**Kvalitetssikring**I modsætning til plandrevne projekter, er der i den agile ikke nogen fastlagte processer ifh. til kvalitetssikring. Det vil sige, at ligesom med risikostyring, arkitektur, og det meste andet, så er det op til det enkelte udviklingsteam at fastlægge sine egne procedurer.

Scrum kommer selv med et par indbyggede muligheder for, at kvalitetssikre sit produkt. Hver user story kommer med en, af product owner defineret, accepttest. Acceptestens krav, samt informationerne på user story´en danner udgangspunkt for, at skrive de UNIT tests der udgør Test First-praktikkens indhold. Derudover er der review indbygget i hvert sprint, så man ofte får rundet produktets kvalitet med product owner, som er den yderste autoritet på området.

I forhold til at kvalitetssikre koden kan man med fordel skæve til de 12 XP praktikker (og 4 værdier), da de alle, i større eller mindre grad støtter op om at sikre en god kodekvalitet. I den forbindelse kan det godt betale sig at være opmærksom på, at såfremt man vælger en eller flere af disse praktikker fra, så opstår der huller i det ”sikkerhedsnet” som alle praktikkerne udgør. Hvis et team f.eks. fravælger parprogrammering, så bør man sikre sig, at man istedet indfører et kodereview, da man ellers er åben overfor fejl.   
  
Samlet set bør man dog være bedre rustet til at opnå et godt kvalitetsniveau, hvis man anvender en Scrum + XP drevet metode fremfor f.eks. UP. Det er naturligvis noget af en påstand, og det forudsætter da også, at man overholder de praktikker der er fastsat (eller har et solidt sikkerhedsnet hvis man vælger nogen af dem fra). Men hvis man gør det, så sikrer man ved hver eneste user story, at såvel de formelle, som de uformelle kvalitetskrav er opfyldt.

[Her kommer to billeder til at illustrere forskellen, men først søndag d. 5/1]

**Risiko og Risikostyring**En ”Risiko” definerés som: ”En mulig negativ planafvigelse”. Altså, en hændelse der kan bringe vores tidsplan, rygte, og i sidste ende, vores job i farezonen.  
For bedst muligt, at kunne ruste teamet imod konsekvenserne af en opfyldt risiko, er det vigtigt, at kunne komme disse risici i forkøbet.   
Risikostyring/analyse dækker over den valgte metode til at identificere, og håndtere, de problemer der eventuelt måtte opstå i forbindelse med afviklingen af et projekt. Selv i mindre projekter er der er en lang række ting der kan gå skævt, og derfor er det også vigtigt, at kunne prioritere de hændelser man forudser.

**Risikostyring i Scrum**  
I et klassisk plandrevet scenarie udfører man en grundig risikoanalyse i starten af et projekt, eventuelt med tilhørende prototyper etc. Det kan man naturligvis også gøre i et Scrum-projekt, og i så fald foregår det i det såkaldte ”Sprint 0”. En alternativ måde at tilgå risikostyringen på er ved, at tage det på et sprint til sprint basis. Hvor man for hver afsat timebox i projektet vurderer hvilke risici der lurer i horisonten, og så håndtere dem som en naturlig del af det kommende sprint.   
En tredje måde at håndtere risiko i et projekt, er at vælge et trin imellem disse to yderligheder. Det hovedsaglige, at man tilpasser sig til situationen.

<http://www.scrumalliance.org/community/articles/2013/2013-may/risk-management-in-agile>

**Vores Perspektiv**

For effektivt at kunne håndtere risiko i vores projekt, har vi i sprint 0 udført en relativt omfattende risikoanalyse, hvor fokus har været på at finde svarene på de følgende spørgsmål:  
\* Hvad er den mulige risiko?  
\* Hvad er den vurderede sandsynlighed for at det hænder?  
\* Hvad er konsekvensen hvis det sker?  
\* Hvordan kan vi håndtere/undgå risikoen?

For at kunne opnå det, har vi kigget på de krav som product owner har ridset op for os, og de teknologier som vi har udledt af de krav, og formuleret en risikoliste:

1. REST
2. Entity Framework
3. Databasesikkerhed
   1. Hashing af databaser
   2. Personligdatasikkerhed
4. Brugervenlighed
5. Test-First Development
6. ASP.NET som teknologi
7. UI design af websiden
8. HTML/CSS
9. Bootstrap

Figur - Udsnit af risikolisten. Se bilag XXX

Ud fra denne, har vi så lavet en tabel med en konkret beskrivelse af den risiko vi forudser, hvor sandsynligt det er for at ske, hvilken effekt (relativt til de andre risici) det har for projektet, samt en relativ skala for hvor højt risikoen skal prioriteres.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Beskrivelse | Sandsynlighed | Effekt | Ranking |
| 1 | Problemer med kodning og opsætning af REST service | 9 | 10 | 95 |
| 2 | Der kan opstå problemer med at bruge Entity frameworket, pga. manglende erfaring | 5 | 9 | 75 |
| 3 | Manglende erfaring med at implementere sikkerhed i forbindelse med brugerdata | 9 | 3 | 35 |
| ... | ... | ... | ... | ... |

Figur - Udsnit af risikomodel - Se bilag XXX

Vi forventer at størstedelen af vores risikomomenter vil opstå som følge af manglende erfaring med de teknologier vi skal arbejde med. For at kunne stå imod når, og hvis, disse problemer opstår har vi opstillet denne tabel, hvor hver risiko er prioriteret efter dens ranking, med en løsningsmodel.  
Som det fremgår har vi til hensigt, at prototype os ud af så meget som vi kan i sprint 0.

|  |  |  |
| --- | --- | --- |
| Risiko Beskrivelse | Ranking | Mitigation |
| 1. Problemer med Kodning og opsætning af Webservice | 95 | Kode simpel hello world prototype af REST service |
| 2. Der kan opstå problemer med at bruge Entity frameworket, pga. manglende erfaring | 75 | Kode simpel prototype med oprettelse og opdatering af database med code-First |
| 6. ASP.NET er en model der ikke er  kendt af udviklergruppen | 70 | Kode simpel hello world prototype  i ASP.NET |
| ... | ... | ... |

Figur - Udsnit af mitigation tabel - Se bilag XXX

**Arkitektur**  
Scrum foreskriver, at man laver lige præcist så meget design forarbejde som er påkrævet. Hvad et udviklingshold så føler der er ”påkrævet” vil givetvis være helt forskelligt fra virksomhed til virksomhed. Det er dog forbundet med en række fordele, at man overvejer en grundlæggende struktur fra starten af et projekt. Som i et traditionelt plandrevet projekt, kan man med fordel tale med kunden. Det produkt som denne ønsker, bør lægge til grund for den grundlæggende arkitektur.   
Der er flere praktiske grunde til, at det kan være en fordel at have en basal arkitektur fra starten. Det kan hjælpe udviklerne til, at estimere projekts størrelse, både i tid og penge. Typisk er der nogle interessenter, være det så banken, product owner, eller virksomhedens ledelsesechelon, der ikke har kapacitet til, at forstå de udfordringer et projekt består af. Hvis man har et tentativt design, så har man også bedre muligheden for at kommunikere disse udfordringer igennem fra start. Derudover kan det også vise sig, at være et nyttigt holdepunkt senere i projektet hvis der opstår tvivl om retningen.

**Vores perspektiv**

I vores tilfælde ønsker product owner et produkt der kan holde dynamisk og persistent da, og som kan tilgåes via en hjemmeside. På længere sigt skal systemet kunne udvides til, at omfatte apps til android, iOs og Windows Phone. Ud fra de kriterier kan vi så som udviklere skimte omridset af en arkitektur, der har en database, og et interface som kan tilgåes fra disse platforme.   
Almindelig sund fornuft siger os, at vi bør sigte efter en lagdelt (omend, det ikke fra starten er præcist besluttet hvor mange lag vi taler om) arkitektur.   
For product owner er udviklingstiden en væsentlig faktor, derfor anser vi det som fornuftigt, at lægge vægt på en slank klient, og en fed backend. Det er ikke en beslutning uden ulemper, men det betyder, at jo mere logik der ligger i backend, jo mindre skal skrives igen når nye klienter skal udvikles.

Med disse overvejelser i hånden, og formaningen fra XP paradigmet om hyppig refaktorering, har vi den mængde prædefinerede arkitektur fra start som vi har til hensigt at have.