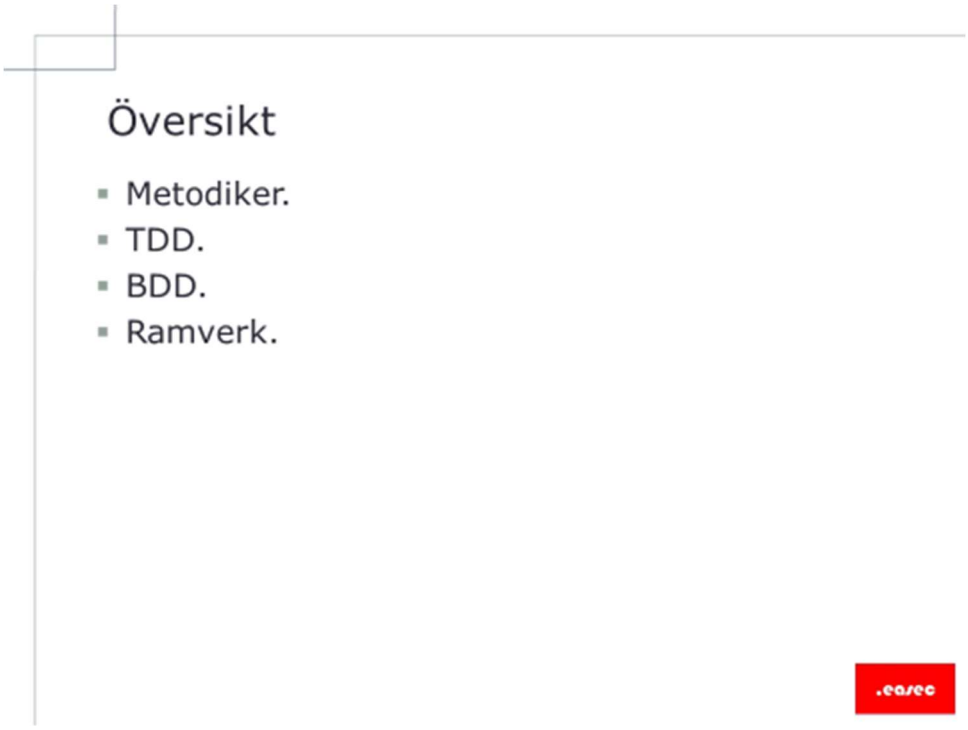
MODUL 14 PROJEKTMETODIK 1

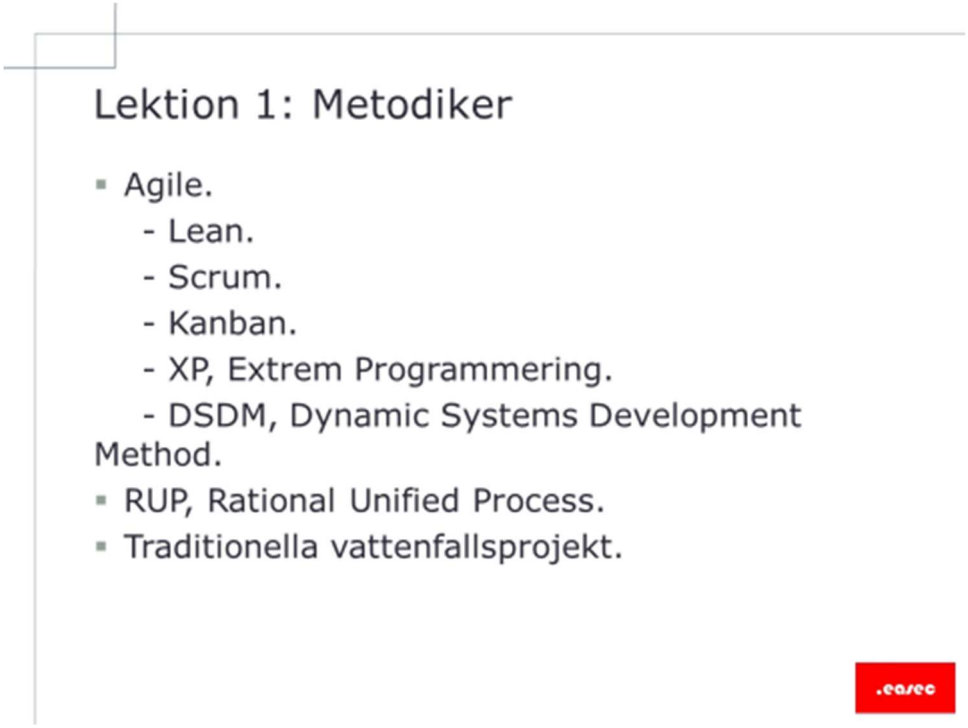
Modul 14 Projektmetodik



Översikt

Denna moduls mål:

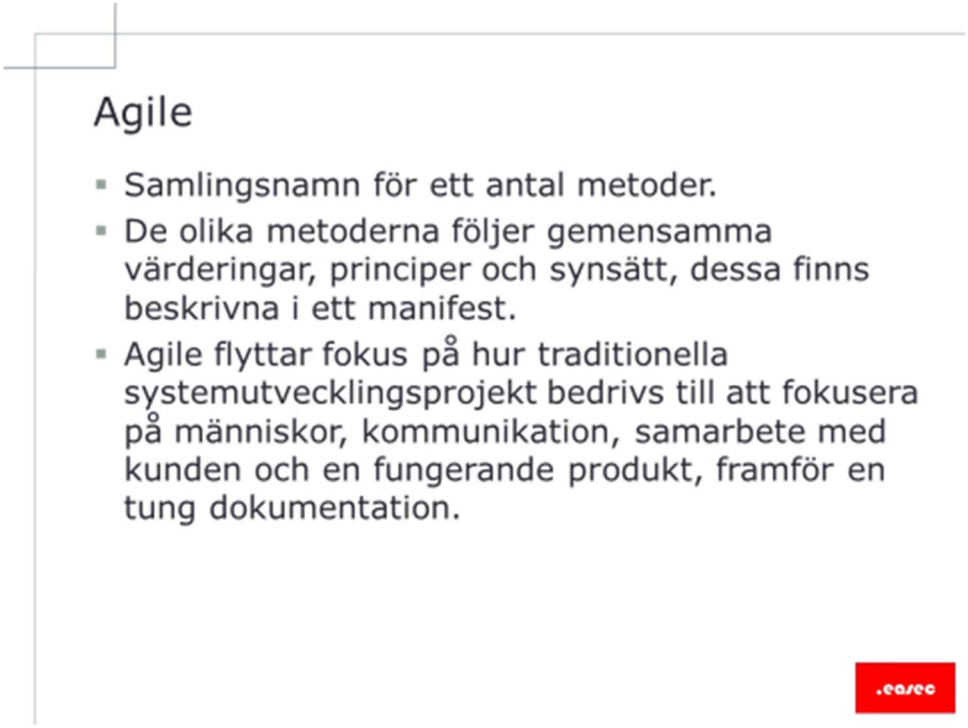
- Få kunskap om olika metodiker för ett projekt.
- Få kunskap om Test-Driven Development (TDD).
- Få kunskap om Behavior-Driven Deplyment (BDD).
- Få kunskap om ramverk.



Lektion 1: Metodiker

I denna lektion skall vi titta på:

- Agile och agila projektmetoder.
- RUP, Rational Unified Process.
- Traditionella vattenfallsprojekt.

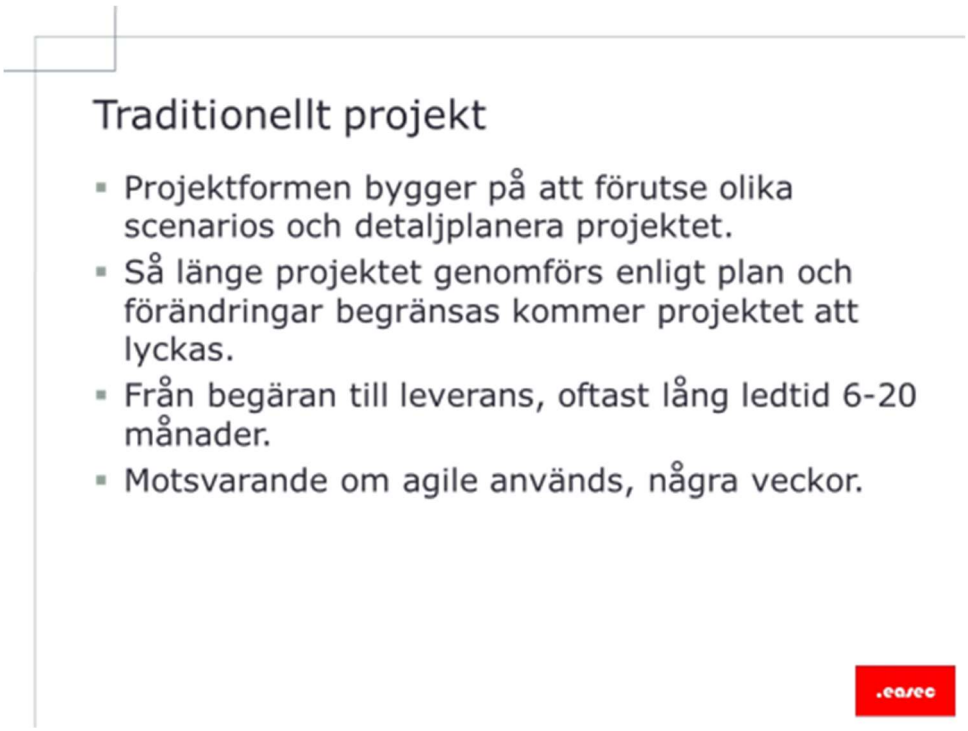


Agile

Samlingsnamn för ett antal metoder.

De olika metoderna följer gemensamma värderingar, principer och synsätt, dessa finns beskrivna i ett manifest.

Agile flyttar fokus på hur traditionella systemutvecklingsprojekt bedrivs till att fokusera på människor, kommunikation, samarbete med kunden och en fungerande produkt, framför en tung dokumentation.



Traditionellt projekt

Projektformen bygger på att förutse olika scenarios och detaljplanera projektet.

Så länge projektet genomförs enligt plan och förändringar begränsas kommer projektet att lyckas.

Från begäran till leverans, oftast lång ledtid 6-20 månander.

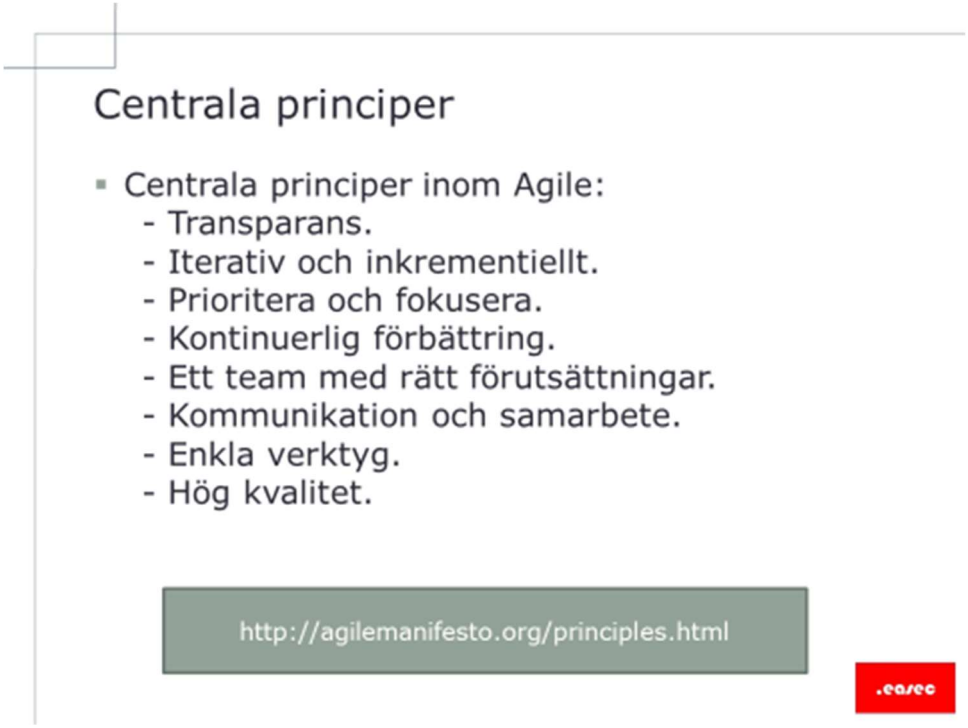
Motsvarande om agile används, några veckor.

Syfte och struktur

Inom Agile finns det ett antal olika metodiker, dessa beskriver hur projektet kan arbeta agilt.

Vanligaste metodiken i dag är Scrum, men även Extrem Programmering, DSDM och Kanban används.

Metodikerna har mycket gemensamt för att skapa de bästa möjliga förutsättningarna för projektet.



Centrala principer

Centrala principer inom Agile:

- **Transparans**, öppenhet mellan kund och beställare skapar tillit, vilket är en av grundstenarna för en bra relation och ett värdefullt samarbete.

- **Iterativ och inkrementiellt – tidiga och kontinuerliga leveranser**, vilket innebär att planer tas fram stegvis, leverans sker tidigt och kontinuerligt i form av del av slutprodukten.

- **Priotitera och fokusera**, prioritera om mellan iterationer för att leverera det som är viktigast just nu. Fullt fokus på vad som skall göras under iterationen utan störande moment eller prioriteringar som ändrats. Detta skapar ett produktivt team som kan hantera förändringar.

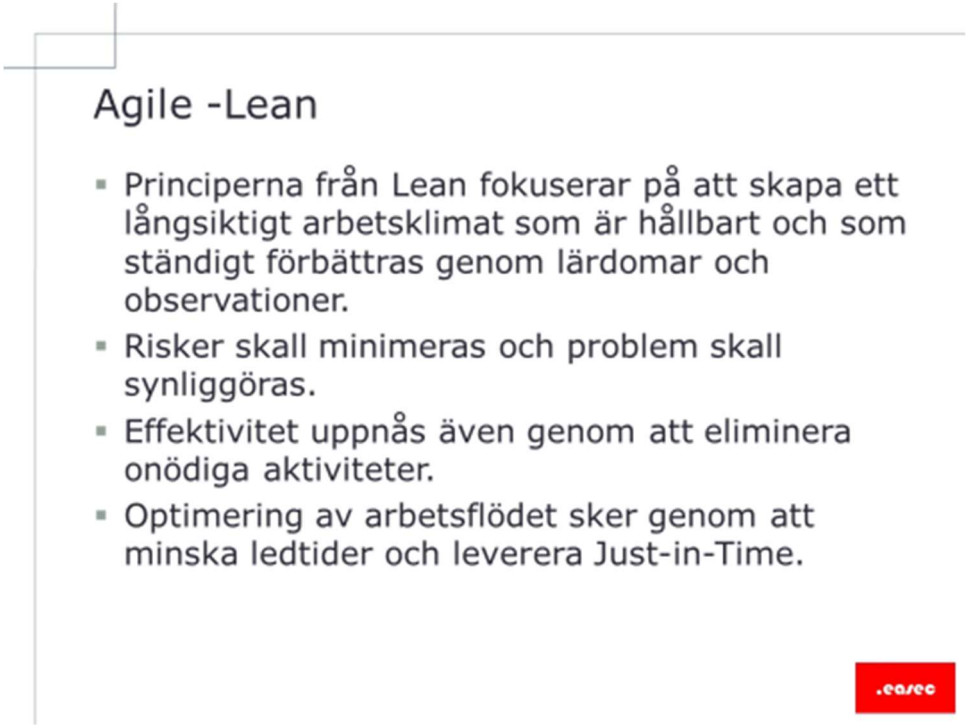
- **Kontinuerlig förbättring**, utvärdera det som har levererats, genom ett öppen diskussion om arbetet i projektet och vad som kan bli bättre leder till ett bättre projekt. Förbättringar är en del av projektet.

- **Ett team med rätt förutsättningar**, ett team med rätt kompetens som organiserar sig själv, skapar den bästa slutprodukten. Ge teamet de rätta förutsättningarna och undanröj eventuella hinder. Detta kommer att ge i slutändan ett högpresterande team.

- **Kommunikation och samarbete**, direkt kommunikation mellan personer är den bästa formen av kommunikation. Samarbeta inom teamet, med kunden och externa partners.

- **Enkla verktyg**, fokusera inte på verktyg som riskerar att begränsa arbetet.

- **Hög kvalitet**, tid och kvalitet är fasta faktorer i en iteration. Prioritera hellre bort funktionalitet än kompromissa med kvalitet och yrkesstolhet.



Agile -Lean

Principerna från Lean fokuserar på att skapa ett långsiktigt arbetsklimat som är hållbart och som ständigt förbättras genom lärdomar och observationer.

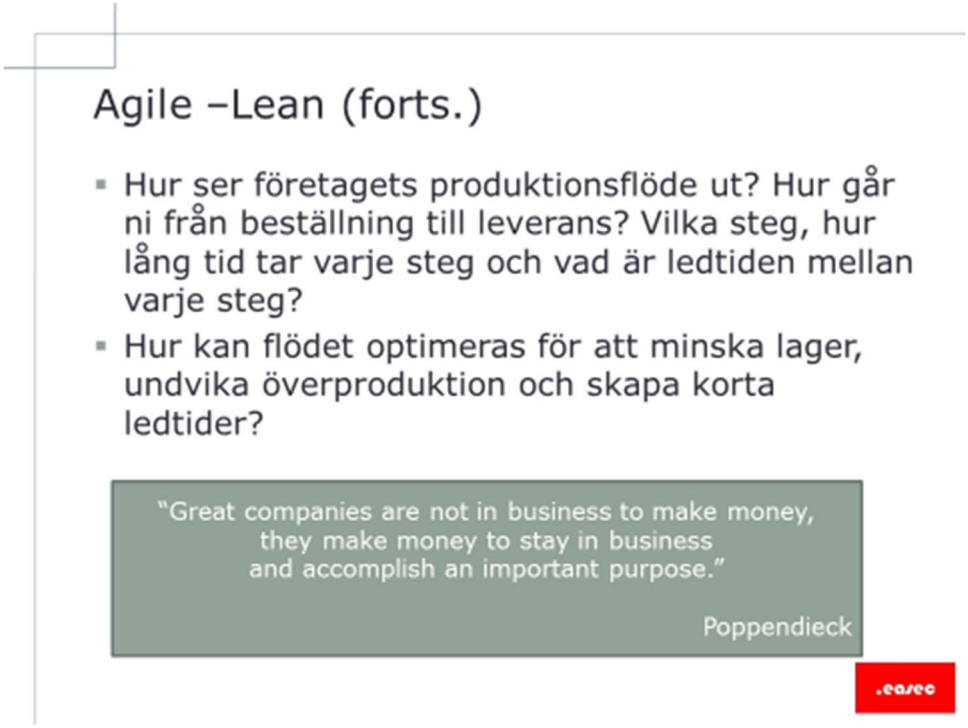
Risker skall minimeras och problen skall synliggöras.

Effektivitet uppnås även genom att eliminera onödiga aktiviteter.

Optimering av arbetsflödet sker genom att minska ledtider och leverera Just-In-Time.

Har sitt ursprung från Toyota, då Toyota med många andra industrier i Japan fick lägga om sin produktion efter andra världskriget. Bilfabrikanten tog fram ett system för produktion, Toyota Production System (TPS), uttrycket Lean myntades 1979 när MIT gjorde en studie som jämförde olika biltillverkare världen över.

Begreppet blev internationellt känt 1988. Modell har sedan dess kommit att användas även utanför tillverkningsindustrin, bland annat inom sjukvården.



Agile –Lean (forts.)

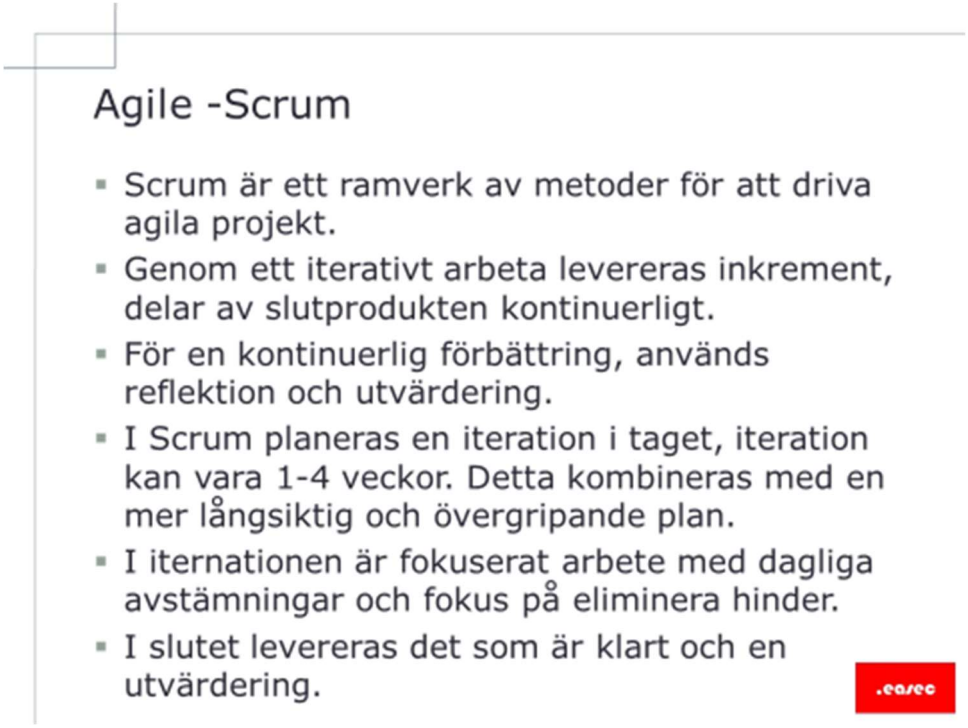
Hur ser företagets produktionsflöde ut? Hur går ni från beställning till leverans? Vilka steg, hur lång tid tar varje steg och vad är ledtiden mellan varje steg?

Hur kan flödet optimeras för att minska lager, undvika överproduktion och skapa korta ledtider?

Lean och programutveckling

En tillämpling av Lean för just programutveckling är gjord av Mary och Tom Poppendieck.

Lean Software Development är en av grundstenarna för Agile.



Agile -Scrum

Scrum är ett ramverk av metoder för att driva agila projekt.

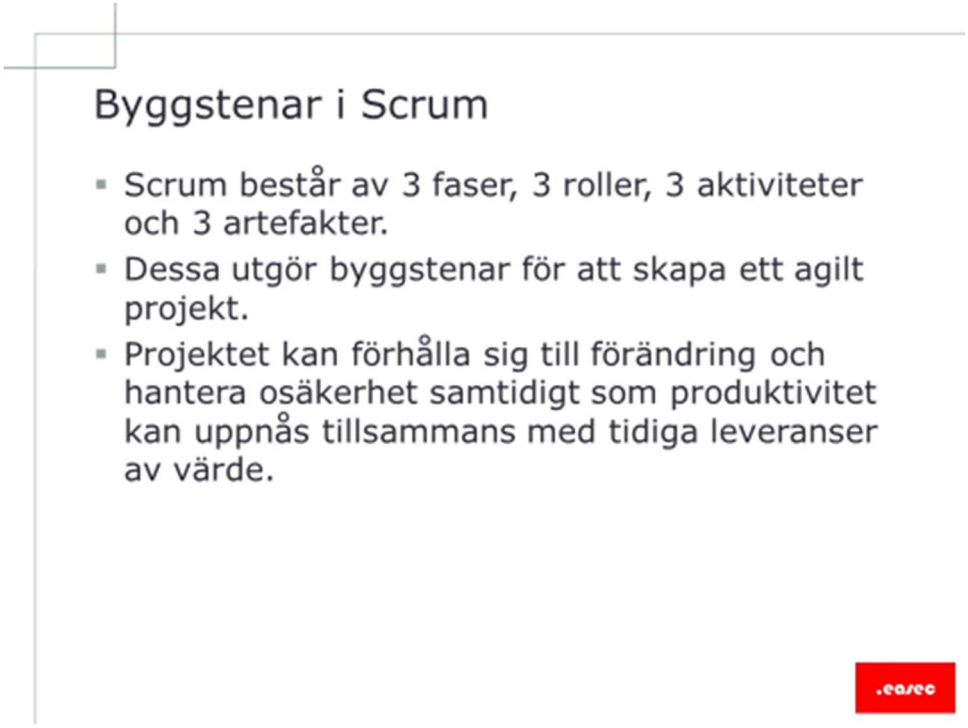
Genom ett iterativt arbete levereras inkrement, delar av slutprodukten kontinuerligt.

För en kontinuerlig förbättring, används reflektion och utvärdering.

I Scrum planeras en iteration i taget, iteration kan vara 1-4 veckor. Detta kombineras med en mer långsiktig och övergripande plan.

I iternationen är fokuserat arbete med dagliga avstämningar och fokus på eliminera hinder.

I slutet levereras det som är klart och en utvärdering.

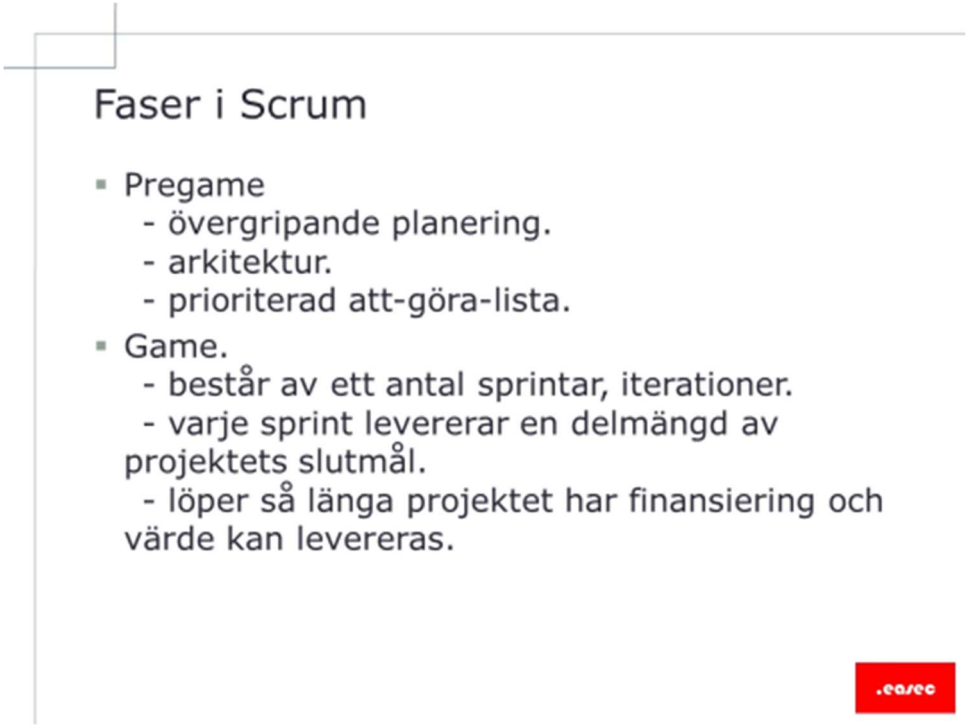


Byggstenar i Scrum

Scrum består av 3 faser, 3 roller, 3 aktiviteter och 3 artefakter.

Dessa utgör byggstenar för att skapa ett agilt projekt.

Projektet kan förbehålla sig till förändring och hantera osäkerhet samtidigt som produktivitet kan uppnås tillsammans med tidiga levereranser av värde.



Faser i Scrum

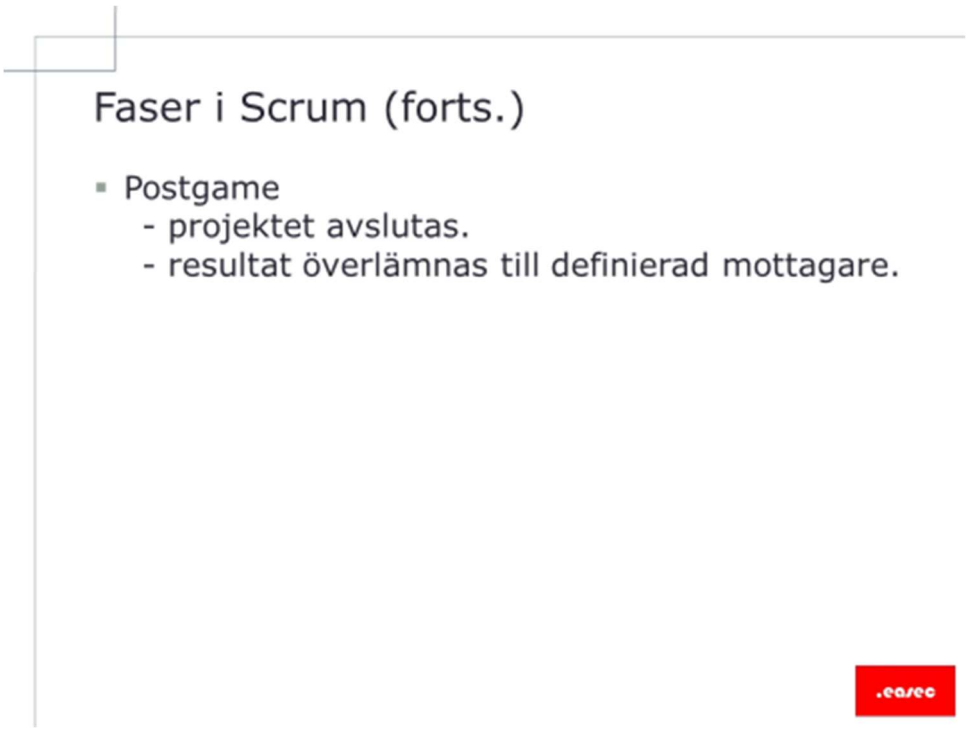
Pregame

- Övergripande planering.
- Arkitektur.
- Prioriterad att-göra-lista.

Att-göra-lista kommer i stor del från user stories.

Game

- Består av ett antal sprintar, iterationer.
- Varje sprint levererar en delmängs av projektets slutmål.
- Löper så länge projektet har finansiering och värde kan levereras.



Faser i Scrum (forts.)

Postgame

- Projektet avslutas.
- Resultat överlämnas till definierad mottagare.

User Story

User Story är en kort beskrivning i vardagligt språk av vad en användare vill uppnå.

Grundidé är att varje story skall vara kort och få plats på en post-it lapp.

Dessa skrivs av eller för beställaren och är beställarens huvudsakligaa metod för att beskriva sina krav och även kunna påverka utvecklingen av mjukvara, systemet eller webbplatsen.

User Story (forts.)

User Stories är ett snabbt sätt att hantera kravställning utan att behöva skapa formell kravspecifikation och arbeta för att hålla dessa uppdaterade.

Dessa kan också skrivas av utvecklare för att t ex beskriva icke-funktionella krav såsom säkerhet och prestanda.

Innan User Story implementeras, skall acceptanskritera skrivas för att se till att målet med User Story uppfylls.

Exempel på User Stories

Skriv enligt:

Som <roll> vill jag <mål/önskan/händelse>.

- Som mobilanvändare vill jag se en responsiv version av webbplatsen

Som <roll> vill jag <mål/önskan/händelse> för att <syfte>.

- Som nyfiken besökare vill jag hitta en intressant utställning för att lära mig mer
- Som mobilanvändare vill jag få en responsiv version av webbplatsen för att den skall ladda snabbare och vara anpassad då man kan använda den i mobilen

User story med acceptanskriterier

User story

- Som besökare vill jag kunna se de senaste twittringarna från företaget direkt på startsidan. Denna funktion är viktig eftersom

den gör att jag kan få all kommunikation från företaget samlad på en plats och inte missar något.

Acceptanskriterier

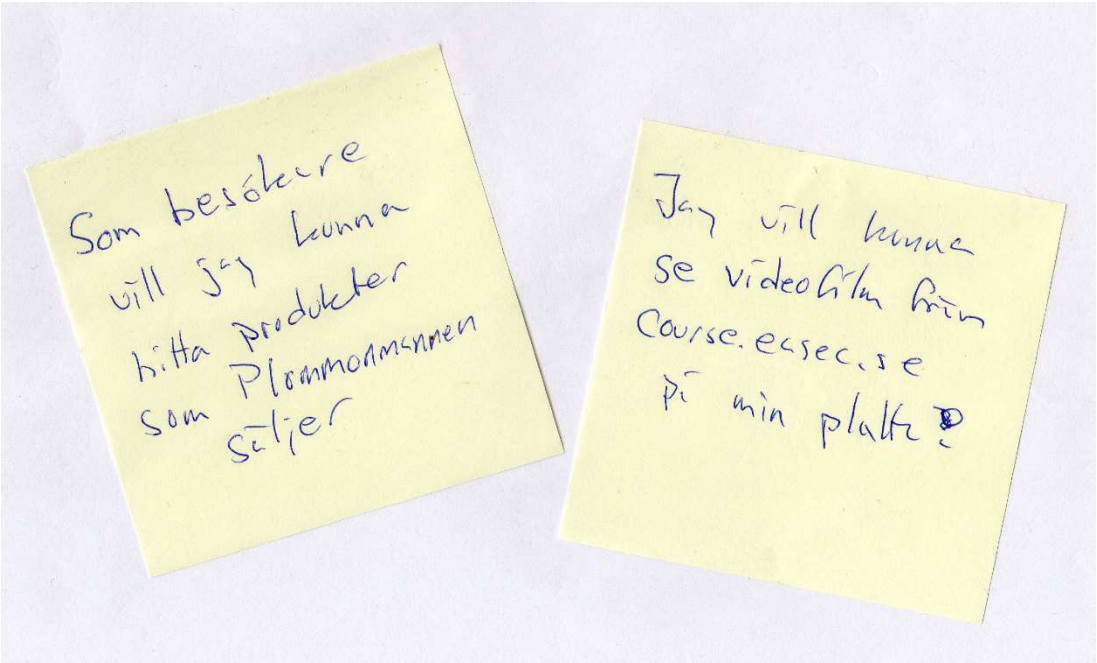
- Webbplatsens startsida visar de 3 senaste tweetsen.
- Tweetsen visas inom 15 minuter från att de twittrades.
- Om en tweet raderas ska den inte visas på webbplatsen.
- Länkar i tweetsen ska fungera.

Förslag till lösning

- Används modulen Twitter_db.
- Sätt cron job att köras var 15:e minut.

Kommentar(er)

Lägg även till en knapp ”Följ oss på Twitter” i anslutning till tweetsen.



Roller i Scrum

Produktägare

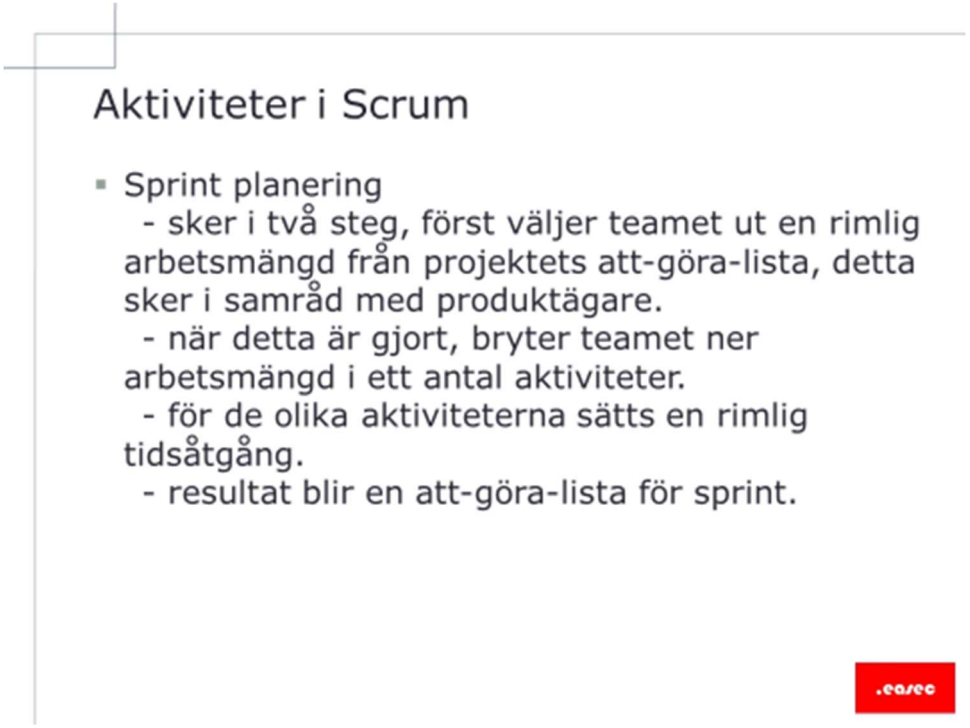
- Ansvarar för att prioritera att-göra-listan.
- Stöttar teamet i deras arbete.
- Är representant för projektets intressenter.

Teamet

- Ansvar för att driva projektet framåt.
- Tar ett gemensamt ansvar för sina leveranser.

Scrum Master

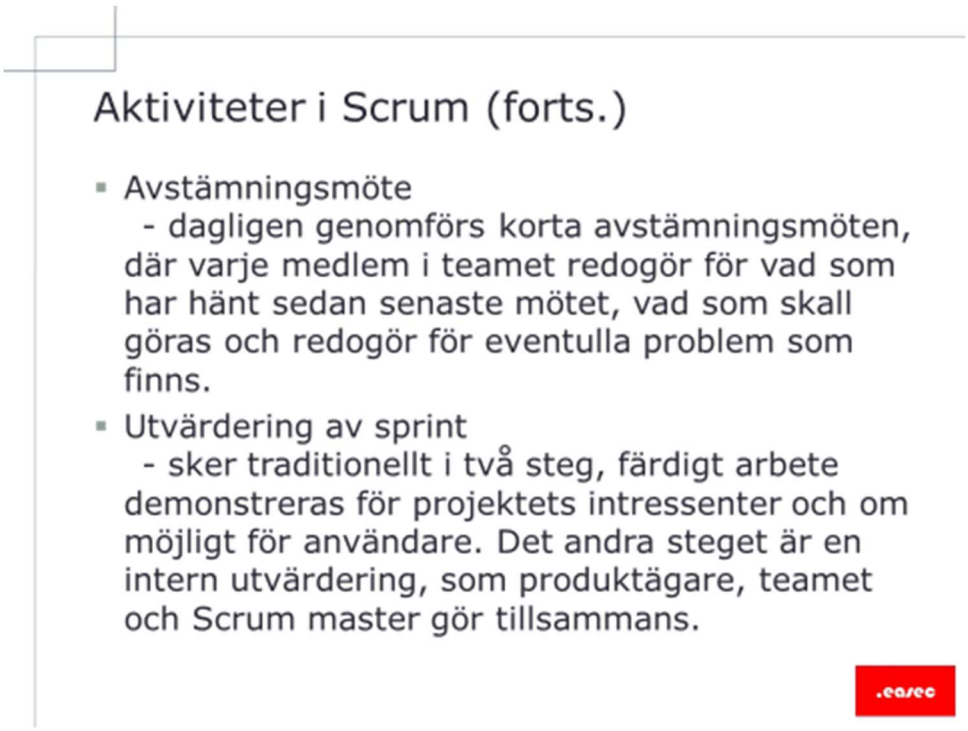
- Coach som stöttar teamet och produktägare.
- Ser till att projektet genomförs inom regelverket och med de metoder som är överenskommet.



Aktiviteter i Scrum

Sprint planering

- Sker i två steg, först väljer teamet ut en rimlig arbetsmängd från projektets att-göra-lista, detta sker i samråd med produktägare,
- När detta är gjort, bryter teamet ner arbetsmängd i ett antal aktiviteter.
- För de olika aktiviteterna sätts en rimlig tidsåtgång.
- Resultat blir en att-göra-lista för sprint.



Aktiviteter i Scrum (forts.)

Avstämningsmöte

- Dagligen genomförs korta avstämningsmöten, där varje medlem i teamet redogör för vad som har hänt sedan senaste mötet, vad som skall göras och redogör för eventuella problem som finns.

Utvärdering av sprint

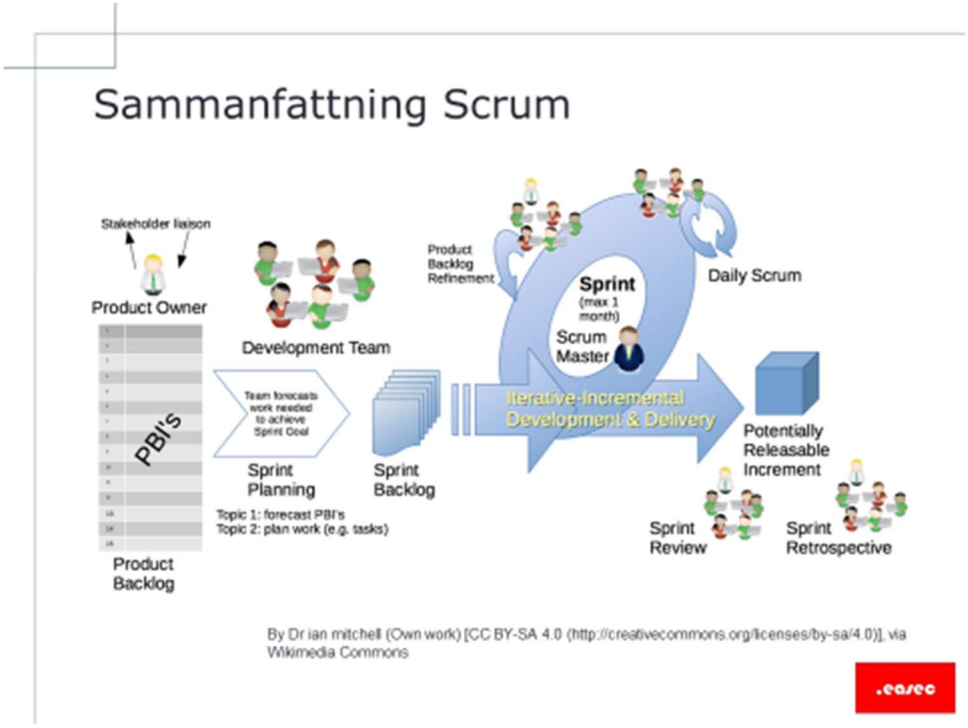
- Sker traditionellt i två steg, färdigt arbete demonstreras för projektets intressenter och om möjligt för användare. Det andra steget är en intern utvärdering, som produktägare, teamet och Scrum master gör tillsammans.

Artefacter i Scrum

Projektets att-göra-lista
- Prioriterad av produktägaren.
- Ju högre prioritet, desto mer konkret.

Aktuell sprints att-göra-lista
- Skapas i början av varje sprint.
- Under själva arbetet görs inga förbättringar av innehållet.

Burn down chart
- Diagram som används för att visualisera framsteg i sprinten.
- I samband med det dagliga avstämningsmötet uppdateras diagrammet för att visa hur mycket arbete som återstå.

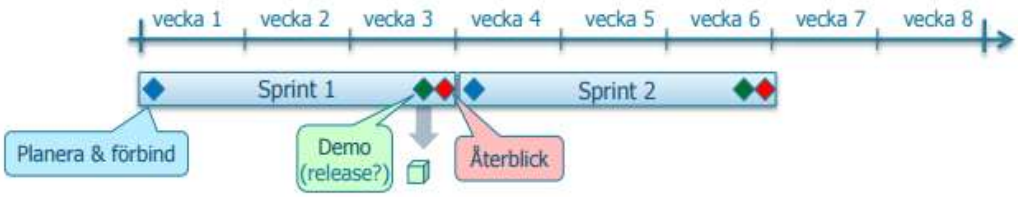


Sammanfattning Scrum

Mönstret i Scrum är enkelt, istället för att detaljplanera projektet skapas en övergripande plan med en tydlig vision, tillsammans med en prioriterad lista (Product Backlog). Denna lista innehåller de funktioner som efterfrågas.

Genom interativt arbete kommer teamet att ansvara för en kontinuerlig levererans av de högst prioriterade önskemålen från Product Backlog. Vad som skall levereras i varje sprint, planeras av produktägaren och utvecklingsteam. Detta sker i början av sprint. Viktigt att tänka på det faktum, att det inte görs några förändringar när väl sprint har påbörjats. Under sprinten finns kunden nära tillhands för att svara på detaljfrågor.

Färdig funktionalitet demonstreras för kunden och användare i slutet av iterationen, för att få feedback som påverkar kommande utveckling. Teamet tillsammans med kunden genomför tillsammans en utvärdering för att hitta förbättringsmöjligheter inför kommande iteration. Inför nästa iteration har kunden uppdaterat Product Backlog med eventuell ny funktionalitet och prioriterat om listan i enlighet med sina önskemål.



Tydlig bild av arbetet, sprint löper inom en tydlig tidsrymd. I slutet av sprint redogörs arbetet och utvärdering görs.

Agile -Kanban

Att införa Scrum innebär oftast en större omställning för organisationen.

Att använda Kanban är ett mindre steg för att inför Agile och använder sig av ett produktionsflöde som är likt Lean.

Hjärtat i Kanban är tavlan som visuliserar arbetsflödet.

I Kanban begränsas antalet pågående aktiviteter och för pågående aktiviteter mäts ledtiden och aktivitet optimeras.

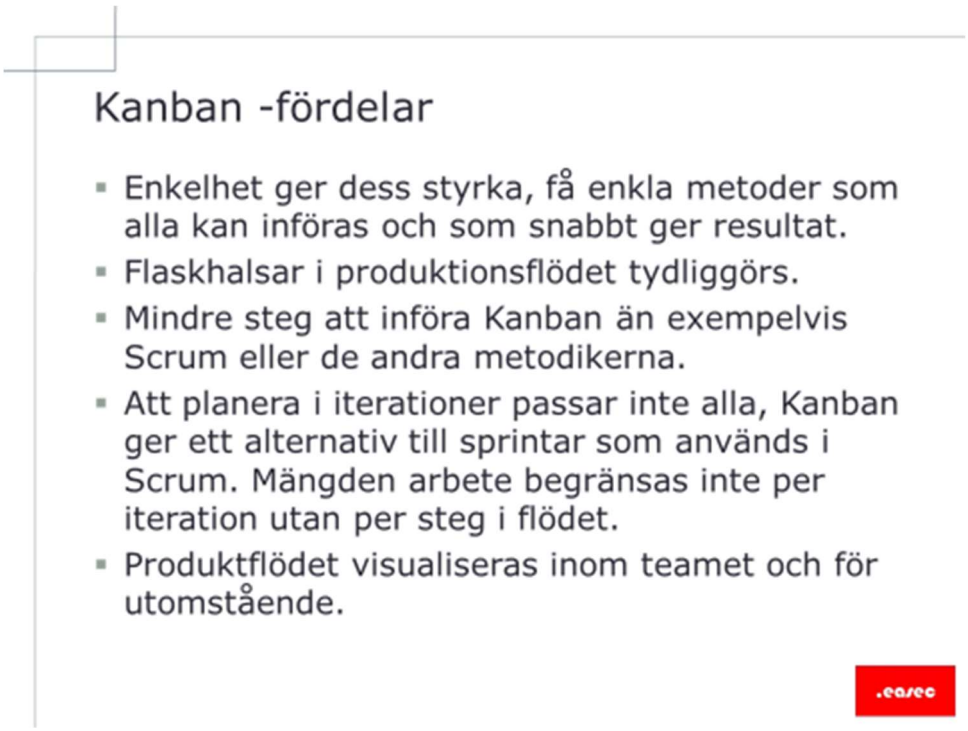
Agile –Kanban (forts.)

Kanban betyder signalkort på japanska.

Används ofta för att införa Just-In-time och för att förhindra överproduktion.

Kanban bygger på tre delar:

- Visualisera arbetsflödet.
- Begränsa antalet pågående aktiviteter.
- Mäta ledtiden och optimera den.



Kanban -fördelar

Enkelhet ger dess styrka, få enkla metoder som alla kan införas och som snabbt ger resultat.

Flaskhalsar i produktionsflödet tydliggörs.

Mindre steg att inför Kanban än exempelvis Scrum eller de andra metodikerna.

Att planera i iterationer passar inte alla, Kanban ger ett alternativ till sprintar som används i Scrum. Mängden arbete begränsas inte per iteration utan per steg i flödet.

Produktflödet visualiseras inom teamet och för utomstående.

Byggstenar i Kanban

Kanban har tre stycken byggstenare:

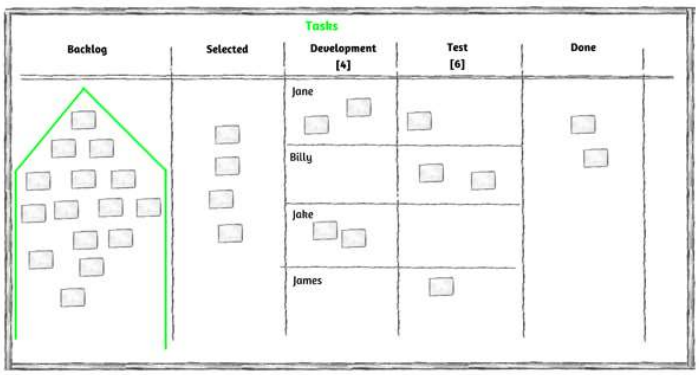
- Visualisering av arbetsflödet.
- Begränsa antalet pågående aktiviteter.
- Mäta ledtiden och optimera denna.

Visualisering av arbetsflödet

Det finns två sätt att illustrera produktionsflödet:

- Kanban tavla.
- Det ackumulerade flödet.

Kanban tavla

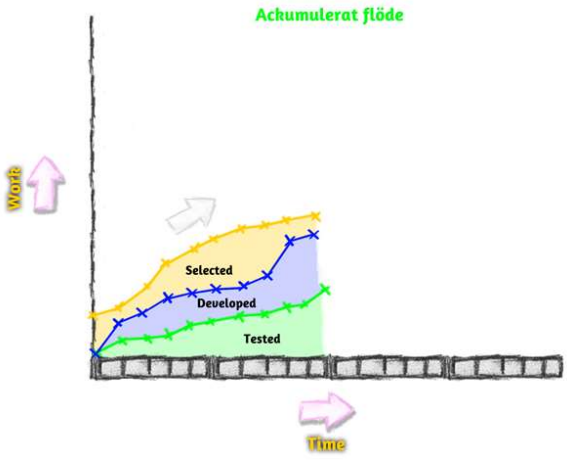


Bilden visar ett exempel på hur en Kanban tavla kan se ut. Denna visar max antal tillåtna aktiviteter i olika faser, detta anges som en siffra under respektive namn. Bilden använder sig även av en Berlic-pyramid för prioriteringen av backlogen.

Visualisering av arbetsflödet (forts.)

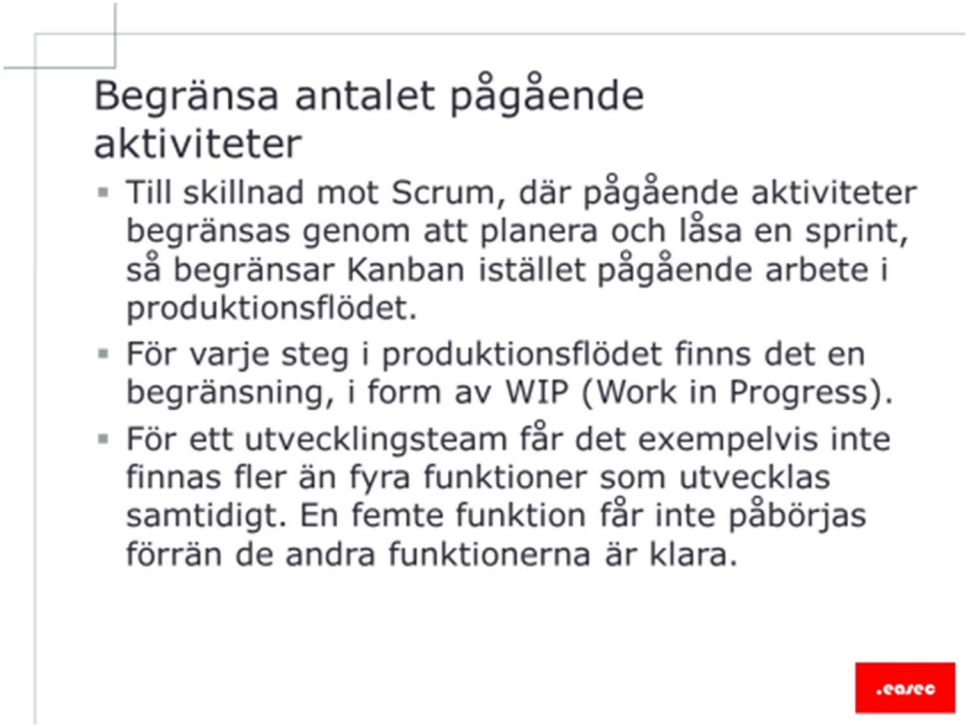
Det ackumulerade flödet besvarar:

- Var finns eventuella flaskhalsar i arbetsfödet?
- Hur lång tid tar det för aktiviteter från en status till en annan?
- Hur fokus skiftar över tiden.



Skapa diagrammet genom att dagligen markera antalet aktiviteter i varje fas, detta görs i ett XY-diagram. Använd X-axel för att ange antalet

aktiviteter och Y-axeln för dagen. Gör gärna som bilden visar, dvs använd olika färger för varje status.



Begränsa antalet pågående aktiviteter

Till skillnad mot Scrum, där pågående aktiviteter begränsas genom att planera och låsa en sprint, så begränsar Kanban istället pågående arbete i produktionsflödet.

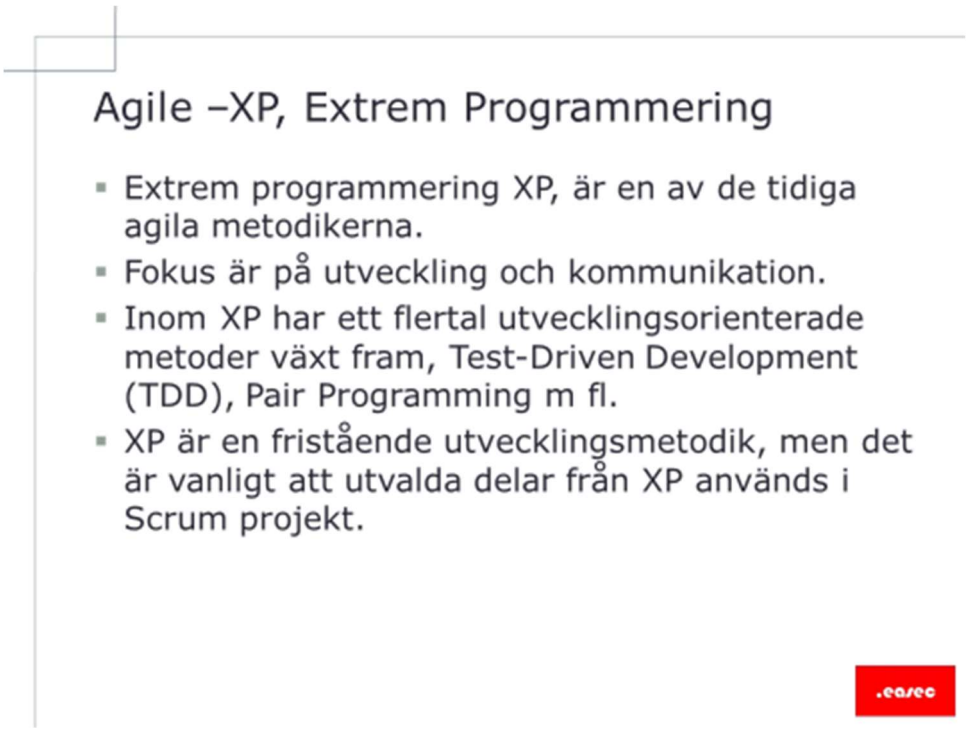
För varje steg i produktionsflödet finns det en begränsning i form av WIP (Work in Progress).

För ett utvecklingsteam får det exempelvis inte finnas fler än fyra funktioner som utvecklas samtidigt. En femte funktion får inte påbörjas förrän de andra funktionerna är klara.

Mäta ledtid och optimera denna

Genom att mäta ledtiden, t ex diagrammet som visar det ackumulerade flödet, kan prognos skapas för hur lång tid varje steg tar.

Målsättning är att ha så korta ledtider som möjligt.



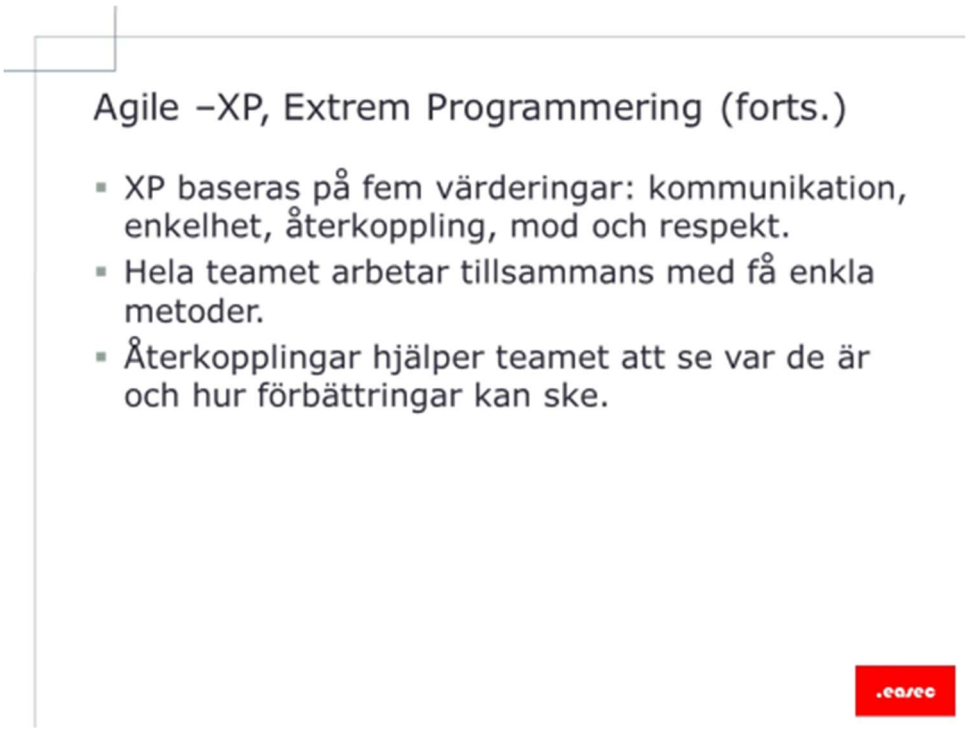
Agile –XP, Extrem Programmering

Extrem programmering XP, är en av de tidiga agila metodikerna.

Fokus är på utveckling och kommunikation.

Inom XP har ett flertal utvecklingsorienterade metoder växt fram, Test-Driven Development (TDD), Pair Programming m fl.

XP är en fristående utvecklingsmetodik, men det är vanligt att utvalda delar från XP används i Scrum projekt.

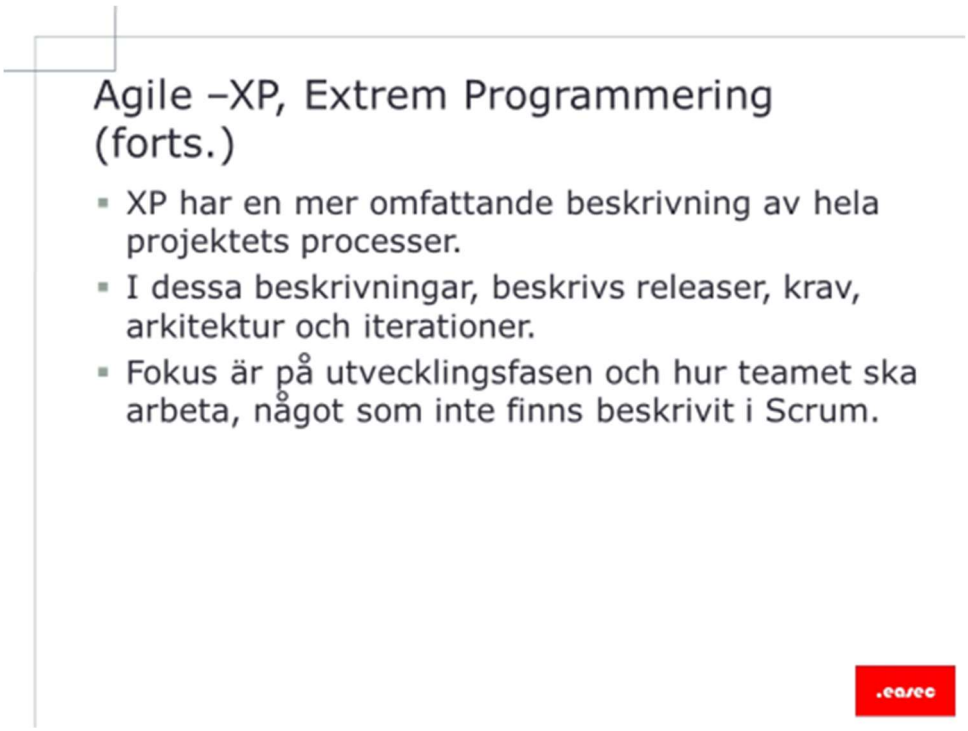


Agile –XP, Extrem Programmering (forts.)

XP baseras på fem värderingar: kommunikation, enkelhet, återkoppling, mod och respekt.

Hela teamet arbetar tillsammans med få enkla metoder.

Återkopplingar hjälper teamet att se var de är och hur förbätringar kan ske.

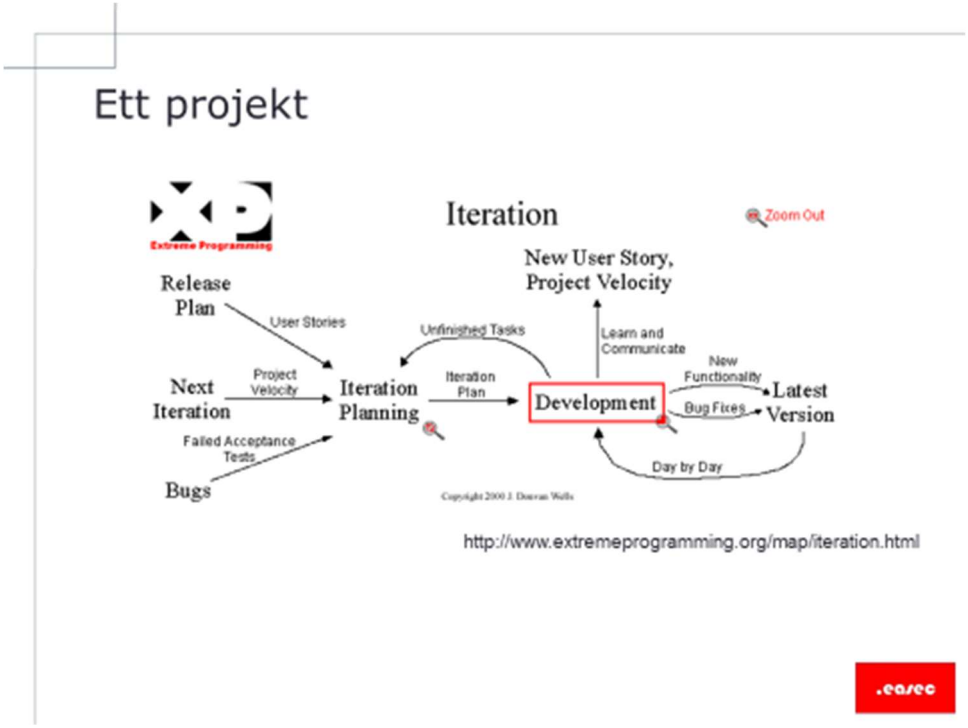


Agile –XP, Extrem Programmering (forts.)

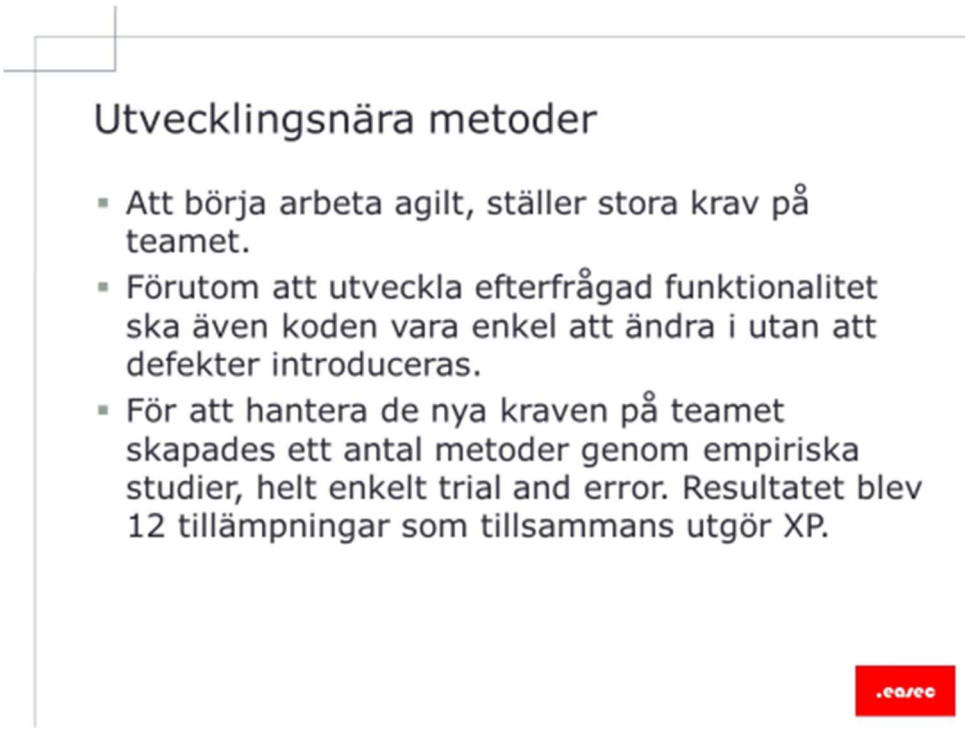
XP har en mer omfattande beskrivning av hela projektets processer.

I dessa beskrivningar, beskrivs releaser, krav, arkitektur och iterationer.

Fokus är på utvecklingsfasen och hur teamet ska arbeta, något som inte finns beskrivit i Scrum.



Ett projekt

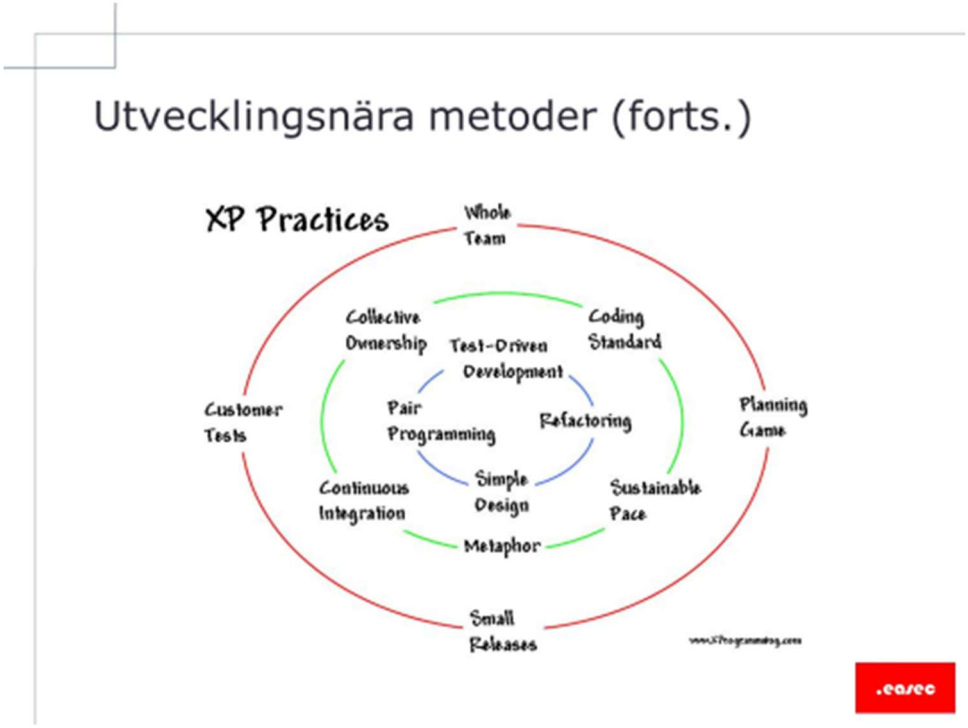


Utvecklingsnära metoder

Att börja arbeta agilt, ställer stora krav på teamet.

Förutom att utveckla efterfrågad funktionalitet ska även koden vara enkel att ändra i utan att defekter introduceras.

För att hantera de nya kraven på teamet skapades ett antal metoder genom empiriska studier, helt enkelt trail and error. Resultatet blev 12 tillämplingar som tillsammans utgör XP.



Utvecklingsnära metoder (forts.)

Dessa tillämplingar är:

Planeringsspelet, funktionalitet i nästa leverans bestäms genom prioriterade verksamhetsberättelser och tekniska bedömningar.

Små leveranser, systemet levereras i små inkrementella versioner.

Metafor, en enkel beskrivning av hur systemet ska fungera.

Enkel design, genom att hålla koden enkel, blir även designen enkel. Ta bort komplexitet från koden kontinuerlig när den upptäcks.

Testning, tester skrivs innan koden utvecklas. Programmerar skriver tester som testar och validerar koden medan kunden skriver tester som testar och validerar verksamhetsberättelser.

Omstrukturering av kod, ta kontinuerligt bort identiska (dubletter) och komplexa kodbitar.

Parprogrammering, programmerare arbetar i par vid en dator. En skriver koden medan den andre granskar koden.

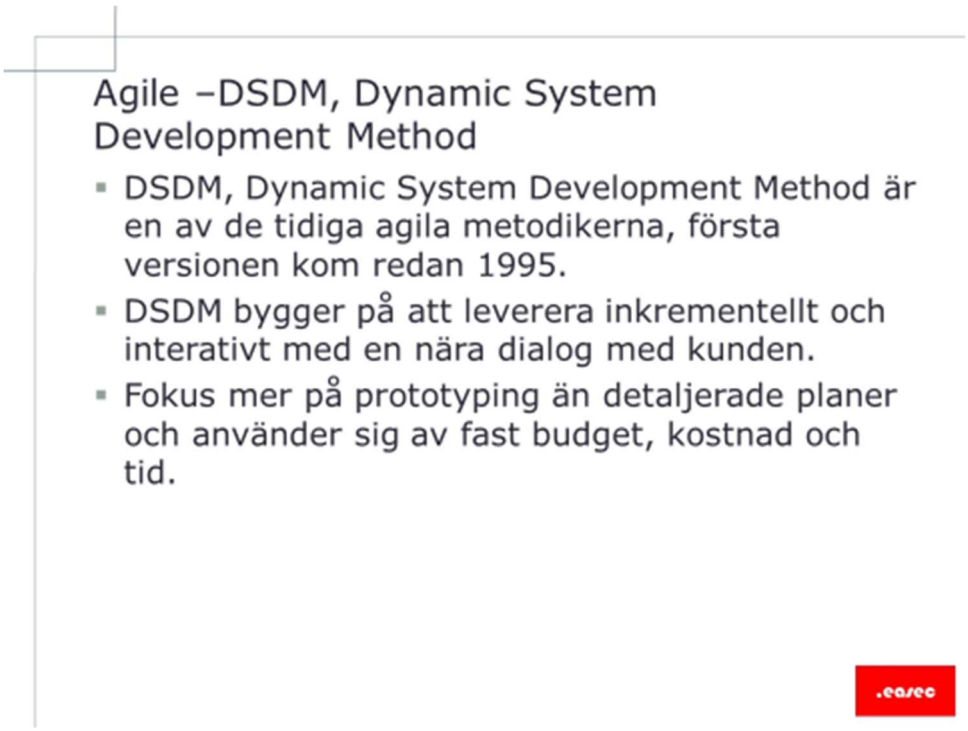
Gemensamt ägandeskap, alla har rätt att ändra i all kod, skriven av dem själva eller någon annan.

Kontinuerlig integration, när en implementationsuppgift är utförd byggs systemet och integreras med den redan färdiga koden. Detta kan ske flera gånger per dag.

40-timmas arbetsvecka, programmerar tänker och därmed arbetar bättre utvilade. Övertid tillåts inte två veckor i rad.

Kund på plats, kunden arbetar med utvecklarna på heltid för att kunna svara på frågor, definiera systemet och skriva tester.

Kodstandard, konsekvent kodstandard som alla programmerar följer.

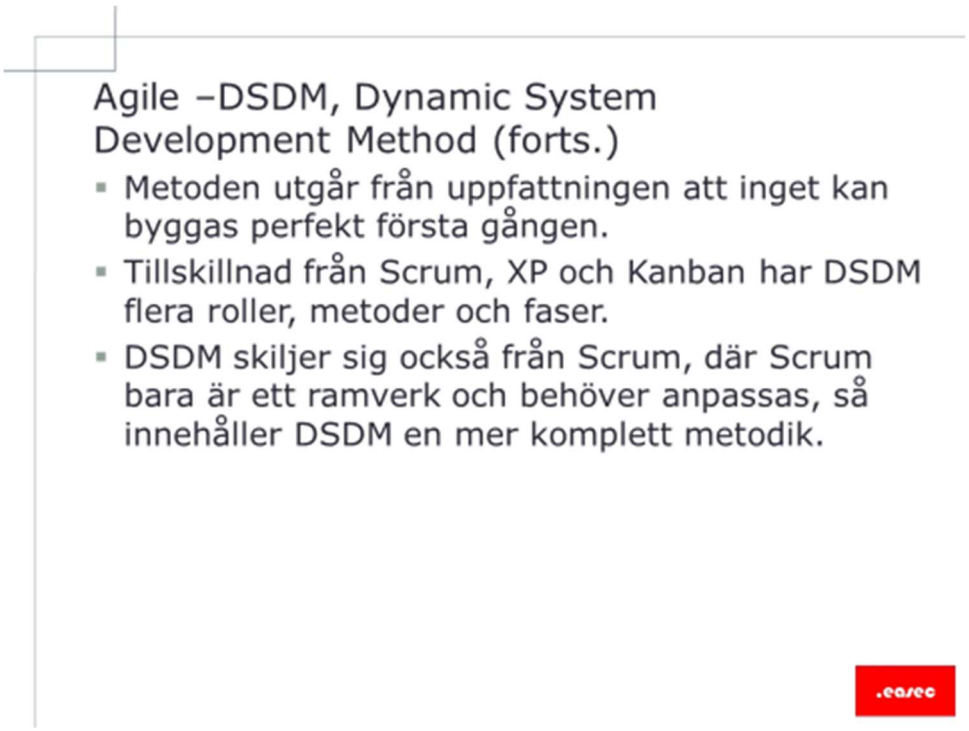


Agile –DSDM, Dynamic System Development Method

DSDM, Dynamic System Development Method är en av de tidiga agila metodikerna, första versionen kom redan 1995.

DSDM bygger på att leverera inkrementellt och interativt med en nära dialog med kunden.

Fokus mer på protyping än detaljerade planer och använder sig av fast budget, kostnad och tid.



Agile –DSDM, Dynamic System Development Method (forts.)

Metoden utgår från uppfattningen att inget kan byggas perfekt första gången.

Tillskillnad från Scrum, XP och Kanban har DSDM flera roller, metoder och faser.

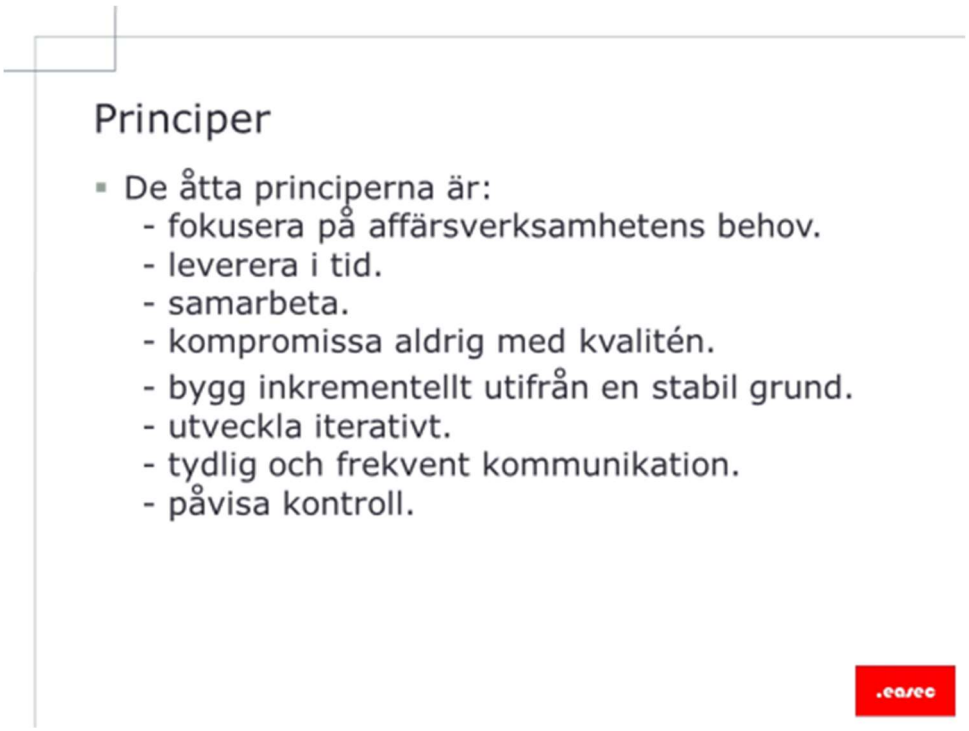
DSDM skiljer sig också från Scrum, där Scrum bara är ett ramverk och behöver anpassas, så innehåller DSDM en mer komplett metodik.

Koppling till Agile manifesto

Metodiken förvaltas av ett konsortium, senaste versionen kom 2007 och går under namnet Atern.

DSDM Atern bygger på åtta principer som berör attityd och inställning som ska prägla ett framgångsrikt projekt.

Många av principerna återfinns direkt eller indirekt i Agile manifesto.



Principer

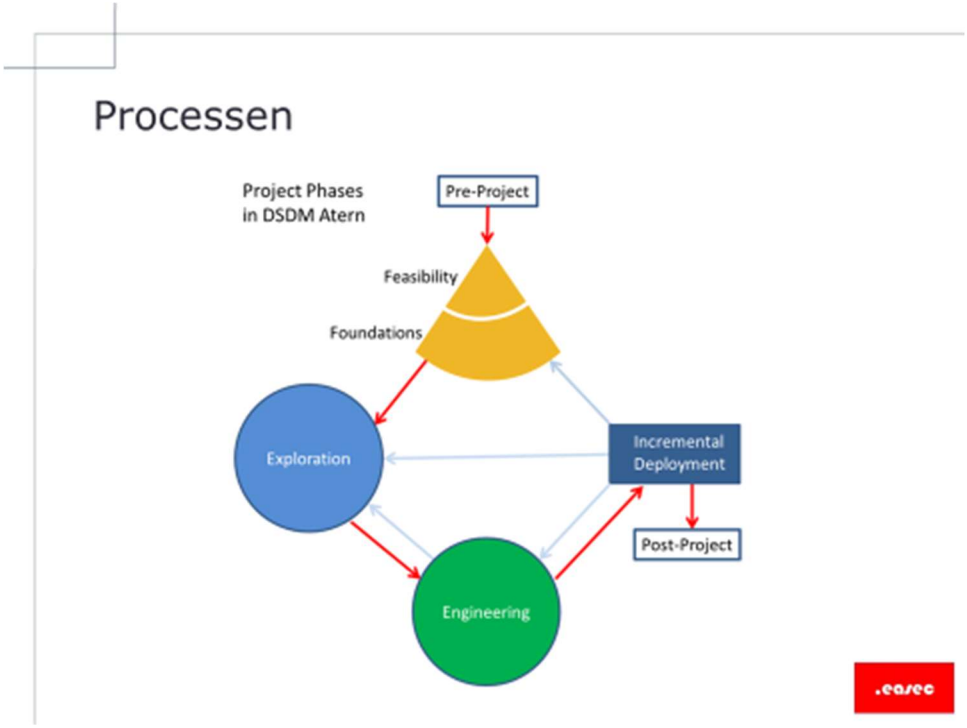
De åtta principerna är:

- **Fokusera på affärsverksamhetens behov**, förstå verksamhetens prioriteringar och garantera att projektets budget räcker för att leverera en tillräckligt viktig delmängd.
- **Leverera i tid**, använd iterationer och leverera kontinuerligt. Fokusera på det som verksamheten prioriterar.
- **Samarbeta**, att involvera användaren är nyckeln till ett lyckat projekt. Även kunden, intressenter och verksamheten ska involveras i projektet genom en nära dialog.
- **Kompromissa aldrig med kvalitén**, var tydliga med vilken kvalité som projektet förväntas leverera. Anpassa design, utveckling och tester efter det målet. Sänk inte kvalitén för att kunna leverera mer funktionalitet.
- **Bygg inkrementellt utifrån en stabil grund**, leverera värde så tidigt som möjligt och verifiera att det som levererats möter verksamhetens och användarnas behov.

- **Utveckla iterativt**, samla in feedback från kunden och användarna i varje iteration och låt produkten utvecklas iterativt genom ökad kundkap om både vad som ska leverers och hur det ska konstrueras. Ett iterativt arbete medför tidiga leveranser och möjlighet att förhålla sig till förändring.

- **Tydlig och frekvent kommunikation**, använd dagliga avstämningsmöten, workshops och använd er av prototyping och modellering för att skapa en visuell dialog. Fokusera på informell kommunikation, ansikte mot ansikte och dokumenter det som är viktigt och ger ett värde.

- **Påvisa kontroll**, använd en lagom nivå av formell uppföljning och rapportering, Planer och projektets aktuella status ska vara tydlig för alla.



Processen

Steg i processen:

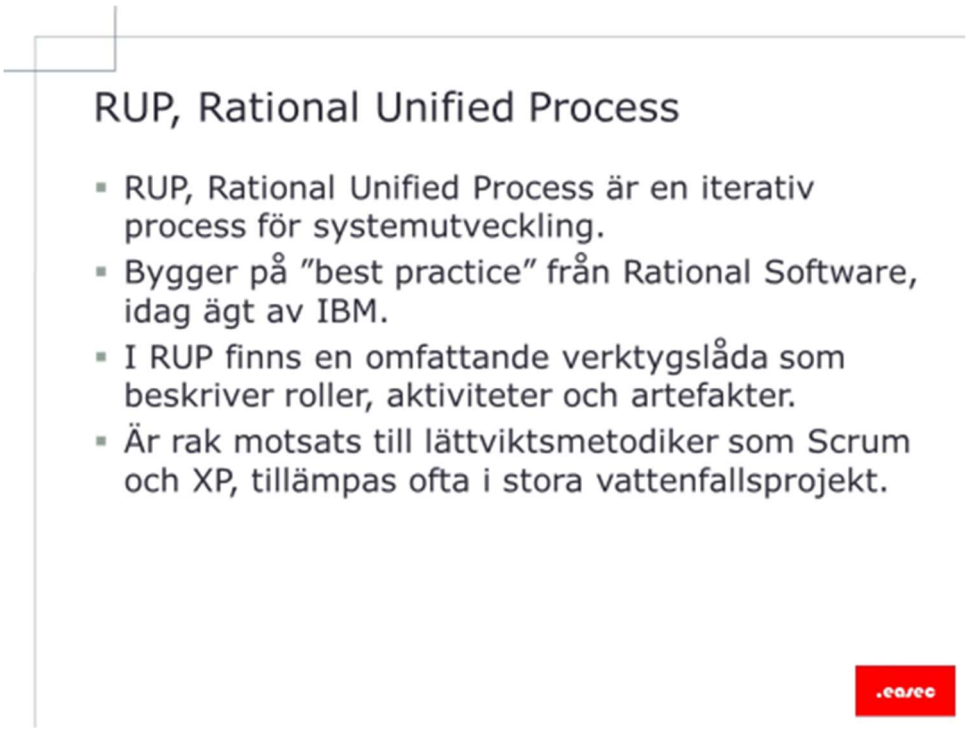
Feasibility, besluta om det förslagna projektet är genomförbart, både för verksamheten och rent tekniskt.

Foundations, skapa en grund för lösningen som gör det möjligt att demonstrera hur verksamhetens behov uppfylls. Viktigt att inte fastna i detaljer.

Exploration, utforska detaljerade krav och skapa lösningar genom ett iterativt och inkrementellt arbete.

Engineering, utveckal produkten enligt lösning som producerades i Explorationfasen.

Deplyment, leverans av lösningen så att den kan användas av målgruppen.



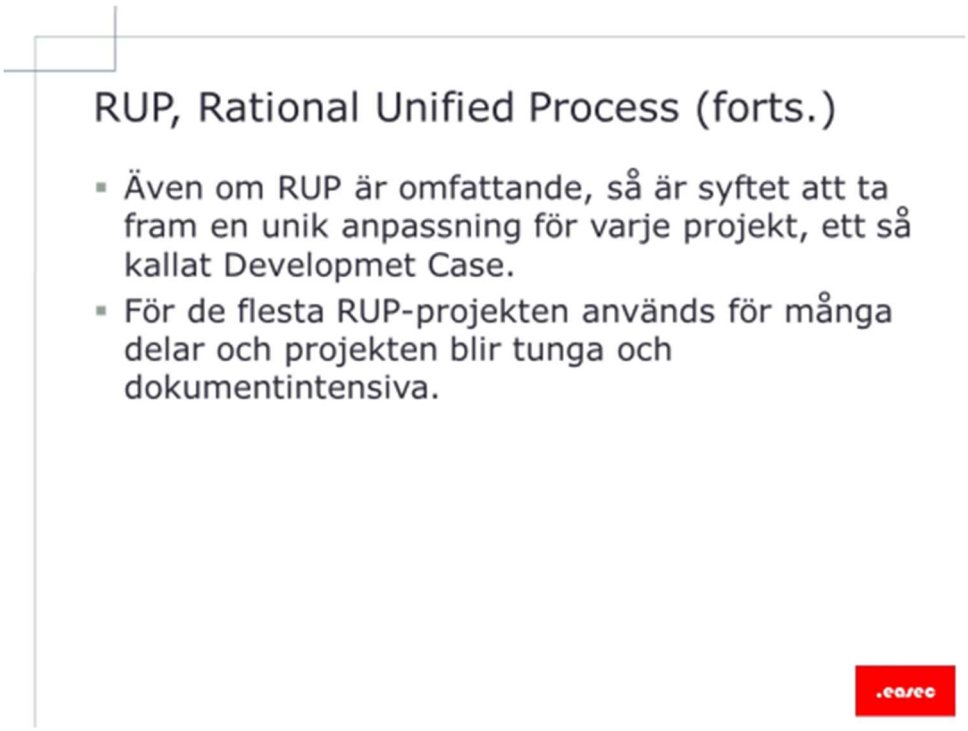
RUP, Rational Unified Process

RUP, Rational Unified Process är en iterativ process för systemutveckling.

Bygger på ”best practice” från Rational Software, som idag ägs av IBM.

I RUP finns en omfattande verktygslåda, denna verktygslåda beskriver roller, aktiviteter och artefakter.

Är rak motsats till lättviktsmetodiker som Scrum och XP, tillämpas ofta i stora vattenfallsprojekt.



RUP, Rational Unified Process (forts.)

Även om RUP är omfattande, så är syftet att ta fram en unik anpassning för varje projekt, ett så kallat Development Case.

För de flesta RUP-projekten används för många delar och projekten blir tunga och dokumentintensiva.

RUP har många likheter med Agile Manifesto

RUP är i grunden en verktygslåda med ett antal ”Best Practice” för systemutveckling av IT-system.

Värderingar som finns i metodiken har många likheter med Agile Manifesto och de principer som Scrum bygger på.

Tillämplingar

Tillämplingar som finns i RUP:

- Utveckla iterativt.
- Hantera systemkrav.
- Använd komponentbaserad arkitektur.
- Modellera visuellt, främst inom UML[1].
- Kontinuerlig kvalitetskontroll.
- Kontrollera förändringar.

[1] http://www.uml.org/

Processen

Metodiken innehåller fyra faser som alla har en väldefinierad milstolpe.

Faserna är:

- Förberedelse (Inception).
- Etablering (Elaboration).
- Konstruktion (Construction).
- Överlämning (Transition).

Traditionella vattenfallsprojekt

Denna metodik eller projektform, har länge varit standard för hur projekt beskrivs.

Under denna tiden har också branchen presterat orväckande dåligt, kraftiga förseningar, dubbel så dyrt och levererat drygt hälften av vad som har efterfrågas.

Agile är en motreaktion mot de traditionella vattanfallsprojektet och dess oförmåga att hantera förändringar samt stora och komplexa IT-projekt.

Vattenfallsmodellen

Modellen bygger på ett logiskt och sekventiellt genomförande av projektet.

Varje steg I processen skall avslutas innan nästa påbörjas.

Nyckelord är kontroll, planering, processer och uppföljning.

Till skillnad från agila projekt, anses ändringar som en risk för projektet.

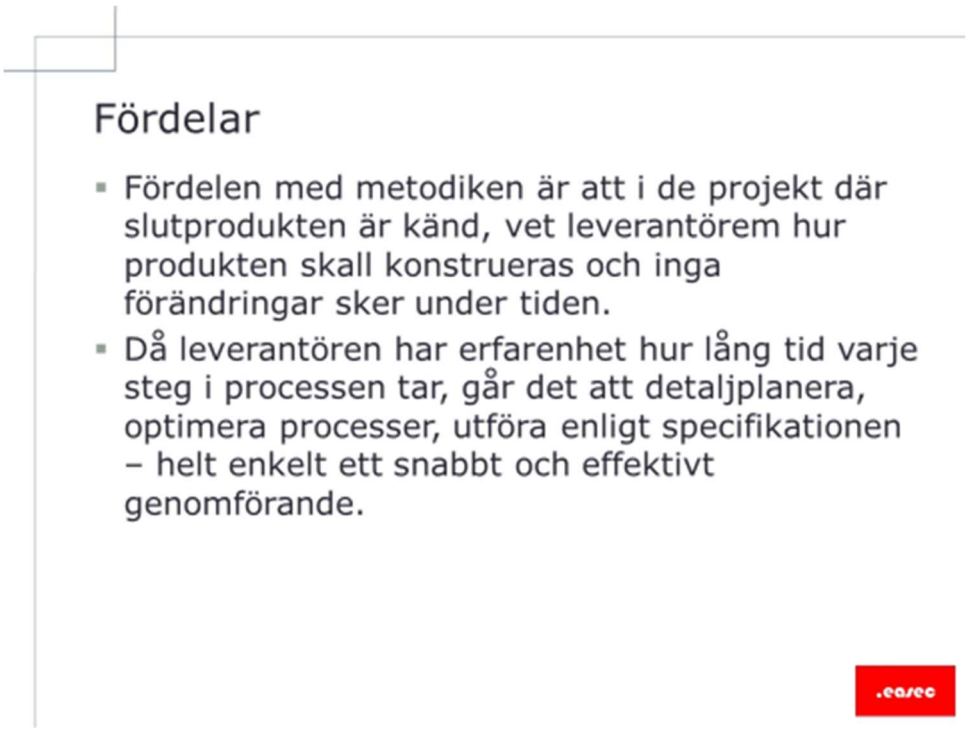
Leverans av slutprodukten sker i slutet av projektet, efter tester av både leverantören och kunden.

Lång tid, många problem

Normalt tar ett vattenfallsprojekt 12-20 månander för att gå från efterfrågad funktionalitet till leverans.

Detta skapar många problem, t ex kan kravmängden växa, liksom att det är högre sannolikhet att något förändras under tiden.

Dokumentation i ett projekt av denna typen tenderar att bli omfattande, primärt för att det är det främsta sättet att kommunicera mellan varje steg i processen.



Fördelar

Fördelen med metodiken är att I de project där slutprodukten är känd, vet leverantören hur produkten skall konstrueras och ingen förändring sker under tiden.

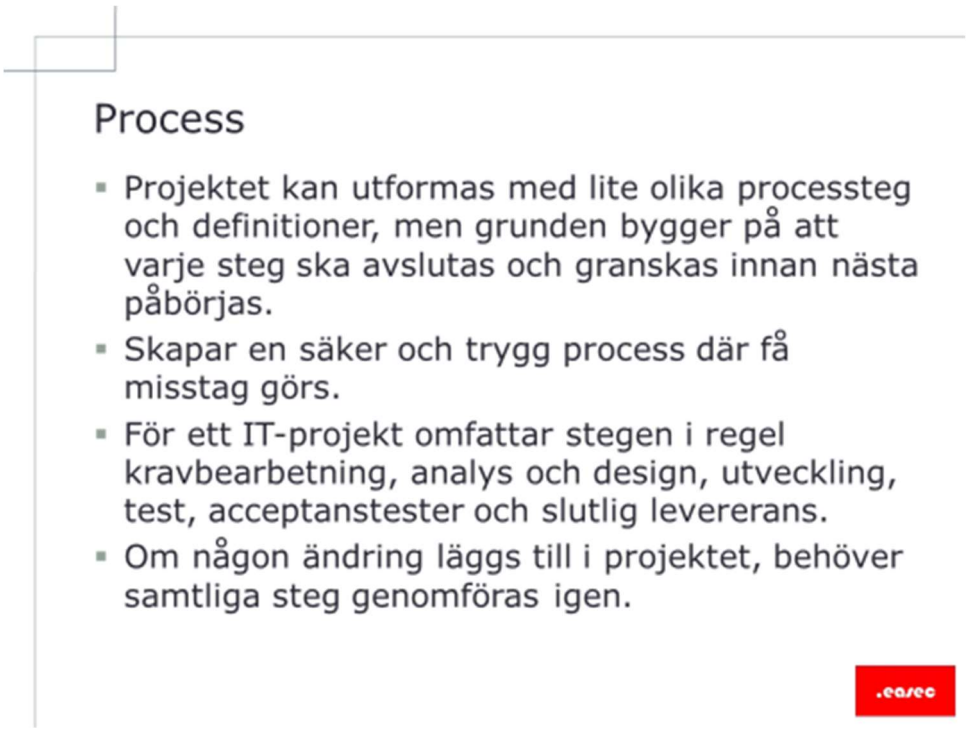
Då leverantören har erfarenhet hur lång tid varje steg i processen tar, fär det att detaljplanera, optimera processer, utföra enligt specifikationen – helt enkelt ett snabb och effektivt genomförande.

Bäst för

Metodiken är bäst för leveranser av standardlösningar med en låg del konfiguration.

IT-projekt är däremot oftast komplexa projekt, med en hög förändringsbenägenhet, unika lösningar där slutprodukten inledningsvis inte är känd, varkens för beställaren eller leverantören.

IT-projekt är ofta ett lärande projekt, där produkten växer fram genom kommunikation och reflektion.



Process

Projektet kan utformas med lite olika processteg och definitioner, men grunden bygger på att varje steg ska avslutas och granskas innan nästa steg påbörjas.

Skapar en säker och trygg process där få misstag görs.

För ett IT-projekt omfattar stegen i regel kravbearbetning, analys och design, utveckling, test, acceptanstester och slutlig leverans.

Om någon ändring läggs till i projektet, behöver samtliga steg genomföras igen.

Lektion 2: Test-Driven Development

I denna lektion skall vi titta på:

- Test-Driven Development (TDD).
- Regelverk.
- Verktyg.
- `PHPUnit`.
- PHPUnit –krav.
- PHPUnit –installation.
- `phar`.
- `composer`.
- Konventioner.
- Steg.

Test-Driven Development

Test-Driven Development = TDD, är en viktig del inom agil utveckling.

Introducerades 2002 av Kent Beck, accepterad och rekommenderad teknik vid programmering, dock med förbehåll. Förbehållet är det som jag var inne på tidigare, finns viss risk att utvecklaren skriver test för att testa sitt sätt att använda programmeringsteknik, inte efter de behov som applikation eller kund har.

TDD förspråkar att efter initial planering, systemdesign, kravställning etc, börjar skriva testfall före implementeringen. Ingen programkod får överhuvudtagit finnas innan kod finns för att testa.

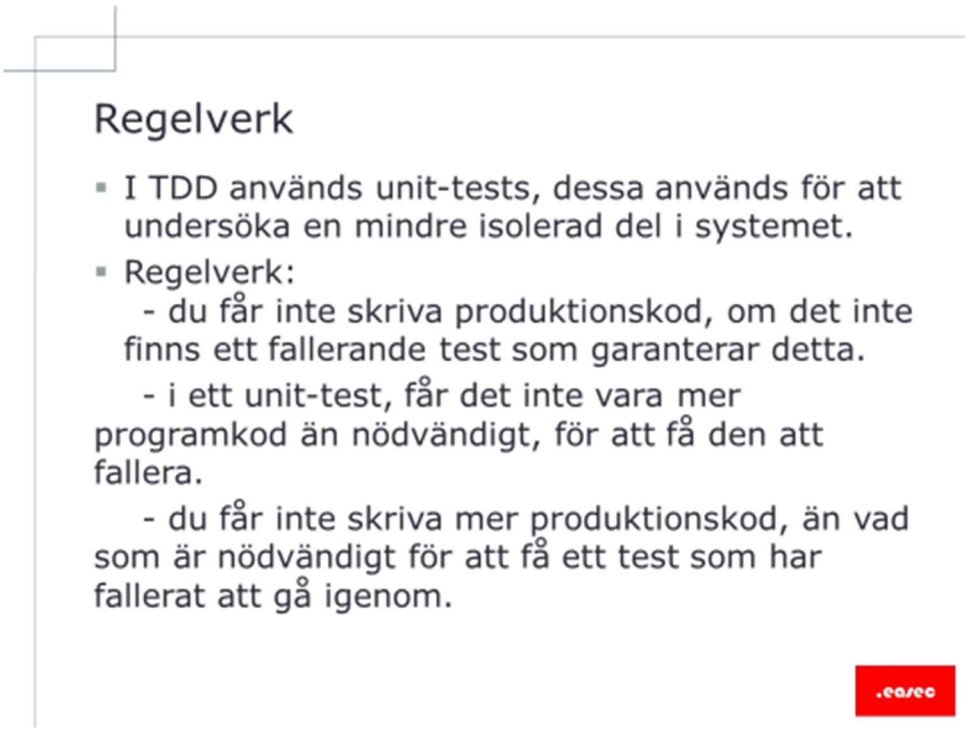
Kod för testning skrivs först

Kod för testning skrivs innan kod för applikationen.

Kod som skrivs för test, läggs i separata filer.

När programkod börjar utvecklas, testas utveckling mot testkod.

Om något går fel, får vi oftast tips om vart det har gått fel.

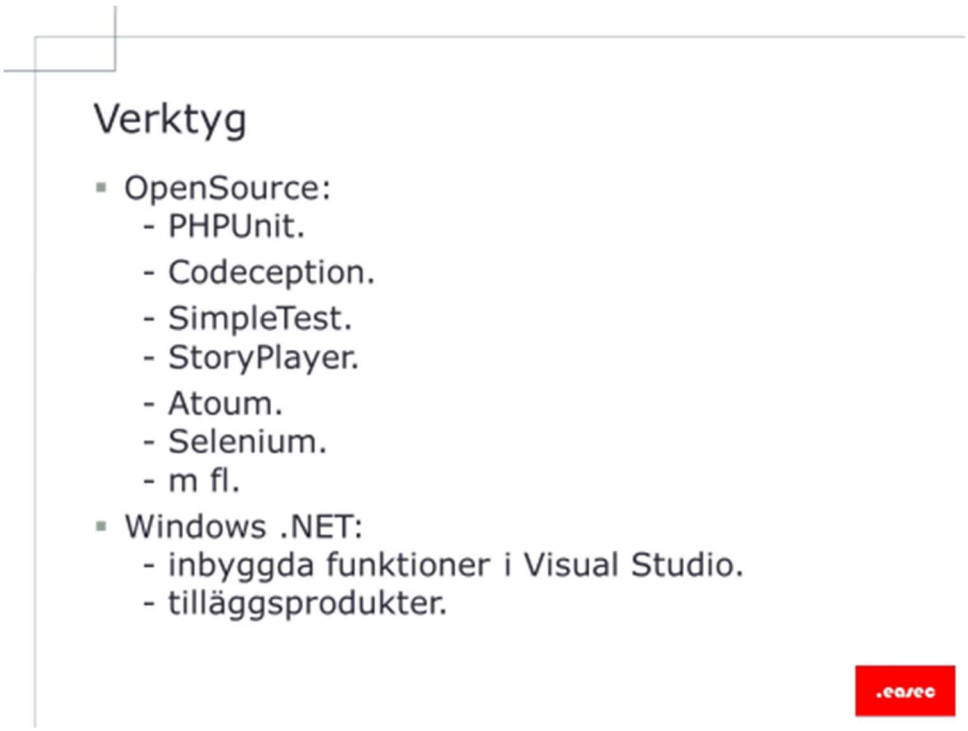


Regelverk

I TDD används unit-tests, dessa används för att undersöka en mindre isolerad del i systemet.

Regelverk:

- Du får inte skriva produktionskod, om det inte finns ett falerande test som garanterar detta.
- Ett unit-test, får inte vara mer programkod än nödvändigt, för att få den att fallera.
- Du får inte skriva mer produktionskod, än vad som är nödvändigt för att få ett test som har fallerat att gå igenom.



Verktyg

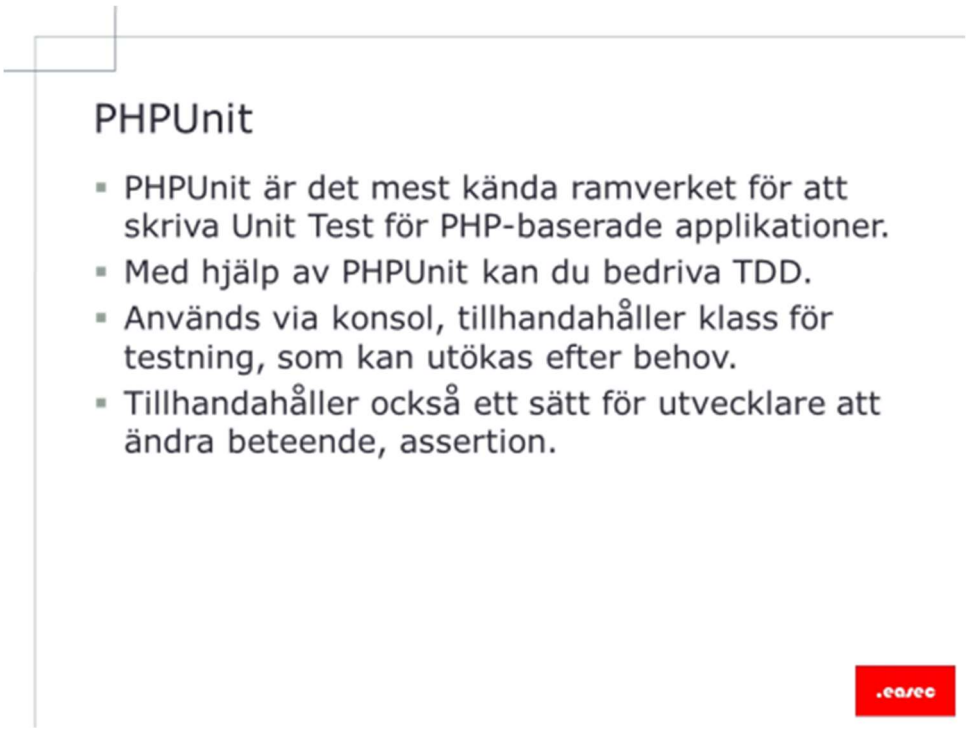
Det finns ett stort antal verktyg tillgängliga.

Inom OpenSource:

- PHPUnit.
- Codeception.
- SimpleTest.
- StorePlayer.
- Atoum.
- Selenium.
- m fl.

Inom Windows .NET:

- inbyggda funktioner i Visual Studio.
- Tilläggsprodukter till Visual Studio.



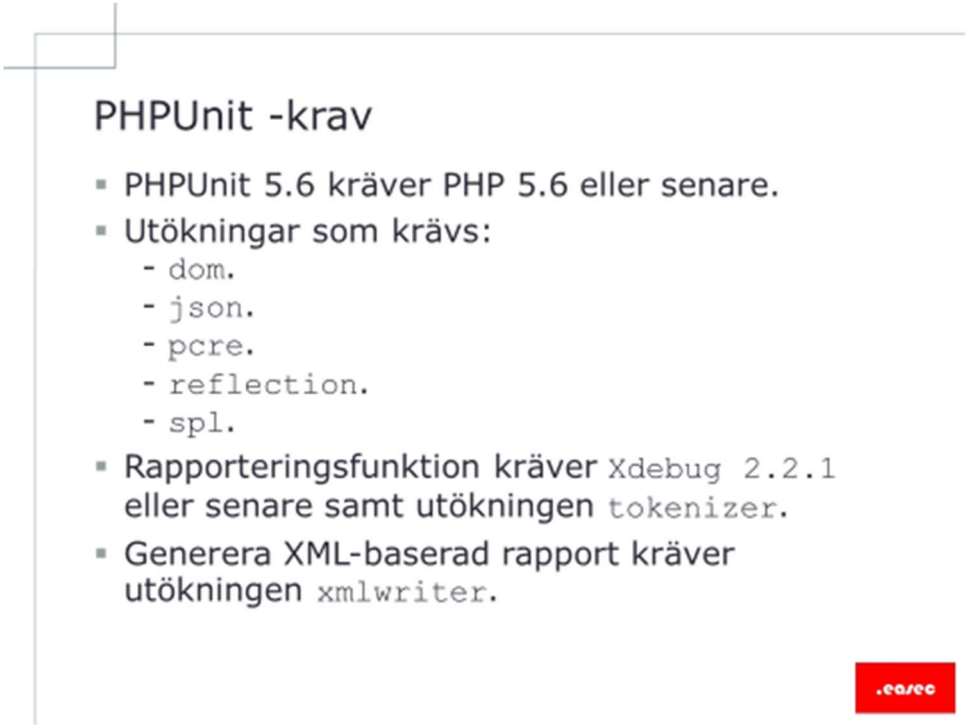
PHPUnit

PHPUnit är det mest kända ramverket för att skriva Unit Test för PHP-baserade applikationer.

Med hjälp av PHPUnit kan du bedriva utveckling enligt principerna för Test-Driven Development.

Används via konsol, tillhandahåller klass för testning, denna klass kan utökas efter behov.

Produkten tillhandahåller också ett sätt för utvecklare att ändra beteende, detta kallas för assertion.



PHPUnit

För att kunna arbeta med PHPUnit 5.6 krävs det PHP 5.6 eller senare.

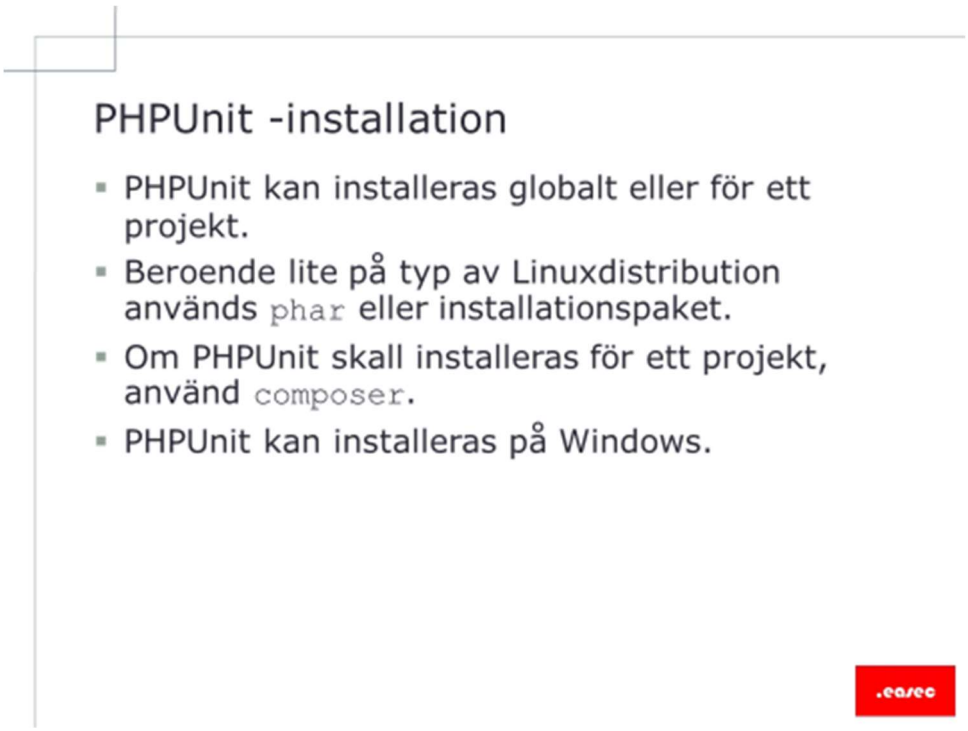
Utökningar som krävs:

- `dom`.
- `json`.
- `pcre`.
- `reflection`.
- `spl`.

Alla utökningar ovan installeras som standard när PHP installeras. Exkludering kan ske när vi själva kompilerar PHP.

Rapporteringsunktionen kräver `Xdebug 2.2.1` eller senare samt utökningen `tokenizer`.

Generera XML-baserad rapport, kräver utökningen `xmlwriter`.



PHPUnit -installation

PHPUnit kan installeras global eller för ett projekt. Globalt i detta sammanhang är att produkten är nårbar för alla projekt. Väljer vi att installera produkten för ett projekt, finns produkten bara tillgänglig för detta projekt.

I övningen som finns för PHPUnit, kommer produkten att installeras globalt, i övningen för BeHat kommer vi att installera BeHat bara för det projektet som vi arbetar med. På ett liknande sätt kan PHPUnit installeras.

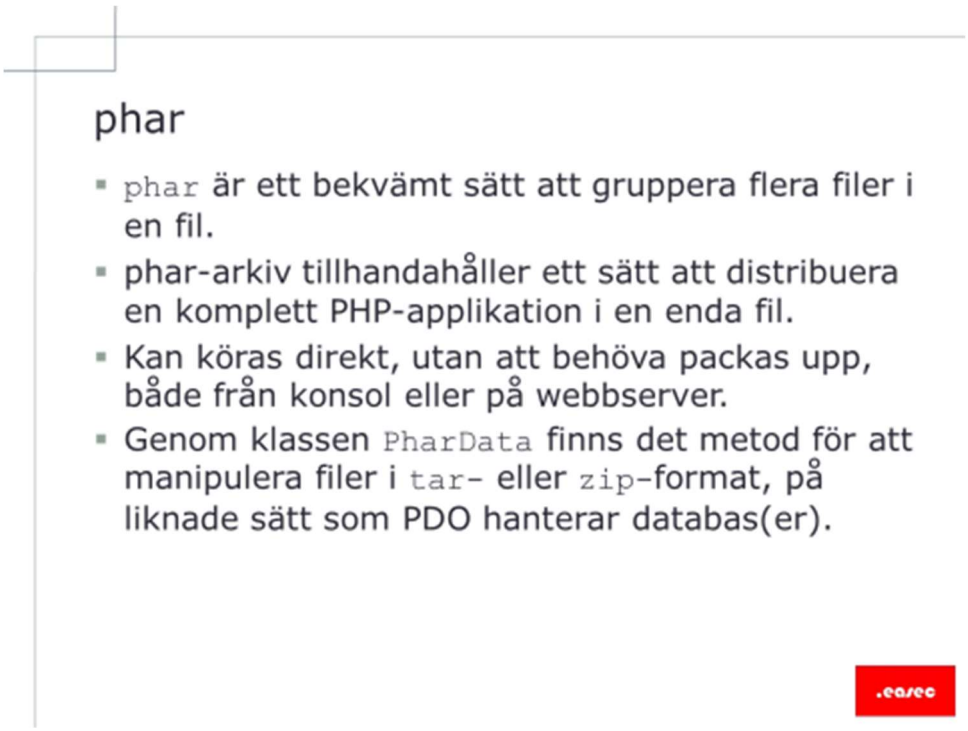
Beroende lite på typ av Linuxdistribution används `phar` eller installationspaket. I vår kursmiljö arbetar vi med ubuntu, för ubuntu finns det ett `apt`-paket för produkten, det är detta paket som vi kommer att installera.

PHPUnit rekommenderar att använda phar-paketet, används detta paket är det alltid den senaste versionen som används. Används `apt` kan det uppstå att det finns en nyare version.

Om PHPUnit skall installeras för ett projekt, använd `composer`. I en speciell fil, i projektets katalog, beskriver du alla eventuella beroende, `composer` ser till att dessa finns tillgängliga.

Konfiguration för programmet kan göras i en .xml-fil med namnet `phpunit.xml`.

PHPUnit kan installeras på Windows.



phar

`phar` är ett bekvämt sätt att gruppera flera filer i en fil.

phar-arkiv tillhandahåller ett sätt att distribuera en komplett PHP-applikation i en enda fil.

Kan köras direkt, utan att behöva packas upp, både från konsol eller på webbserver.

Genom klassen `PharData` finns det metod för att manipulera filer i `tar`- eller `zip`-formatet, på liknande sätt som PDO hanterar databas(er).

composer

`composer` är ett verktyg för att hantera beroende i PHP. Är inget verktyg för att paketera.

Du deklarerar vilka beroende som finns för ditt projekt, composer hanterar installation och uppdateringar av dessa.

Beroende deklareras i filen `composer.json`.

```
GNU nano 2.5.3                          File: composer.json
{
    "require-dev": {
        "behat/behat": "^3.2",
        "phpunit/phpunit": "^5.6"
    }
}
```

`composer` kräver PHP 5.3.2 eller senare.

Kan installeras global eller för ett projekt.

Kan köras i Windows.

Konventioner

När det gäller att arbeta med TDD generellt, eller med PHPUnit i synnerhet, finns det ett antal konventioner. Du kan strunta i dessa eller tolka dessa lite mer brett. Men detta kommer att försvåra ditt arbete och innebär oftast också att mer arbete än nödvändig får läggas ner.

Katalogstruktur:

Projekt skall alltid finnas i en egen katalog, döp katalog till ett namn som det är lätt att referera till applikationen.

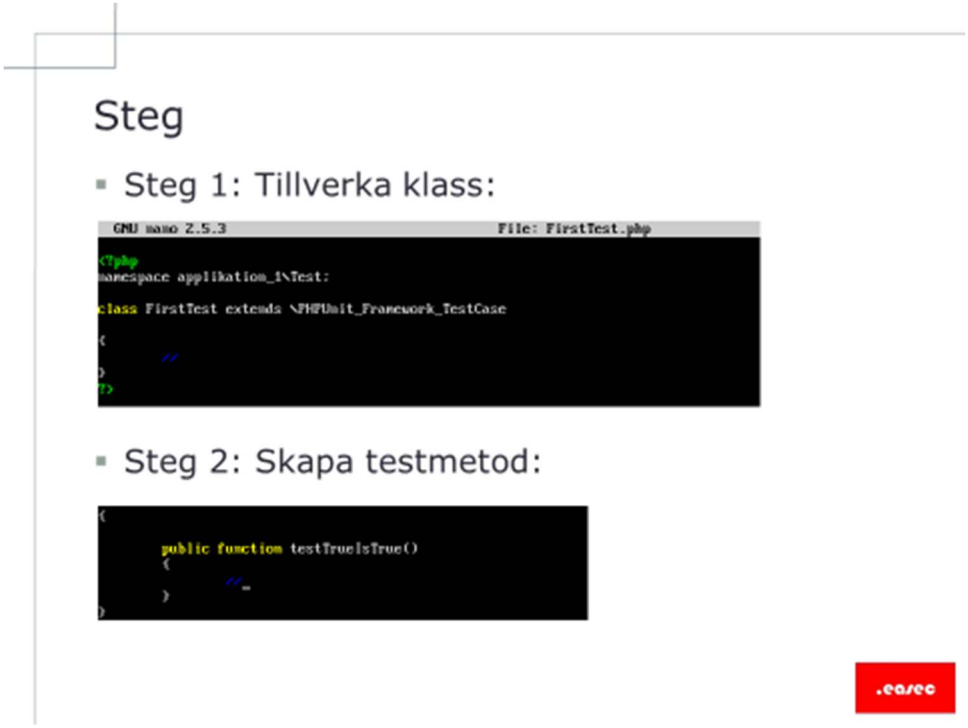
I denna katalog skapas en katalog för din programkod, i denna katalog skapas underkatalog med namnet Test, i denna katalog kommer dina olika tester att finnas. Är separerade från din ordinarie programkod.

Filstruktur

Filstrukturen skall spegla din kodbas, med sufixet Test. Test alltid med stor bokstav.

Exempelvis: `ApplikationTest.php`

Klassnamnet skall vara detsamma som filnamnet (Är inga konstigheter, har vi arbetat med hela tiden ☺)



Steg

Följ dessa steg:

Steg 1: Tillverka klass:

```
 GNU nano 2.5.3                                  File: FirstTest.php

<?php
namespace applikation_1\Test;

class FirstTest extends \PHPUnit_Framework_TestCase

{
        //
}
?>
```

Steg 2: Skapa testmetod:

```
{

        public function testTrueIsTrue()
        {
                //_
        }
}
```

Steg (forts.)

Steg 3: Lägg till testkod:

```
{
        public function testTrueIsTrue()
        {
                $foo = true;
                $this->assertTrue($foo);
        }
}
```

Steg 4: Kör!

```
mats@ubuntu:~/projekt/applikation_1/Test$ phpunit --colors FirstTest.php
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 24 ms, Memory: 4.00Mb

OK (1 test, 1 assertion)
mats@ubuntu:~/projekt/applikation_1/Test$ _
```

Övning arbeta med TDD

Installation av phpUnit

Arbetsuppgift 1: Starta och logga på din ubuntu baserade maskin.

Steg 1: Starta och logga på din ubuntu baserade maskin.

Steg 2: Skriv in kommandot: `sudo su`, klicka därefter på Enter. Ange lösenordet för root, klicka på Enter.

Arbetsuppgift 2: Installera produkt.

Steg 1: Skriv in kommandot: `apt-get update`, klicka därefter på Enter. Vi gör detta för att uppdatera referenslistor för olika distributionspunkter.

Installation av beroende.

Steg 2: Skriv in följande kommando: `apt-get install phpunit`, klicka därefter på Enter.

Klar att användas.

Arbeta med phpUnit

Arbetsuppgift 1: Starta och logga på din ubuntu baserade maskin.

Steg 1: Starta och logga på din ubuntu baserade maskin.

Arbetsuppgift 2: Skapa struktur

På din ubuntu baserade maskin, skapa följande struktur i din hemmakatalog:

projekt

|

---------- applikation_1

|

----------Test

Katalog skapar du med kommandot `mkdir namn_på_katalog`.

Arbetsuppgift 3: Skriv ditt första test, provkör detta test

Steg 1: Om du inte redan står i katalogen Test, förflytta dig till denna.

Steg 2: Skriv in kommandot: `nano FirstTest.php`, klicka därefter på Enter.

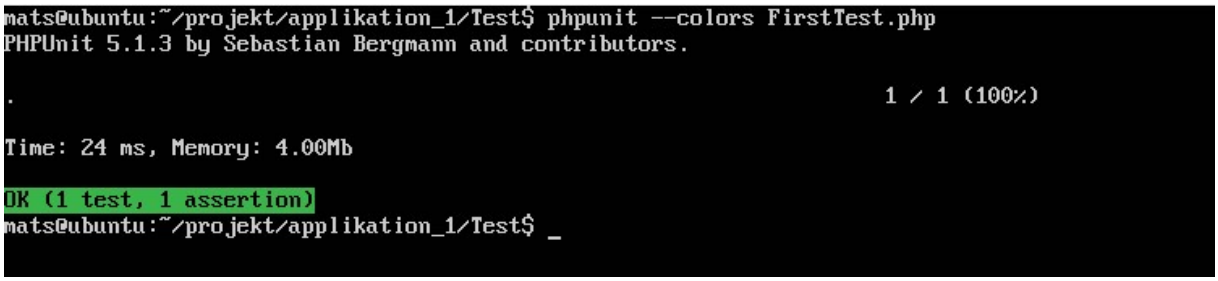
Steg 3: Skriv in PHP-kod enligt nedan:

```
<?php
namespace applikation_1\Test;

class FirstTest extends
\PHPUnit_Framework_TestCase

{
    public function testTrueIsTrue()
    {
        $foo = true;
        $this->assertTrue($foo);
    }
```

```
}
?>
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 4: Skriv in kommandot: `phpunit --colors FirstStep.php`, klicka därefter på Enter.



Ditt första lyckade test!

Assertion som vi la till verifierar att uttrycket är lika med true.

Arbetsuppgift 4: Ändra i ditt test och provkör

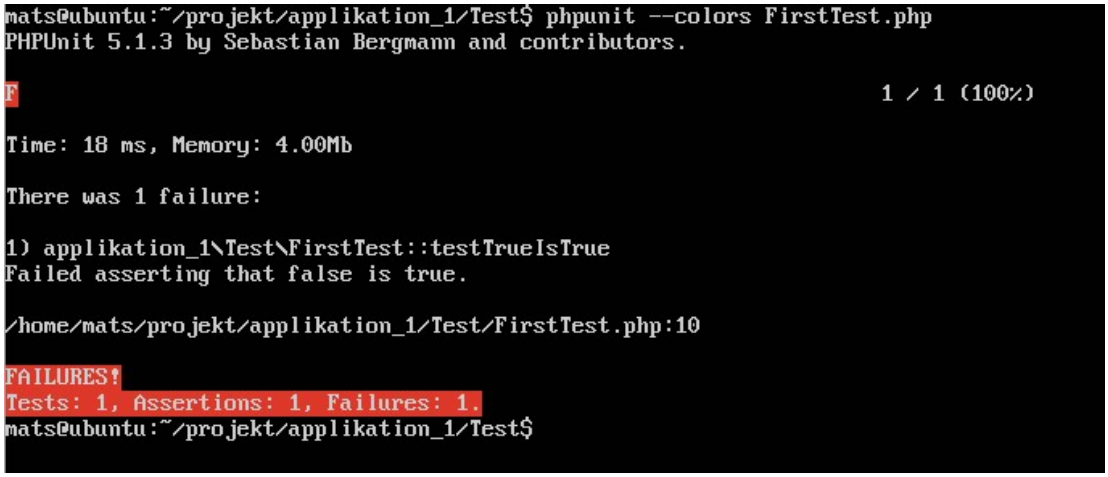
Steg 1: Skriv in kommandot: `nano FirstTest.php`, klicka därefter på Enter.

Steg 3: Ändra i din kod enligt nedan:

```
public function testTrueIsTrue()
{
    $foo = false;
    $this->assertTrue($foo);
}
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 4: Skriv in kommandot: `phpunit --colors FirstStep.php`, klicka därefter på Enter.



Vårt första misslyckade test!

Vad händer om vi ändrar assert till false är false (if (false == false))?

Låt oss prova.

Arbetsuppgift 5: Ändra i ditt test och provkör

Steg 1: Skriv in kommandot: `nano FirstTest.php`, klicka därefter på Enter.

Steg 3: Ändra i din kod enligt nedan:

```
public function testTrueIsFalse()
{
    $foo = false;
    $this->assertFalse($foo);
}
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 4: Skriv in kommandot: `phpunit --colors FirstStep.php`, klicka därefter på Enter.

Testet lyckades, pga assertion är lika med true även om metoden kallas för `assertFalse()`.

PHPUnit levereras med över 90 stycken olika assertion, vanligaste som man använder är kanske: `assertHasKey()`, `assertEquals()`, `assertFalse()`, `assertSame()` och `assertTrue()`.

Arbetsuppgift 6: Skapa ny klass och ett nytt test

Vi skall skapa ett program som av en rad som ges in skapar utdata där eventuella mellanslag ersätts med bindestreck.

Exempelvis: "This string will be sluggified" blir "this-string-will-be-sluggified".

Steg 1: Gå till katalogen projekt/applikation_1.

Steg 2: Skriv in kommandot: `nano URL.php`, klicka därefter på Enter.

Steg 3: Skriv in kod enligt nedan:

```
<?php
namespace applikation_1;

class URL
{
        public function sluggify($string, $separator = '-', $maxLength =96)
        {
                $title = iconv('UTF8', 'ASCII//TRANSLIT', $string);
                $title = preg_replace("%[^-/+|\w ]%", '', $title);
                $title = strtolower(trim(substr($title, 0, $maxLength), '-'));
                $title = preg_replace("/[\/_|+ -]+/", $separator, $title);

                return $title;
        }
}
?>
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 4: Skapa test, gå ner till katalogen Test, skriv in kommandot: `nano URLTest.php`, klicka därefter på Enter.

Skriv in följande kod:

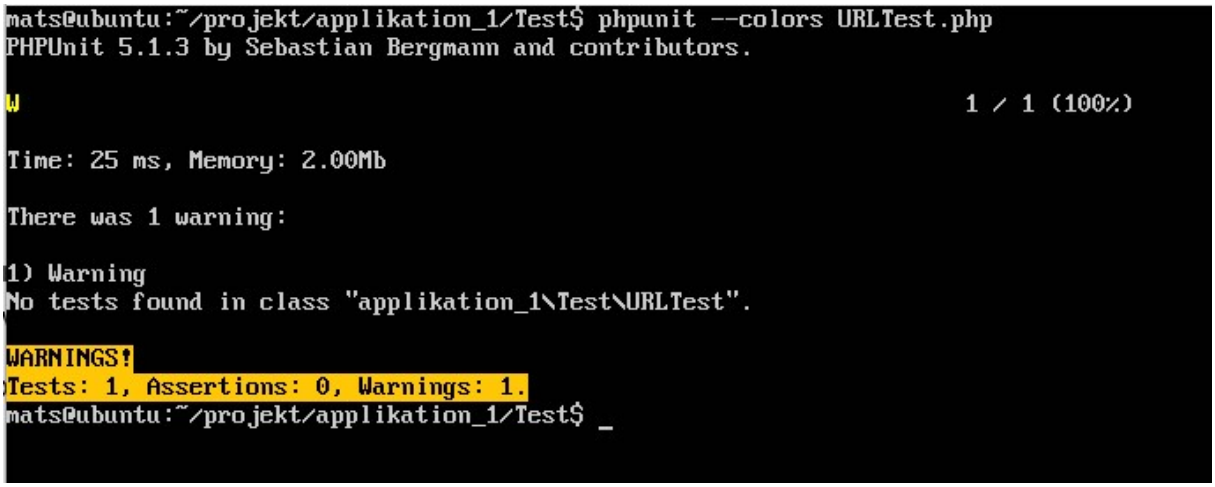
```
<?php

namespace applikation_1\Test;

class URLTest extends
\PHPUnit_Framework_TestCase

{
```

```
}

?>
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 5: Skriv in kommandot: `phpunit --colors URLTest.php`, klicka därefter på Enter.

Testet kommer att få ett varningsmeddelande, då det inte finns några testkod i din klass. Skälet till detta steg är att kontrollera så det inte är några bekymmer med grundkonstruktionen av klassen.



Steg 6: Skriv in kommandot: `nano URLTest.php`, klicka därefter på Enter.

Lägg till markerad kod nedan:

```
<?php

namespace applikation_1\Test;

class URLTest extends
\PHPUnit_Framework_TestCase

{

    public function
testSluggifyReturnsSluggifiedString()
```

```
        {

        }

}

?>
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 7: Skriv in kommandot: `phpunit --colors URLTest.php`, klicka därefter på Enter.

Testet lyckades. Vi har test funktionalitet för att kunna gå vidare.

Arbetsuppgift 7: Bygg ut ditt test

Steg 1: Skriv in kommandot: `nano URLTest.php`, klicka därefter på Enter.

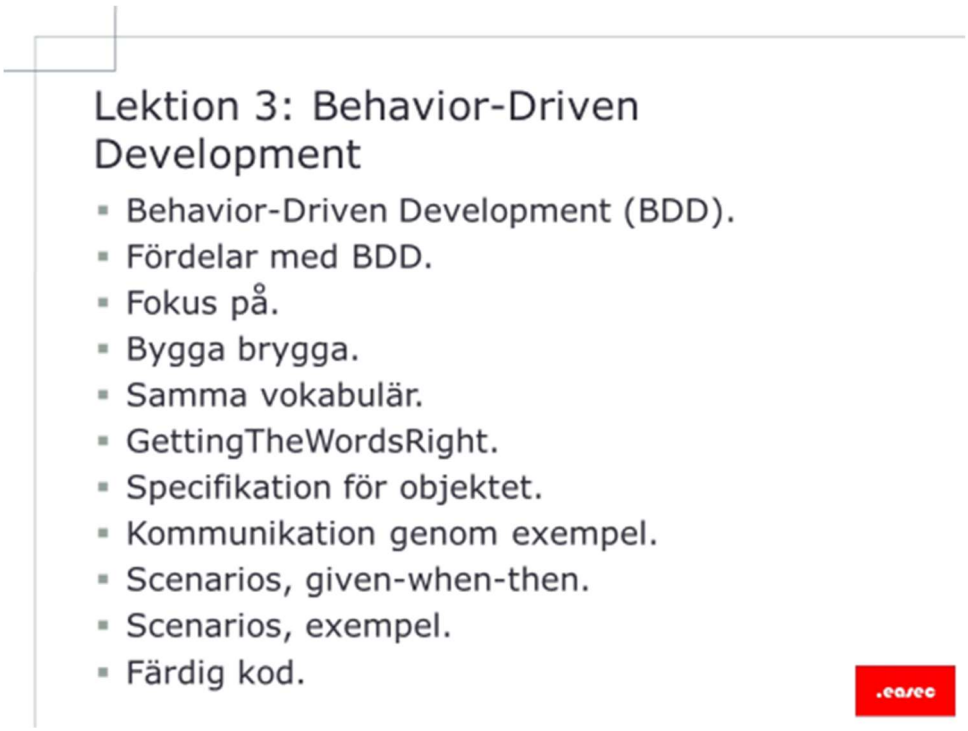
Lägg till markerad kod nedan:

```
<?php
namespace applikation_1\Test;

require_once __DIR__.'/../URL.php';

use applikation_1\URL;

class URLTest extends \PHPUnit_Framework_TestCase
{
        public function testSluggifyReturnsSluggifiedString()
        {
                $originalString = 'This string will be sluggified';
                $expectedResult = 'this-string-will-be-sluggified';

                $url = new URL();
                $result = $url->sluggify($originalString);

                $this->assertEquals($expectedResult, $result);


        }
}
?>
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 2: Skriv in kommandot: `phpunit --colors URLTest.php`, klicka därefter på Enter.

Testet lyckades.



Lektion 3: Behavior-Driven Development

I denna lektion skall vi titta på:

- Behavior-Driven Development (TDD).
- Fördelar med BDD.
- Fokus på.
- Bygga brygga.
- Samma vokabulär.
- GettingTheWordsRight.
- Specifikation för objektet.
- Kommunikation genom exempel.
- Scenarios, given-when-then.
- Scenarios, exempel.
- Färdig kod.

Lektion 3: Behavior-Driven Development (forts.)

BeHat.

BeHat –krav.

BeHat –installation.

Behavior-Driven Development (BDD)

Är en utvecklingsteknik som har sitt ursprung från Test-Driven Development (TDD).

BDD använder sig av ett enkelt språk, domain-specific scripting language (DSL), för att konvertera satser till exekverbara satser.

Resultatet kommer närmre kriteria för acceptans för en given funktion, testet annvänds för att validera funktionalitet.

Naturlig utökning av TDD.

Fördelar med BDD

Allt utvecklingsarbete kan spåras direkt till mål för utvecklingen.

Utvecklingen svarar på vad användare behöver. Vilket i slutändan kommer att ge nöjda kunder och för oss en bra affär.

Effektiv prioritering – kritiska funktioner levereras först.

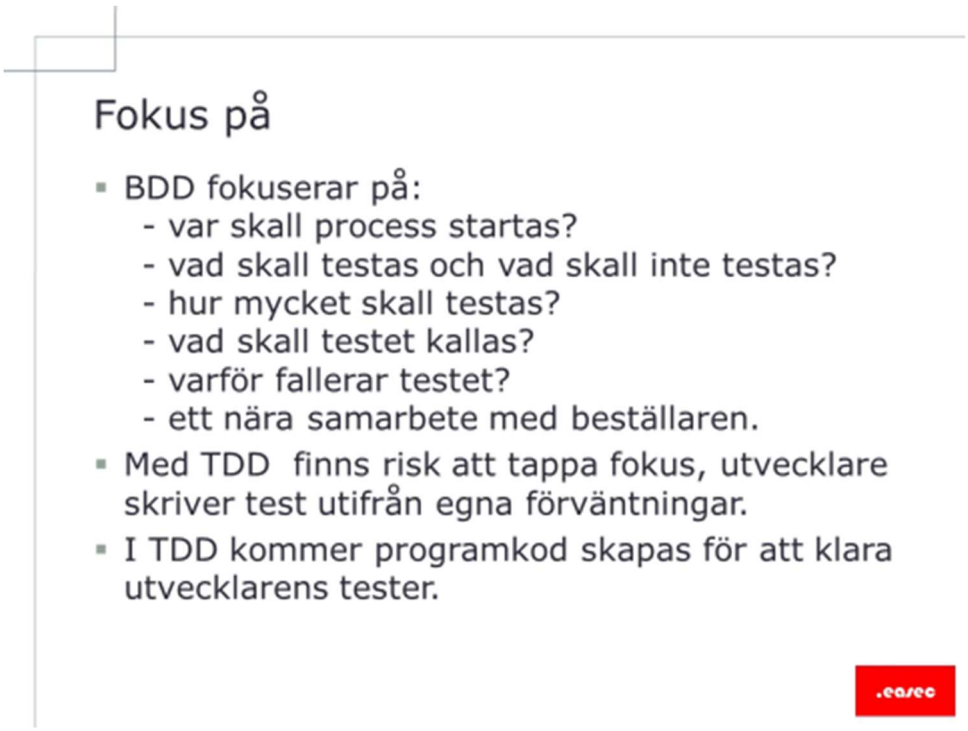
Alla involverade delar samma uppfattning om projektet och kan vara delaktiga i kommunikation kring projektet.

Delat språk ser till att alla har grundlig insyn i projektets utveckling.

Fördelar med BDD (forts.)

Resultatet av mjukvarudesign matchar existerande och stödjer kommande behov för företaget.

Förbättrad kvalité för programkoden och reducerar risker för projektet.



Fokus på

BDD fokuserar på:

- Vad skall process startas?
- Vad skall testas och vad skall inte testas?
- Hur mycket skall testas?
- Vad skall testet kallas?
- Varför fallerar testet?
- Ett nära samarbete med beställaren.

Med TDD finns det risk att tappa fokus, utvecklaren skriver test utifrån egna förväntningar. Sitt eget sätt att skriva programkod.

I TDD kommer programkod skapas för att klara utvecklarens tester.

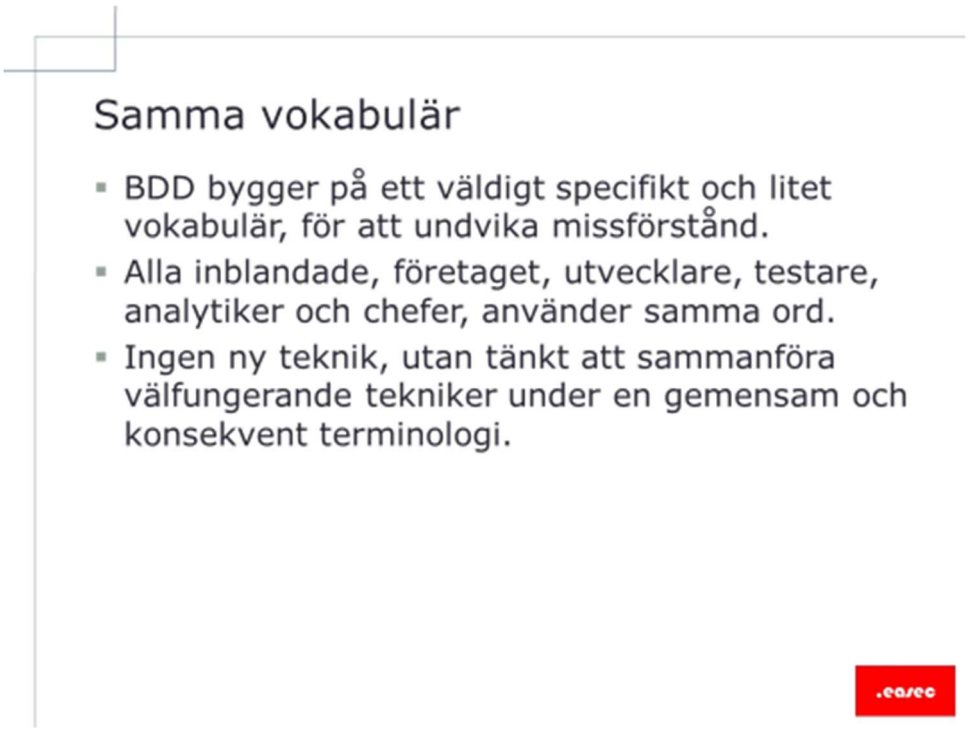
Bygga brygga

BDD försöker bygga brygga mellan olika synsätten på ett datorsystem, mellan de som skall använda systemet och de som utvecklar det.

Utgår från TDD och influerat av Domain Driven Design.

Fokuserat på att minimera hindren mellan specifikation, design, implementering och acceptans, för ett system.

Ger möjlighet till inkrementel leverans av ett system och ger möjlighet till utvecklingsteam att snabbt ta till sig agila metoder.

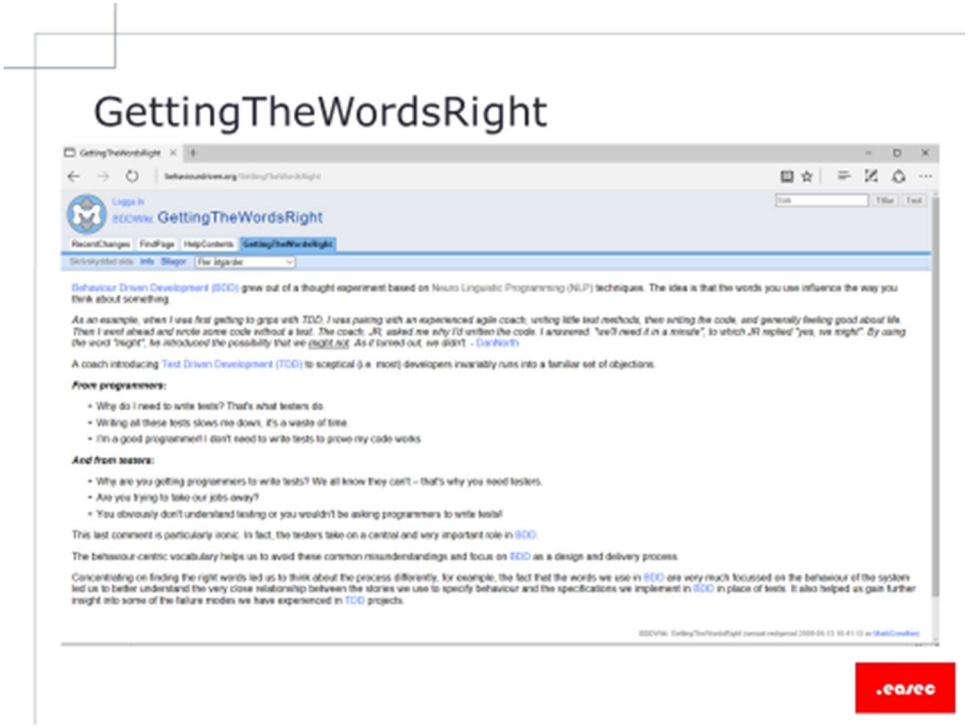


Samma vokabulär

BDD bygger på ett väldigt specific och litet vokabulär, allt för att undvika missförstånd.

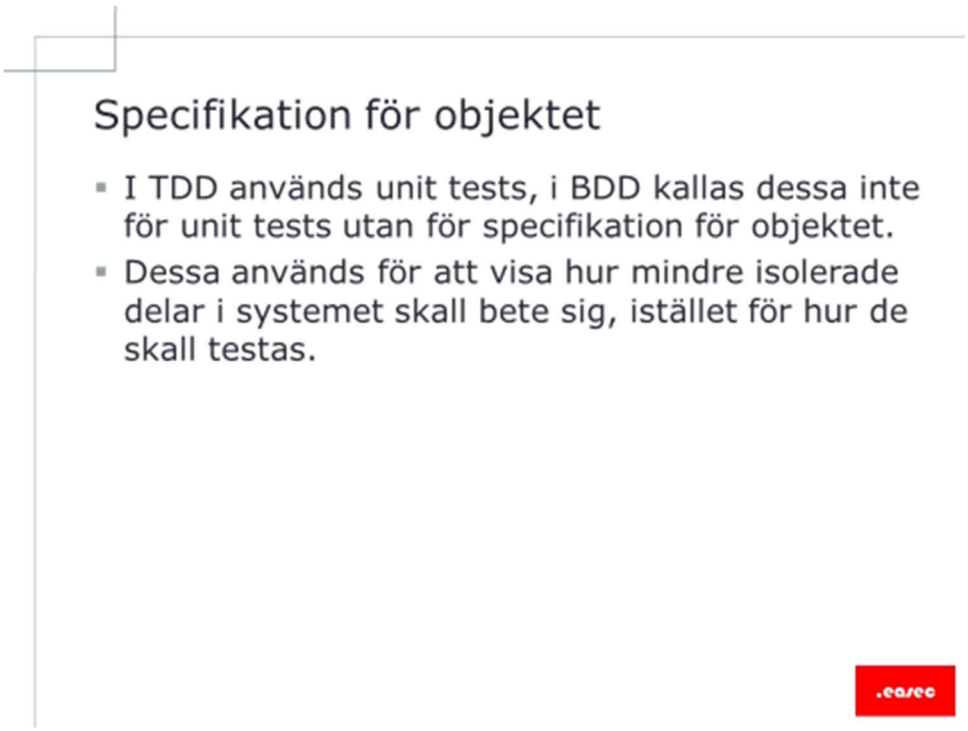
Alla inblandande, dvs företaget, utvecklare, testare, analytiker och chefer, använder samma ord.

Ingen ny teknik, utan tänkt att sammanföra välfungernade tekniker under en gemensam och konsekvent terminologi.



GettingTheWordsRight

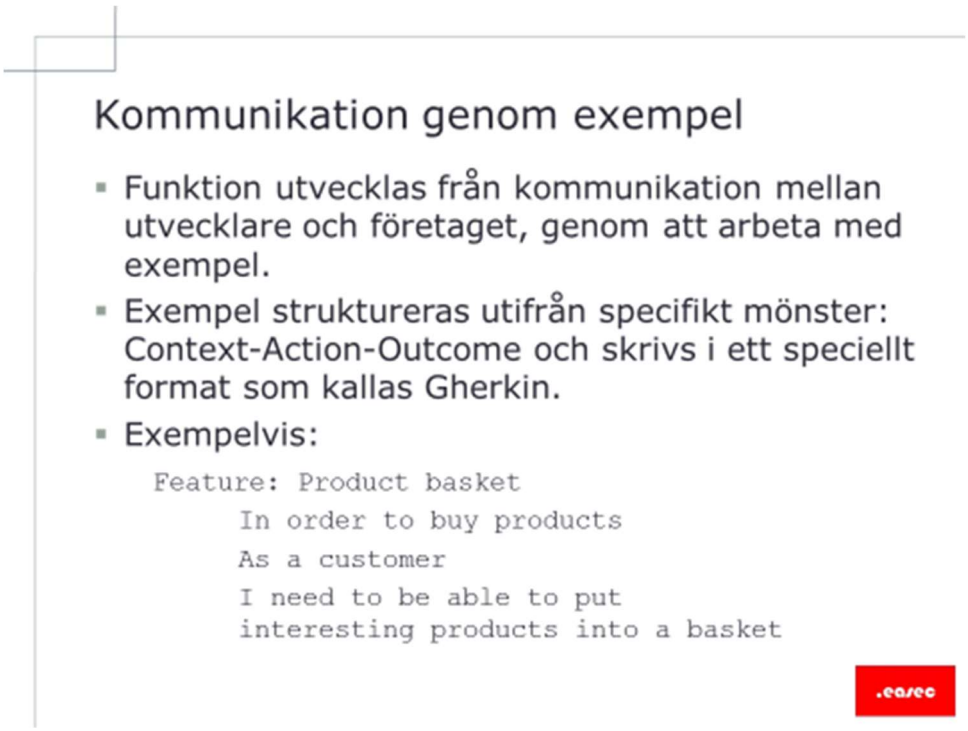
Klassisk situation, eller hur? ☺



Specifikation för objektet

I TDD används unit tests, i BDD kallas dessa inte för unit tests, utan för specifikation för objektet.

Dessa används för att visa hur mindre isolerade delar i systemet skall bete sig, istället (som i TDD) hur de skall testas.



Kommunikation genom exempel

Funktion utvecklas från kommunikation mellan utvecklare och företaget, exempel används för att få en gemensam plattform för kommunikationen.

Exempel struktureras utifrån specifikt mönster: Context-Action-Outcome och skrivs i ett speciellt format som kallas Gherkin.

Du skall utveckla en webbshop, i kommunikationen kommer ni fram till att det skall finnas en varukorg, I denna varukorg skall kunden kunna lägga ett antal produkter.

Bra och tydlig avgränsning för exemplet.

Detta kan se ut på detta sättet:

```
Feature: Product basket

    In order to buy products

    As a customer
```

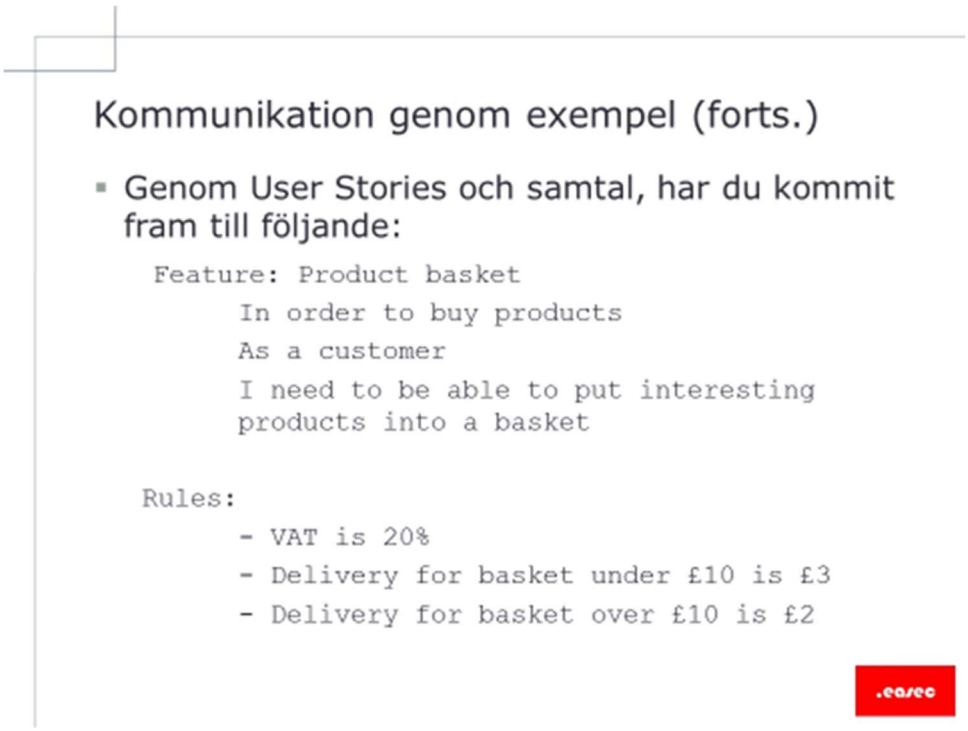
```
I need to be able to put
interesting products into a basket
```

Exemplet beskriver problemet enligt det mönster som skall användas, Context-Action-Outcome.

Context: Product basket.

Action: kunna lägga produkt i varukorg som kund.

Outcome: att produkt skall finnas i varukorgen.

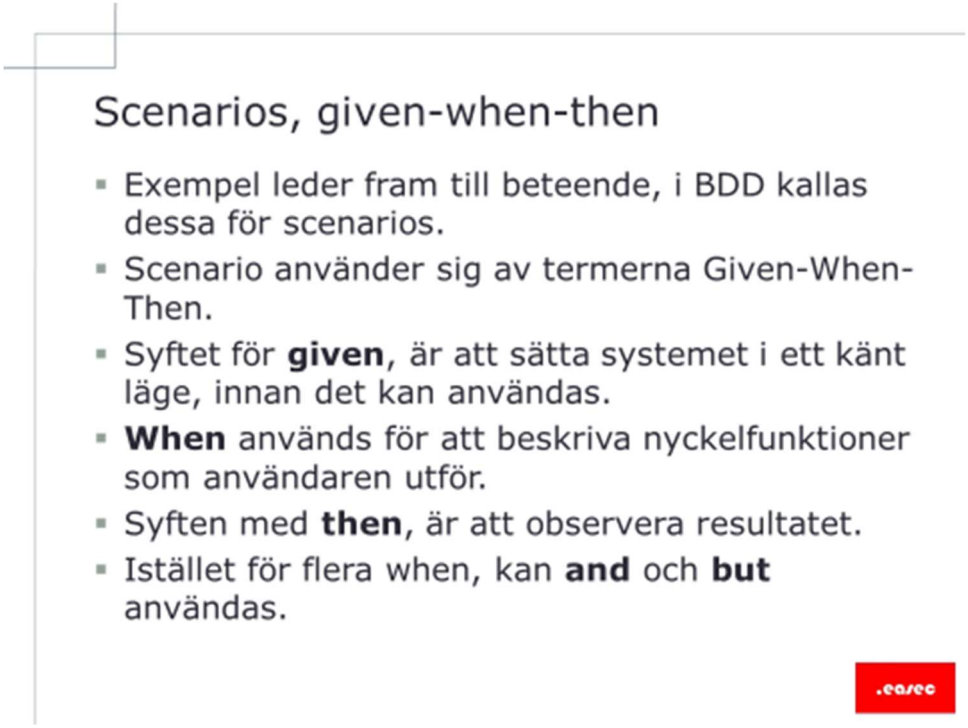


Kommunikation genom exempel (forts.)

Genom User Stories och samtal, har du kommit fram till följande:

```
Feature: Product basket

        In order to buy products

        As a customer

        I need to be able to put interesting
        products into a basket

    Rules:

        - VAT is 20%

        - Delivery for basket under £10 is £3

        - Delivery for basket over £10 is £2
```

Dvs förutom den tidigare informationen, finns det även information om regelverk, i detta fallet är det påslag av moms och kostnad för frakt. Kostnaden för frakt kan vara olika, beroende på priset för produkten.



Scenarions, given-when-then

Exempel leder fram till ett beteende, i BDD kallas dessa för scenarios.

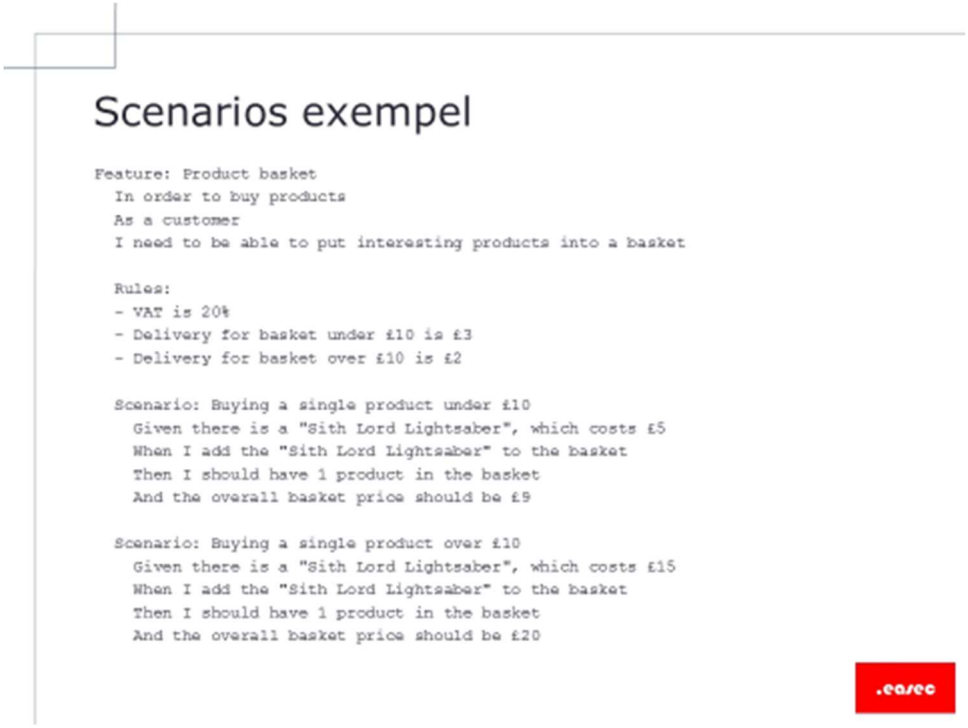
Scenario använder sig av termerna Given-When-Then.

Syftet för **given**, är att sätta systemet i ett känt läge, innan det kan användas.

When används för att beskriva nyckelfunktioner som användaren utför.

Syften med **then**, är att observera resultatet.

Istället för flera when, kan **and** och **but** användas.



Scenarios exempel

```
Feature: Product basket

  In order to buy products

  As a customer

  I need to be able to put interesting products
into a basket

  Rules:

  - VAT is 20%

  - Delivery for basket under £10 is £3

  - Delivery for basket over £10 is £2

  Scenario: Buying a single product under £10

    Given there is a "Sith Lord Lightsaber",
which costs £5

    When I add the "Sith Lord Lightsaber" to the
basket
```

```
    Then I should have 1 product in the basket

    And the overall basket price should be £9

  Scenario: Buying a single product over £10

    Given there is a "Sith Lord Lightsaber",
which costs £15

    When I add the "Sith Lord Lightsaber" to the
basket

    Then I should have 1 product in the basket

    And the overall basket price should be £20
```

Färdig kod

Utifrån scenario kommer färdig kod att genereras. Denna kod får vi åtgärda på ett liknade sätt som i all annan programmering.

Någon metod kanske kan förändras, göra koden mer lättläst etc. Det som i dagliga talet kallar för refactoring.

```
// features/bootstrap/Basket.php

final class Basket implements \Countable
{
    private $shelf;
    private $products;
    private $productsPrice = 0.0;

    public function __construct(Shelf $shelf)
    {
        $this->shelf = $shelf;
    }

    public function addProduct($product)
    {
        $this->products[] = $product;
        $this->productsPrice += $this->shelf->getProductPrice($product);
    }

    public function getTotalPrice()
    {
        return $this->productsPrice
            + ($this->productsPrice * 0.2)
            + ($this->productsPrice > 10 ? 2.0 : 3.0);
    }

    public function count()
    {
        return count($this->products);
    }
}
```

Övning arbeta med BDD

Installation av composer

Jag har gjort ett litet bash skript som installerar `curl`, `php-cli`, `git`, `unzip` och `composer`.

`composer` är lite speciell för man skall ange SHA-384 hash för att verifiera installationspaketet. I skriptet har jag lagt in denna, men tänk på att denna hash ändras om ny version släpps.

Arbetsuppgift 1: Ladda ner skriptet och flytta scriptet till LAMP-server

Steg 1: På din dator, öppna webbläsare och skriv in URL:en https://course.easec.se/installcomp.tar, klicka därefter på Enter.

Steg 2: Starta WinSCP, anslut och flytta filen till servern.

Arbetsuppgift 2: Packa upp filen, flytta filen

Steg 1: I konsol, skriv in följande kommando: `sudo su`, klicka därefter på Enter. Skriv in lösenordet för root, klicka därefter på Enter.

```
root@ubuntu:/home/mats# tar -zxvf installcomp.tar
installcomp.sh
root@ubuntu:/home/mats# _
```

Steg 2: Skriv in kommandot: `tar -zxvf installcomp.tar`, klicka därefter på Enter.

Flytta skript.

Steg 2: Skriv in kommandot: `cp /home/ditt_namn/installcomp.sh /bin`, klicka därefter på Enter.

Förflytta dig till `/bin`.

Steg 3: Skriv in kommandot: `cd /bin`, klicka därefter på Enter.

Ändra rättigheter.

Steg 4: Skriv in kommandot: `chmod 707 installcomp.sh`, klicka därefter på Enter.

Exekvera.

Steg 5: Skriv in kommandot: `installcomp.sh`, klicka på Enter.

```
root@ubuntu:/home/mats# cp installcomp.sh /bin
root@ubuntu:/home/mats# cd /bin
root@ubuntu:/bin# chmod 0707 installcomp.sh
root@ubuntu:/bin# installcomp.sh
Kommer att installera curl php-cli git och unzip
Läser paketlistor… Färdig
Bygger beroendeträd
Läser tillståndsinformation… Färdig
git is already the newest version (1:2.7.4-0ubuntu1).
php-cli is already the newest version (1:7.0+35ubuntu6).
unzip is already the newest version (6.0-20ubuntu1).
curl is already the newest version (7.47.0-1ubuntu2.1).
0 att uppgradera, 0 att nyinstallera, 0 att ta bort och 62 att inte uppgradera.
Laddar ner och installerar composer
```

Skriptet:

```
 GNU nano 2.5.3                    File: installcomp.sh

#!/bin/bash
echo "Kommer att installera curl php-cli git och unzip"
apt-get install curl php-cli git unzip
cd ~
echo "Laddar ner och installerar composer"
curl -sS https://getcomposer.org/installer -o composer-setup.php
php -r "if (hash_file('SHA384', 'composer-setup.php') === 'e115a8dc7871f15d853148a7fbac7da27d6c0030b848d9b3dc09e2a0388afed865e6a3d6b3c0fad45c4
echo "Verifierar installationsarkivet. OBS! SHA-384 Hash ändras om ny installation, verifiera och ev ändra."
php composer-setup.php --install-dir=/usr/local/bin --filename=composer
echo "Startar composer"
composer
```

Signatur kan ändras, kontrollera på:
https://composer.github.io/pubkeys.html .

Arbeta med Behat/phpunit

Arbetsuppgift 1: Skapa ny katalog med namnet projekt, sätt rättigheter och förflytta dig till den nyss skapade katalog

Steg 1: Skriv in kommandot: `sudo mkdir projekt`, klicka därefter på Enter. Ange lösenord för root, klicka därefter på Enter.

Steg 2: Skriv in kommandot: `sudo chmod 777 projekt`, klicka därefter på Enter.

Steg 3: Skriv in kommandot: `cd projekt`, klicka därefter på Enter.

```
mats@ubuntu:/$ sudo mkdir projekt
mats@ubuntu:/$ sudo chmod 777 projekt
mats@ubuntu:/$ cd projekt
mats@ubuntu:/projekt$
```

Arbetsuppgift 2: Klona arkivet på github som innehåller scenario

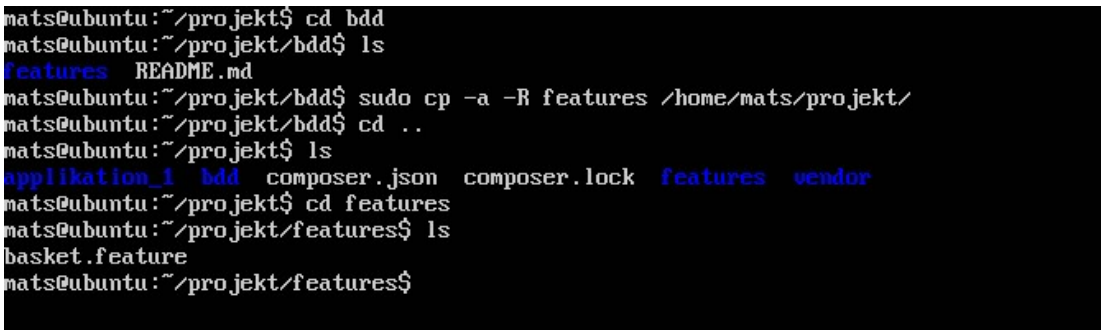
Steg 1: Skriv in kommandot: `git clone https://github.com/easec/bdd.git`, klicka därefter på Enter.

```
mats@ubuntu:/projekt$ git clone https://github.com/easec/bdd.git
Cloning into 'bdd'...
remote: Counting objects: 10, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (10/10), done.
Checking connectivity... done.
mats@ubuntu:/projekt$ _
```

Arbetsuppgift 3: Kopiera katalogen features (som finns under katalogen bdd) till /projekt/features

Steg 1: Skriv in kommandot: `cd bdd`, klicka därefter på Enter.

Steg 2: Skriv in kommandot: `sudo cp -a -R features /home/ditt_namn/projekt/features`, klicka därefter på Enter.

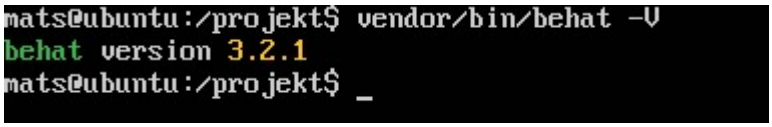


Steg 3: Förflytta dig till katalogen projekt, genom att skriva kommandot: `cd ..`, klicka därefter på Enter.

Arbetsuppgift 4: Installera Behat och verifiera installationen

Steg 1: Skriv in kommandot: `composer require --dev behat/behat`, klicka därefter på Enter.

Steg 2: Skriv in kommandot: `vendor/bin/behat -V`, klicka därefter på Enter.



Arbetsuppgift 5: Initiera Behat

Steg 1: Skriv in kommandot: `sudo vendor/bin/behat --init`, klicka därefter på Enter.

```
mats@ubuntu:~/projekt$ sudo vendor/bin/behat --init
+d features/bootstrap - place your context classes here
+f features/bootstrap/FeatureContext.php - place your definitions, transformations and hooks here

mats@ubuntu:~/projekt$ _
```

Behat skapar ny katalog med namnet bootstrap, denna katalog finns under katalogen features. Det är i denna katalog som vi lägger vår programkod.

Behat talar även om vad som saknas i ditt projekt.

Arbetsuppgift 6: Börja arbeta med Behat

Steg 1: Skriv in kommandot: `vendor/bin/behat`, klicka därefter på Enter.

```
mats@ubuntu:/projekt$ vendor/bin/behat
PHP Warning:  file_put_contents(/tmp/behat_gherkin_cache/v4.4-dev/c707fee4147642a39bc5fc2cc7a5396f.feature.cache): failed to open stream: Perm
sion denied in /projekt/vendor/behat/gherkin/src/Behat/Gherkin/Cache/FileCache.php on line 95
Feature: Product basket
       In order to buy products
       As a customer
  I need to be able to put interesting products into a basket

       Rules:
  - VAT is 20%
  - Delivery for basket under £10 is £3
  - Delivery for basket over £10 is £2

  Scenario: Buying a single product under £10
    Given there is a "Sith Lord Lightsaber", which costs £5
    When I add the "Sith Lord Lightsaber" to the basket
    Then I should have 1 product in the basket
    And the overall basket price should be £9

  Scenario: Buying a single product over £10
    Given there is a "Sith Lord Lightsaber", which cost £15
    When I add the "Sith Lord Lightsaber" to the basket
    Then I should have 1 product in the basket
    And the overall basket price should be $20

  Scenario: Buying two products over £10
    Given there is a "Sith Lord Lightsaber", whish costs £10
    And there is a "Jedi Lightsaber", which costs £5
    When I add the "Sith Lord Lightsaber" to the basket
    And I add the "Jedi Lightsaber" to the basket
    Then I should have 2 products in the basket
    And the overall basket price should be £20

3 scenarios (3 undefined)
14 steps (14 undefined)
0m0.01s (6.83Mb)

 >> default suite has undefined steps. Please choose the context to generate snippets:

  [0] None
  [1] FeatureContext
 >
```

Steg 2: Skriv `0` som svar på frågan Please choose the context to generate snippets. Klicka därefter på Enter.

```
> 0

--- Use --snippets-for CLI option to generate snippets for following default suite steps:

    Given there is a "Sith Lord Lightsaber", which costs £5
    When I add the "Sith Lord Lightsaber" to the basket
    Then I should have 1 product in the basket
    And the overall basket price should be £9
    Given there is a "Sith Lord Lightsaber", which cost £15
    And the overall basket price should be $20
    Given there is a "Sith Lord Lightsaber", whish costs £10
    And there is a "Jedi Lightsaber", which costs £5
    And I add the "Jedi Lightsaber" to the basket
    Then I should have 2 products in the basket
    And the overall basket price should be £20

mats@ubuntu:/projekt$
```

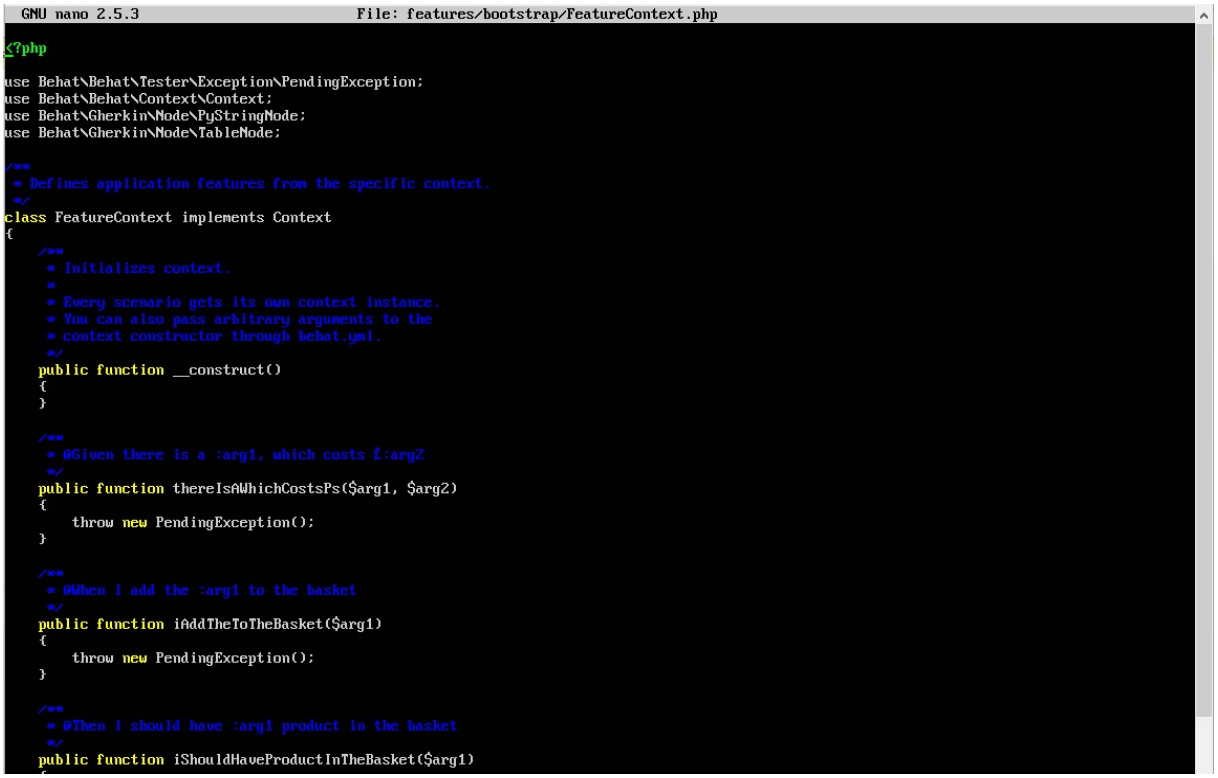
Arbetsuppgift 7: Låt Behat lägga till snippets

Steg 1: Skriv in kommandot: `sudo vendor/bin/behat --dry-run --append-snippets`, klicka därefter på Enter.

Steg 2: Skriv `1` som svar på frågan Please choose the context to generate snippets och att vill göra detta för `FeatureContext`. Klicka därefter på Enter.

```
>> default suite has undefined steps. Please choose the context to generate snippets:

  [0] None
  [1] FeatureContext
> 1

u features/bootstrap/FeatureContext.php - `there is a "Sith Lord Lightsaber", which costs £5` definition added
u features/bootstrap/FeatureContext.php - `I add the "Sith Lord Lightsaber" to the basket` definition added
u features/bootstrap/FeatureContext.php - `I should have 1 product in the basket` definition added
u features/bootstrap/FeatureContext.php - `the overall basket price should be £9` definition added
u features/bootstrap/FeatureContext.php - `there is a "Sith Lord Lightsaber", which cost £15` definition added
u features/bootstrap/FeatureContext.php - `the overall basket price should be $20` definition added
u features/bootstrap/FeatureContext.php - `there is a "Sith Lord Lightsaber", whish costs £10` definition added
u features/bootstrap/FeatureContext.php - `I should have 2 products in the basket` definition added
mats@ubuntu:/projekt$ _
```

Steg 3: Nu är det hög tid att titta på koden som har genererats, skriv in kommandot `sudo nano features/bootstrap/FeatureContext.php`, klicka därefter på Enter.



Som du ser, så utifrån scenario som vi beskrev i textfilen, har Behat skapat programkod åt oss. Låt nano vara öppet.

Nu kan vi själva ta över, eller låta Behat fortsätta ...

Trodde väl det ☺, vi fortsätter!

Arbetsuppgift 8: Ändra i programkoden

Steg 1: Ändra koden enligt markeringarna nedan:

```
<?php

use Behat\Behat\Tester\Exception\PendingException;
use Behat\Behat\Context\SnippetAcceptingContext;
use Behat\Gherkin\Node\PyStringNode;
use Behat\Gherkin\Node\TableNode;

/**
 * Defines application features from the specific context.
 */
class FeatureContext implements SnippetAcceptingContext
{
    private $shelf;
    private $basket;
```

```
    /**
     * Initializes context.
     *
     * Every scenario gets its own context instance.
     * You can also pass arbitrary arguments to the
     * context constructor through behat.yml.
     */
    public function __construct()
    {
    $this->shelf = new Shelf();
    $this->basket = new Basket($this->shelf);
    }

    /**
     * @Given there is a :arg1, which costs £:arg2
     */
    public function thereIsAWhichCostsPs($product, $price)
    {
        $this->shelf->setProductPrice($product,
floatval($price));
    }

    /**
     * @When I add the :arg1 to the basket
     */
    public function iAddTheToTheBasket($product)
    {
        $this->basket->addProduct($product);
    }

    /**
     * @Then I should have :arg1 product in the basket
     */
    public function iShouldHaveProductInTheBasket($count)
    {
        PHPUnit_Framework_Assert::assertCount(
         intval($count),
            $this->basket
        );
    }

    /**
     * @Then the overall basket price should be £:arg1
     */
    public function theOverallBasketPriceShouldBePs($price)
    {
        PHPUnit_Framework_Assert::assertSame(
           floatval($price),
           $this->basket->getTotalPrice()
     );
    }
}
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Som du märkte så la vi till 2 objekt, Shelf och Basket. Detta första objektet är ansvarig för att lagra produkter och pris, det andra objekt representerar kundens varukorg. De steg som ytterligare la tills var att deklarera priset för produkt och för att lägga till produkt till varukorg.

Nästa steg är att använda PHPUnit, för att kontrollera att objektet Basket innehåller vad vi hade tänkt oss. Detta görs genom att använda PHPUnit assertions.

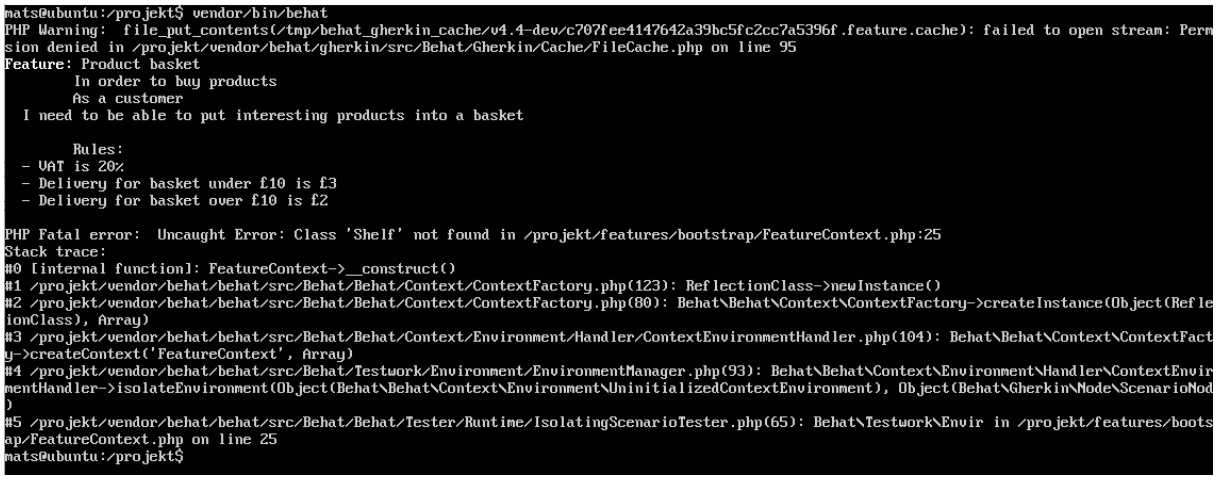
```
PHPUnit_Framework_Assert::assertCount(

   intval($count),

   $this->basket
);


PHPUnit_Framework_Assert::assertSame(
   floatval($price),
   $this->basket->getTotalPrice()
);
```

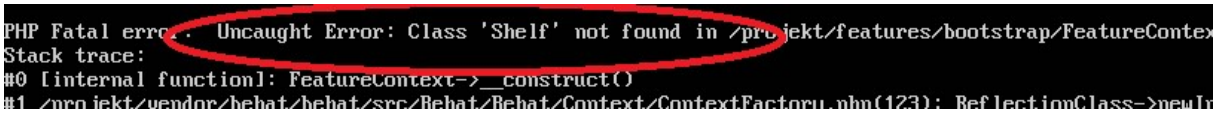
Arbetsuppgift 9: Installera PHPUnit

Steg 1: Skriv in kommandot: `sudo composer require --dev phpunit/phpunit`, klicka därefter på Enter.

Arbetsuppgift 10: Kör Behat tillsammans med PHPUnit

Steg 1: Skriv in kommandot: `sudo vendor/bin/behat`, klicka därefter på Enter.





Obeservera att du får ett Fatal error, klassen Shelf saknas!

Arbetsuppgift 11: Skapa klassen Shelf

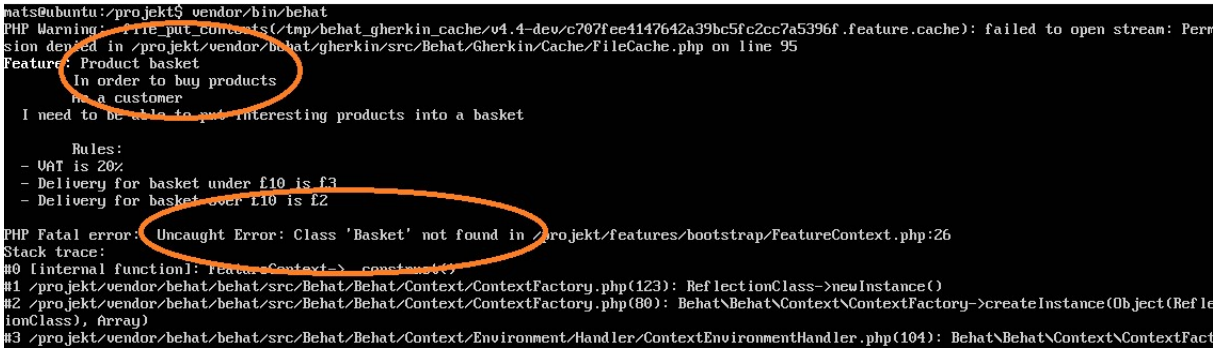
Steg 1: Skriv in kommandot: `sudo nano features/bootstrap/Shelf.php`, klicka därefter på Enter.

Steg 2: Skriv in enligt nedan:

```
<?php

final class Shelf

{

}

?>
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 3: Skriv in kommandot: `sudo vendor/bin/behat`, klicka därefter på Enter.



Obeservera att du får ett Fatal error, klassen Basket saknas!

Arbetsuppgift 12: Skapa klassen Basket

Steg 1: Skriv in kommandot: `sudo nano features/bootstrap/Basket.php`, klicka därefter på Enter.

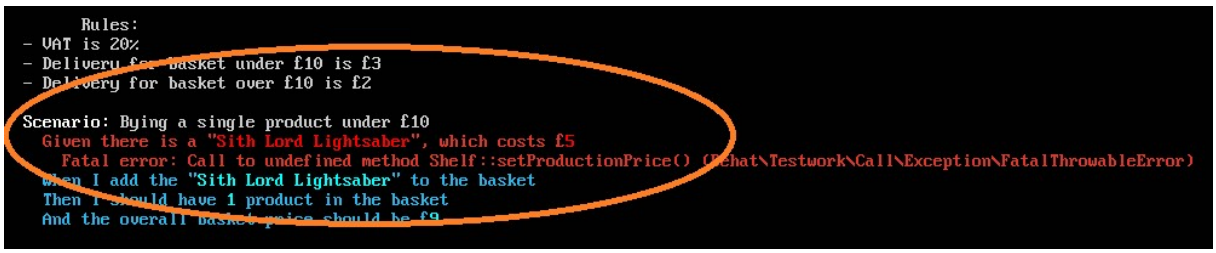
Steg 2: Skriv in enligt nedan:

```
<?php

final class Basket

{

}

?>
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 3: Skriv in kommandot: `sudo vendor/bin/behat`, klicka därefter på Enter.

Nytt bekymmer dyker upp ☹ `Call to undefined method Shelf::setProductPrice().`



Arbetsuppgift 13: Ändra klassen Shelf

Steg 1: Skriv in kommandot: `sudo nano features/bootstrap/Shelf.php`, klicka därefter på Enter.

Steg 2: Lägg till enligt nedan:

```
<?php

final class Shelf

{

    private $priceMap = array();

    public function setProductPrice($product,
$price)
    {
        $this->priceMap[$product] = $price;
    }

    public function getProductPrice($product)
    {
        return $this->priceMap[$product];
    }


}

?>
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

Steg 3: Ändra klassen Basket, skriv in kommandot `sudo nano features/bootstrap/Basket.php`, klicka därefter på Enter.

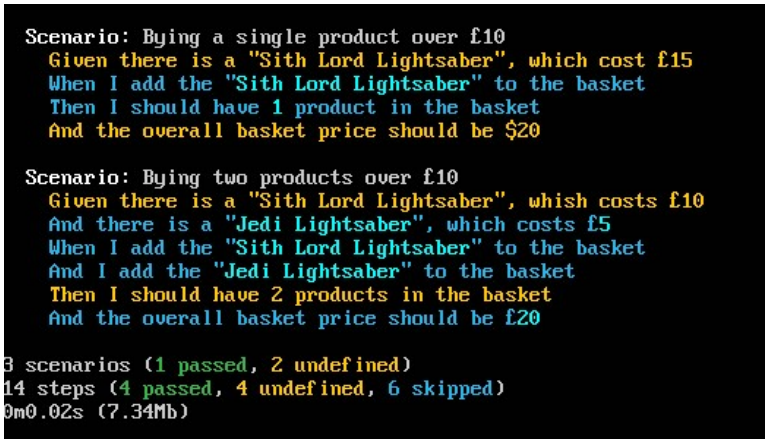
Steg 4: Lägg till enligt nedan:

```
<?php

final class Basket implements \Countable
```

```
{
    private $shelf;
    private $products;
    private $productsPrice = 0.0;

    public function __construct(Shelf $shelf)
    {
        $this->shelf = $shelf;
    }

    public function addProduct($product)
    {
        $this->products[] = $product;
        $this->productsPrice += $this->shelf-
>getProductPrice($product);
    }

    public function getTotalPrice()
    {
        return $this->productsPrice
            + ($this->productsPrice * 0.2)
            + ($this->productsPrice > 10 ? 2.0 :
3.0);
    }

    public function count()
    {
        return count($this->products);
    }
}

?>
```

Klicka på ctrl+o för att spara, bekräfta med Enter att du vill spara, klicka därefter på ctrl+x för att avsluta nano.

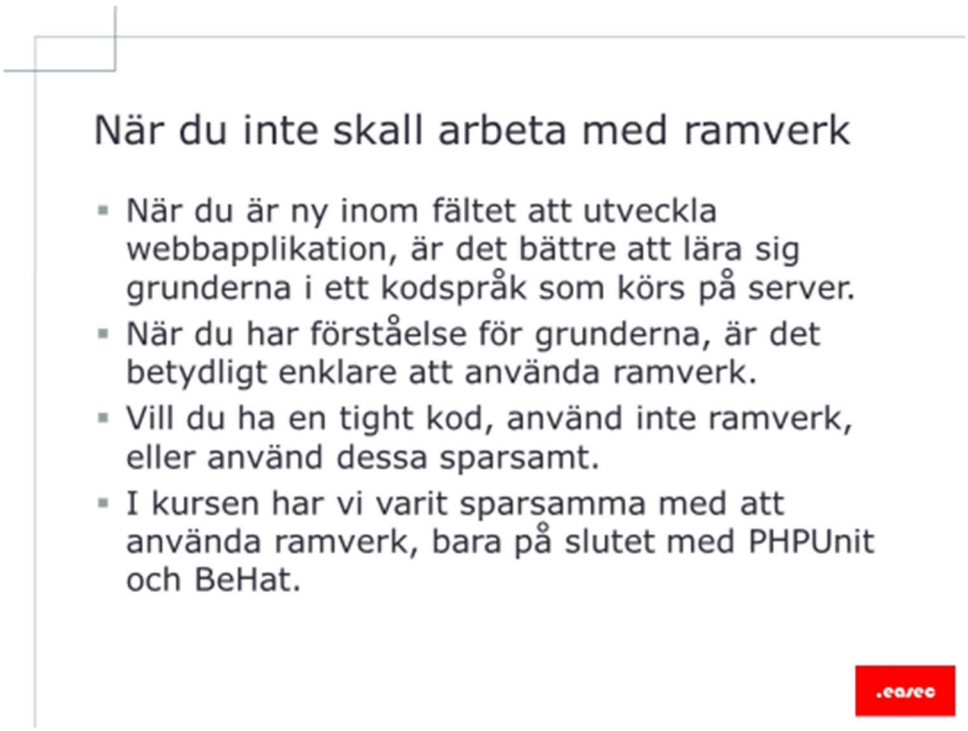
Steg 5: Skriv in kommandot: `sudo vendor/bin/behat`, klicka därefter på Enter.



Lektion 4: Ramverk

I denna lektion skall vi titta på:

- När du inte skall arbeta med ramverk.
- Storlek på programkod.
- När du skall arbeta med ramverk.
- Vanliga ramverk för PHP.



När du inte skall arbeta med ramverk

När du är ny inom fältet att utveckla webbapplikation, är det bättre att lära sig grunderna i ett kodspårk som körs på server.

När du har förståelse för grunderna, är det betydligt enklare att använda ramverk.

Vill du ha en tight kod, använd inte ramverk, eller använd dessa sparsamt.

I kursen har vi varit sparsamma med att använda ramverk, egentligen bara i denna modulen som vi har tittat på PHPUnit och BeHat. När vi har tittat på kommunikation med SQL Server och PDO, kan man säga att PDO är också ett ramverk, även om den ingår i PHP.

Storlek på ramverk

Precis som för .NET-utvecklarna kan man se att storlek på programkod ökar markant när ramverk används.

Samma utveckling inom PHP-utveckling, ju mer ramverk används desto större blir programkoden.

Sedan kan vi kanske försvara det lite genom att tänka att i dagens värld är uppgifterna lite mer komplexa, än när PHP startades.

En annan aspekt är vad som händer när vi arbetar med ramverk, har vi full kontroll på bakomliggande programkod?

När skall du arbeta med ramverk

När du har fått erfarenhet inom fältet, kan du börja titta på ramverk.

Du har fått förståelse för hur dessa ramverk fungerar.

Krav blir alltmer komplexa, krav går att tillgodose med ”egna” utvecklingar, med det är lättare att arbeta med ett ramverk.

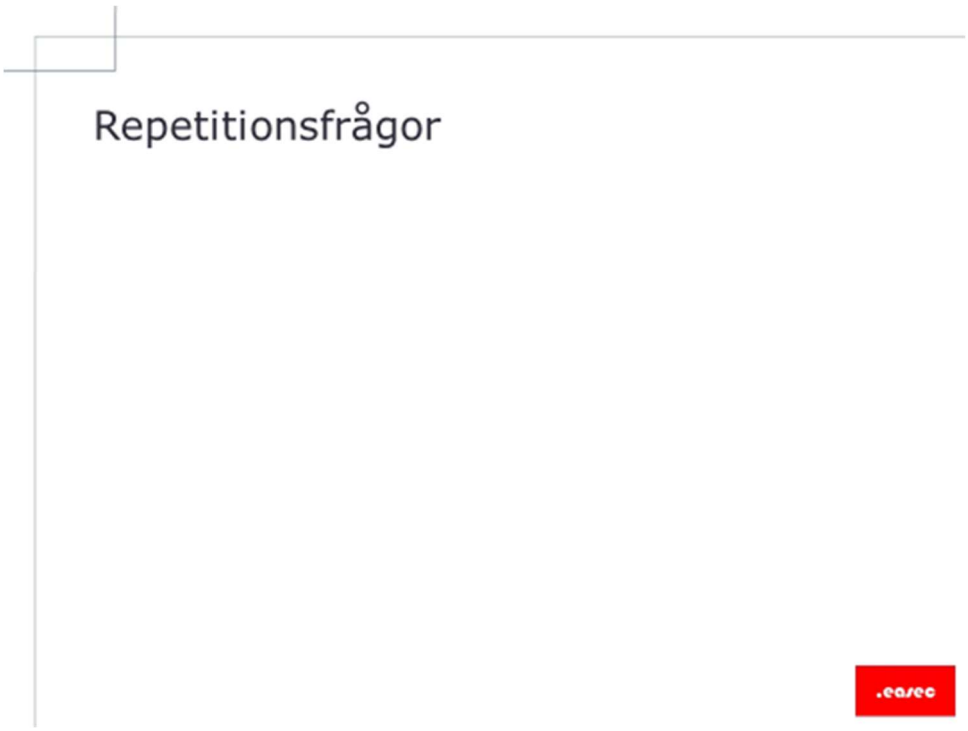
Det är också lättare att hantera eventuella förändringar eller uppdateringar, om ett ramverk används.

Vanliga ramverk för PHP

Det finns ett stort antal ramverk för PHP.

De vanligaste är:

- Zend Framework.
- Symfony.
- Cake PHP.
- Code Igniter.
- Laravel.
- Phalcon.



Repetitionsfrågor

1) Vilka projektmetodiker anses vara agila?

   _____
   _____
   _____

2) Vad är det största skillnaden mellan metoder som baseras på agile och vattenfallsmodellen?

   _____
   _____
   _____

3) Vad används user stories till?

_____
_____
_____

4) Du vill arbeta efter TDD-modellen, vad skall du börja med?

_____
_____
_____

5) Vad är ett unit test?

_____
_____
_____

6) I TDD har vi unit test, vad heter motsvarigheten i BDD?

_____
_____
_____

Appendix