, och kan därför lämpligen vara en grupp av personer. Men då är det viktigt att gruppen hela tiden ”pratar med en mun” när det gäller prioriteringar. Dubbelkommando är förödande för produktivitet och kvalitet.

Utvecklarna

Individerna i ett team har inte egna roller i Scrum. Anledningen är att man vill att teamet gemensamt ska ta på sig ansvaret att arbeta och leverera utifrån backloggen. Oftast krävs det mer än en kompetens för att bli klar med något. Kanske behövs det en användargränssnittsexpert plus en programmerare och en testare innan en sak är på plats. Inte bara deras kompetenser var och en för sig utan ett fungerande samarbete mellan dessa.

En lämplig storlek på utvecklargruppen är tre till tio personer som bör vara 100% dedikerade till gruppen. Det går att ha personer som är med i flera team men det blir då svårt för en person att förpliktiga sig till båda teamens mål. Det är dessutom demoraliserande för ett team, som är beroende av att alla personer ger järnet, när en eller flera säger att de måste prioritera en annan uppgift. Tänk dig ett fotbollslag där målvakten spelar två matcher samtidigt. Det kan fungera när det ena laget anfaller men ger prioriteringssvårigheter när båda är på defensiven. Bättre är att dedikera de personer som man har svårt att knyta till ett team på heltid

till ett av teamen, och låta det andra, vid behov och när det passar sig, låna personen. Detta brukar få till följd att det andra teamet snabbt lär sig det teamet behöver, och använder personen mer som lärare än utförare.

Blanda kunskap i tvärfunktionella team

Tänk dig ett fotbollslag med bara målvakter eller ett med bara vänsterforwards. Det skulle inte gå så bra för det laget – ändå är det så många utvecklingsteam ser ut. Scrum ser hellre att grupperna byggs upp med olika kompetenser som behövs för att ro projektet i hamn. Man pratar om att en bra teammedlem är både specialist och generalist. Varje medlem har en eller flera spetskunskaper, men kan även kavla upp ärmarna och hugga in och hjälpa till på andra områden. Till exempel kan en användargränssnittsexpert vara med och testa applikationen. Detta kallas även att man har en T-formad kompetens snarare än en I-formad.

Det är moralhöjande att tillsammans få upp hög fart och inte bromsas av att ständigt vänta på tillgång till gemensamma resurser. Ursäkter i stil med ”vi gick inte i mål eftersom vi inte fick tillgång till den eller de resurser vi behövde” ska inte behöva förekomma.

Systemet ska byggas i den ordning kunden vill, för att optimera kundnyttan, inte i den ordning som maximerar varje resurs. Scrum försöker därför komma bort från bestämda roller eftersom det innebär en stelhet. I stället för roller finns experter i teamet men huvudfokus är att lyckas uppnå uppsatta mål som ett team, inte som individer. Genom att sitta i teamet sprider experter sin kunskap till de övriga utvecklarna. På så sätt försvinner flaskhalsar samtidigt som företaget blir mindre beroende av enstaka personer.

En vanlig fråga är ”blir det inte väldigt spretigt om det inte finns en arkitektgrupp och databasgrupp som sätter standarder?” Även om experterna sitter spridda i olika utvecklingsteam bildar de virtuella grupper som träffas regelbundet för att synkronisera. Den virtuella gruppen för scrum-masters brukar kallas Scrum av Scrum. Där syncar scrum-mastrarna saker som berör flera team.

Självorganiserande team (self-organizing/managing team)

Det dagliga arbetet som utvecklare innebär att ständigt fatta beslut. Effektivast blir det om de som sitter djupast med händerna i syltburken också fattar besluten. Chefen får tid att ägna sig åt de långsiktiga målen

medan medarbetarna jobbar med de kortsiktiga. Om teamen själva ansvarar för att på bästa sätt organisera sig för att uppnå optimalt resultat blir medlemmarna mer aktiva och drivande samtidigt som kreativiteten ökar. Det ger motiverade personer som arbetar engagerat och effektivt eftersom det ligger i deras eget intresse.

Men det är frihet under ansvar. För att teamet ska få behålla sin frihet måste de visa att de tar sitt ansvar på allvar. Det gör de genom att leverera högkvalitativa produkter, ständigt jobba på att höja effektiviteten och genom att följa de strategiskt bestämda riktlinjerna. Dessutom ansvarar de för att få förtroende från resten av företaget. Peter Scholtes beskriver i sin bok ”The Leader’s Handbook” vad som krävs för att bygga förtroende. Kort går det ut på att visa att man har samma mål och är kompetent att uppfylla målen. Rätt inställning är bra men den måste också kommuniceras och bevisas.

Det gör teamet genom att vara transparanta och tala affärsspråk. Att vara sin egen ledare innebär också ett ansvar att skaffa sig den information man behöver – inga fler ”vi får ingen information”.

Gemensamt ansvar

Många chefer har svårt att ge utvecklingsgruppen eget och inom gruppen delat ansvar. Men tänk på vad som är chefens uppgift och även vad de flesta chefer tycker är roligast. Är det att peta i detaljer och tala om för varje person vad han ska göra och hur han ska göra det, inte bara under en upplärning utan dagligen? Är det det bästa sättet att utnyttja chefens tid? Eller vore det bättre om chefen ägnade tiden åt att sätta upp strategiska mål, kommunicera och följa upp dem?

Även det delade ansvaret har vissa chefer svårt att leva med. De verkar tro att det viktigaste är att det finns en person att peka ut som syndabock om något går fel. Vi tycker det är bättre att det blir rätt istället, och sju personers hjärnor tänker bättre än en. Någon annans ansvar är inte mitt ansvar. Gruppens ansvar är mitt ansvar om gruppen klarar av att självorganisera sig.

Sitter tillsammans (co-located)

Ett team fungerar bäst om medlemmarna sitter samlade. Om alla sitter tillsammans och det bara tar några sekunder att rulla fram stolen till en

annan utvecklares skärm fungerar samarbetet lättare. Scrum försöker att göra det så smidigt och enkelt som möjligt att samarbeta.

Scrum-mästaren

Scrum-mästaren är lagkapten, coach, ordningsvakt och vaktmästare i en och samma person. Han eller hon ser till att teammedlemmarna kan jobba strukturerat, ostört och effektivt och med möjlighet att ständigt förbättra sin process.

Om någon eller något stör teamet är det effektivast om en representant för teamet får i uppgift att få bort det störande elementet. För att slippa diskussioner om vem som ska representera är det scrum-mästarens jobb. Störande element behöver inte bara vara personer utan även andra hinder som onödiga tidrapporter, krånglig process, långsamma datorer och dålig miljö, både fysisk och teknisk. Scrum-mästaren behöver inte själv ta bort hindren men ansvarar för att frågan drivs mot en lösning och inte faller mellan stolarna.

Scrum-mästaren ansvarar också för att den bestämda processen efterföljs. Om vi inte följer en process kan vi inte utvärdera om den fungerar eller ej och heller inte förbättra den eftersom vi inte vet vårt utgångsläge.

En bra scrum-master är en person som upptäcker och löser problem utan att göra en stor sak av det.

Produktbacklog

Produktbacklog är produktens att-göra-lista, en produktutvecklingsplan. Den är strikt prioriterad där det bara finns en punkt som är prio ett, en punkt som är prio två och så vidare. Därmed är det alltid tydligt vad som ska jobbas på härnäst. Punkterna på listan kallas för ”backlog items” och kan vara en ny funktion, en teknisk förbättring eller en buggfix. När det gäller en funktion kallas det ofta för ”user story” eller bara ”story”. Storyn ska beskrivas utifrån användarens eller kundens perspektiv och inte hur den ska lösas rent tekniskt. Användarna ska kunna förstå vad de kan förvänta sig. En produktägare äger listan och bestämmer vad som läggs dit och dess prioritering. Listan är alltid levande. När som helst kan produktägaren lägga till nya saker till listan och omprioritera dess

ordning. Om produktbackloggen är synlig kan alla på företaget se var just deras favoritförbättringar finns på listan och vilka saker som anses mer prioriterade. En väl synlig och lättillgänglig produktbacklog minskar organisationens frustration. Därmed går mindre tid åt till att förklara vad som görs för dem som anser att ”ingentinghänder”. Transparens istället för hemlighållande!

Arbetet med produktbackloggen

Att underhålla en produktbacklog innebär en hel del jobb. För att inte lägga för mycket tid på saker som kanske aldrig blir utvecklade bör du lägga mest tid på det som ligger överst. De har högst sannolikhet att bli utvecklade. Saker högst upp bör vara väl genomtänkta medan saker längre ner i prioriteringen kan lämnas lite mer diffusa. De saker som kan förväntas bli utvecklade under de närmaste sprintarna bör vara nedbrutna i så små delar att de inte bör ta mer än en halv sprint att utföra. Dock ska nedbrytningen ske så att det fortfarande finns ett värde kvar i uppgiften. Det kan vara en direkt kundfördel men även riskminskning och lärande har ett värde.

Prioritera saker i produktbackloggen

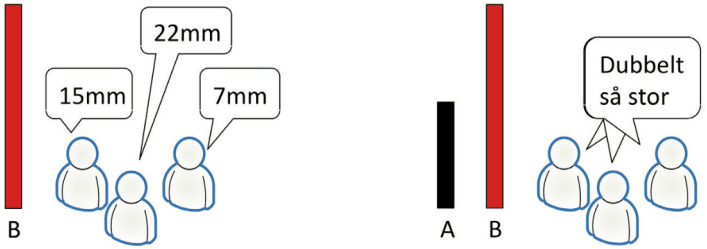
Vid prioritering mellan uppgifterna kan det vara bra att tänka på fyra saker:

1. Vilket värde tillförs?
2. Vad är kostnaden?
3. Minimeras någon risk genom att den utvecklas?
4. Kan vi lära oss något genom att få den utvecklad?

Storleksuppskattning

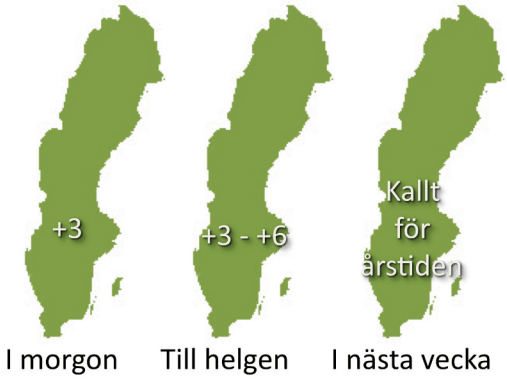
För att produktägaren ska kunna prioritera måste denne veta kostnaden för varje story/backlog item. Dess storlek behöver uppskattas. Att estimera är svårt men här kommer tre knep som i alla fall gör det lättare:

1. Det görs lämpligen av de som senare ska bygga funktionen, det vill säga utvecklarna. För att veta vad som ska göras krävs god kommunikation (möten) mellan produktägaren, som vet vad han vill ha, och teamet som bäst vet hur det ska byggas.



Det är lättare att få ett korrekt estimat om estimatet görs relativt.

2. Vi är dåliga på att tidsuppskatta i absoluta tal som ”Det tar 3 dagar att skapa denna modul”. Det är lättare att estimera relativt så som ”Det här är ungefär dubbelt så stort som det där”
3. Ta fram estimaten under en diskussion för att få in flera vinklar. Om bara en person gör uppskattningarna finns risken att den personen glömmer att ta med viktiga delar i beräkningarna. En vanlig teknik, för att få hela teamet att tidsuppskatta, är att spela planning poker. Se www.crisp.se/planningpoker ifall du vill veta mer.



Vi brukar få en rätt bra uppskattning av vädret i morgon, på någon grad när. Vi få lite mindre precision för några dagar framåt, och för flera veckor framåt får vi inga prognoser alls. För varje dag som går tillkommer ny information som gör att vi ändrar vår prognos.

Omvandla estimerad storlek till tidsåtgång med velocity

Scrum är en empirisk process. Med det menas att vi istället för att uppskatta exakt hur lång tid någonting tar så gissar vi hur lång tid det tar, därefter mäter vi hur lång tid det faktiskt tog, och justerar våra andra uppskattningar utifrån det uppmätta värdet.

låt oss ta ett exempel: hur lång tid tar det att läsa en bok?

Ett sätt att bedöma det är att uppskatta dess storlek. Till exempel kan vi snabbt kolla hur många kapitel boken har (låt oss säga, femton). Därefter läser vi ett genomstort kapitel, och mäter tiden för hur lång tid det tog (fyra timmar). Vår uppskattning av hur lång tid det tar att läsa hela boken är femton gånger denna tid. Efter att bara ha läst en femtondel, ett medelstort kapitel, kan vi alltså veta hur lång tid hela boken tar att läsa. Man skulle kunna säga att vår hastighet är ett kapitel per 4 timmar och att hela boken tar ungefär sextio timmar att läsa.

Tidsuppskattning enligt Scrum fungerar på samma sätt. Man uppskattar de uppgifter man har att göra i relativ storlek. För att komma igång är det lämpligt att välja en vanlig arbetsuppgift och använda den som norm, det alla andra arbetsuppgifter ska relateras till. Om den normgivande arbetsuppgiften är medelstor kan vi bestämma att den har en storlek av 8 story points. Story points är en påhittad storlek utan någon norm. Den betyder olika för varje team. Därefter jämförs de olika saker vi ska göra relativt denna medelstora sak. Någon sak som uppskattas vara en fjärdedel så stor ges storleken 2 story points. Något som är dubbelt så stort får storleken 16 story points.

I början av en sprint åtar sig sedan teamets medlemmar den mängd story points de tror sig klarar av, säg 50 (uppskattad förmåga). Därefter arbetar teamet en sprint och ser hur många story points de faktiskt hann med under sprinten, säg 35 (uppnådd förmåga). Då har vi ett mått på hur snabbt de för närvarande arbetar, det vill säga deras nuvarande hastighet (velocity) är 35 story points per sprint. Denna hastighet har vi därefter som utgångspunkt för hur mycket vi hinner med i nästa sprint.

Efter några få sprintar brukar hastigheten svänga in sig mellan några värden, till exempel 30–40. Därmed har vi den information vi behöver för att planera framtiden. Om en intressent undrar när de saker de vill ha gjorda är klara, och de är uppskattade till 90 story points blir svaret

antagligen om tre sprintar, max fyra. Eller om varje sprint är 3 veckor är vi klara om 9–12 veckor.

Velocity är alltså något som beror på dels hur teamet tidsuppskattar, dels hur de arbetar. Detta gör att det inte enkelt går att jämföra ett teams hastighet med ett annat. Velocity har vi inte för att jämföra teams produktivitet med varandra, utan för att kunna få en förutsägbarhet i när saker blir klara. Det är ju lätt att få hög hastighet genom att helt enkelt bara öka uppskattningarna.

Definition av klar – (Definition of Done, DoD)

För att kunna göra en tidsuppskattning måste alla vara överrens om vad som behöver göras för att en uppgift ska anses färdiggjord. Är det när en programmerare har kodat klart, när en testare har verifierat saken, eller när vi gått live på en sajt? Den vanligaste definitionen är ”potentiellt releasebar”. Det betyder färdigkodad, färdigtestad, buggfixad, dokumenterad, packad och klar. Enklaste sättet att säkerställa att ”klart” betyder lanseringsklart är helt enkelt att lansera. Om inte kunderna är redo för en lansering, sätt upp en testmiljö och lansera dit. Om testmiljön är likadan som produktionsmiljön kan vi vara ganska lugna att den riktiga lanseringen senare kommer att gå bra.

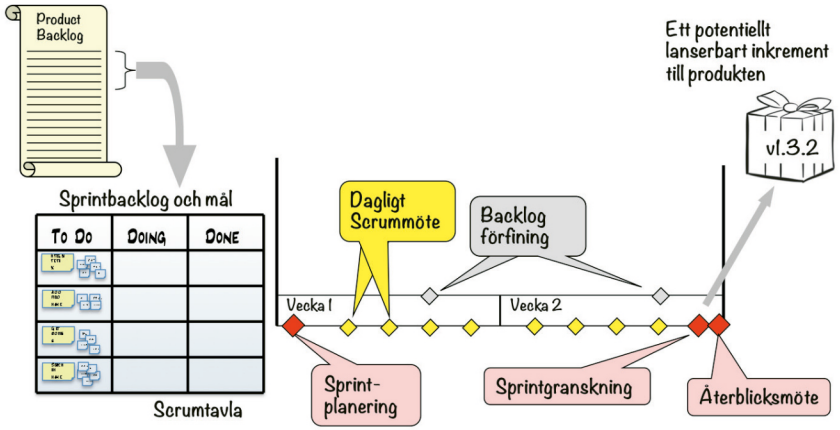
Begränsa tiden och variera innehållet (Timeboxing)

Att tidsuppskatta är svårt eftersom det i utveckling nästan alltid rör sig om att göra något nytt och därmed okänt. Man kan då välja ett av två spår. Antingen håller man innehållet fixerat och låter det ta den tid det tar. Eller så levererar man på utsatt datum oavsett hur mycket man har hunnit med. Scrum tror på att lansera på utsatt tid och istället tumma på vad som lanseras. Timeboxing istället för scopeboxing.

Ett datum kommer alla ihåg och det är lätt att notera i en kalender. Vad som ingår är svårare att komma ihåg speciellt eftersom det viktigaste byggs först och finns med även vid en försening. Om arbetet inte avstannar utan fortsätter, kommer det inom kort uppdateringar med förbättringar, ibland enligt ursprunglig plan och ibland utifrån återkoppling från dem som använder de levererade delarna av systemet. Genom att

fokusera på tider, som att en sprint alltid är av en fix längd, fås en trevlig bieffekt – man får en puls. En puls som teamet och så småningom hela företaget anpassar sig efter.

Scrumprocessen



Som vi tidigare nämnt jobbar scrum-team i iterationer (kallas sprintar) på till exempel två veckor. Sprintens längd kan skilja mellan olika team beroende på vad som passar deras arbetsuppgifter och intressenternas behov för snabba omprioriteringar. Dock är det lämpligt att hålla sig till en längd för att kunna jämföra resultat mellan sprintar.

Sprinten består av en sprintstart med planeringsmöte, dagliga synkroniseringsmöten och ett sprintslut. Sprintslutet består av en granskning (review) för att visa upp vad teamet åstadkommit under sprinten och ett återblicksmöte (retrospective) för att utvärdera och förbättra processen.

Starta sprinten

Sprintplaneringsmöte

Sprinten startar med ett planeringsmöte. Ett sprintplaneringsmöte är typiskt två timmar per sprintvecka, så om man har en sprint som är två veckor är mötet fyra timmar. Tidsbegränsat, som vanligt. Dit kommer produktägaren med en grovt tidsuppskattad och prioriterad produktbacklog som presenteras för utvecklarna. Utvecklarna ska under mötet bestämma sig för hur mycket de tror sig kunna utföra under sprinten. Till sin hjälp har de produktägaren som svarar på frågor. Stora funktioner kan brytas ned till mindre och även brytas ned till vad som krävs rent tekniskt. Utvecklarna gör det som behövs för att känna sig komfortabla med sin prognos. När teamet bestämt vad de tror sig klara av under sprinten är det dags att planera utförandet. Eftersom de är självorganiserande listar de tillsammans ut hur de bäst tar sig fram till målet. T.ex. hur ser arkitekturen ut och vilka frågor behöver besvaras? Stories bryts ner till tekniska arbetsuppgifter (tasks). När planeringsmötet är slut börjar sprinten.

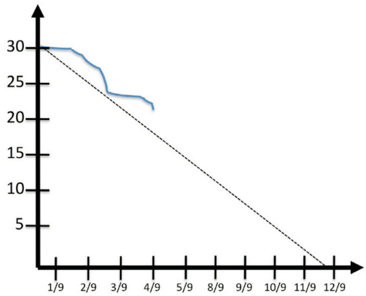
Sprintbacklog

Sprintbackloggen är den del av produktbackloggen teamet tror sig kunna färdigställa under en sprint. Normalt är det här de högst prioriterade sakerna. Till skillnad från produktbacklogen är dess innehåll och prioritering fryst efter sprintplaneringsmötet och ändras inte under sprinten utan att både produktägare och utvecklare godkänt ändringen. Det ger en bra balans mellan lättrörlighet och möjligheten för utvecklarna att fokusera och planera sitt arbete.

Sprint Burndown

Det arbete som teamet tagit in i sprinten summeras vid sprintens start. Det kan vara summan av antalet timmar för arbetsuppgifter som är kvar eller helt enkelt bara antalet arbetsuppgifter som är kvar. Siffran ger en punkt på y-axeln i ett diagram. X-axeln utgörs av dagarna som ingår i sprinten. Varje dag räknas en ny siffra fram och med tiden syns en kurva som förhoppningsvis tenderar att korsa x-axeln vid slutet av sprinten. Det viktiga är inte att få fram en snygg kurva utan att kunna se en trend och

tidigt meddela produktägaren om det finns risk att några stories inte hinns med eller om produktägaren bör förbereda fler stories eftersom utvecklingsarbetet gått bättre än förväntat.



Sprint burndown chart är ett vanligt sätt att hålla koll på hur man ligger till i sprinten. För varje dag prickar man in en punkt för hur mycket arbete man har kvar. På så sätt ser hela teamet, och de som passerar teamrummet, hela tiden ifall man ligger före (under det diagonala strecket) eller efter (över det diagonala strecket).

Sprintens gång

Dagligt Scrum-möte

För att tillsammans planera dagen möts gruppen i det dagligt Scrum-mötet som varar max 15 minuter, samma tid varje dag. Under mötet skapar vi samsyn på var vi är, vad vi bör göra idag och om något hindrar oss. T.ex. kan var och en besvara frågorna:

1. Vad gjorde jag igår, som hjälper oss att nå sprintmålet.
2. Vad kommer jag göra idag, som hjälper oss att nå sprintmålet.
3. Detta hindrar mig eller oss, från att nå sprintmålet.

Det viktiga är att mötet resulterar i en samförståelse inom teamet på hur dagens arbete kan organiseras för att teamet effektivast ska nå sprintmålet. Det är inte ett status och avrapporteringsmöte till produktägare eller scrum-mästaren. För att försäkra sig om att mötet inte drar ut på tiden står alla upp under mötet. Därför kallas mötet ibland för stå-upp-möte.

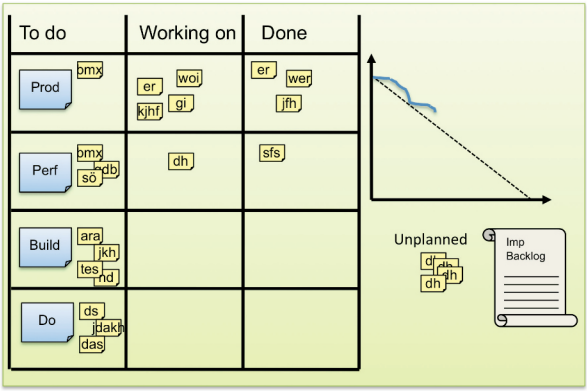
Fokus är på hinder, och dessa hinder diskuteras efter mötet med de personer som kan lösa problemet. Mötet är i första hand till för att teamet ska koordinera sig. Alla är välkomna att komma på mötet som åhörare men får inte störa för att mötet ska kunna hållas effektivt. Har de något

viktigt att tillägga ska de naturligtvis säga det. Relevant information är så värdefull att det inte spelar någon roll vem som sitter inne med den. Efter mötet uppdateras sprint-burndown så alla kan se hur sprinten fortlöper och hur troligt det är att teamet når sitt mål.

Sprint task board

En princip som är ständigt återkommande inom Agile är att visualisera problem och status. Ju synligare man gör saker och ting, desto lättare är det att komma överens om vad som ska göras härnäst. Om teamet fysiskt lyckas synliggöra saker blir teamet dessutom effektivare: utomstående behöver inte störa teamet för att få reda på hur det ligger till. Man talar om informationsradiatorer, det vill säga informationstavlor som sprider information i ett rum på samma sätt som element sprider värme.

Allt arbete som tagits in i sprinten sätts upp på en vägg eller whiteboard. Huvuddelen är ett rutnät med en rad per story och tre kolumner. Den första kolumnen är saker som ingen påbörjat, den andra är det som det jobbas på och den tredje är det som är klart. Burn down chart är ett sätt att se hur man ligger till tidsmässigt. Y-axeln har mängden arbete som är kvar och x-axeln visar sprintens dagar. Varje dag markerar man en punkt för det återstående

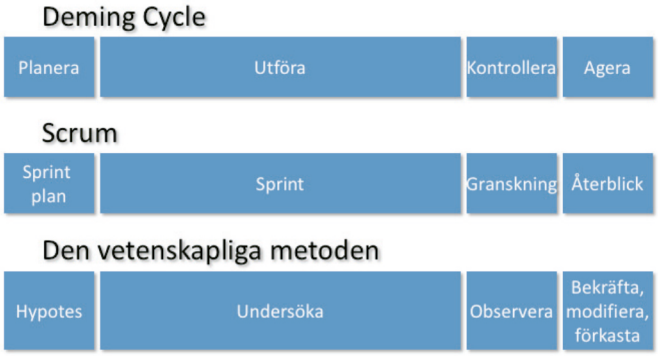


Ett exempel på hur det kan se ut på en vägg i ett Scrum-teams rum. Dels vilka saker som är påbörjade respektive avslutade i kolumnerna till vänster. Dels vilka störningar man haft under sprinten under rubriken "Unplanned Items". Dels hur man ligger till tidsmässigt via burndown-diagrammet.

arbetet. Ligger man över det diagonala strecken så ligger man efter, och det kan vara bra att göra produktägaren uppmärksam på det.

Sprintavslutning – tid för reflektion och anpassning (inspect and adapt)

När sprinten är avslutad ska vi förbättra oss genom att utvärdera och lära av det som hänt under sprintens gång. Det gäller både produktens och processens utformning. Detta är ett återkommande arbetssätt som har många namn, ett är Deming Cycle eller Plan-Do-Check-Act (PDCA).



Både Lean, Scrum och vetenskapen använder den så kallade Deming cycle för att säkr kvaliteten. Genom att repetera cykeln kan vi komma närmare den perfekta processe och resultatet.

Sprintgranskning (Review)

Som tidigare nämnts jobbar man i Scrum i sprintar på mellan en och fyra veckor (beroende på vad som passar organisationen). Efter en sprint ska utvecklarna leverera fungerande funktioner som demonstreras för intressenter. Hit är alla välkomna dels för att se vad som gjorts, få reda på framtida planerade leveranser men också för att sammarbeta om hur pro dukten kan vidareutvecklas. Eftersom leveransen är något som kan testas är det enkelt för intressenterna att förstå hur slutprodukten kommer att se ut och redan tidigt korrigera eventuella missförstånd. Korrigering kan också behövas för att ny kunskap och omvärldens

förändringar gör att ändringar i planen måste till. Ändringar är faktiskt bra: det innebär ju att kunden får något bättre än vad kunden från början tänkt sig.

Granskningsmötet är också ett utmärkt tillfälle att bygga förtroende för teamet och deras förmåga att välja rätt beslut om både process- och produktutveckling. Därför passar teamet på att berätta om hur de tolkar marknadsplatsen och användarnas behov samt berättar om vilka problem de stött på och hur de löst dem.

Ett väl genomfört sprintgranskningsmöte minskar behovet för styrgruppsmöten.

Återblicksmöte (Retrospective)

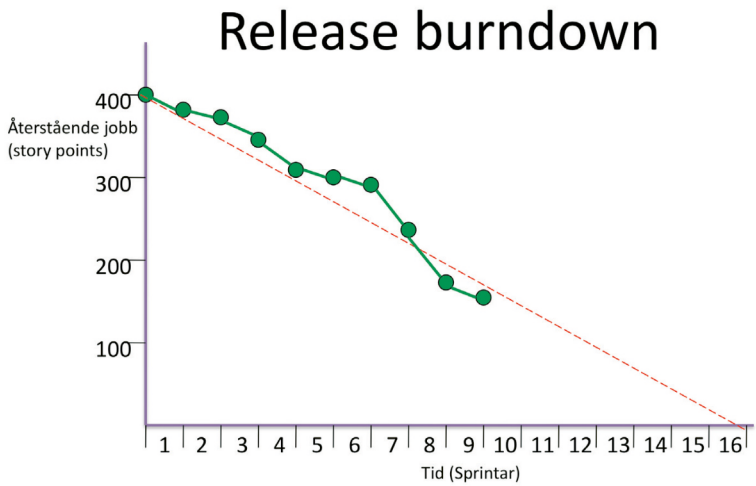
Efter sprintgranskningen tar teamet och produktägaren ett möte för att förbättra sättet att arbeta. Man reflekterar över vad som fungerat bra och vad som kan förbättras i nästa sprint för att tillsammans bli effektivare. Eftersom iterationerna är korta hinner teamet förbättra sig många gånger under ett projekt. Med dessa mötet behöver man inte vänta till projektslutet för att lösa saker som inte har fungerat. Bättre att förbättra processen under gång än att tänka att ”i nästa projekt ska vi göra rätt från början”. Ett angreppssätt är att brainstorma fram vad som varit bra, vad som varit mindre bra och vad vi kan göra för att förbättra eller eliminera de mindre bra sakerna. Mötet resulterar i en prioriterad lista med förbättringsförslag kallad hinderlista (impediment backlog) som scrum-mastern har som uppgift att se till att den blir avbetad. Om förbättringsarbetet riskerar att ta betydande resurser i anspråk under sprinten är det lämpligt att först diskutera med produktägaren när arbetet bör göras. Förhoppningsvis ser produktägaren den långsiktiga fördelen och har inga problem att acceptera en tillfällig kapacitetsminskning för en långsiktig kapacitetsökning. Som vanligt handlar det om kommunikation.

Därefter är det dags för nytt planeringsmöte och en ny sprint.

Lanserings-burndown (Release burndown)

Även om det normala utvecklingsarbetet är en ständig ström av förbättringar av en produkt, förekommer projekt med tydliga start och slut. Som tidigare nämnts föreslår Scrum att spika datumet (timebox) och

Låta innehållet variera. För att ändå få en fingervisning av hur projektet går i sin helhet och inte bara inom varje sprint, kan en lanserings-burndown tas fram. Den fungerar som sprint burndown men y-axeln anger totala antalet story points som återstår i projektet och x-axeln anger tidsenhet i veckor, månader eller sprintar. Efter varje sprint uppdateras diagrammet och vi kan se om vi ligger på budget eller om innehållet måste ändras. Om innehållet är låst och tiden är variabel kan lanseringsdatumet uppskattas genom att dra en linje som visar trenden.



En lanserings-burndown visar hur mycket jobb som återstår efter varje sprint. Med hjälp av grafen går det att få en bra känsla för när ett projekt kan lanseras.

Kontrakt med fast pris

En fråga som väldigt ofta kommer upp är hur Scrum fungerar med kontrakt med fast pris och fast innehåll. Svaret är att Scrum fungerar i alla fall inte sämre än andra vanliga processformer vid den sortens kontrakt. Med en lanserings-burndown kan kund och leverantör tidigt se om projektet verkar bli klart i tid eller ej. Om projektet ser ut att bli försenat kan de tidigt vidta de åtgärder som de tror mest på och förhoppningsvis minimera skadan. Det finns fyra variabler att styra med: antalet resurser (kostnaden), projektets omfattning, lanseringsdatum och kvaliteten. Den

sistnämnda är den som är svårast att se och ofta den som blir lidande. Samtidigt är det den som kostar mest i förlängningen. Därför föreslår Scrum att det är omfattningen som ska ändras i första hand och i andra hand lanseringsdatumet.

Bättre vore att utnyttja rörligheten i Scrum-projekt och erbjuda det som alternativ till fast innehåll. Kunden borde sätta mer värde vid att få ut en produkt som ger ett högt värde vid lansering snarare än en produkt med exakt det innehåll som beställdes.

Distribuerat Scrum - till exempel för off shoring

Traditionellt har den allmänna uppfattningen varit att Scrum endast fungerar när alla sitter på samma plats. Jeff Sutherland (Scrums skapare) har tillsammans med det Nederländska företaget Xebia gjort lyckade försök då teamdeltagarna suttit spridda inte bara på olika våningsplan i samma byggnad utan även i andra länder i annan tidzon och med annan kultur.

Det blir betydligt svårare och kräver erfarna utvecklare med god kunskap om mekanismerna bakom Scrum tillsammans med mycket god kommunikation. Men just svårigheterna kan även vändas till styrkor då alla måste anstränga sig för att det ska fungera.

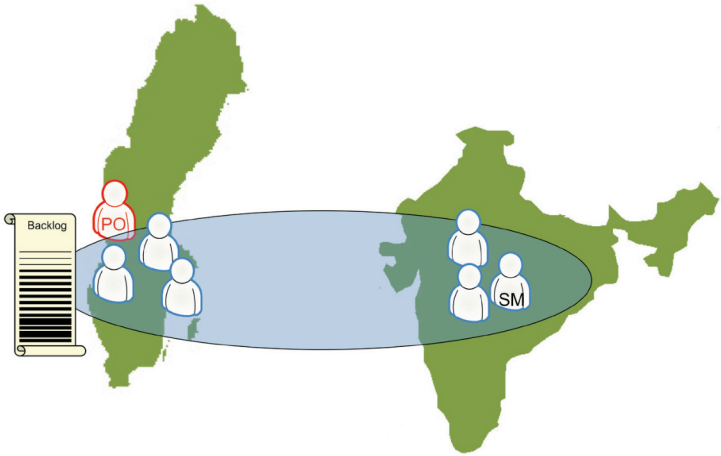
Två vanliga mönster man brukar använda sig av är distribuerade team och produktägare på annan ort.

Distribuerat team

Då kommunikationen mellan produktägaren och teamet är central väljer vi hellre att splitta utvecklingsteamet och sätta dem på olika platser, snarare än att låta teamet vara på en plats och produktägaren på en annan. Detta kan synas lite ointuitivt men har visat sig fungera bäst. Anledningen är att teamen tvingas lära sig samarbeta för att fungera. Det som visat sig vara den sämsta lösningen är att sätta vissa funktioner, t.ex. testning, på annan plats än övriga utvecklare.

Kommunikationen är A och O i samarbete så lägg stor insats där. Video- konferens är bättre än bara ljud. Kanske kan det vara en idé att ha en videokonferens som är ständigt uppkopplad så teammedlemmarna hela tiden ser varandra och lätt kan ställa frågor och diskutera lösningar.

För att införa ett distribuerat team är det viktigt att de först får lära känna varandra. När vi startar låter vi dem sitta på samma ort tillsammans med produktägaren i några sprintar, för att lära känna varandra och få till ett bra samarbete. Därefter sprider vi ut dem geografiskt.

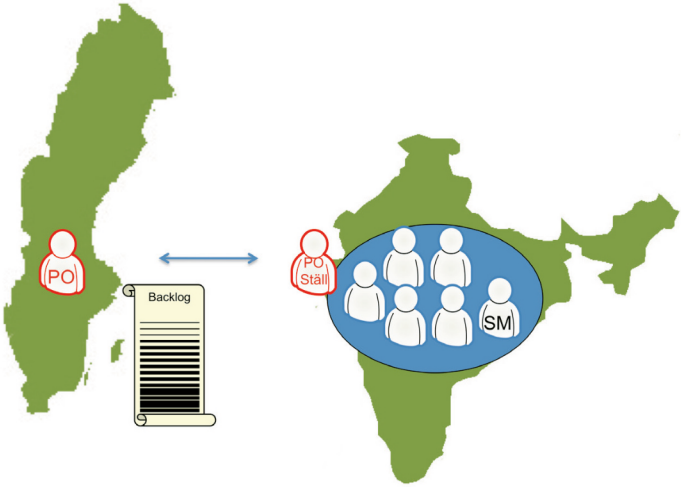


Delar av teamet sitter på annan geografisk plats än produktägaren och resterande team. Videokonferenser, telefonmöten och många resor hjälper till att överbrygga kommunikationsproblemen.

Finns möjligheten är det bra om hela teamet träffas vid varje sprintövergång. Hos en av våra kunder gjorde vi så att vi la sprintslut på en tisdag och sprintstart på en onsdag, vilket gjorde att de som satt på annan ort bara behövde göra en resa per sprint.

De försök som gjorts har visat att, om Scrum görs rätt, utan påtryckning och med team som lyckas självorganisera trots att de sitter på olika platser, då kan produktiviteten bli nära nog densamma som för team som sitter tillsammans. Intressant nog visade det sig att kundens önskemål fick mer fokus vid distribuerad Scrum. Förklaringen ligger i att den del av teamet som sitter nära produktägaren och kunden tvingades lära sig kundens önskemål så bra att de också kunde förklara för den halva av teamet som sitter långt borta.

Produktägare på annan ort med ställföreträdare (proxy)



En produktägare som befinner sig för långt bort för de dagliga arbetet utser en ställföreträdare som tar alla beslut som måste tas tills dess att man kan synkronisera sig nästa gång

Ibland har man personen som är ansvarig för prioriteringar på annan geografisk plats än teamet. Ett sätt att säkra snabba svar angående innehåll och prioritering är att ha en ställföreträdare (proxy) i samma lokaler som teamet. Proxyn tar de snabba dagliga beslut som dyker upp och har mandat att ta dessa beslut, tills dess att en uppdatering kan ske med den verkliga produktägaren. När man kan nå den riktiga produktägaren är det dock bäst att prata med denne, så att vi inte får en viskleksseffekt. Det är helt centralt att produktägaren och ställföreträdaren har förtroende för varandra, helst väljer produktägaren själv sin ställföreträdare.

Scrum för stora företag

I Scrum-guiden beskrivs hur arbetet ska utföras inom ett (1) Scrum-team. Naturligtvis går det att ha flera samarbetande scrum-team och de grundläggande scrum-principerna kan skalas. Det har gjorts på många ställen under många år. Ur dessa olika erfarenheter har det vuxit fram återkommande mönster och bra erfarenheter som olika personer har fångat upp, sammanfattat och givit namn åt. Tre olika sådana ramverk är LeSS, SAFe och Scrum@Scale.

Gemensamt för LeSS, SAFe och Scrum@Scale

I alla tre ramverken utgår man från att man i botten har vanliga Scrumteam som är tvärfunktionella och självorganiserande.

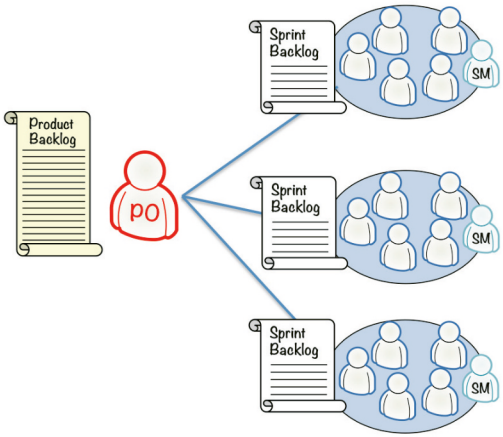
Man utgår också från att vi alltid försöker bryta ner kraven vertikalt, så att varje inkrement blir så litet som möjligt men ändå kan driftsättas separat.

Underförstått är även att man kör kontinuerlig integration och automatiserad regressionstestning, och att man efter varje sprint har en produkt som går att driftsätta ifall man så väljer.

LeSS (Large Scaled Scrum)

LeSS är ett skalningsramverk som kommer från Craig Larman och Bas Vodde och baserar sig på det jobb de gjorde inom finans- och telekomindustrin.

Minimalism och minsta möjliga process är något som genomsyrar hela LeSS. Lägg till så lite process som bara är mjöligt för att få fler-team-Scrum att fungera. LeSS består av ett minimalt antal regler och sedan tillkommer det en hel del guider och exempel på tillämpningar som man kan ösa ur i takt med att man växer.

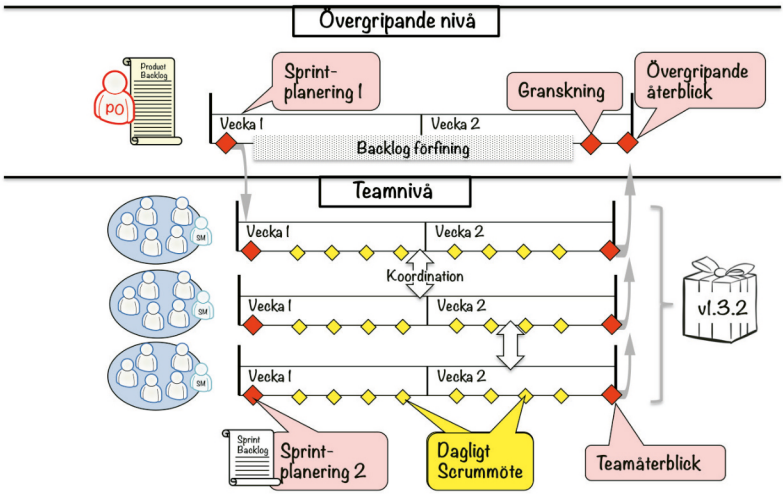


En produktägare med en produktbacklog och flera team, med var sin sprintbacklog

I grund och botten är LeSS ett förtydligande av hur Scrum är tänkt att fungera. Till exempel förtydligas det att teamen ska vara tvärfunktionella featureteam som är långlivade även över flera projekt och sitta tillsammans. Tanken är att hantera den komplexa situation, storskalig utveckling utgör, genom att förenkla så mycket det bara går. Därför föreslår LeSS att team kör synkroniserade sprintar och att alla teamen jobbar med samma produktägare och utifrån samma produktbacklog.

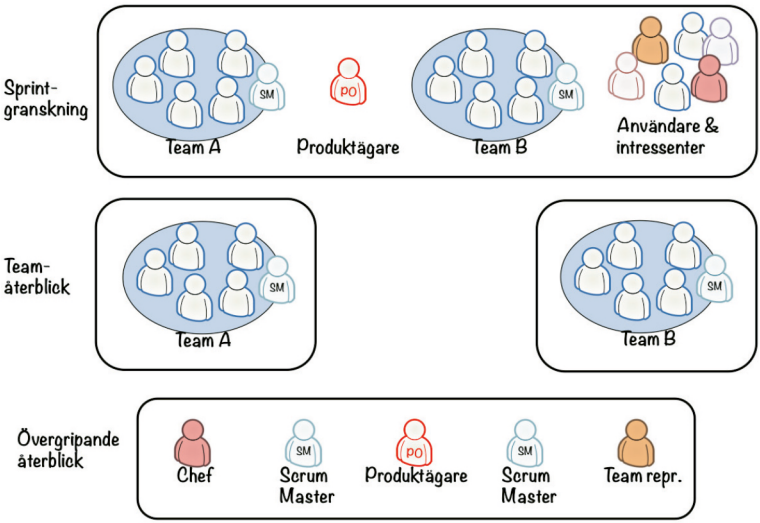
Det kan tyckas omöjligt att en enda person ska klara av att vara produktägare för upp till åtta team, men filosofin i LeSS är att genom detta upplägg kan den personen inte lägga sig i så mycket detaljer och behöver istället överlämna mer till teamen att ta reda på. Detta gör att utvecklarna tvingas komma närmare användarna vilket ger dem bättre förståelse för sina användares vardag, deras behov och problem. Därmed får man högre engagemang och mer anpassade lösningar med högre kvalitet och man arbetar effektivare. “Get more with LeSS” helt enkelt.

Processmässigt följer LeSS vanlig Scrum (tidsboxade iterationer, sprintplanering, dagliga scrummöten, sprintgranskning och återblicksmöte) med det tillägget att sprintplaneringens första del görs aning annorlunda: Under första delen träffas representanter från teamen och produktägaren för att välja ut vilka saker som de tror kan göra klart under sprinten. Andra delen av sprintplaneringen ägnar teamen åt att planera hur de ska utveckla funktionerna, precis som för vanlig Scrum. Om det finns många beroenden mellan team är de lämpligt att de sitter i samma rum när de planerar. Allt för att underlätta kommunikation mellan teamen.



Översikt över LeSS med synkroniserade sprintar.

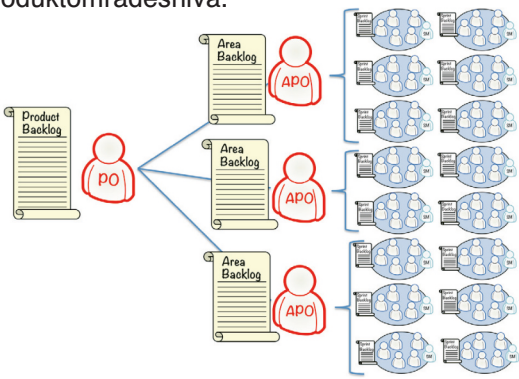
Även sprintslut behöver synkroniseras. Detta görs genom att Sprint Review är gemensamt för samtliga team. Återblicken delas upp i två delar på liknande sätt som Sprintplanering. Varje team har först ett teamåter-blicksmöte och sedan har man ett gemensamt återblicksmöte med repre-sentanter från samtliga team där man tar upp saker som inte går att lösa på enskild teamnivå.



Gemensamt sprintslut med alla team i LeSS

LeSS Hugh

Om vi har fler än åtta scrum-team delas produktbackloggen upp i olika kravområden med 4-8 team i varje. Varje område har då en områdes-produktägare (eng: area product owner) och en backlog precis som för vanlig LeSS. Rollen produktägare finns kvar och denne ansvarar för alla områden tillsammans. Processen (sprintplanering, daglig Scrum, granskning och återblick) körs på både övergripande och produktområdesnivå.



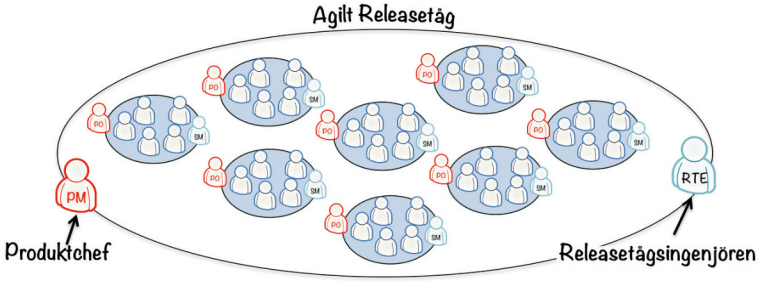
Produktägare med sina områdesproduktägare och deras team.

SAFe

SAFe är ett ramverk som kommer från Dean Leffingwell. Version 1 kom 2011 och 2019 var ramverket uppe i version 5.0 genom tilläggen om Design Thinking och Business Agility. Vi antar att ramverket fortsätter att utvecklas. Texten nedan baseras på SAFe version 5.0.

Organisation

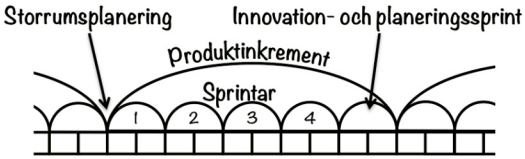
Det centrala i SAFe är att dela upp arbetet i värdeströmmar. En värdeström är de återkommande steg som sker i ett företag för att löpande leverera värde till kunder och användare. I en värdeström kan typiskt mellan 5 och 15 team jobba ihop. Denna gruppering av team kallas releasetåg. Ett releasetåg kan innehålla upp till 150 personer. Därefter rekommenderas det att ha flera Agila releasetåg inom samma värdeström.



Agila releasetåg är alltså en organisationsform, och releasetågsingengör är en faciliterande coachande roll, som i andra organisationer ibland kallas Agile coach.

Process

Inom det Agila releasetåget jobbar teamen enligt vanlig Scrum men SAFe tillför också programinkrement vilket är en överliggande planeringshorisont på typiskt 10 veckor även om kortare och längre intervaller förekommer. Skaparna av SAFe rekommenderar en

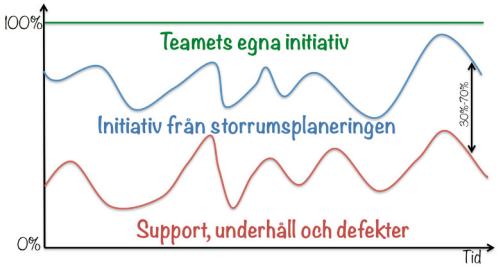


Vecka för vecka hur sprintar och programinkrement hänger ihop

planeringshorisont på mellan 8 till 12 veckor baserat på deras erfarenheter inom området. Produktinkrementet börjar med storrumsplanering, ett evenemang på 1-2 dagar då teamen tillsammans planerar de närmsta 10 veckorna. Typiskt börjar mötet med att produktcheferna sätter visionen följt av att teamen gör planeringen och löser beroenden sinsemellan. Om oklarheter uppstår finns produktcheferna hela tiden på plats för att omedelbart fatta beslut. Produktinkrementet faciliteras av en releasetågsingenjör.

För att få en balans mellan förutsägbarhet, tekniska behov och flexibilitet föreslår SAFe två saker:

1. Planera inte sista sprinten i produktinkrementet utan spara den sista sprinten till att göra klart saker som inte hunnits med, hjälpa produktägare planera kommande inkrement och utöver detta testa nya saker.
2. Fyll inte sprintarna till 100% utan snarare 30-70%. Detta för att ge rum åt att rätta defekter, användarsupport och annat oförutsägbart. Det ger också utrymme för att göra saker som produktledningen tar för givet görs, men som ofta inte får någon dedikerad tid, t.ex. Uppgraderinga av verktyg och anpassningar till nya plattformar.

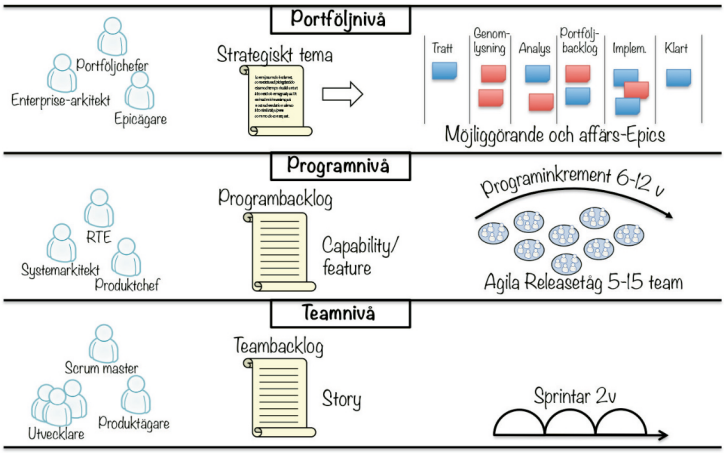


Ett typiskt scrum-team har tre typer av saker i sin sprintbacklog. Buggar som går före allt, saker från storrumsplaneringen, samt sådant teamet vet borde göras men ingen annan har egentlig insikt eller åsikt om (t.ex. uppgradering av operativsystem eller databaser).

Portföljhantering

För att säkra att företagets alla initiativ syftar till samma mål har SAFe en portföljnivå. Där jobbar arkitekter, portföljhanterare och initiativägare tillsammans utifrån den strategi och vision företagsledningen tagit fram.

De håller reda på alla initiativ med hjälp av en Kanbantavla som innehåller både möjliggörare (d.v.s. tekniska initiativ som underlättar framtida utveckling) och affärsinitiativ (d.v.s. funktioner som kunderna direkt märker). På portföljnivå har man kontroll över hela kund- och användar-processen, från idétratten till lanserad funktionalitet. När en epic hamnar i statusen "portföljbacklog" är den en kandidat för att brytas ner och hamnar i backloggen som en eller flera förmågor/funktioner. De epics som helt eller till del utvecklas i ett produktinkrement samlas i kolumnen "implementation".



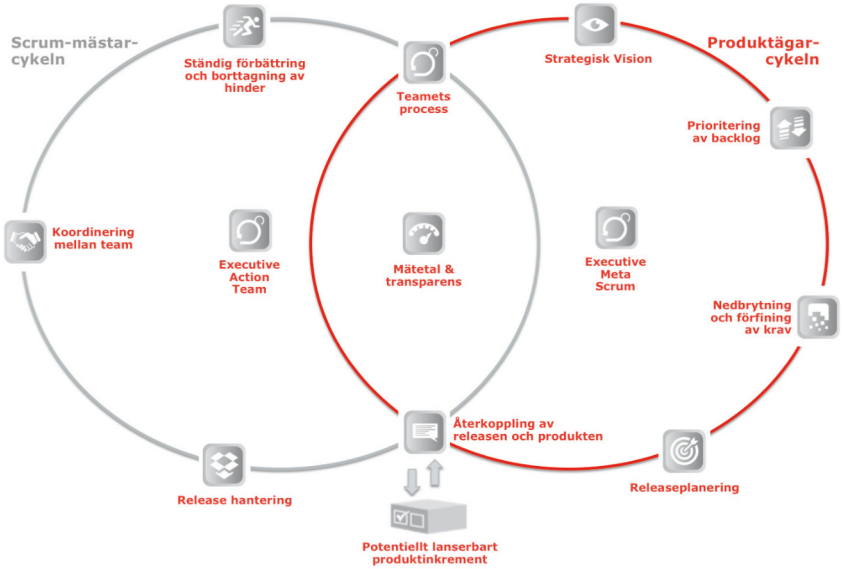
Sammanställning av SAFe i en bild som beskriver de tre nivåerna (storskalig SAFe har även nivån "värdeström" som ligger mellan portfölj- och programnivån), vilka roller som typiskt finns på varje nivå, deras arbetssätt och vad kraven kallas på respektive nivå.

Många roller är det

Vid en första anblick kan det kännas övermäktigt med alla roller, nivåer, möten och artefakter. Detta gör att det finns risk för att man gör en SAFe-implementation som blir både stel och tung. Men man bör skilja på individ och roll. En person kan med fördel ha flera roller och rollerna i bilden ovan kan bemannas av ett fåtal personer. Om man har en bra förståelse för de grundläggande principerna i Lean och Agile kan man anpassa SAFe och få till en både smärt och smidig implementation.

Scrum@Scale

Scrum@Scale kommer från en av Scrums grundare, Jeff Sutherland, och är den lättviktigaste av de skalningsramverk vi tar upp. Den beskriver hur organisationen bör skalas och vilka områden som behöver lösas för att få Scrum att skala utan att gå djupt in i detaljerna.



Områdena är uppdelade i scrum-mästarcykeln (vänstra) och produktägarcykeln (högra) samt några principer kring mätvärden och transparens. Bild med tillstånd från Jeff Sutherland och Scrum Inc.

Scrum-mästarcykeln – koordinera ”hur”

- **Koordinering mellan team** – Representanter, oftast Scrum-mästarna, från varje team, utgör den virtuella gruppen Scrum of Scrums (SoS). De samlas dagligen i det skalad dagliga Scrum-mötet som hålls efter teamens egna dagliga Scrum-möten. Här avhandlas hinder, beroenden och den kommande gemensamma releasen. Facilitator för SoS är Scrum of Scrums mästaren. Vid fler än fem team kan det vara läge att fundera på att skapa flera SoS. Då kan det vara bra med en övergripande grupp, här kallad Scrum of Scrum of Scrums.
- **Executive action team (EAT)** - Högsta nivån av Scrum-mästarcykeln. Består av personer med beslutsmandat som tar tag i de förbättringar

som teamen, produktägare och scrum-mästare inte själva har möjlighet att genomföra. EAT ansvarar också för att omvandla hela organisationen till att bli Agil i sitt affärstänk.

- **Ständig förbättring och bortagning av hinder** – Hur säkrar vi att alla team är medvetna om sin förmåga och ständigt försöker förbättra den? Hur säkrar vi att det som teamen kommer på blir löst även om det gäller problem som ligger utanför deras sfär av påverkan?
- **Releasehantering** – Hur synkar vi mellan team för att få till smärtfria lanseringar?

Produktägar-cykeln – koordinera ”vad”

- Prioritering – Precis som Scrum mästarna, skalar även produktägarna upp genom att produktägarna skapar ett team. Här kallas nivåerna över teamet för MetaScrum och den som leder MetaScrum-grupperna kallas Chief Produkt Owner (CPO). På MetaScrum-nivå beslutas om vilken prioritering som gäller för samtliga team som ingår.
- **Executive metascrum(ems)** - Högst upp i hierarkin av produktägare finns personer som tar ett perspektiv på vision och roadmap som berör samtliga team. Sätter företagets gemensamma strategier.
- **Strategisk vision** – Vart ska vi med företaget och våra produkter?
- **Nedbrytning och förfining av krav** – Hur fördelas arbetsuppgifterna mellan teamen och hur genomför vi sessioner för förberedelse av dem?
- **Releaseplanering** – Hur ska vi planera för våra leveranser? Vad kommer när?

I båda cyklerna finns

- **Teamets process** - Baserat på ramverket Scrum avgör teamet själva hur de jobbar.
- **Återkoppling av releasen och produkten** – Hur får vi återkoppling från användandet och mottagandet av det som lanserats och hur får vi ut den informationen till teamen?
- **Mätetal och transparens** – Hur ska vi mäta att vi bygger rätt saker och hur vi ser vi till att alla har tillgång till dessa mätvärden?

SAFe, LeSS eller Scrum@Scale eller något eget?

Oavsett vilket ni väljer så kommer ni att behöva ändra och komma på egna lösningar då ingen företagskultur är den andre lik.

LeSS är en bra startpunkt när man har Scrum på plats och börjar skala upp med fler team, ett i taget.

I LeSS skalar man upp med minimal extra process jämfört med ett-team Scrum. Som jämförelse kan man kan säga att SAFe är mer ett komplett ramverk, och sedan tar man bort det som inte behövs.

Två exempel på anledningar att välja SAFe före LeSS:

1. Man behöver koordinera en väldigt stor organisation av hundra- eller kanske tusentals människor. SAFe ger en tydlig väg framåt här för hela kedjan från företagsstrategi till produktstrategi till taktik till vad vi skall göra nästa sprint.
2. Man är utan någon djupare erfarenhet av Agil utveckling och vill ha ett ramverk med förslag på hur man kan hantera nästan varje aspekt av en helt Agil organisation. SAFe har mycket “så här kan man göra” beskrivet.

Scrum@Scale ser vi mest som ett diskussionsramverk som är användbart för de som redan idag har flera scrum-team på plats och som fungerar till-sammans men som gärna vill hitta saker och områden att förbättra och att föra diskussioner kring. Alternativt om du själv vill beskriva er process, så kan du utgå från Scrum@Scale för att inte missa något perspektiv.

Som alltid när det gäller Agilt så är det viktigt att anpassa metoden efter era förutsättningar, att ständigt reflektera över situationen och förbättra hur ni jobbar tillsammans. Ifall ni precis börjat med Scrum och ni har en kultur där det är viktigt med roller och befogenheter är troligen SAFe ett bra val. SAFe kanske också är rätt val ifall samarbetet mellan teamen är på gång att haverera och ni har svårt att komma överens om en väg framåt. Genom att välja SAFe tar ni många beslut om roller och processen i ett beslut och det kan vara enklare att ledas av en välbeskriven metod än att gemensamt besluta om arbetssätt.

Scrum Pattern

Bakgrund

Eftersom Scrum är ett ramverk snarare än en färdig process lämnar det mycket till varje team, avdelning och företag att komma på hur de ska forma sin process så den fungerar i deras sammanhang. För att få inspiration samlar några personer i Scrum-gemenskapen på framgångsrika tips som de publicerar som mönster (patterns) på sidan http://www.scrumplop.org/.

Här listar vi några av de vanligaste mönstren:

- Stabila team - Alla ingår i teamet till 100% under en längre tid.
- Gårdagens väder - Du hinner med lika mycket i denna sprint som förra.
- Daglig städad kod - Fixa funna buggar inom ett dygn för att ha buggfria sprintreleaser.
- Team som når sitt sprintmål innan sprintens slut accelererar sitt lärande och bygger självförtroende. De som tar på sig för mycket stagnerar i sitt lärande.
- Välbefinnande index - Se nedan.
- Svärma - Se nedan.
- Avbrottsmönstret - Se nedan.
- Oförutsedd händelse - Se nedan.
- Scrumming the scrum - Se nedan.

Välbefinnandeindex (Happiness metric)

Under återblicksmötet fokusera man på den sak som kommer att öka teamets välbefinnandet mest och implementera den i nästa sprint. Typiskt går det till så att varje teammedlem säger på en femgradig skada hur tillfreds de är och vilken förbättring i arbetsätt skulle öka hens värde. Teamet tar sedan och kommer fram till vilken sak som skulle öka den totala nivån mest.

Svärma (Swarm)

Att hela teamet "svärmar" runt en story betyder att de, som ett team, jobbar med en sak i taget. Det blir som det som inom Lean kallas "one piece flow". Här kommer några av fördelarna med att "svärma":

- Den blir redo för test tidigare vilket ger en mer utjämnad arbetsbelastning för de som testar.
- Kunskap sprids inom teamet
- Teamet blir tightare då samarbete krävs
- Det blir roligare daily Scrum-möten eftersom alla är mer beroende av och har kunskap om varandras arbetsuppgifter
- För centrala personer i teamet (t.ex. experter och produktägare) blir det mindre ställtider (eng: switching time)

Avbrottsmönstret (Interrupt)

Även om det bästa vore att aldrig bli störd så är det sällan fallet. Därför är det en fördel att ha en uttalad process som beskriver hur störningar ska hanteras och som samtidigt ger en anledning att minska dem. En första åtgärd kan vara att minska sprint-längden. Det som inte kan vänta fyra veckor kanske kan vänta två veckor. När det inte går att minska sprinten mer kan du ta till den här processen för att hantera störningar i form av oplanerat arbete som ska utföras.

Under sprintplaneringen kommer produktägaren och utvecklarna överens om hur mycket av sprinten som ska läggas i en buffer för oplanerat arbete. T.ex. 20%.

När akuta saker dyker upp bestämmer produktägaren om det ska göras nuvarande sprint eller om det kan vänta. Om det måste genomföras nuvarande sprint kan det antingen tas med teamet genast, vilket blir en kraftig störning, eller tas vid nästa dagliga Scrum, vilket är en mindre störning. När jobbet med den oplanerade uppgiften är klar räknas bufferten ner med hur många poäng som arbetet faktiskt tog.

Om bufferten tar slut avbryts sprinten och teamet får samlas till ett nytt sprintplaneringsmöte. Just den här åtgärden är den kraft som gör att produktägaren och intressenter håller hårt emot när behov för oplanerat arbete uppstår. Ingen vill bryta en sprint i onödan då de inte får de planerade sakerna levererade.

Oförutsedd händelse

Om bufferten har tagit slut eller alla signaler visar att sprintmålet inte kommer att uppnås är det dags att ta till processen för nödsituation. Nedan är ett exempel på hur en process för nödsituationer skulle kunna se ut (gör bara så mycket som behövs):

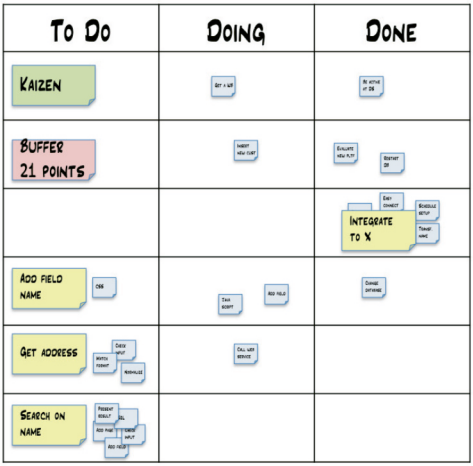
1. Ändra hur arbetet utförs. Gör något annorlunda.
2. Få hjälp, oftast genom att lägga över delar av jobbet till någon annan.
3. Säkra att det går snabbt att få svar på frågor.
4. Minska mängden arbete.
5. Avbryt sprinten och planera om.
6. Informera ledningen hur lanseringsdatum kommer att påverkas.

De bästa teamen fokuserar på att hitta lösningar enligt punkterna 1 till 3.

Scrumming the Scrum

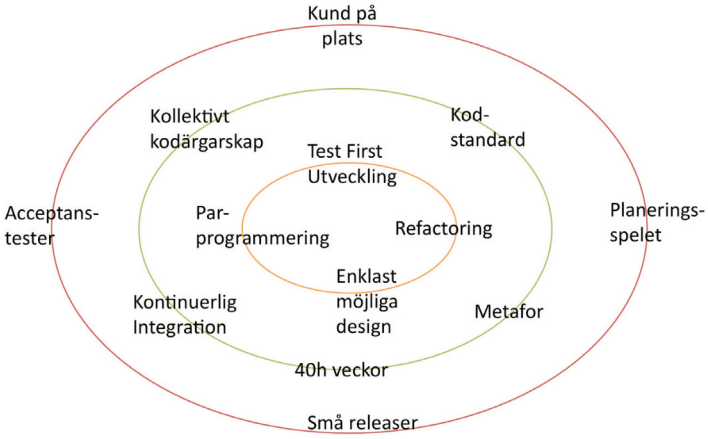
Från retrospektiven har teamet ofta med sig konkreta saker som de tror kommer förbättra teamets arbetsätt. För att se till att det faktiskt blir utfört så ser många team till att det är en del av sprintplanen, och placeras på Scrum-tavlan som vilken arbetsuppgift som helst. Eftersom förbättringar är centralt i Scrum är rekommendationen att ha förbättringsarbetet, ”Kaizen”, som prio nummer 1.

Sammantaget ger de utvalda mönstren en Scrum-tavla som kan se ut så här ut så här:



En Scrumtavla som följer Scrummönstrena Scrumming the Scrum (följ upp förbättringar, ”kaizen” på tavlan) samt Avbrott (ha en förbestämd storlek, ”buffer”, på hur mycket avbrott som kommer tillåtas under sprinten).

XP – eXtreme Programming



XPs praktiska moment. De flesta i den yttre cirkeln (Kund på plats, planneringsspelet små releaser) överlappar det Scrum har. Acceptanstester plus de inre kan alla med fördel användas inom ett Scrum-team.

Extreme programming är som sagt en helt egen metod. Den liknar Scrum men innehåller mycket mer. Själva projektplaneringen skiljer sig inte s mycket från Scrum, men XP tar även upp hur man som team bör organisera sig för att arbeta effektivt. En allt vanligare kombination är Scrum + valda delar av XP.

Att införa Scrum ställer höga krav på programmerarna eftersom de ska samarbeta i samma kodbas och iterativt ta fram en funktion i taget. Koden måste vara mycket lätt att ändra i utan att man inför en massa defekter.Detta ställer höga krav på både samarbetet inom teamet och på kodkvaliteten. XP är en metod som ser till att detta verkligen fungerar.

Allt fler kombinerar Scrum med XP helt enkelt för att det kan vara svårt att få Scrum att fungera bra utan XP:s tekniker, framförallt parprogrammering och automatiserade tester.

XP har drygt tio olika moment. Här tar vi upp några få – för en mer komplett bild rekommenderas Kent Becks bok ”eXtreme Programming eXplained” Second Edition och sajten http://www.xprogramming.com/xpmag/whatisxp.htm

En av utmaningarna när man arbetar lättrörligt är att man hela tiden behöver ändra i sin kod. Ibland inte bara små förändringar, utan tämligen stora eftersom man hela tiden lägger till nya delar inkrementellt. XP:s olika moment ser till att samverka för att göra detta så smidigt och billigt som möjligt.

Parprogramming

Par-programmering innebär att varje rad kod som ska bli en del av produkten, produktionskod, ska skrivas tillsammans med en kollega. Det är två personer, en dator och gemensam design och kodning.

Detta kan tyckas vara ett långsamt sätt att arbeta på. Men det beror på vilket perspektiv man har till kodning. Om målet är att knacka ner så många rader kod man kan på en dag, gör två personer på varsitt håll detta snabbare än två personer vid samma dator. Men så går det inte till när man programmerar, även om många – även programmerare – tror det.

Ser man över en period, kommer en programmerare att vilja bolla idéer, göra design, få den bekräftad av andra, diskutera vad kraven fler

egentligen betyder, så småningom skriva kod, hitta buggar, rätta buggar, rätta buggar och så vidare. Bara en del av all tid går åt till att skriva själv koden.
Det mesta är andra saker.

När man parprogrammerar gör man allt det här på en gång vid datorn i små, små bitar. Det kan kännas som om man är mindre produktiv, eftersom man ibland sitter längre tid vid tangentbordet och dessutom är två. Men räknat över en period sparar man tid, eftersom man gör design och förstår saker bättre, kodar bättre, skapar färre buggar och hittar fel snabbare när man är två.

Går det dubbelt så fort när man är två med allt det här? Ja, faktiskt och ibland ännu fortare. Räknar man dessutom med den tid man sparar genom att andra inte drabbas av programmerarens misstag såsom felrapportmöten, testare och kunder blir tidsbesparingen ännu större.

Parprogrammering sprider kunskap om den kod som skrivs och vad som är sund programmering. Vill man ha ett företag där alla lär sig hela tiden, och där man undviker att koden kollapsar för att någon blir sjuk eller säger upp sig, då är parprogrammering ett effektivt sätt.

Till sist får man dessutom ut en produkt av högre kvalitet. Eftersom kvalitet är en nyckelegenskap och nödvändig för att man ska kunna arbeta Agilt lönar sig parprogrammering ytterligare. Det är enklare att skriva kod med hög kvalitet från början än att i efterhand höja kvaliteten genom testande.

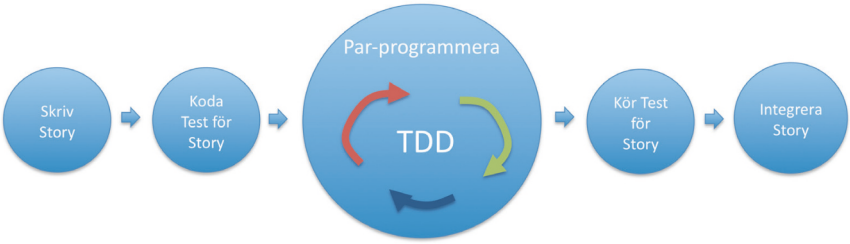
Testdriven utveckling (TDD)

Testdriven utveckling är att skriva små automatiserade tester samtidigt som man skriver sin kod. Först ett litet test, därefter lite kod, ett litet test till, lite mer kod och så vidare.

Att göra detta på ny kod är inte svårt, det är till och med så att man snabbt blir mer produktiv på detta sätt, och kvaliteten blir oftast högre (färre defekter) än vad man är van vid.

Så fort man ska göra en ändring i koden, kör man först alla befintliga tester för att se att allting är som det ska. Därefter skriver man ett litet test som testar den nya funktionaliteten man vill ha in och kör det (det fallerar). Därefter skriver man lite kod, till dess att även det nya testet

fungerar. Till slut städar du upp i koden (refactor) så att den är snygg och prydlig för framtiden. Medan du gör detta ser du till att inget test går sönder. Typiskt hinner man fyra fem sådana här cykler på en timme.



Arbetsflödet i ett team. Först tar vi en story från backloggen, därefter skriver vi ett acceptanstest för denna story. Sen har vi några timmars testande och kodande, varvid vi avslutar med att köra acceptanstestet och därefter integrerar vår kod med projektets.

Fördelarna med eXtreme Programming

Parprogrammering gör att man får:

- En tydligare, enklare och klarare design.
- Kontinuerlig kodgranskning.
- Ett gemensamt ägaransvar för koden.
- Ständig kunskapsöverföring.

Testdriven utveckling gör att utvecklarna blir trygga i att inte ha sönder något medan de håller på att ändra. En bieffekt när utvecklarna känner sig
trygga är att de vågar ta ut svängarna och blir därmed effektivare.
Det XP gör är bland annat att definiera ett arbetsflöde internt i ett team. Ett område som Scrum lämnar helt öppet, vilket gör att XP passar som handen i handsken tillsammans med Scrum.

Ett par avslutande ord

Vi hoppas du uppskattat boken och att vi givit dig lite inspiration till att förändra det sätt som ni jobbar på. Kom ihåg att Agile och Lean inte bara är ett nytt sätt att jobba på utan också ett nytt sätt att tänka och organisera sig. Även om de regler och principer vi beskrivit kan tyckas enkla att tillämpa krävs ofta en stor insats som genomsyrar hela företaget för att få det att fungera. Vår kollega, Henrik Kniberg, skrev på sin blogg att Scrum är som schack: tar ett par timmar att lära sig men en livstid att behärska. Vi kan inte annat än hålla med. Men även ett litet steg i rätt riktning är bättre än att stå stilla.

Lycka Till!

Litteraturlista

Mary & Tom Poppendieck: Implementing Lean Software Development
Mike Cohn: Agile Estimating and Planning
Kent Beck: Extreme Programming Explained (second edition)
Peter R. Scholtes: The Leader's handbook
Patrick Lencioni: Five Dysfunctions Of A Team
Jeffrey Liker: The Toyota Way
Henrik Kniberg: Scrum and XP from the Trenches
Tomas Björkholm: Kanban in 30 days
Eric Ries: The Lean Startup
Peter Senge: The Fifth Discipline
Jeff Sutherland: Scrum: The Art of Doing Twice the Work in Half the Time
Jeff Patton: User Story Mapping
John Seddon: Freedom from Command and Control
Daniel Pink: Drive
David J. Anderson: Kanban
Niklas Modig, Pär Åhlström: Detta är Lean
Stanley McChrystal: Team of Teams
Stephen Bungay: Art of Action
David Marquet: Turn the Ship Around
Carol Dweck: Minset
Amy Edmondson: The Fearless Organization