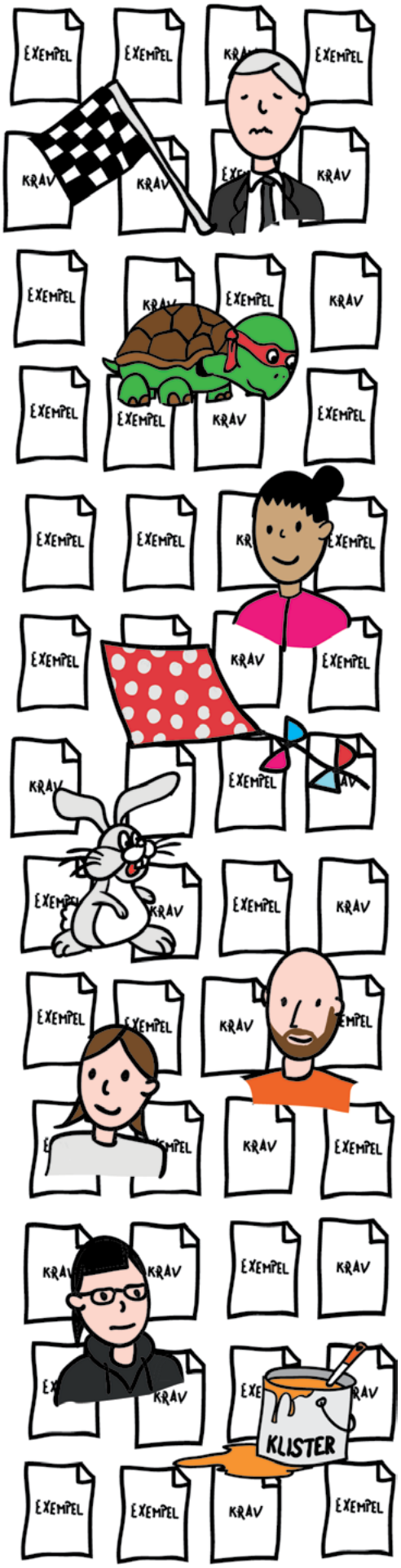
RÄTT IT-SYSTEM OCH FÅ BUGGAR MED HJÄLP AV

BDD OCH KONKRETA EXEMPEL

agero.



SAMMANFATTNING

Att bygga IT-system är förknippat med många utmaningar. Självklart ska vi hålla budget och tid, men vi ska bygga rätt lösning och göra det med att använda sig av bra kod (ej onödig). Kanske ännu viktigare är att hela teamet förstår och har samma bild av kraven. För ett projekt kan ha de senaste utvecklingsverktygen, jobba med en avancerad förändringsvänlig arkitektur och ha en fantastisk teamkänsla. Men om de som ska bygga systemet inte förstår kraven, kommer projektet att misslyckas. Vi kommer helt enkelt bygga fel lösning.

I det här dokumentet får du veta varför kravarbete med konkreta exempel (även kallat Specification by Example eller BDD) ökar möjligheten att bygga rätt system med få buggar. Du får veta hur du kommer igång och hur det påverkar utvecklingsprocessen. Och du får veta när det lönar sig och även när det inte lönar sig.

Dan Waltin
Titel: IT-arkitekt
dan.waltin@agero.se

Dan är en pedagogisk och pragmatisk problemlösare med genuin vilja att förstå verksamhetens behov. Dan är en driven konsult inom teknisk projektledning, arkitektur och systemutveckling.

agero.

INNEHÅLLSFÖRTECKNING

INTRODUKTION 4

1. NYTTAN MED KONKRETA EXEMPEL 9

2. HUR KOMMER MAN IGÅNG OCH HUR GÖR MAN? HUR PÅVERKAR DET MIN ORGANISATION? 7

3. NÄR SKA MAN BÖRJA. NÄR LÖNAR DET SIG? 11

agero.

INTRODUKTION

Inte nu igen! Jag som kände att detta IT-projekt gick bra, vi trodde ju att vi byggde rätt system. Vad hände?! Det blev dessutom onödigt stort med för många buggar. Kanske känner du igen dig, kanske har du suttit i ett IT-projekt som inte gått som ni tänkt er? Ni är i alla fall inte ensamma, om det är någon tröst. Idag slutar fortfarande alltför många IT/utvecklingsprojekt med att man byggt fel lösning på fel sätt. Vårt mål är att hjälpa så många som möjligt att ändra på det här. Hur bygger vi på tid/budget utan onödig kod och med få buggar? Men framförallt hur ser vi till att det är rätt lösning vi bygger?

Att investera i BDD/Specification by Example betyder att vi skapar förutsättningar för trygghet och förutsägbarhet i projektet/förvaltningen. Det innebär att vi ser till att de som ska bygga systemet förstår kraven, vilket borgar för ett lyckat projekt.

Med detta dokument tar vi oss an utmaningarna ovan genom att diskutera hur specificering av krav med konkreta exempel kan hjälpa dig att lyckas i ett IT-projekt. Först kommer en inledning om konkreta exempel. Därefter diskuteras fördelarna, var i utvecklingsprocessen exemplen kommer in och hur man gör, och slutligen när det lönar sig och när man ska sätta igång.

Konkreta exempel

När någon frågar dig ”hur menar du nu?”, så är det naturliga svaret ”jo, om du till exempel tänker att…”. Det är med konkreta exempel vi kommunicerar i vardagen, därför att det är med konkreta exempel som vi förstår världen.

Ett exempel på det är följande. På Skatteverkets hemsida finns information om regler för avdragsgilla förbättringsutgifter vid försäljning av bostadsrätt. En av dessa regler är detta:

”Reparation och underhåll som återställer till nyskick är avdragsgilla, om det är gjort senast fem år innan försäljning, och om underhållet under ett år överstigit 5000 kronor. Genom brukande och ålder konsumerar du det förbättrade skicket, och du gör själv en bedömning hur mycket”.

Vad betyder det? Vi tar ett par konkreta exempel, också de från Skatteverkets hemsida, med konkreta värden instoppade:

Säg att en bostadsrätt såldes 2014. När den köptes 1998 fanns en tvättmaskin i lägenheten.

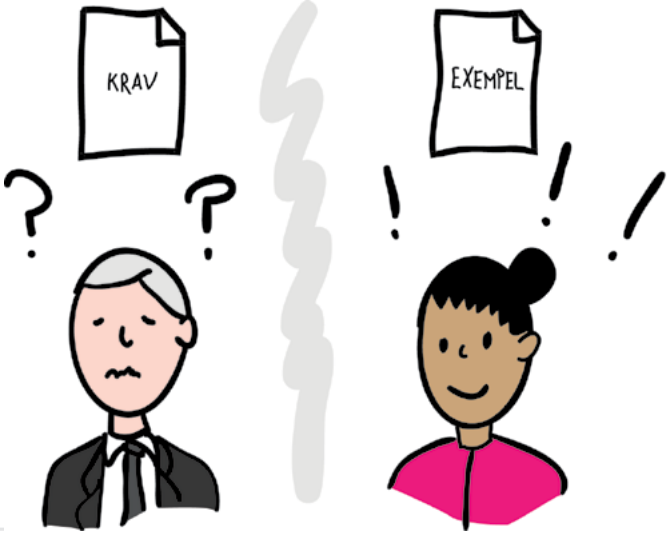
EXEMPEL 1

2008 byttes tvättmaskinen för 5900 kronor. Det ger inget avdrag eftersom det är före 2009, även om kostnaden är mer än 5000 kr.

EXEMPEL 2

2012 tapetserades vardagsrum och kök för 17000 kronor Det är gjort 2 år innan försäljning, men slitage gör att avdraget måste minskas. Säljarna gör själva bedömningen att 13000 är avdragsgillt.

Vad var lättast att läsa och att förstå? Regeln eller exemplen?



agero.

Viskleken

I många utvecklingsprojekt och -metoder sker kommunikation med hjälp av dokument. Kravanalytiker pratar med verksamheten och skriver kravdokument. Testavdelningen läser kravdokumenten och skriver testspecifikationer. Arkitekter läser kravdokumenten och skriver arkitekturdokument. Utvecklarna läser (i bästa fall!) alla dokument och utifrån sin tolkning bygger de ett system, som i många fall tyvärr inte är rätt. Anledningen är att massor av förståelse försvinner i dokumenten. Det blir "viskleken" över det hela.

För att hålla alla dessa dokument uppdaterade och i synk med koden krävs mycket arbete! Sannolikheten är stor att inom kort tid beskriver kravdokumenten något annat än det systemet gör.

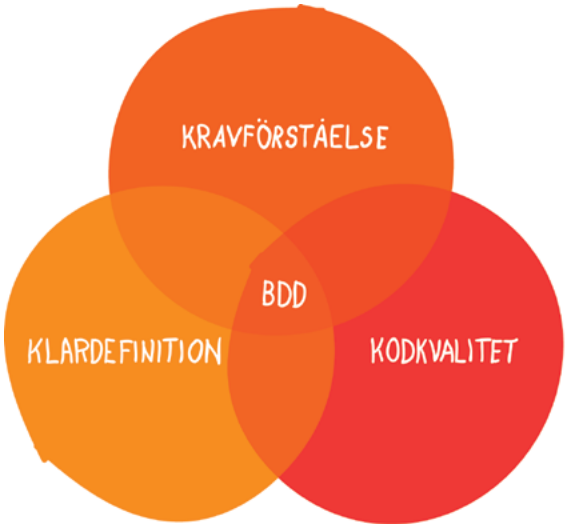
Det finns många processer som försöker minimera förlusterna i varje övergång. Men grundproblemet är själva viskleken, i förlusterna av förståelse i varje dokument. Om den går att eliminera är oerhört mycket vunnet.

Exempel som krav på IT-system

Tänk om man kunde diskutera och dokumentera krav på IT-system på samma sätt, med konkreta exempel? Och tänk om man kunde minimera viskleken och dess problem? Vad skulle det få för konsekvenser?
Det går faktiskt att göra, och några positiva konsekvenser när man gör så är dessa:

- ✓ Ökad kravförståelse, vilket leder till att rätt lösning byggs
- ✓ Tydlig klardefinition, vilket leder till att onödig kod undviks
- ✓ Förbättrad kodkvalitet, vilket leder till att utvecklingen kan hålla ett bra tempo, med få buggar

Detta white paper kommer att diskutera hur konkreta kravexempel hjälper till att driva utvecklingen av ett IT-system. Varför det är bra, när och hur i utvecklingsprocessen detta kommer in, och när det är läge att föra in det i en organisation.



1. NYTTAN MED KONKRETA EXEMPEL

Ökad kravförståelse

Det här är den allra viktigaste punkten! Ett projekt kan ha de senaste utvecklingsverktygen, jobba med en avancerad förändringsvänlig arkitektur och ha en fantastisk teamkänsla. Men om de som ska bygga systemet inte förstår kraven, kommer projektet att misslyckas.

Genom att ha workshops där personer med kompetens inom verksamhet, krav, användbarhet/UX, test, och utveckling diskuterar kraven, med konkreta exempel, så kommer den gemensamma förståelsen att öka. Genom att även utvecklingsteamet är med och tar fram kraven, minimeras "viskleken". Dessutom får utvecklarna en djupare förståelse för verksamheten när de sitter med på workshoparna, och hör hur verksamheten resonerar.

När kravförståelsen ökar, minimeras risken att utvecklingsteamet lägger tid och pengar på att bygga fel saker.

I ett projekt när vi använde detta hade verksamheten kommit med ett önskemål om en ny påminnelsefunktion. När vi satte oss och tog fram konkreta exempel insåg verksamheten att funktionen egentligen inte behövdes!

Utan workshopen hade utvecklingsgruppen lagt tid och pengar på att bygga en funktion som löste fel problem.

Tydlig klardefintion

De konkreta exemplen används som klardefinition, eller acceptanskriterier. Detta ger ett bra fokus för att driva utvecklingen, och man får ett tydligt kvitto på när man är klar. När alla exempel fungerar, då är systemet klart !
Dessutom vet[1]man vad som är klart! Om någon frågar vad som är klart, är det bara att visa exemplen, för det är det som är klart!

Med en bra klardefinition minimeras risken att man skriver för mycket kod. Man skriver bara den kod som behövs och undviker "bra att ha"-kod.

Förbättrad kodkvalitet

Kravförståelsen är som sagt det allra viktigaste, men ytterligare en stor fördel är att det går att använda de konkreta exemplen för att verifiera systemet automatiskt. Det finns olika verktyg (t.ex. SpecFlow om man jobbar i .Net-världen eller Fitnesse i Java-världen), som med utgångspunkt från exemplen automatiskt anropar systemets funktioner och verifierar att systemet uppfyller exemplen.

Det innebär att så fort en funktion är klar, kommer den fortsätta att vara klar! Den kommer aldrig att gå sönder, inte utan att det märks i de automatiska testerna. Det innebär att det är möjligt att hela tiden "refaktorisera" koden, dvs hålla den i ett bra och förvaltningsbart skick.

Med hjälp av de automatiska testerna och att koden hela tiden håller en bra kvalitet, är det möjligt att hålla ett stadigt och högt tempo i utvecklingen över lång tid med låg risk att införa buggar.

---

[1] Det finns förstås typer av krav som hanteras bättre med andra metoder än Specification by Example. Dit hör till exempel krav på användbarhet och prestandakrav.

2. HUR KOMMER MAN IGÅNG OCH HUR GÖR MAN? HUR PÅVERKAR DET MIN ORGANISATION?

Men hur gör man då, rent konkret? Och hur påverkar detta utvecklingsprocessen? Det blir lite olika ansatser om man skriver ett nytt system från grunden, eller om man förändrar ett existerande system. Vanligast är antagligen det senare, så det är det fallet som kommer att behandlas här.

Om vi antar att det finns ett existerande system som används och förvaltas. Vi antar också att utvecklingen bedrivs enligt någon form av agil process, att man har en prioritering av ärenden (antagligen visualiserad på en scrum- eller kanbantavla).

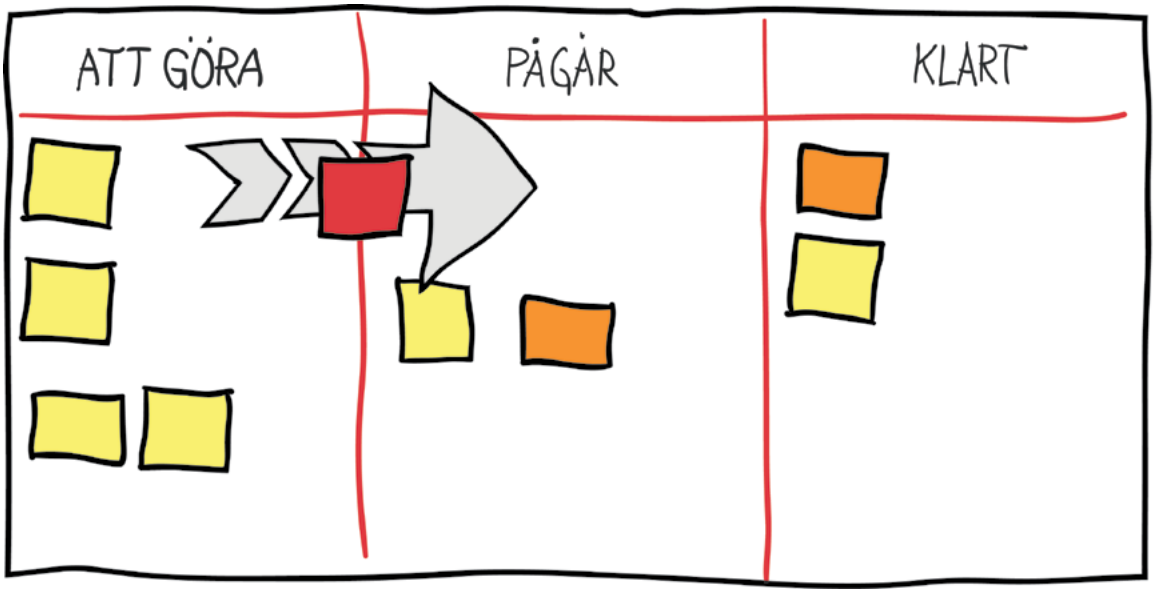
Ärenden kommer in och prioriteras, det kan vara buggar som ska rättas, en existerande funktion som ska ändras, eller ny funktion som ska läggas till.
För att hänga upp diskussionen på något konkret kommer vi utgå från följande, väldigt enkla, case:

Ett system där användarna kan visa innehållet på sitt eget språk. Det finns också en administrativ del där språk hanteras och språkfiler kan läsas in.

Nu kommer önskemål från verksamheten om en ny funktion:

- Man ska kunna att skilja på icke publicerade och publicerade språk.
- En användare ska ha möjlighet att välja bland de aktiva språken, som ska visas i bokstavsordning.
- En administratör skall kunna publicera språk

Detta ärende har nu kommit högt upp på prioriteringen och det är dags att börja jobba med det. Det är nu det är dags att börja ta fram konkreta exempel för att specificera kraven och dokumentera kravförståelsen.



agero.

Workshop för gemensam förståelse

Börja med att kalla till en workshop för att diskutera funktionen *Publicera språk.* (I praktiken kan det vara lämpligt att ha mer än ett ärende på workshopen, t.ex. beroende på hur lätt det är att få tag på verksamheten.) Som med alla möten och workshops är det viktigt att det finns en person som driver mötet, håller fokus, har koll på tiden osv.

Syftet med workshopen är att få en **gemensam** kravförståelse. Både verksamhet och IT ska förstå kraven på samma sätt.

Målet med workshopen är ett eller flera konkreta exempel som illustrerar de nya kraven. Det ska vara tillräckligt många exempel för att de ska kunna fungera som acceptanskriterier, dvs. när alla exempel är implementerade är funktionen klar.

Deltagare är så många olika relevanta roller som möjligt: verksamhet, test, krav, användbarhet/UX, utvecklare.

På workshopen skriver man på whiteboard, skissar och använder postit-lappar. Var noga med att använda konkreta exempel! Istället för att prata om att språken ska presenteras i bokstavsordning, rita upp det på tavlan med konkreta språk! Det kanske framgår att verksamheten vill ha språken i bokstavsordning, men att svenska och engelska alltid ska vara överst (kanske därför att användbarhetsarbetet har visat att de flesta användarna har svenska eller engelska som modersmål)!

Eller så kanske deltagarna när de ser de konkreta exemplen kommer på att det behövs möjlighet för administratören att specificera vilka språk som ska särbehandlas och hamna högst i listan. Det kanske tas med nu, eller skjuts till ett senare ärende. Det är en av styrkorna med att använda just konkreta exempel, att det gör att deltagarna kommer på nya varianter, nya frågor, gränsvärden mm.

> Förbereda några exempel innan för att få igång diskussionen, men var beredd på att de kan ändras

Var noga med att använda rätt begrepp. Heter det publicera eller heter det aktivera? Ärendet i exemplet ovan innehöll båda orden! Var konsekvent.

Var noga med att använda realistiska exempel och inte påhittade testexempel. Har ni en workshop om verksamhetsroller, använd riktiga roller som verksamheten använder (Controller, CFO, Developer...) och inte påhittade roller (Roll 1, Fotbollsdomare, Exempel 12...).

Lite då och då under workshopen är det lämpligt för workshopledaren att stämma av att alla har samma kravförståelse. Ett bra sätt att göra det är att fråga gruppen om ett par konkreta exempel som ännu inte skrivits upp på whiteboarden, och se hur deltagarna svarar.

En fråga här skulle kunna vara: Vad händer när man laddar in en språkfil för ett nytt språk, ska det språket vara publicerat eller inte?

Svarar alla likadant har man antagligen en gemensam förståelse, annars behöver man utreda varför det blev olika svar.

agero.

Dokumentera med Givet, När, Så

När workshopen är klar dokumenteras resultatet, de konkreta exemplen. Dokumentationen mejlas till de som var med på workshopen för att se om de känner igen exemplen och fortfarande håller med. När man fått acceptans på dokumentationen är det dags att börja implementera funktionen.

Skulle man stanna här, vid den gemensamma kravförståelsen, och sen börja bygga funktionen, så har man vunnit oerhört mycket. Men det går att komma ännu längre!

Istället för att gömma dokumentationen i ett word-dokument på en filarea eller i Sharepoint där ingen kommer att läsa den, kan man utnyttja de konkreta exemplen för att *automatiskt* validera att systemet gör rätt!

Ett bra sätt att dokumentera exemplen är med en "Givet, När, Så"-syntax.

```
Givet en viss kontext
När något händer
Så ska en viss sak hända
```

Följande två scenarios (exempel) skulle kunna vara dokumentationen av workshopen kring publicering av språk:

EXEMPEL 1

```
Scenario: När en administratör publicerar
ett språk så blir det valbart

Givet följande språk
    | Språk   | Publicerat |
    | Danish  | Ja         |
    | English | Nej        |
    | Italian | Nej        |
    | Chinese | Ja         |
    | Swedish | Nej        |

När administratören publicerar Italienska
Och användaren vill byta språk

Så ska följande språk vara möjliga att
välja
    | Språk   |
    | Danish  |
    | Italian |
    | Chinese |
```

EXEMPEL 2

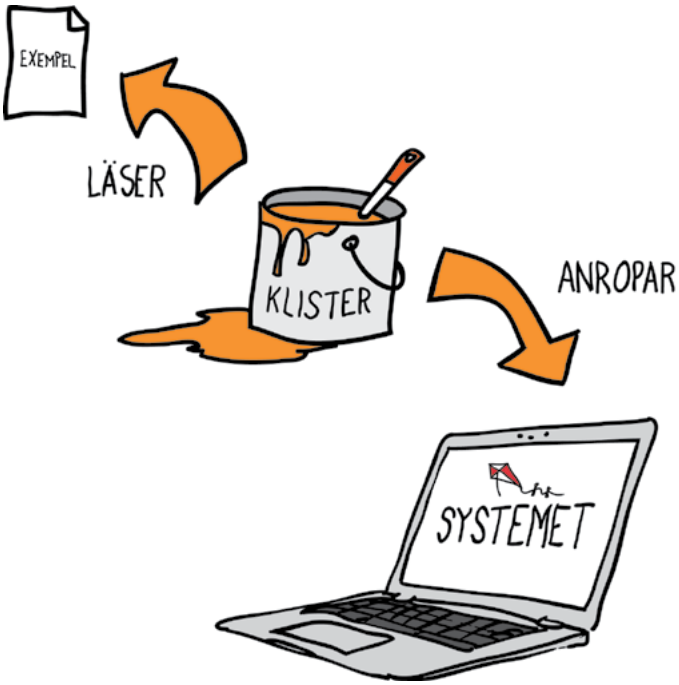
```
Scenario: Språk ska visas i bokstavsordn-
ing, men engelska och svenska ska visas
först, när en användare vill byta språk

Givet följande språk
    | Språk   | Publicerat |
    | Danish  | Ja         |
    | English | Ja         |
    | Italian | Ja         |
    | Chinese | Ja         |
    | Swedish | Ja         |

När användaren vill byta språk

Så ska följande språk vara möjliga att
välja
    | Språk   |
    | English |
    | Swedish |
    | Danish  |
    | Italian |
    | Chinese |
```

> Var två personer som dokumenterar exemplen. För att exemplen ska bli lättlästa

> Skriv exemplen på koncernspråket, så att rätt begrepp används!

Skapa tester utifrån exemplen

Nu när vi har fått en gemensam förståelse av behoven och dokumenterat kraven med konkreta exempel, så kan vi gå vidare och skapa en automatisk verifiering av systemet. Det går att skriva hur mycket som helst om det, men här kommer vi bara skrapa lite på ytan.



Det man gör är att skriva en sorts “klisterkod”, som läser exempelfilerna, rad för rad, och anropar funktioner i systemet.

Givet-steget är ett sätt att beskriva det testdata som behövs. När klisterkoden läser Givet-raderna så ser den till att skapa detta testdata, kanske genom att anropa funktioner i systemet, eller kanske genom att skriva direkt till en databas.

När-steget är den faktiska testen. Här gör klisterkoden anropet till den funktion i systemet som skall verifieras.

Så-steget är verifiering av att det förväntade resultatet uppnåddes. Här kanske klisterkoden tittar på att databasen har uppdaterats korrekt, eller att rätt saker visas i gränssnittet.

Klisterkoden ska ses som en del av programkoden, precis som enhetstester. Den är lika viktig som produktionskoden. För system där den här sortens tester används kommer en ganska stor andel av koden vara klisterkod. Och en ganska stor del av tiden kommer att ägnas åt att skriva och underhålla exempel och klisterkod. Det kan vid en första anblick verka onödigt och dyrt, men är i själva verket ett nödvändigt grundarbete för att skriva bra system. Jämför med att kakla om ett badrum. Det är ju kaklet man är ute efter, men underarbetet med att spackla och att sätta upp fuktspärr är helt nödvändigt för ett lyckat resultat.

Den automatiska verifieringen av exemplen kopplas sedan in i den automatiska byggprocessen, så att man vid varje förändring av koden får ett kvitto på om allt fortfarande fungerar.

3. NÄR SKA MAN BÖRJA. NÄR LÖNAR DET SIG?

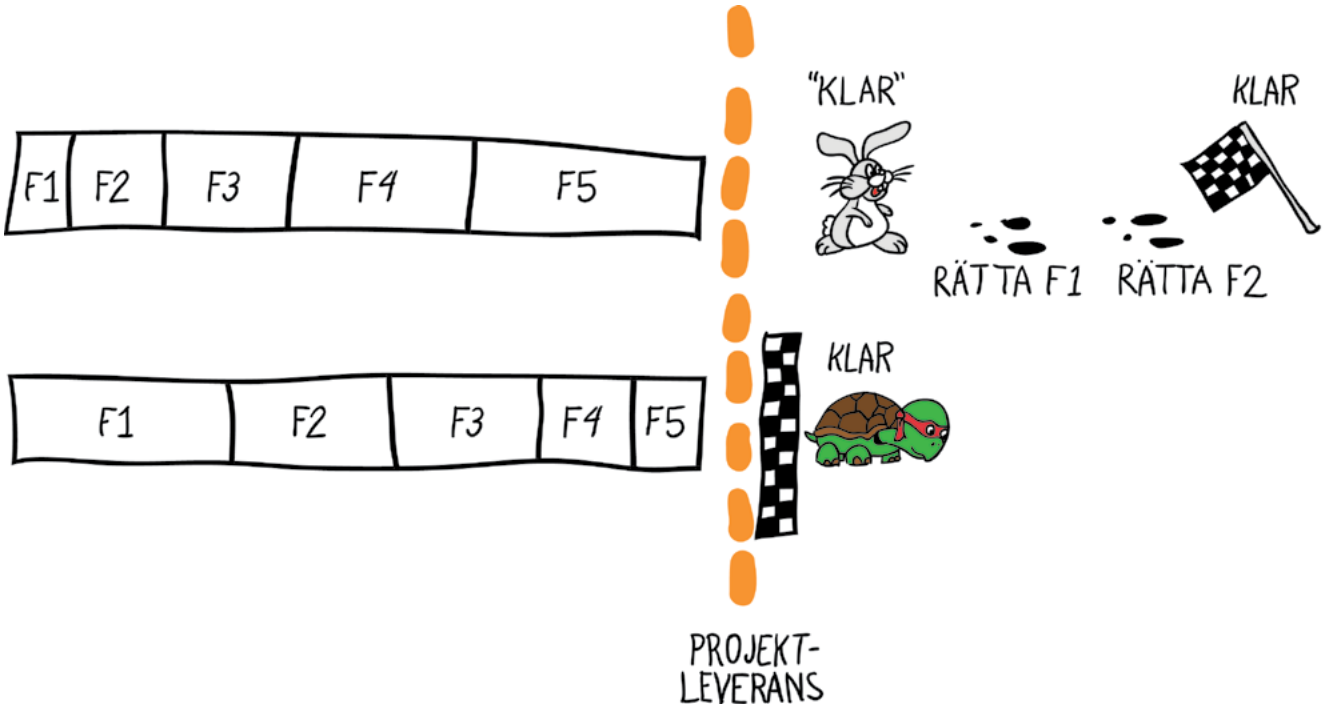
Nu vet vi lite mer om *varför* och *hur* konkreta exempel används i utvecklingsprojekt. Men när ska man börja? När är rätt läge att ändra sitt sätt att jobba? Det korta svaret är att rätt tid att börja är *nu*! Eller lite mer nyanserat, nästa gång något ska förändras i systemet. Det kan vara en ny funktion som ska införas, en existerande funktion som ska förändras, eller en bugg som ska rättas.

Långsiktiga eller kortsiktiga lösningar

När du har ett system som förvaltas över tid är arbete med exempelbaserade krav och automatisk verifiering en investering i trygghet och förutsägbarhet. De första funktionerna man använder det på kommer att ta längre tid än man är van vid. Det kan kännas läskigt, och det kan därför kännas frestande att lösa den där buggen "lite snabbt", men undvik den frestelsen!

Det kommer alltid att finnas de som säger att det inte finns tid: För just det här ärendet finns inte tid – det är bättre att lite snabbt fixa en lösning; I just den här sprinten finns inte tid – det är bättre att trycka in ett par funktioner till; I just det här projektet finns inte tid – det kostar för mycket att lära nytt...

Undvik frestelsen att acceptera sådana kortsiktiga tankar. Varför? Därför att efter ett antal sådana ärenden eller iterationer eller projekt, så har man passerat gränsen för när investeringen hade betalat sig! Därefter blir den ackumulerade kostnaden av de "billiga" lösningarna högre!



agero.

När kommer investeringen att betala sig?

Men när kommer den här investeringen i trygghet och förutsägbarhet betala sig? Som vanligt är svaret “det beror på”.

Det finns två delar. Den ena delen är att jobba med kravförståelsen med hjälp av konkreta exempel tillsammans med verksamheten. Den andra delen är att automatisera verifieringen av exemplen. Det är den första delen som är den allra viktigaste, det är den som säkerställer att rätt system byggs. Den andra delen säkerställer att systemet byggs på rätt sätt med hög kvalitet och få buggar.

Ska man bara välja en av de två delarna så är det den första som ger mest tillbaka på investerad tid. Men full effekt får man först när även den andra delen kopplas in.

Det finns många faktorer som påverkar hur lång tid det tar:

- Finns förståelse i verksamheten att den behöver avsätta tid och resurser för workshoparna?
- Finns det stöd från projektledning och chefer att pröva detta? Detta är helt nödvändigt, speciellt i början när ärenden tar längre tid än man är van vid.
- Finns det vilja i utvecklingsteamet att förändra sitt arbetssätt? Att börja arbeta med konkreta exempel, och automatiserad verifiering av dessa, kommer innebära en förändring av arbetssätt. Fokus kommer skifta från teknik till krav. Det är en bra sak, men förändringsarbete är alltid en utmaning.
- Har någon eller några i utvecklingsteamet erfarenhet av automatiserade tester, t.ex. enhetstester? Om inte kan en utbildningsinsats och/eller coachning behövas.
- Finns en teknisk infrastruktur på plats för automatiserade byggen? Det är viktigt att verifiering sker automatiskt vid varje förändring.
- Är koden “testbar”? I de flesta fall kommer viss förändring av koden behövas för att det ska vara möjligt att verifiera exemplen automatiskt. Beroende på kvaliteten i koden sedan tidigare tar detta olika lång tid.

Om du bygger ett nytt system från grunden och det systemet ska förvaltas, så kommer den här investeringen *definitivt* att löna sig.

Om du har ett existerande stabilt system där det endast läggs några timmar i månaden på förvaltning, kommer en sådan investering knappast att löna sig.

Om du i ett existerande system ska göra en större förändring som kan förväntas ta fyra till sex man-månader, så är det rimligt att förvänta sig att investeringen kommer att betala sig.

Om du har ett existerande system i aktiv förvaltning, där en heltidsresurs eller mer jobbar med systemet och lägger till nya funktioner, så är det rimligt att förvänta sig att investeringen kommer att betala sig efter fyra till sex månader.

Avslutning

Vi människor kommunicerar och förstår varandra genom konkreta exempel. Att använda konkreta exempel i IT-projekt ger flera fördelar, som *bättre kravförståelse* och *färre buggar.* Exemplen tas fram gemensamt av verksamhet och it, inklusive utvecklare, i workshops. De dokumenteras i ett "givet, när, så"-format, vilket kan användas för automatisk verifiering av exemplen. Arbetet med konkreta exempel är ett komplement till grundläggande kravanalys och användbarhetsarbete, och görs iterativt under utvecklingen. Att skapa den automatiska verifieringen har en viss initialkostnad, men görs det på rätt sätt lönar det sig i det långa loppet.

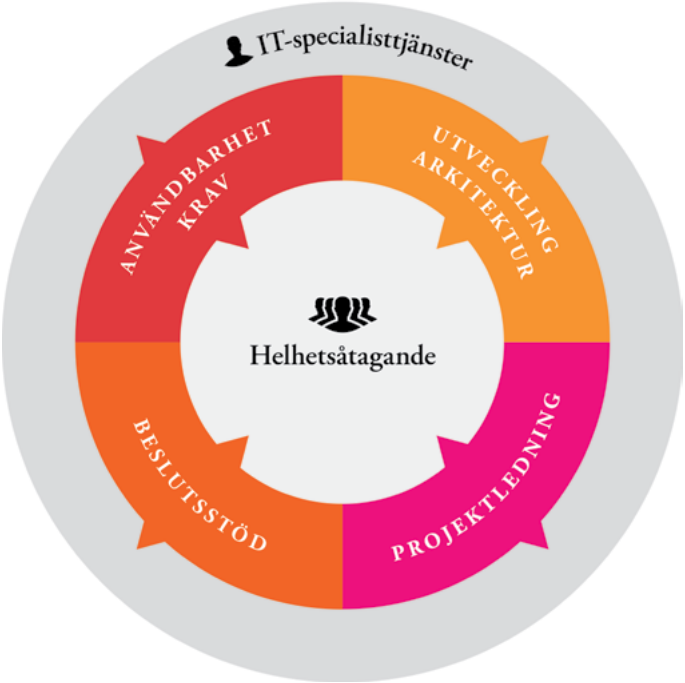
För ett nytt system eller för ett system som ska förvaltas mer än några månader finns det ingen anledning att vänta. Bästa tiden för att bygga rätt system på rätt sätt är nu!

Vidare läsning

- http://specificationbyexample.com/
- https://en.wikipedia.org/wiki/Specification_by_example

VILKA ÄR AGERO?

Agero är ett IT-tjänsteföretag som hjälper företag och organisationer att göra det som de är bäst på. Den effekten når vi genom att utveckla och förvalta digitala processer med agila arbetssätt. Vi arbetar med helhetsåtaganden där vi skapar skräddarsydda system som stödjer kundens specifika verksamhetsprocesser. På så sätt får våra kunder konkurrensfördelar och kan fokusera på sin kärnverksamhet. Våra kompetenser återfinns inom våra fyra affärsområden som bilden illustrerar.



Jag vill veta mer om Agero

agero.