Laboration 4 – 1DV404

Lucas Wik

Iteration 1 – Planering:

(Jag har bestämt mig för att börja om ifrån början och inte använda någon av mina tidigare laborationer som start.)

Introduktion: Så till att börja med så vill jag bygga en grund till hela systemet i min första iteration, denna grund ska innehålla lag registrering, editering och loggin möjligheter för admins. Admins ska också bara vara de som kan ändra poäng/ lagnamn osv. Jag anser att denna kommer att vara den som tar längst tid eftersom det är ganska mycket som ska implementeras.

Iteration 1 – Mål:

- Registrering av lag, ett lag ska kunna registrera sig och dess X mängd av medlemmar. 2 lag ska INTE kunna ha samma namn.

- Editering av lag, admins ska kunna editera lag. Därför måste det implementeras system som kan utföra de tjänsterna.

- Admin konton ska skapas då de ska ha möjlighet att editera dessa lag. Allt ska kunna ändras av en admin, lagnamn, medlem, träff plats. Allt. Ta detta i tanke vid uppsättningen av systemet.

- Systemet behöver inte se till talande ut men de olika valen användaren har ska framgå tydligt vad de gör.

Tidsberäkning: 10-11 tim. (Motivation för beslut: det är mycket att göra men jag är bekväm med det mesta och därför ser inga stora problem som jag kan stöta på. Enda som är nytt är att jag ska göra mycket mera i mina klasser men tror jag inte får problem som jag inte kan lösa snabbt.)

Iteration 2 – Planering:

Introduktion: Nu när basen är helt färdig så måste vi slipa kanterna. Det jag vill fokusera på i denna iteration är att lägga till mera system krav och stila upp det lite. Till och börja med så ska poängen räkningen implementeras så att ett lags poäng räknas ut genom att addera ihop alla medlemmarnas poäng i laget och att en medlems poäng i en gren kan ges av en sekreterare. En medlems poäng är summan av alla poängen den personen har fått i alla grenar hon/ han har deltagit i.

Träffar behöver också bestämmas så att när en registrering sker så måste de också ange vilken stad de vill tävla i samt vilken träff.

Iteration 2 – Mål:

- Poängberäkningen, en admin(sekreterare) ska kunna ge en medlem poäng i varja liga han/ hon deltar i. Därefter så ska systemet räkna ut varje medlems poäng genom att addera alla poängen personen har fått i alla grenar, ett lags poäng är summan av alla dessa slut poäng adderat ihop.

- Vid detta stadiet så behöver inte systemet se snyggt ut men meny uppsättningen och valen ska vara bra arrangerat.

Tidsberäkning: 9 tim. (Motivation för beslut: eftersom dessa mål är saker som ska läggas till mitt i applikationen så förväntar jag mig att behöva omstrukturera någon del.)

Iteration 3 – Planering:

Introduktion: Så nu när de mesta kraven är ifyllda så är det dags göra de sista tilläggen. Det jag hade tänkt mig för iteration 3 är att användaren har valet att se en lista av alla lag, med alla medlemmars poäng och lagens poäng och ordna dem från laget med mest poäng överst i listan och laget med minst poäng längst ner i listan. Detta blir ett val användarna kan vilja se vid slutet av en säsong eller bara kolla hur de ligger till i deras träff. Samt så ska systemet se grafiskt tilltalande ut.

Iteration 3 – Mål:

- Systemet ska se grafiskt till talande ut och meny valen ska vara bra arrangerat.

- En användare ska kunna se en lista över alla lag och medlemmar då de arrangeras i lednings order, dvs. att de som leder är överst på listan.

Tidsberäkning: 9 tim. (Motivation för beslut: även om det inte är så mycket att göra så tror jag att toppa av programmet med lite design kan vara en tidsslukare.)

Slutkrav för projektet (sammanfatting):

- Man ska kunna registrera och admins ska kunna editera lag.
- Grafiskt tilltalande.
- Ett poängberäknings system som underlättar för sekreteraren.

Iteration 1

Detaljerad planering:

Krav – analysera hur den totala uppsättningen kommer att se ut. (30 min)

Implementation – Identifiera klasser och deras uppgifter samt implementera. (6 tim)

Test – Identifiera olika test fall för en aktör och deras mening. Implementera tester (3tim)

Reflektion – Reflektera på iterationen. (45 min)

Steg 1 – Krav:

Funktionella krav:

- Ska kunna registrera ett lag samt editera dem.
- Admin konton ska skapas och bara de ska ha möjlighet att kunna editera lag. Mer edit möjligheter och poäng tillägg implementeras i senare iterationer.
- Behöver inte se grafiskt tilltalande ut.

Klasser:

- RegisterCheck, registerar ett lagnamn och ser till att det inte redan finns registrerat samt kollar om max mängd av lag redan är registrearade.
- Editor, denna klassen kommer vara lite av en data samling, den kommer läsa in och innehålla det mesta av all data, så som poäng, medlems namn osv.
- LoggIn, denna klass kontrollerar loggin uppgifter och har en metod som retunerar true om de stämmer, false om de inte stämmer.

Aktörer:

- Tävlare/ klubbar, behöver register funktionen för att kunna registrera sitt lag.

- Sekreterare, behöver denna funktion så hon/ han kan vara ensam om att editera.

Flöden:

- Tävlare/ klubbar:
    - Startar appliaktionen
        - Krashar - > Misslyckat försök.
    - Anger valet registrera lag.
    - Anger registrerings uppgifter.
        - Råkar skriva in tomt fält, användaren blir noterad om att ett fel skedde och får skriva om.
    - Lyckad registrering.
- Sekreterare:
    - Startar appliaktionen.
        - Krashar. - >Misslyckat försök.
    - Anger loggin valet och skriver in användar namn och lösen.

        - Anger fel uppgifter -> blir meddelad om misstag.
    - Blir lyckat inloggad, ska nu visas för att göra användaren medveten om att hon/ han är inloggad.

Steg 2 – Implementation:

Klasspecifikationer:

- Editor:
    - Egenskaper och fält:
        - 2 konstanter, MaxNumberofAllowedRegisters, teamSize. Används för att bestämma array storleker.
        - 2 int variabler: Index & position(Pekar ut vilken plats som ska ändras i multi dimensionella arrays.)
        - 1 string array, teamNames, innehåller alla namn.
        - 1 multidimensionel array, string, innehåller alla medlemmars namn.
    - Metoder:
        - ReadMemberNames: läser in lagnamn och registrerar dem i arrayen, denna kallas bara på när ett lag har gått igenom RegisterCheck.
        - ChangeNames: Grund metoden för att editera namn, den kallar på funktioner som utför själva ändringarna.
        - ChangeTeamName: Byter ut lag namnet på utvald array position. (Gjordes under arbetets gång.)
        - ChangeTeamMember: Byter ut en medlems namn på utvald position. (Gjordes under arbetets gång.)
        - ShowAllTeams: Visar alla registrerade lag. (Gjordes under arbetets gång.)
        - ShowChosenTeam: Visar ett utvalt lag. (Gjordes under arbetets gång.)
- RegisterCheck:
    - Metod:
        - teamRegister: Registrerar ett lag och kollar så att namnet inte redan är taget.
        - ChangeTeamName: Byter ut ett lag namn på angiven array plats.
        - getTeamNames: returnerar Lagnamns arrayen. (Gjordes under arbetets gång.)
        - setTeamNames: setar den lokala TeamNames arrayen till det medsickade värdet. (Gjordes under arbetets gång.)

    - Egenskaper och fält:
        - En konstant för max mängden av tillåtna registrerade lag samt en array för alla lag namn.

- LoggIn:
    - Metod:
        - ValidateInfo: Kollar så att den inskriva infon över stämmer med inloggnings info, returnerar true om sant, false om falskt.
    - Metod: getPassword
        - Kod för att få lösenordet att framstå som asterisker och retunerar lösenordet som en sträng.
    - Egenskaper och fält:
        - AccountName: ett fält som tar värdet av det med sickadekonto namnet.
        - AccountPasswrod: ett fält som tar värdet av det medsickade konto lösenordet.
        - AccountNames: Array som innehåller konto namn, konto namnets lösen har samma array index som AccountPasswords.
        - AccountPassword: Array som innehåller konto lösenord, lösenorderts index i arrayn är samma som dess konto namn i AccountNames.
        - Index: får värdet som ges av Array.IndexOf.

Steg 3 – Test:

Testfall:

- (Test 1)Vad händer om användaren skriver in fel logg-in info? Hur meddelas dem? Var händer?
    - Förväntat resultat: Användare meddelas om att inloggnings infon är fel, samt vilken av dem som är fel, dvs. kontonamn eller lösen.

- (Test 2)Kan man skriva in ett konto namn och sen lyckas använda ett annat kontos lösen och fortfarande loggas in?
    - Förväntat resultat: Nej, detta ska inte lyckas. Endast 1 lösen ska funka för varje konto namn.

- (Test 3)Kan 2 lag ha samma namn? Om ett lag namn som redan är registrerat försöks registreras igen, hur hanterar programmet detta?
    - Förväntat resultat: användaren får ett medelande om att lagnamnet redan är registrerat.

- (Test 4)Vad händer om max antalet tillåtna registrerade lag har nåtts? Vad händer?
    - Förväntat resultat: användaren meddelas om detta.

Testdata:

Test:	*Vad som testas:*	*Kommentar:*	*Pass/fail*	*Datum:*
Test 1, fel loggin info blir angiven.	Metoden ValidateInfo i LoggIn	Förväntar att metoden retunerar false.	Pass	2015-01-07
Test 2, om man kan logga in med 1 lösen som tillhör ett annat användare namn.	Metoden ValidateInfo i Loggin.	Förväntar att metoden retunerar false.	Pass	2015-01-07
Test 3, om 2 lag kan ha samma namn	Metoden teamRegister i klassen RegisterCheck.	Förväntar att metoden retunerar samma nummer som förra registreringen eller 0. Samt meddelar användaren.	Misslyckat	2015-01-07
Test 4, om ytligare en registrering försöks göra om maxantalet registrerade redan har nåtts.	Metoden teamRegister i klassen RegisterCheck.	Förväntat att registrering stoppas och att användaren blir meddelad om varför.	Lyckat	2015-01-07

Re:Testdata:

Test 3, om 2 lag kan ha samma namn.	Metoden teamRegister i klassen RegisterCheck.	Förra failade pga. att jag testade en ny metod för att jämföra lagnamn och den kunde inte hantera stora/ små bokstäver.	Lyckat	2015-01-07

Reflektion:

Nu när jag är klar med min iteration 1 så inser jag att jag kanske hade lite mycket i den. Inte bara har det varit smått tidspressande så ha det varit saker jag glömt eller slarvat med, t ex. så fick jag lägga till några flera fält/ metoder än vad som var i min planering, vissa av dem visste jag till och med att jag behövde göra men hade bara inte kommit till tanke när jag skrev planeringen. Vilka det var nämnde jag inom parenteser som ni säkerligen sett.

Jag kommer också ändra min uppsättning lite i nästa iterations plan, lite mer detaljerat om vad jag ska spendera tiden på istället för hur mycket tid jag tänker spendera på var steg samt så kommer jag vara mer noggrann i min loggbok så det blir lättare att hålla koll vad jag spenderar tid på. Men vad jag menar med att vara mer detaljerad i min planering så är det t ex. att jag ska bryta ut tiden jag spenderar på implementation av målen samt implementationen av tester eftersom de tar upp mycket tid i de stegen.

På tal om tester så tycker jag att jag har problem att komma på bra sådana, jag brukar alltid ha i baktanke när jag jobbar ”om vad händer om de gör så här, eller så här?” Test 3 misslyckades egentligen bara för att jag testade en ny metod för att kolla om ett namn var taget, men den kunde inte hantera om 2 namn var lika fast bara med olika versaler/ germener.

Iteration 2

Detaljerad planering:

- Krav:
  - Analysera användnings fall och aktörer samt flöden (45 min)
  - Planera uppsättningen (30 min)
- Design och implementation:
  - Identifiera klasser/ metoder/ egenskaper (30 min)
  - Implementera (2 ½ tim)
- Test
  - Analysera testfall (30 min)
  - Test design(30 min)
  - Implementation(1 tim)
  - Exekvering av tester och analysera testdata(45 min)
- Reflektion
  - Reflektera (45 min)

Steg 1: Krav

Funktionella mål:

- Poängräkning för lag och individuella medlemmar. Ett lags poäng är alla lagets medlemmars poäng ihop adderat.
- Arrangerad meny.

Aktörer:

- Sekreterare: behöver poängräkningen implementerad för lättare underhåll av alla medlemmars poäng.

Flöden:

- Sekreterare:
  - 1.(efter inloggning) Väljer meny alternativ poäng Lägg till/ ändra deltagandes poäng.
    - Applikationen kraschar -> misslyckat försök.
  - 2. Väljer vilket lag som ska editeras enligt en meny av alla registrerade lag.
    - Inga lag registrerade, användaren tvingas tillbaka till huvudmeny.
  - 3. Väljer ut vilken medlem poängen ska ändras på/ läggas till på.
  - 4. Lyckat flöde.

Steg 2: Design och implementation

Klasspecifikationer:

- (Tillägg till Editor klassen)
    - Fält/ egenskaper:
        - Points,innehåller int värdet av poängen som ska ges.
        - AllMemberPoints, multidimensionell int array som innehåller alla medlemmars poäng.
        - TeamPoints, en array med alla lags poäng.
    - Metoder:
        - ChangeMemberPoints, en meny för poäng redigering(Gjordes under arbetets gång.)
        - ResetMemberPoints, resettar en medlems poäng. (Gjordes under arbetets gång.)
        - ConfirmChoices, frågar om användaren är nöjd med valen. (Gjordes under arbetets gång.)
        - UpdateTeamPoints, räknar ut ett lags poäng efter uppdatering av en av dess medlems poäng.
        - AddMemberPoints, räknar ut en medlems poäng eller resetar det om så önskat.

Steg 3: Test

Testfall:

- (Test 1)Felaktig poängsumma ges. (ett 0-10 möjliga givna poäng per gren?)
    - Förväntat resultat: Points egenskap kastar ett fel.

- (Test 2)Den totala lag poäng summan uppdateras rätt och vid behövt tillfälle.
    - Förväntat resultat: Poängsumman uppdateras alltid efter reset av poäng eller addering av poäng.
- (Test 3)Användaren väljer valet att lägga till/ resetta poäng och inga lag är registrerade.
    - Förväntat resultat: Användaren får medelande om att inga lag än är registrerade.
- (Test 4)Om en *utloggad* användare väljer alternativ 3.
    - Förväntat resultat: användaren blir meddelad om att detta meny val in finns tillgängligt.

Testdata:

Test:	Vad som testas:	Kommentar:	Pass/ fail:	Datum:
Test 1, för hög poäng summa	Egenskapen Points tilldelas värdet 11.	Ett fel förväntas att kastas	Pass	2015-01-08
Test 2, uppdatering av lagpoäng.	Att uppdaterings metoden körs vid behov.	3 medlemmar ges manuellt 5 poäng var och sedan resettar en av deras poäng till 0.	Pass	2015-01-08
Test 3, väljer meny val vid icke möjlighet.	Menyval 3 körs när inga lag är registrerade.	Användaren ska bli meddelad om att inga lag än är registrerade.	Pass	2015-01-08
Test 4, en utloggad användare väljer ett menyval de inte har tillgång till.	Om en ej befogad användare försöker komma åt menyval	Användaren ska bli informerad om att det menyvaler inte finns tillgängligt.	Acceptabel	2015-01-08

Steg 4: Reflektion

I denna iteration så blev det betydligt mindre saker jag fick lägga till under arbetets gång. Det var 3 metoder som jag anser ska göra saker smidigare, först och främst den första ChangeMemberPoints som är en start meny för att redigera någons poäng som därefter kallar på ResetMemberPoints eller AddMemberPoints. Dessa två kallar sedan på metoden ConfirmChoices, denna frågar om användaren är nöjd med de ingivna valen. Varför jag kände att det behövdes här är för att om man t ex ska lägga på poäng på en medlem så om man skriver in fel så har man möjligheten att avbryta om man inte är nöjd med sitt val. Eller om man också kanske väljer ut fel medlem i ett lag.

Detta är något jag skulle kunna implementera för editeringen valet också men i så fall så gör jag det på iteration 3 som kan innehålla lite extra beroende på hur det går med tiden eftersom det skulle kräva en del omstrukturering.

Iteration 3

Detaljerad planering:

- Krav:
  - Analysera användningsfall, aktörer och flöden. (45 min)
  - Planera implementation. (30 min)
- Implementation:
  - Identifiera klasser, metoder och fält. (30 min)
  - Implementera kod. (3 ½ tim)
- Test
  - Identifiera testfall. (30min)
  - Designa tester. (30 min)
  - Implementera tester. (45 min)
  - Exekvering av tester och analysera testdata. (30 min)
- Reflektera
  - Skriva reflektion (45 min)

Steg 1 Krav:

Funktionella mål:

- Ett menyval som låter användaren se alla lag i poängordning, ledaren på toppen och sista laget på botten.
- Applikationen ska se någorlunda grafiskt tilltalande ut, menyer med överskrift mm.

Aktörer:

- Familjer och vänner, använder användningsfallet för att se hur det går för deras nära och kära.
- Klubbar och medlemmar, använder användningsfallet för att se hur de ligger till i ligan.

Flöden:

- En utloggad användare väljer meny valet.
  - Applikationen krashar -> misslyckat försök.
- Användaren får se alla lag i poänglista.
  - Inga lag är registrerade. -> misslyckat försök.
- Lyckat Flöde!

- En inloggad användare väljer meny valet.
  - Applikationen krashar -> misslyckat försök.
- Användaren får se alla lag i poänglista.
  - Inga lag är registrerade. -> misslyckat försök.
- Lyckat flöde.

Steg 2 Implementation:

Klasspecifikationer:

- Fält/ egenskaper:
    - TeamPointsClone, en klon av lag poängs arrayen som används för att ordna poängen i storlekordning.
- Metoder(lägger till i klassen editor):
    - DisplayTeamOrder, visar upp alla lag i poäng ordning för användaren.

(Mycket av kodningen i denna iteration kommer att vara implementation av design och snygga till hörnen.)

Steg 3 Tester:

Testfall:

- (Test 1)Användaren väljer att se listan av lagen i poäng ordning när inga lag är registrerade.
    - Förväntat resultat: användaren meddelas om att inga lag än finns registrerade. *Kan bara testas manuellt.*
- (Test 2)Användaren väljer att se listan av lagen i poäng ordning när inga lag än har fått poäng.
    - Förväntat resultat: Användaren blir meddelad om detta och åker tillbaka till huvudmenyn.
- (Test 3)Alla meny val fungerar som de ska efter ändringen av meny uppsättningen.
    - Förväntat resultat: Alla meny val gör vad det står att de ska göra. *Kan bara testas manuellt.*

Testdata:

Test:	Vad som testas:	Kommentar:	Pass/ Fail:	Datum:
Test 1	Hur det nya meny valet hanterar om det inte finns några lag registrerade.	Måste testas manuellt.	Pass	16/1-2015
Test 2	Hur det nya menyvalet agerar om inga lag än har fått några poäng.	Testar genom att sicka en array med några namn till metoden medans teampoints arrayen inte har några värden.	Fail	16/1-2015
Test 3	Om den nya meny uppsättningen fungerar så som den ska.	Måste testas manuellt.	Pass	16/1-2015

Re:Testdata:

Test 2	Hur det nya menyvalet agerar om inga lag än har fått några poäng.	Testar genom att sicka en array med några namn till metoden medans teampoints arrayen inte har några värden.	Pass	16/1-2015

Reflektion:

Denna iteration framstod mer problem än vad jag hade trott det skulle göra. Var ganska självsäker men hade några små problem när jag gjorde DisplayTeamOrder metoden men hann i slutändan endå med att göra lite design på programmet.

Små problem som uppstod var bl a. att jag glömde att sicka med lag namns arrayen med till metoden, samt så tänkte jag tvät om när jag använde Arr.Sort(), den lägger lägsta högst upp, inte högsta så fick också lägga till en reverse. Jag gjorde också en omdesign på min meny, bytte plats på meny val osv för att få en lite mer ordning på det.

En till sak var att när jag kom på mitt test 2 så visste jag redan att det skulle misslyckas. Men blev chokad för att det misslyckades inte redigt så som jag hade trott det skulle, i vilket fall som helst så löste jag det med att använda en boolean datatyp som kollar om ett lag har blivit givet poäng ännu.

Sista Reflektion

Det jag vill första ta upp på denna reflektion är att jag inte redigt höll till mig om vad jag sa Editor klassen skulle göra. Förutom att sköta edit funktioner som namn byte eller poäng givning så tar den också hand om t ex. visa upp alla lag i en lista och att visa alla lag i poängordning. Detta kunde jag nog ha gjort i en annan klass men eftersom jag hade samlat alla arrayer och data där så det underlättade att göra det där eftersom denna applikation inte är stor. Men om jag skulle ta i tanke att det skulle vara 10000+ rader kod så skulle mitt sätt som jag har nu vara väldigt förvirrande.

Jag tycker också att jag har haft lite svårigheter att komma på bra tester samt att göra upplägg för de jag har, några stycken kunde jag bara testa manuellt och det är nog inte något önskar krav man vill ha för sina tester. Om man har 500+ tester och sen 2 som måste testas manuellt känns lite sådär liksom.

Tiden har också varit lite sådär under denna iteration, jag tror inte att jag kommer gå ihop 32 timmar men om jag skulle försöka trycka in någon mer funktion eller nått i den stilen så skulle jag nog överstiga med tiden ganska mycket. Jag ligger inte så mycket under gränsen så nöjer mig med det jag har.

Implementationerna har jag inte haft någon tids överflöd eftersom jag har lagt en ganska tung mängd av tid på dem i planeringen, vilket i slutändan blev ett bra val eftersom jag hade svag tidspress någon iteration men det var allt.

Att fortsätta så här på varje iteration istället för att bara göra 1 som vi gjort på de förra labbarna har visat sina nya tankesätt man får använda. I framtiden så behöver jag lägga mer tanke på hur de kommande iterationer ska läggas in. Det var nog det största problemet med detta projekt att jag fick byta ut min huvudmeny med en ny uppsättning för att underlätta.