

1DV404
Iterativ mjukvaruutveckling
Laboration 4
Av: Roy Nilsson
rn222cx@student.lnu.se

Planering

Översikt.

Projektet kommer att börjas om från början och inte någon fortsättning på föregående projekt.

Målet är att skapa en klass för varje iteration och implementera så mycket som möjligt inom en viss tid samt testning av varje iteration.

För varje klass tillkommer det en testklass med passande namn.

Iteration 1.

Vad:

Implementera registrerings formulär som är anpassad för de olika aktörerna samt testa funktionalitet som behövs för att kunna registrera sig i systemet.

Analys kommer göras för att hitta olika fält och metoder som krävs för systemet.

Registrerings formuläret planeras vara helt färdig i denna iteration.

Mål:

- Skapa en klassen UserRegister.
- Skapa de olika fälten för:
 - Fullständigt namn.
 - Lösenord.
 - Email.
 - Val av aktör.
 - Person nummer.
- Regex kontroll på samtliga fält eller liknande.
- Samtliga uppgifter lagras på en fil i hårddisken.
- Testning av UserRegister klassen
- Lyckad registrering av aktör

Tid:

Iterationen kommer ta 10 timmar att implementera och testa

Iteration 2.

Vad:

Implementering av ett inlogginssystem som hämtar information från föregående iteration som skapar en fil med information om aktören. Testning av olika funktionaliteter som krävs för att kunna logga in på systemet.

Mål:

- Skapa klassen UserLogin
- Hämtar information från fil på hårddisken.
- Testning av klassen UserLogin
- Lyckad inloggning av aktör

Tid:

Iterationen kommer ta 7 timmar att implementera och testa

Iteration 3.

Vad:

Implementering av ett system som kan skapa nya ligor och lägga till nya medlemmar i ligorna.
Endast sekreteraren kommer ha behörighet till systemet.
Testning av olika funktionaliteter som krävs för att kunna registrera ligor och medlemmar i systemet.

Mål:

- Skapa klassen AddLeague
- Läs information från fil som ligger på hårddisken
- Information skrivs in i fil som ligger på hårddisken
- Endast sekreteraren har behörighet till systemet.

Tid:

Iterationen kommer ta 7 timmar att implementera och testa

Iteration 1

Översikt

Implementering av Registrerings formulär för aktörerna.

Krav

- Analysera användningsfallen, 30 minuter.
- Specifiera kraven, 30 minuter.

Design/Implementation

Implementation av:

- klassen UserRegister. 1 minut.
- fält och egenskaper, 1 timme.
- konstruktorerna, 15 minuter.
- regex kontroll på samtliga egenskaper, 2 timmar.
- Functionen som lägger in information till fil, 2 timmar.

Test

- Testar egenskaperna i samband med implementering genom enskild konstruktor för att se om de fungerar, 15 minuter
- Test design, 1 timme.
- Test specification, 30 minuter.
- Test implementation, 30 minuter.
- Test exekvering, 30 minuter.

Reflektion

- Reflektera 1 timme.

Total tid: 10 timmar

Krav:

- Regex kontrollera samtliga uppgifter om de är godkända.
- Inga fält får lämnas tomma.
- Lösenord måste upprepas och vara lika.
- Uppgifterna ska sparas i en fil vid lyckad registrering.

Reflektion

Det svåra är att skapa en iteration som ska passa inom en viss tidsram samt att det ska gå ihop med två andra iterationer som kommer implementeras. Jag började skissa på olika typer av användningsfall och försökte hitta användningsfall som passar bra till tidsramen samt att det finns kunskaper nog att implementera dem.

Den bästa idén som jag kom på är att helt enkelt börja från början och skapa ett användningsfall som kan bygga vidare på de flesta användningsfallen senare och det är just ett registreringsformulär.

Tack vare föregående laboration har man mer kunskaper som krävs till denna laboration och det känns som en tung sten har lättats och dokumentationerna går mycket smidigare än väntat.

Det känns skönt och börja med en större iteration för att kunna banta ner de nästkommande iterationerna.

Aktörer

Lag ledare

De kan registrera nya lag med medlemmar samt registrera vilka grenar, ålder och kön de tillhör. De kan även ta bort lag som de har registrerat.

Sekreterare

Handhåller Tävlingspoängen och räkna ut medelpoängen. Sekreteraren kan skriva in och ändra poängen i systemet.

Domare

Domaren kan logga in och ändra i schemat, exempelvis vilka domare som ska döma vilka lag samt tider.

Flöden

Användningsfalls beskrivningar

Användaren fyller i samtliga fält och klickar på registrera ny användare.

Previllkor:

- Användaren har gått in på webbsidan och hittat registrerings länken.

Postvillkor:

- Registrering av användare är lyckad.
- Alla data som är ifyllt har skickats till databasen(filen).

Primära:

- Bekräftelse på en lyckad registrering (Den perfekta världen)

Alternativa flöden:

- Fält som inte är korrekt ifyllt kastas ett undantag.
- Fält som är tomma kastas ett undantag.

Klassspecifikationer

Klass UserRegister

Klassen är avsedd för aktörer som registrera sig i systemet innehållande fält, metoder, egenskaper och konstruktorer.

Fält

string _fullname;

Innehåller för och efternamn på användaren.

string _password;

Innehåller användarens lösenord.

string _password2;

Innehåller användarens upprepade lösenord.

string _email;

Innehåller användarens email.

string _actor;

Innehåller användarens roll.

string _pin;

Innehåller användarens personnummer.

Egenskaper

String Fullname

Ger fältet _username ett värde av typen string samt ett krav på att namnet ska innehålla mellan 4-25 tecken.

String Password

Ger fältet _password ett värde av typen string samt går igenom metoden PasswordIsValid ().

String Password2

Ger fältet _password2 ett värde av typen string samt kollar så att värdet stämmer överens med Password.

String Email

Ger fältet _email ett värde av typen string samt går igenom metoden EmailIsValid ().

String Actor

Ger fältet `_actor` ett värde av tre möjliga.

String pin

Ger fältet `_pin` ett värde av typen `string` samt går igenom metoden `PinIsValid ()`.

Konstruktörer**Register()**

Skapa ett nytt objekt av klassen `Register`.
Inparametrar är samtliga fält.

Metoder**PasswordIsValid ()**

Kollar så att `_password` innehåller mellan 6 till 25 tecken samt inkludera minst en siffra genom reguljära uttryck.

EmailIsValid ()

Kollar om `_email` är en valid mail genom reguljära uttryck.

PinIsValid()

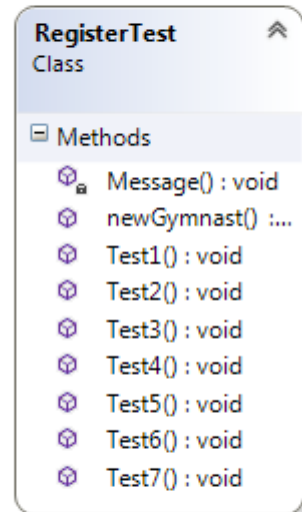
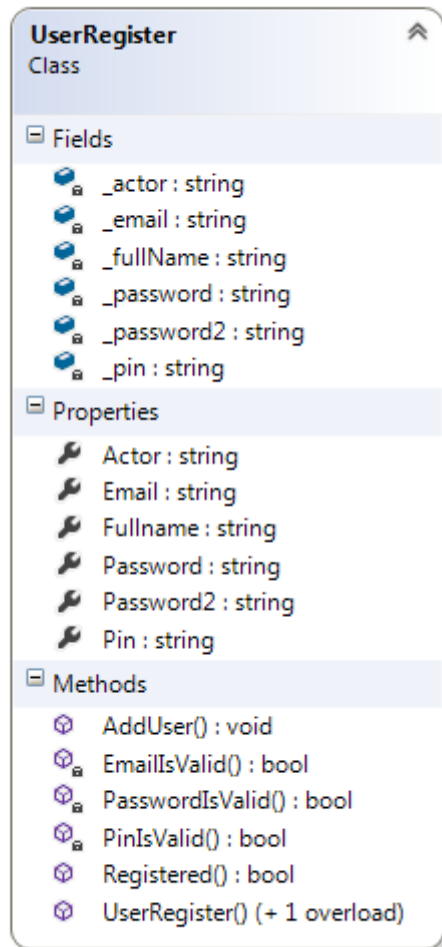
Kollar om `_pin` är ett giltigt personnummer genom reguljära uttryck.

AddUser()

Lägger in samtlig information till en text fil

Registered()

Returnera ett booleanskt värd för att se om registreringen har lyckats.



Testdokumentation

Testningen går ut på att säkerställa att alla delar i klassen fungera som planerat och uppnår dokumenterade krav. Testningen kommer även se till att minimera eventuella buggar.

Testsvit 1 – Förväntade resultat av klassen UserRegister

Nr	Typ av test	Testdata	Metod	Kommentar	Förväntat Resultat
1	Kontroll av registrering	Namn = "roye" Lösenord = "haxx0r" Email = "roy@lnu.se" Aktör=" IAg LeDaRE" Pin="871131-2739"	Skapa en instans av klassen UserRegister och skapa en ny konstruktör som innehåller samtliga parametrar.		Ny Medlem registreras.
2	Kontroll av namn	Namn = "roy"	Fullname()	Kräver minst 4 tecken	Inte godkänt
3	Kontroll av lösenord	Lösenord= "haxxor"	Password()	Kräver en siffra minst	Inte godkänt
4	Kontroll av Email	Email="roy.87@msn.com"	Email()	Både 2 och 3 bokstäver efter sista punkt ska godkännas	Email skapas
5	Kontroll av aktör	Aktör="domare"	Actor()	Val av aktör är inte beroende av att vara gemener eller versaler	Aktör skapas
6	Kontroll av personnummer	Pin="871331-2739"	Pin()	Värdet överstrider antal månader "13" och kommer inte att godkännas	Inte godkänt
7	Kontroll av upprepat lösenord	Lösenord="password"	Password2()	Värdet har inte samma värde som password() och därför falera det	Inte godkänt

Testsvit 1 – Resultat av klassen UserRegister

Nr	Typ av test	Testdata	Metod	Förväntat	Verkliga
----	-------------	----------	-------	-----------	----------

				Resultat	Resultat
1	Kontroll av registrering	Namn = "roye" Lösenord = "haxx0r" Email ="roy@Inu.se" Aktör=" IAg LeDaRE" Pin="871131-2739"	Skapa en instans av klassen UserRegister och skapa en ny konstruktör som innehåller samtliga parametrar.	Ny Medlem registreras.	Godkänt
2	Kontroll av namn	Namn = "roy"	Fullname()	Inte godkänt	Inte godkänt
3	Kontroll av lösenord	Lösenord= "haxxor"	Password()	Inte godkänt	Inte godkänt
4	Kontroll av Email	Email="roy.87@msn.com"	Email()	Email skapas	Godkänt
5	Kontroll av aktör	Aktör="domare"	Actor()	Aktör skapas	Godkänt
6	Kontroll av personnummer	Pin="871331-2739"	Pin()	Inte godkänt	Inte godkänt
7	Kontroll av upprepat lösenord	Lösenord="password"	Password2()	Inte godkänt	Inte godkänt

Sammanfattning av tester.

Klassen är uppdelad i sviter och de testas främst genom deras metoder för att snabbt och enkelt kunna testa deras funktionalitet.

Testerna körs och returnera om testet är

godkänd eller inte godkänt genom en try- catch sats.

Samtliga tester är inkapslade i en metod som innehåller en try- catch sats för att utgöra om testet blir godkänt eller inte och där efter hämtar en metod som färgkodar testerna genom boolenskt värde sant eller falskt.

Testerna ger en bra visuell bild av samtliga tester i konsol fönstret

Iteration 2

Översikt

Implementering inloggningsystem för aktörerna.

Krav

- Analysera användningsfallen, 30 minuter.
- Specifiera kraven, 30 timme.

Design/Implementation

Implementation av:

- klassen UserLogin, 1 minut.
- Fält, metoder och egenskaper, 1 timma.
- Function som hämtar ut information från fil, 1 timme.
- Function som kollar om login information stämmer överens, 1 timme.

Test

- Test design, 30 timme.
- Test specification, 30 minuter.
- Test implementation, 30 minuter.
- Test exekvering, 30 minuter.

Reflektion

- Reflektera 1 timme.

Total tid: 7 tim

Krav:

- Logga in med email samt angivet lösenord.
- Inga fält får lämnas tomma.
- Lösenord måste vara beroende av gemener och versaler.
- Email ska inte vara beroende av gemener och versaler.
- Användaruppgifterna ska hämtas från en fil.

Reflektion

Det svåra med denna iteration var att försöka hitta en lösning att skapa iterationen i klassnivå men ändå kunna hämta ut information från förra iterationen.

Jag tycker jag har hittat en lagom mängd kod att implementera samt att får jobba med något som jag inte har arbetat med innan med att skapa och läsa från textfiler.

En lite bättre genomtänkt strategi kunde behövas då det inte funkade helt som jag ville i slutet av implementeringen men till hjälp av en del guider från internet kunde jag förbättra implementeringen.

Att få dessa två iterationer sammanhängande lyckades jag med tillslut och implementationen blev bra i slutändan även att det stannade upp en hel del.

Aktörer

Lag ledare

De kan registrera nya lag med medlemmar samt registrera vilka grenar, ålder och kön de tillhör. De kan även ta bort lag som de har registrerat.

Sekreterare

Handhåller Tävlingspoängen och räkna ut medelpoängen. Sekreteraren kan skriva in och ändra poängen i systemet.

Domare

Domaren kan logga in och ändra i schemat, exempelvis vilka domare som ska döma vilka lag samt tider.

Flöden

Användningsfalls beskrivningar

Användaren fyller i inloggnings fälten och klickar på logga in.

Previllkor:

- Användaren har registrerat sig i systemet.

Postvillkor:

- Innloggningen är lyckad.
- Alla data har hämtats från databasen(filen) och jämförts med inskrivet värde.

Primära:

- Bekräftelse på en lyckad inloggning (Den perfekta världen)

Alternativa flöden:

- Fält som inte har korrekt värde kastas ett undantag.
- Fält som är tomma kastas ett undantag.

Klassspecifikationer

Klass UserLogin

Klassen är avsedd för aktörer som loggar in sig i systemet innehållande fält, metoder och egenskaper.

Fält

string _loginName;

Innehåller email adress från registreringen.

string _loginPassword;

Innehåller lösenord från registreringen.

string _password2;

Innehåller användarens upprepade lösenord.

Egenskaper

String LoginName

Ger fältet _loginName ett värde av typen string samt ett krav på att fältet inte får lämnas blankt

String LoginPassword

Ger fältet _password ett värde av typen string samt ett krav på att fältet inte får lämnas blankt

Metoder

Login ()

Innehåller två parametrar som ska föreställa användarens inloggnings namn och lösenord. Metoden läser in från en fil som lagrar användaruppgifter som sedan jämförs med värdet som skrivs in från parameterna.

Testdokumentation

Testningen går ut på att säkerställa att alla delar i klassen fungera som planerat och uppnår dokumenterade krav. Testningen kommer även se till att minimera eventuella buggar. Innan testning har en användare registrerat sig för att kunna fullfölja inloggnings test.

Testsvit 2 – Förväntade resultat av klassen UserLogin

Nr	Typ av test	Testdata	Metod	Kommentar	Förväntat Resultat
----	-------------	----------	-------	-----------	--------------------

8	Kontroll av inloggning	Namn = "roy@lnu.se" Lösenord = "haxx0r"	Login()	Skapa en instans av klassen UserLogin och prova metoden Login()	Medlem loggas in
9	Kontroll om namn går att använda istället för email	Namn = "roye" Lösenord = "haxx0r"	Login()	Bör inte funkar	Godkänns inte
10	Kontroll av felangivet lösenord	Namn = "roy@lnu.se" Lösenord = "password"	Login()	Bör inte funka	Godkänns inte
11	Kontroll om lösenord är case sensitive	Namn = "roy@lnu.se" Lösenord = "Haxx0r"	Login()	Bör inte funka	Godkänns inte
12	Kontroll om email är case sensitive	Namn = "Roy@lnu.se" Lösenord = "haxx0r"	Login()	Ska funka	Godkänt
13	Kontroll av tomma fält	Namn = "" Lösenord = ""	Login()	Bör inte funka	Godkänns inte

Testsvit 2 – Resultat av klassen UserLogin

Nr	Typ av test	Testdata	Metod	Förväntat Resultat	Verkliga Resultat
8	Kontroll av inloggning	Namn = "roy@lnu.se" Lösenord = "haxx0r"	Login()	Medlem loggas in	Godkänt
9	Kontroll om namn går att använda istället	Namn = "roye" Lösenord = "haxx0r"	Login()	Bör inte funkar	Inte Godkänt

	för email				
10	Kontroll av felangivet lösenord	Namn = "roy@lnu.se" Lösenord = "password"	Login()	Bör inte funka	Inte Godkänt
11	Kontroll om lösenord är case sensitive	Namn = "roy@lnu.se" Lösenord = "Haxx0r"	Login()	Bör inte funka	Inte Godkänt
12	Kontroll om email är case sensitive	Namn = "Roy@lnu.se" Lösenord = "haxx0r"	Login()	Ska funka	Godkänt
13	Kontroll av tomta fält	Namn = "" Lösenord = ""	Login()	Bör inte funka	Inte Godkänt

Sammanfattning av tester.

Klassen är uppdelad i sviter och de testas främst genom deras metoder för att snabbt och enkelt kunna testa deras funktionalitet.

Iteration 3

Översikt

Implementering av system som skapar nya ligor samt registrera medlemmar till ligorna. Systemet är avsedd för sekreteraren som är den enda aktören som har behörighet till systemet.

Krav

- Analysera användningsfallen, 30 minuter.
- Specifiera kraven, 30 timme.

Design/Implementation

Implementation av:

- klassen AddLeague, 1 minut.
- Fält, metoder och egenskaper, 1 timma.
- Kod som kontrollera behörighet, 30 min
- Function som hämtar ut information från fil, 30 minuter.
- Function som lägger in Ligor och medlemmar i fil på hårddisken, 1 timme

Test

- Test design, 30 timme.
- Test specification, 30 minuter.
- Test implementation, 30 minuter.
- Test exekvering, 30 minuter.

Reflektion

- Reflektera 1 timme.

Total tid: 8 tim

Krav:

- Endast sekreteraren har behörighet till systemet.
- Registrerad liga ska läggas in i fil
- Registrerade medlemmar ska läggas in under ligans namn i samma fil
- Ligor och lag skrivs inte över när nya registreringar utförs

Reflektion

Den tredje iterationen var den svåraste att planera då den ska implementeras med de andra iterationerna och hämta information från dessa.

Jag hade ide torka om vad jag skulle implementera i systemet som passar systemet och håller fast vid tidsramen.

Det var svårt att försöka hålla sig till en tidsram och inte överskrida den då det inte kommer en nästa iteration man kan lägga på eller ta bort tid i.

Dokumentationen blev som enklast i sista iterationen då man har fått dokumentera en hel del på vägen.

Aktörer

Sekreterare

Handhåller information om ligor och medlemmar samt registrera nya i systemet.

Ligor

Lämna information till sekreteraren.

Blir registrerade och har ingen påverkan i systemet.

Flöden

Användningsfalls beskrivningar

Sekreteraren fyller i fälten liga och medlemmar och klicka på registrera

Previllkor:

- Sekreteraren har registrerat sig och loggat in på systemet.

Postvillkor:

- Registrering av liga och medlemmar är lyckad.
- Alla data som är ifyllt har skickats till databasen(filen).

Primära:

- Bekräftelse på en lyckad registrering (Den perfekta världen)

Alternativa flöden:

- Vid fel typ av behörighet kastas ett undantag.

Klassspecifikationer

Klass AddLeague

Klassen är avsedd för sekreteraren som registrera nya ligor och medlemmar i systemet innehållande fält, metoder, egenskaper och konstruktorer.

Fält

string _league;
Innehåller ligans namn.

Egenskaper

String League
Ger fältet _league ett värde av typen string.

Konstruktorer

AddLeague ()
Skapa ett nytt objekt av klassen AddLeague.
Inparametrar är league.


Metoder

void AddLeagues()
Kollar så att användaren är en sekreterare och om så är fallet skapas en ny liga och läggs in i databasen(filen).


void AddMembers()
Kollar så att användaren är en sekreterare och om så är fallet kan fritt antal medlemmar skrivas in samt läggs in i databasen(filen).

AddLeague
Class


Fields


 _league : string


Properties

 League : string

Methods


 AddLeague()


 AddLeagues() : void


 AddMembers() : void


LeagueTest
Class


Methods


 Message() : void

 Test14() : void

 Test15() : void

 Test16() : void

 Test17() : void

 Test18() : void

Testdokumentation

Testningen går ut på att säkerställa att alla delar i klassen fungera som planerat och uppnår dokumenterade krav. Testningen kommer även se till att minimera eventuella buggar.

Testsvit 3 – Förväntade resultat av klassen AddLeague

Nr	Typ av test	Testdata	Metod	Kommentar	Förväntat Resultat
14	Kontrollera om domare kan lägga till ligor	Aktör="domare" Liga = "NewLeague"	AddUser() AddLeagues()	Skapa en ny aktör och försöker skapa en liga.	Godkänns inte
15	Kontrollera om sekreterare kan lägga till ligor	Aktör="sekreterare" Liga = "NewLeague"	AddUser() AddLeagues()	Skapa en ny aktör och försöker skapa en liga.	Godkänns inte
16	Kontrollera om sekreterare kan lägga till medlemmar	Aktör="sekreterare" Medlem ="Roy"	AddUser() AddMembers()	Endast lag ledare kan lägga till medlemmar	Godkänns inte
17	Kontrollera om lag ledare kan lägga till ligor	Aktör="lag ledare" Liga = "NewLeague"	AddUser() AddLeagues()	Lag ledare ska kunna lägga till liga	Liga läggs till
18	Kontrollera om lag ledare kan lägga till medlemmar	Aktör="lagledare" Skrivs in manuellt Medlem ="Roy, Åke"	AddUser() AddMembers()	Lag ledare ska kunna lägga till medlemmar	medlemmar läggs till
19	Kontrollera om lag ledare kan lägga till liga samt medlemmar	Aktör="lagledare" Liga = "NewLeague" Skrivs in manuellt Medlem ="Roy, Åke"	AddUser() AddLeagues() AddMembers()	Lag ledare ska kunna lägga till liga samt medlemmar	Liga läggs till Samt medlemmar

Testsvit 3 – Resultat av klassen AddLeague

Nr	Typ av test	Testdata	Metod	Förväntat Resultat	Verkliga Resultat
14	Kontrollera om domare kan lägga till ligor	Aktör="domare" Liga = "NewLeague"	AddUser() AddLeagues()	Undantag kastas	Inte godkänt
15	Kontrollera om sekreterare kan lägga till ligor	Aktör="sekreterare" Liga = "NewLeague"	AddUser() AddLeagues()	Undantag kastas	Inte godkänt
16	Kontrollera om sekreterare kan lägga till medlemmar	Aktör="sekreterare" Medlem ="Roy"	AddUser() AddMembers()	Undantag kastas	Inte godkänt
17	Kontrollera om lag ledare kan lägga till ligor	Aktör="lag ledare" Liga = "NewLeague"	AddUser() AddLeagues()	Liga läggs till	Godkänt
18	Kontrollera om lag ledare kan lägga till medlemmar	Kontrollera om lag ledare kan lägga till medlemmar	Kontrollera om lag ledare kan lägga till medlemmar	medlemmar läggs till	Godkänt
19	Kontrollera om lag ledare kan lägga till liga samt medlemmar	Aktör="lagledare" Liga = "NewLeague" Skrivs in manuellt Medlem ="Roy, Åke"	AddUser() AddLeagues() AddMembers()	Liga läggs till Samt medlemmar	Godkänt

Sammanfattning av tester.

Klassen är uppdelad i sviter och de testas främst genom deras metoder för att snabbt och enkelt kunna testa deras funktionalitet.

Testerna körs och returnera om testet är

godkänd eller inte godkänt genom en try- catch sats.

Samtliga tester är in kapslade i en metod som innehåller en try- catch sats för att utgöra om testet blir godkänt eller inte och där efter hämtar en metod som färgkodar testerna genom boolenskt värde sant eller falskt.

Testerna ger en bra visuell bild av samtliga tester i konsol fönstret.

Slut reflektion

En stor svårighet av projektet var att planera tre iterationer som var av en lagom mängd kod och som hade en samhörighet med varandra. Jag kände att jag hade dålig fantasi av att välja användningsfall samt att vissa användningsfall hade varit för svåra för mig att implementera av brist på kunskap eller tid.

Jag har inte valt att implementera projektet i C# för att jag känner mig säkrast i det språket utan att föregående laborationer har varit väldigt svårtydliga att utföra i andra språk och jag vill inte riskera att göra om projektet i ett annat språk.

Det är svårt att dokumentera en tidslogg då man ofta tappa bort tiden och glömmer bort hur långa pauser man tar.

Jag tyckte det var för lite information om vad som skulle dokumenteras under iterationerna och jag fick leta mer information från forumet samt andra klasskamrater för att få en komplett uppfattning om vad som ska dokumenteras.

Det hade kunnat vara bra med någon slags mall eller exempel hur projektet kan se ut.