**Definér kodestandarder**

**Solution**Der bliver oprettet en solution, kaldet "LevelUp". Denne solution indeholder de følgende projekter:  
DataAccess,   
Controller  
RESTService  
Client  
UnitTest

De fire første sikrer en fornuftig lagdeling af arkitekturen, med DataAccess som modellag, Controller, som controllerlag, RESTService som servicelag og Client som klientlag. Da der er tale om en ASP.Net klient, og da denne har indbygget et specifik arkitektur, benytter vi MS standard. Det sidste projekt benytter vi i forbindelse med Test First-udviklingsprocessen, og alle metoder til test opbevares heri.

**Klasser**Alle klasser skrives som UpperCamelCase. I modellaget er der blot et sigende navn, uden specifik endelse. Kontrollaget slutter alle klasser med "Controller", mens alle interfaces i servicelaget starter med stort "I" efterfulgt af modelnavnet. [OI! Ikke log!! fix it]  
I klienten har hver side navn efter dens primære funktion, og derfor må den godt adskille sig fra modelnavnene. Så længe det er klart hvilken primær model den benytter.  
f.eks. CreateUser for siden til at oprette bruger.  
I App\_Code benyttes endelsen "Calls" på de klasser der har til ansvar at hidkalde servicen. Mens hjælpeklasser har endelsen "Helper"

**Metoder**Alle metoder skrives som UpperCamelCase. Vi benytter .Net inline properties til get/set på instansvariabler.   
På metoder åbnes og lukkes tuborgklammerne under metodeheaderen. På kontrolblokke åbnes klammerne under statement, og afsluttes under.

Metoderne skal have sigende navne. Hvis man søger et objekt, men ikke laver ændringer til det, skal metoden starte med "Get". Hvis man laver en ændring skal metoden starte med "Update". Ved oprettelse af et objekt skal den starte med "Create" og ved sletning skal det være "Delete".  
I tilfælde der falder uden for CRUD skal der anvendes Sund Fornuft ved oprettelsen.

**Variabler**  
Alle almindelige variabler skrives lowerCamelCase, mens konstanter skrives STORT\_MED\_UNDERSTREGNING.

**Begrund valg af metode**

I forbindelse med udarbejdelsen af dette projekt, har vi valgt, at bruge agile udviklingsmetoder. Fordi vi er frie fugle på den måde.

**Kanban**

Kanban kommer fra Japan hvor Kan betyder *visuel*, og Ban betyder *kort*, og det er faktisk præcist det det er.  
Det er en helt simpel metode til at visualisere og dermed effektivisere og overskueliggøre en arbejdsprocess. Således er metoden ikke indskrænket til brug for software udvikling, men benyttes også i forbindelse med f.eks tilberedning af Sushi, og fremstilling af biler.

Kanban sit fulde fokus på flow, det handler hele tiden om, at komme videre, og hvis der opstår et problem, så er det alle man på opgaven, så den kan blive løst, før man fortsætter. Selv om man kan bruge kanban alene, så er der tale om en letvægtsmetode, der gør sig bedst i samspil med et større framework, som f.eks. Scrum. På figur 1 kan man se et bud på hvordan sådan et kunne tage sig ud til softwareudvikling.



Figur 1 - Kanban board

Umiddelbart ligner det ret meget det Scrum board vi i forvejen har, blot er der på kanban-boardet tilføjet tal i parantes på *In Progress* og *Review.* Disse tal er kendt som ”Work in Progress”-limits, eller WiP-limits, og signalerer klart og tydeligt hvor mange opgaver der må forefindes i den givne kolonne. I det ovenstående tilfælde ligger det f.eks fast, at for at projektet kan siges at køre på skinner, er der ingen der kan starte en ny opgave fra *backloggen*, og der er heller ingen der kan afslutte deres *In Progress*-opgaver, før der bliver fjernet en (eller flere) opgaver fra *Review.*

På den måde kan man på en let og overskuelig måde se om ens projekt skrider fremad som planlagt. Hvis der pludselig er for mange på en kolonne. F.eks hvis en task flyttes fra *In Progress* til *Review,*  så er det umiddelbart klart for enhver i teamet, at noget er skævt, og alle kan så samles og bidrage til, at løse kniben.

Det er vigtigt at WiP-limits bliver fastsat efter antal udviklere i teamet. Hvis de sættes for lavt, er der risiko for, at der er teammedlemmer som bliver låst inaktive, fordi der ikke må startes nye opgaver. Den slags forøger produktionstiden (lead time). Hvis man sætter WiP for højt, kunne man lige så godt lade være med at bruge kanban, da eventuelle flaskehalse alligevel først opdages på et senere tidspunkt. Bruger man Scrum og XP, så opdages den slags af andre veje.

* Afhjælper uklarheder ifb med udviklingen, f.eks kan man indsætte et kanban-board i sit Scrum-board, og på den måde indsætte flere trin i for eksempel *Igang*-koloonen for at repræsentere de trin *igang* dækker over, og samtidigt få nytte af WiP.   
  Selvom brugen af Scrum til en vis grad sikrer isolering af opgaver, i.e. det er givet hvilke User Stories der forventes løst i løbet af et sprint. Så er det nærmest umuligt at undgå, at skifte frem og tilbage mellem de mindre tasks. Det er især gældende hvis man samtidigt anvender XP-praktikkerne. Det kan give flaskehalse i løbet af et sprint, men ved at have WiP implementeret inde i selve *doing-*kolonnen, ville man kunne bevare overblikket på task-niveau
* Samtidigt ville det hjælpe en eventuel projektleder, eller product owner til at forstå hvor i processen en given opgave befandt sig.
* Slutteligt forhindrer en implementeret kanban halvfærdige opgaver. Pga. WiP, er man konstant tvunget til at levere 100% færdige opgaver. Scrum i sig selv kræver blot færdige opgaver på den yderste dag af et sprint.

**Sprint 3**

**Planlægning**

Dette sidste sprint i projektperioden er en dag kortere end de forgående. Derfor har vi blot afsat 12 story points til, at dække sprintet. Dermed skulle vi gerne holde os til vores ”37”-timers uge Igen i dette sprint har vi valgt at undlade parprogrammeringen. En af udfordringerne i det sidste sprint var, at det kollektive kodeejerskab blev sat lidt i baggrunden. Alle kan naturligvis rette i koden, uden at skulle spørge nogen om tilladelse, men det har typisk været de samme gruppemedlemmer der har stået for den samme type opgave, igennem hele forløbet. I dette sprint har vi målrettet besluttet, at ryste posen, så vi på den måde kunne sprede kompetencerne. I øvrigt har vi en hensigtserklæring i gruppen om, at oppe kommunikationsniveauet, så vi hurtigere kan overkomme eventuelle hindringer.

I den forbindelse har vi tidligere haft diskuteret hvorvidt vi burde anvende KanBan-metoden til at styre workflowet på vores tasks. Det har tidligere været en kilde til ærgrelse, at opgaver har trukket ud, hvor det ikke har været nødvendigt. Det ville vi i højere grad være istand til at gennemskue hvis vi også implementerede et KanBan-board. Det havde vi valgt at gøre hvis sprintet havde haft en normal længde. Men over en så kort periode vurderer vi, at processen med at implementere denne model, ville overskygge den værdi som vi kunne få ud af den. I øvrigt er Kanban ikke at betragte som en magisk løsning. Pga. Opgavens kompleksitet vil vi også fremadrettet rende ind i opgaver der går i hårdknude. Kanbans opgave ville blot være at identificere de hårdknuder for os, og måske hurtigere end vi ellers ville have set dem.

Som i de tidligere sprints, har vi fokus på kundeinvolvering, dels fordi det er en kernedel af udviklingsmetoden, men især fordi alle erfaringer peger på, at en hurtig forventningsjustering i fællesskab med kunden, kan medvirke til afhjælpe et ellers katestrofalt skred i tidsplanen (SE SPRINT 1)

Sprintbackloggen:

* Sikkerhed på brugernavn og password (x SP)
* (Opret??) Achievements (x SP)
* Administrere brugerprofil privatindstillinger (orig navn?) (x SP)
* Login skærm, find bruger (x SP)

**Review**

Det stod klart på andendagen af sprintet, at vi ikke ville kunne nå at lave den samlede ”administrer brugerprofil”-story. Dette skyldes at opgavens kompleksitet var større end antaget. I stedet blev vi, sammen med product owner, enige om, at splitte den op i to user stories, så vi til dette sprint kun skulle færdiggøre ”Ændre kodeord”, mens den resterende del af den originale userstory blev fremskudt til et teoretisk 4. sprint. Dog kunne vi demo det meste af det vi havde planlagt på. Hvad angår story points, så var den originale story på 5. Baseret på de ting som er lavet i systemet, UI, databasefelter, controller-metoder. Så har gruppen vurderet at 3 story points er brugt, hvorfor den uløste opgave er delegeret videre med en værdi på 2 story points.

De resterende opgaver fremviste vi på prototype niveau, hvilket vil sige, at vi har implementeret en simpel hashing af brugerens password, hvilket er nok til at opfylde accepttesten på den user story. En mindre fejl i koden gjorde, at vi var nødt til at vise effekten af ”ændre kodeord” i databasen i stedet for klienten.

”Opret Achievements” volder nogle problemer i klienten med visningen. Men da disse problemer er direkte forbundet til den user story som vi splittede op, så er det product owners overbevisning, at acceptesten er bestået ved visning af oprettelse i databasen.

Alt i alt har vi brændt 8 ud af 10 point i dette sprint. De 2 resterende tilhører den opgave der blev fremskudt

**Retrospective**

Selv om det i sig selv var et irritationsmoment , at vi var nødsaget til at splitte en story, så er vi dog ret godt tilfredse med, at have identificeret problemet hurtigt. Det gik både hurtigere og nemmere end i nogen af de tidligere tilfælde.  
Vi synes også estimeringen af velocity var god denne gang, med kun en opdelt task og ingen uforudsete spikes. Grunden skal findes i, at vi var meget bedre til, at arbejde på flere parallelle tasks. Vores fortsæt for sprintet angående en bedre kommunikation blev holdt, og derfor var vi istand til at nå opgaverne.  
Samlet viser både vores time burndown, og sp-burndown, at dette var det sprint hvor vi nåede mest af det vi havde estimeret.  
I øvrigt har dette været sprintet hvor vores enfant terrible; REST-servicen, ikke har givet anledning til hverken spikes, eller alvorlige bugs.

Det gik mindre godt med, at estimere hvilke tasks der hører med til de forskellige user stories, og det sker for tit, at vi er nødt til, at skrive nye task-sedler under sprintet. Nogle gange fjerner vi også en task der ikke giver nogen mening når man får den påbegyndt. Når vi stadig kan være nogenlunde på målet for sprintet, så skyldes det, at vi specifikt har planlagt vores velocity efter det.

Der har også været en del ændringer på eksisterende metoder, når vi tilføjer ny funktionalitet, hvilket har forårsaget en slem kattepine både i forbindelse med den opsplittede story, og i forhold til vores Achievement-story. Fordi vi religiøst følger Scrum-princippet om at alle user stories er uafhængige af hinanden, og derfor kan laves i præcist den rækkefølge product owner fastlægger, så er vi ofte nødt til at hardcode værdier ind i metoder, da den story som skulle forsyne dataen ikke er lavet endnu. Samtidigt har vi ikke har lavet nogle af de klassiske UP designdokumenter, og derfor ændrer vi ret ofte i vores design. Det betyder nogen gange at ting bliver glemt, og det skaber problemer. Vores Unit Tests er ikke nok til at afdække alle de problemer som dette medfører, da mange af dem kun er synlige fra klienten.

Vi kunne enten have en slags user story der kun var til for at knytte de her vertikale søjler af materiale sammen, men det ville være et brud på Scrum, da en sådan jo af gode grunde ikke ville kunne stå alene. Alternativt kunne vi planlægge at sætte tid af til det i hver story, som en latent task. Det ville nok være det bedste.

Projektplanlægning - ???