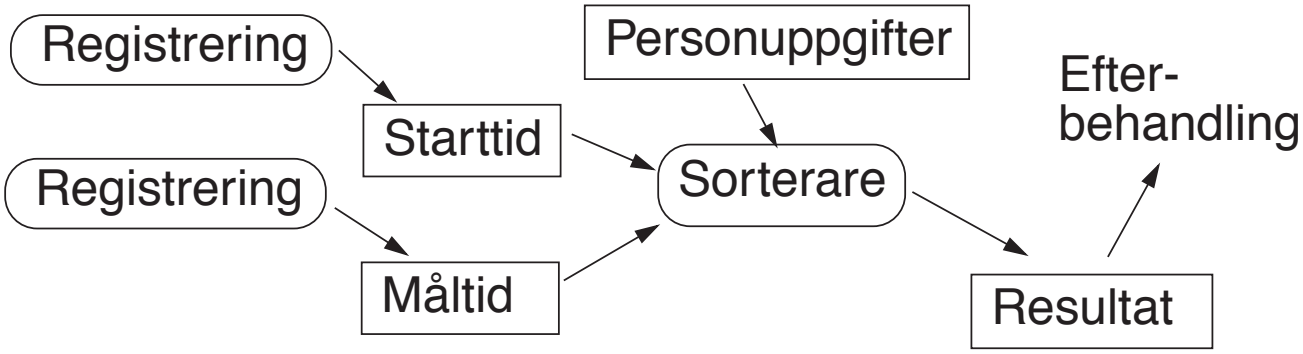
Proj-Iteration1

PVG/Coaching

Boris Magnusson
Datavetenskap LTH

Arkitektur alt. 1



Tre program som kan köras oberoende av varandra. Kommunikation via filer som ev kan transporteras på USB-sticka.

Resultatet kan kunden sedan sortera och formattera på olika sätt för olika behov.

Vi utnyttjar arkitekturpatterns: - Pipes & Filters

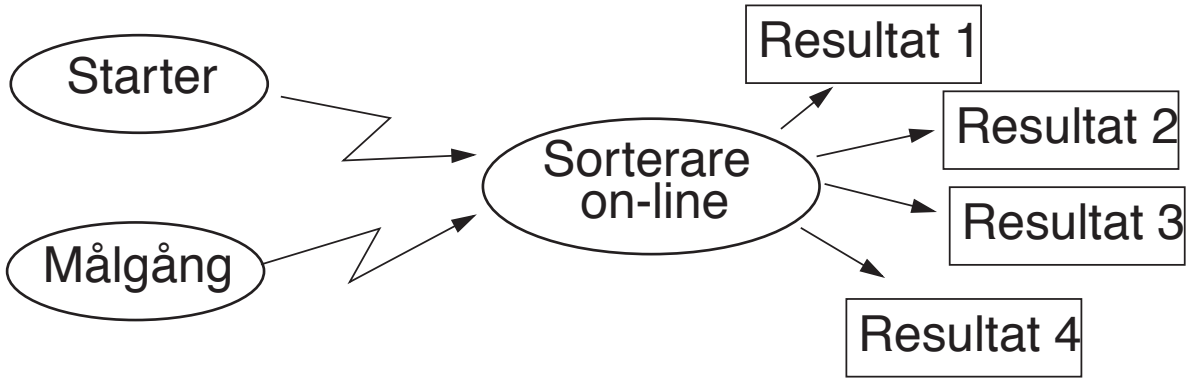
- Model/view i programmet Registrering

Arkitektur alt. 2

Programmen “Starter” och “Målgång” skickar sina registreringar direkt till en Server. Detta gör att man enklare och snabbare kan ta ut mellanresultat.

Arkitekturpatterns: dessutom Client/Server

Arkitektur alt. 3



Programmen “Starter” och “Målgång” skickar sina registreringar direkt till en Server som i alt2. Här gör dock server direkt presentation av resultat i olika format, t ex slut och mellanresultatat, ställning som uppdateras vid varje målgång, information till förare om placering, varvtider etc.

Nollte-iteration

Görs av coacherna. Kan:

Generera start-fil med EN registrering
Generera mål-fil med EN registrering
Generera resultat med EN åkare

Resultatet är ett system som kan exekveras och som innehåller tillräckligt mycket implementation för att flera par skall kunna börja jobba samtidigt. Men inte nödvändigtvis några färdiga stories. Funktionalitet kan vara fejkad. T.ex. påhittade tidpunkter för registrering, påhittade resultat som inte stämmer med start och måltid.

Använd Ark1 och strukturera Registreringsprogrammet enl MVC-arkitekturen.

Format för filer

Kunden vill ha möjligheten att kunna editera start/mål/resultatfiler för hand och att kunna använda det nya systemet tillsammans med tidigare teknik. Därför skall filformat vara enl följande (EXCEL – semikolon för att separera fält):

Start-fil (och Mål-fil) rader med par: StartNr; StartTid

Ex:
```
1; 12.00.00
2; 12.01.00
```

Resultat-fil: Första raden innehåller rubriker för kolumnerna, därefter en rad per deltagare

Ex:
```
StartNr; Totaltid; Starttid; Måltid
1; 1.23.34; 12.00.00; 13.23.34
2; 1.14.16; 12.01.00; 13.15.16
3; 1.03.06; 12.02.00; 13.05.06
```

Acceptanstester

Den mycket ambitiöse och ovanligt datorkunnige kunden har satt samman en svit tester som han delar med sig i elektronisk form. Detta sparar naturligtvis mycket arbete och tid i projektet. Se /usr/local/cs/EDA260/Acceptanstester09

Testerna skall dock inte tas bokstavligt:

- de är gjord för hand och kan innehålla fel
- formateringen är inte tvingade avseende t ex blanktecken och dylikt

Om er utmatning inte blir helt exakt som beskrivet i testerna ta en titt på skillnaderna. Är det bara småsaker, eller fel i testerna som skiljer, så använd er egen utmatning att jämföra med i fortsättningen. Om ni är tveksamma, diskutera skillnaderna med kunden! Det viktiga är att tidigt upptäcka oavsiktliga förändringar i resultatet!

Iteration 1: Stödja maratonlopp

Kunden vill i första iterationen fokusera på att ta fram ett system som stödjer maratonlopp.

Motsvarar följande ursprungliga stories (från F7):

Registrera start för en åkare

Registrera målgång (men bara en sträcka)

Generera resultatlista (men ännu inte klasser, tidsordning)

Användarvänligt

Kunden har formulerat nya, mer precisa stories för detta:

Kunden har itererat fram denna beskrivning, och av denna anledning börjar numreringen på 3 :-)

3: Enkel resultatlista (utan totaltid)

4: Personuppgifter

5: Totaltid

6: Felaktiga registreringar

7: GUI för registreringar

8: Registrering av tider

Story 3: Enkel resultatlista

Kunden vill kunna ta handkonstruerade filer för start och måltider och från dem få en enkel resultatlista där start och måltider för samma förare parats ihop.

Totaltiden behöver inte beräknas utan ersätts tills vidare med “--.--.--”

Exempel på resultatlista:

```
StartNr; Totaltid; Starttid; Måltid
1; --.--.--; 12.00.00; 13.23.34
2; --.--.--; 12.01.00; 13.15.16
3; --.--.--; 12.02.00; 13.06.06
```

Kunden har redan tagit fram input och outputfiler för att storyn skall kunna testas. Se Acceptanstest3.

Kom ihåg:

"Stories are promises for conversation"

Om något är minsta oklart i en story, ta kontakt med kunden och diskutera saken (på planeringsmöte eller långlabb).

Task-indelning för Story 3

Kunden har anlitat en XP-expert som har gett följande förslag till task-indelning för Story 3:

- Task 3.1: Datastruktur för att lagra start och måltider.
- Task 3.2: Läs in start och målfiler och fyll i datastrukturen.
- Task 3.3: Para ihop tider och skriv ut resultatfilen.
- Task 3.4: Anpassa huvudprogrammet så att rätt filer läses in och skrivs ut.
- Task 3.5: Acceptanstesta storyn.

Experten rekommenderar vidare att dessa tasks utförs parallellt av 2-3 par (även om man annars i normalfallet låter ett par ta hand om en hel story):

- Storyn är central, många andra stories bygger på den. Bra att få färdigt den så fort som möjligt.
- Och det kan annars vara svårt att sysselsätta alla par i början.

Task 3.1 Datastruktur

Implementera en datastruktur för att lagra alla start och måltider.

Expertens tips:

- Nollte iterationen av systemet innehåller antagligen en nollte iteration av datastrukturen, med ett enkelt API, en tom eller förenklad implementation, och kanske något enhetstest. Börja där och arbeta vidare med TDD.
- Hur skall datastrukturen användas? Prata med paren som läser in inputfiler och skriver ut resultatfiler.
- Se Enhetstest1 för exempel på hur man kan enhetstesta datastrukturen.
- Vilka enhetstester behövs? Vilka olika kombinationer av start och måltider är relevanta?

Task 3.2 Läs in filer

Se till att startfiler och målfiler kan läsas in och att datastrukturen fylls i på rätt sätt.

Expertens tips:

- Enhetstesta fil-läsningen genom att läsa in filer med olika innehåll och kolla att datastrukturen anropas på rätt sätt och/eller får rätt innehåll.
- Titta på Enhetstest1 för att se exempel på hur datastrukturens innehåll kan testas. Och prata med paret som implementerar datastrukturen.

Task 3.3 Skriv ut på resultatfil

Skriv ut innehållet i datastrukturen på en resultatfil.

Expertens tips:

- Enhetstesta fil-skrivningen genom att skriva ut olika instanser av datastrukturen med olika innehåll. Testa att output-filen har förväntat innehåll.
- Behöver datastrukturens API vidareutvecklas? Diskutera med paret som implementerar datastrukturen så att det är klart vem som implementerar vad.

Task 3.4 Anpassa huvudprogrammet

Se till att huvudprogrammet läser in och skriver ut lämpliga filer.

Expertens tips:

- Fundera på hur användaren skall köra programmet. Vad är enkelt och praktiskt i denna första iteration av systemet? Förbestämda filnamn? Argument på kommandoraden?
- Task 3.5 (acceptanstestning) är mycket relaterad.

Task 3.5 Acceptanstesta storyn

Systemet skall klara Acceptanstest3.

Expertens tips:

- Denna task kan inte fullföljas förrän alla andra tasks är färdiga. Men man kan börja innan allt är klart.
- Fundera på hur acceptanstesten skall utföras. Hur skall användaren köra systemet för att utföra acceptanstesten? Hur ser man enkelt om acceptanstesten går igenom eller inte?
- Om man väntar på Task 3.2 (läsa in filer) kan man fejka huvudprogrammet så länge: Man kan anropa datastrukturens API direkt i huvudprogrammet (istället för att läsa in start och målfiler). Blir resultatfilen rätt?

Story 4: Personuppgifter

Kunden har en fil med personuppgifter enl följande format. Första raden innehåller Kolumnnamn, följande rader innehåller uppgifter med en rad per deltagare.

Ex:
```
StartNr; Namn
1; Anders Asson
2; Berit Bsson
```

Systemet skall utnyttja denna fil för att skriva ut namn i resultatfilen.

Ex:
```
StartNr; Namn; Totaltid; Starttid; Måltid
1; Anders Asson; --.--.--; 12.00.00; 13.23.34
```

Kunden har förberett acceptanstest för detta: Acceptanstest 3_4

Taskindelning för Story 4

Experten föreslår följande tasks:

- Task 4.1 Designa datastruktur för att lagra personuppgifter
- Task 4.2 Läs in och lagra raden med kolumnnamn
- Task 4.3 Läs in personuppgifterna
- Task 4.4 Ändra så att resultatfilen innehåller Namn också
- Task 4.5 Acceptanstesta (Acceptanstest 3_4).

Story 5: Totaltid

Beräkna och skriv ut Totaltiden på filen

Ex:
```
StartNr; Namn; Totaltid; Starttid; Måltid
1; Anders Asson; 1.23.34; 12.00.00; 13.23.34
2; Bengt Bsson; 1.14.16; 12.01.00; 13.15.16
3; Chris Csson; 1.03.06; 12.02.00; 13.05.06;
```

Använd Acceptanstest5

Alt Acceptanstest3_5 om Story 4 ej är färdig:

```
1; 1.23.34; 12.00.00; 13.23.34
...
```

När Story 4 & 5 är färdiga används bara Acceptanstest5.
Acceptanstest 3, 3_4 och 3_5 är då överspelade.

Story 6: Felaktiga registreringar

Flera olika slags felregistreringar och inkonsistenser kan uppstå vid en tävling. T.ex. kan fel förarnummer matas in så att en förare får för få tider och en annan för många. Genom att upptäcka inkonsistenta data och markera i resultatfilen kan tävlingsledningen enklare hitta felen.

Markera i resultatfilen på följande sätt:

- Task 6.1: Ingen måltid - skriv “Slut?” istället
- Task 6.2: Ingen starttid - skriv “Start?” istället
- Task 6.3: Flera måltider - skriv en extra kolumn med texten: “Flera måltider?” följt av den/de extra måltiderna
- Task 6.4: Flera starttider - pss “Flera starttider?”
- Task 6.5: Upptäck om Totaltiden är mindre än 00.15.00 skriv då “Omöjlig Totaltid?” efter Måltiden.

Task 6 Exempel på resultat

Ex:

```
StartNr; Namn; Totaltid; Starttid; Måltid
1; Anders Asson; --.--.--; 12.00.00; Slut?
2; Bengt Bsson; --.--.--; Start?; 13.15.16
3; C C; 1.03.06; 12.02.00; 13.05.06; Flera måltider? 13.07.08
4; D D; 1.09.07; 12.03.00; 13.12.07; Flera starttider? 12.12.00
5; E E; 0.13.07; 12.03.00; 12.16.07; Omöjlig Totaltid?
```

- Task 6.6: Acceptanstest 6

Story 7: GUI för registering

Systemet skall ha ett interaktivt program för att registrera start och måltider. Det grafiska användargränssnittet för detta program skall vara enkelt och lätt att använda. Det skall finnas:

- Ett fält där Startnummer kan skrivas in
- Knapp för att utföra registrering
- Fält för att visa de senaste registreringarna

Acceptanstest: låt Kunden titta på resultatet

Registreringsprogrammen kommer att köras på bärbara datorerer PC/Mac. Av portabilitetshänsyn använder vi AWT eller Swing för gränssnittet.

Story 8: Registrering av tider

Registreringsprogrammet skall producera en fil med start-nummer och klocktider - en rad för varje registrering.

Acceptanstest görs interaktivt: Kör programmet och gör några registeringar som du själv antecknar. Kontrollera att programmets resultat stämmer.

Experten föreslår följande tasks:

- Task 8.1 För varje registrering, skriv ut Startnummer och en fejkad tid: “12.00.00” på utdatafilen.
- Task 8.2 Läs av klockan och skriv ut verklig tid vid registrering
- Task 8.3 Uppdatera fältet med de senaste registreringarna. Se till att programmet Först skriver på fil och Sedan uppdaterar användar-gränssnittet för att minska risken för förlorad data vid ev haveri.
- Task 8.4 Acceptanstesta.

Iteration 2

Inför Release 1 bör följande stories också vara med:

Hantera Namnfiler som innehåller flera kolumner

Registrera starttid vid masstart

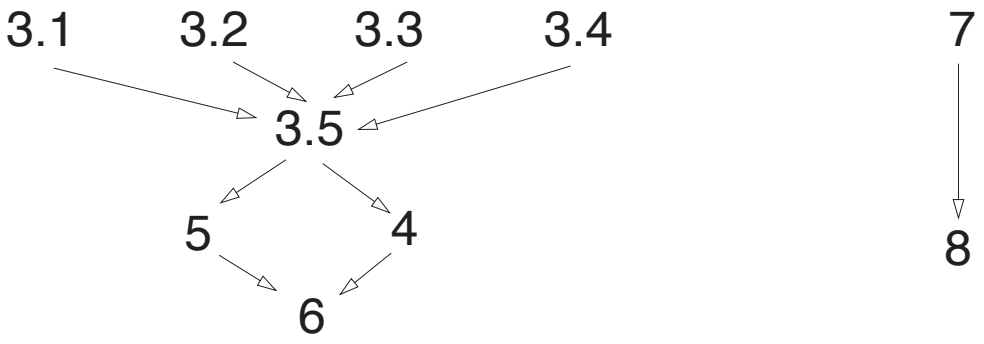
Användarvänligt: Användarmanual och installations-beskrivning

Förbereda Release

Tips från XP-experten

De presenterade storysarna kan implementeras ganska par-allellt. Det är bra eftersom kodbasen från iteration noll är liten och det annars kan vara ett problem att börja jobba flera samtidigt.

Exempel på i vilken ordning man kan ta dessa stories:



Tracker

Story/Task	Vem	Tid?	Tid!
3: Enkel resultatlista			
3.1: Datastrukturer			
3.2: Läs in filer			
3.3: Skriv ut			
3.4: Anpassa huvudprogram			
3.5: Acceptanstesta			
4: Personuppgifter			
5: Totaltid			
6: Felaktiga registreringar			
7: GUI för registreringar			
8: Registrering av tider			