62531, 62532, 02312

Udviklingsmetoder til IT-systemer, Versionsstyring og Testmetoder, og Indledende Programmering

---

CDIO 1

01. Oktober 2021

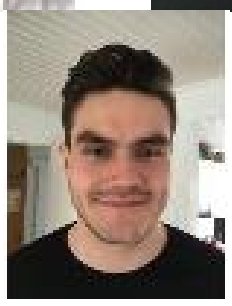
Sebastian N. B. E.

Christian H.

Valdemar N.

Sebastian G.

Lasse. J



Resumé

- Vi skal, som udviklere for spillefirmaet IOOuterActive, analysere, designe, implementere og teste et terningespil, som opfylder kundens krav og vision.

Timeregnskab

- Sebastian G: 12 timer
- Sebastian E: 8 timer
- Christian: 7 timer
- Valdemar 8 timer
- Lasse 3 timer

Indholdsfortegnelse

Indledning

- I denne rapport vil vi komme ind på udviklingen af et terningspil og vores analyse deraf. Det indeholder på risikoanalyse, overvejelser, modeller og mere.

Projektplanlægning

- Udarbejd et terningespil efter kundens vision som skal være klar d. 01/10/2021. Udvikling af projektet foregår men løbende opdateringer på Github og en tidlig star

Krav

- Kunden vil se resultatet af terningekastet på under 333MS.
- Kunden skal have mulighed for at redigere og se inspicere udviklingen af koden.
- Der er blevet bedt om test, så mængden af de mulige udfald af antal øjne er så ligeligt fordelt som muligt, og at antallet af "par" slået under testen blev optalt.

Analyse

Risikoanalyse:

Identificér, estimér og prioritér risici fra projektet;

- Der er en fra gruppen som ikke kan deltage i projektet
- Vores færdige produkt virker ikke på DTU's computere i databar på grund af manglende software.
- Mistolkning af kundens ønsker. Derved lave et fejlfyldt produkt.
- Undervurdering af projektets omfang
- ikke aflevere det færdige produkt i den givne tidsramme.
- Tab af data pga tekniske fejl
- Fejlprioritering af krav

Videre beskrivelse af risici

1. Dette kunne ske ved for eksempel ved en akut familiesituation, der kræver stort tidsforbrug. Dette kan være et problem, da vi derved får færre hænder til at lave vores projekt. Vi har vurderet at der er lav sandsynlighed for at der her sker, da disse ting jo ikke sker super ofte. Desuden er det ikke det største problem da vi stadig er fire, og personen der er i en nødsituation måske kan hjælpe lidt alligevel.
2. Dette kan ske ved at kommandoprompten ikke kan læse vores program ordentligt. Dette kan ske ved at vi aflevere filer på en forkert måde, eller der sker når vi overfører noget data. Sandsynligheden for dette er ikke særlig stor, da hvis vi kan compile filen på en af vores windows computere. burde en anden computer med samme compiler heller ikke have noget problem.
3. Det kan ske hvis vi ikke får spurgt kunden i tide omkring hvad der præcis menes i funktionskravene. Og vi derved aflevere et program der ikke gør som kunden har bedt om. Der er en mellem sandsynlighed for at dette sker på et højt nok niveau til at blive et problem. Dette vil have en lidt stor impact da kunden vil være utilfreds.
4. Denne undervurdering kan ske, ved at kigge på softwaren og vurdere opgaven som nem, selvom der er meget mere end bare kode. Dette kan give et stort tidspres få dage før afleveringen, hvilke kan resultere i en masse små dumme fejl. Vi vurdere denne til at have en lav risiko, da vi internt har talt om det i gruppen. Dog kan det ske og hvis det sker kan det have meget store konsekvenser.
5. Her vurdere vi en lavere risiko, da det er en af de ting vi er mest observante omkring. Da DTU's regler ikke var venlige over for personer der afleverer for sent. Dette ser vi som en lavere risiko da vi vil aflevere noget. Men konsekvensen er den største, da vores projekt så vil være en fiasko.
6. Dette kunne ske hvis man ikke når at committe noget man har arbejdet på i flere

timer, og mister det man har lavet. Risikoen for dette er, svær at vurdere da det er meget individuelt hvor grundig man er. Vi har dog valgt at give den en lille sandsynlighed, da man også skal glemme og gemme og noget galt skal ske med ens computer. Dog er der ikke stor udfordring ved det da alle skal comite noget, derfor bliver projektet lavet i bidder

7. Dette kan ske hvis vi bruger for meget tid på mindre vigtige ting, og vi derved ikke når det centrale dele i projektet. Sandsynligheden for dette er ikke super stor, da der kun er en use case i vores projekt, og vi derfor altid vil arbejde på den vigtigste use case. Derfor er det svært at arbejde med noget der er helt forkert. Og hvis der arbejdes på noget forkert, ender det med at være tidspress der bliver et problem. Derfor vurdere vi dette til at have lav risiko.

- Sortér risici

Sorteret risici-scenarier	Sandsynlighed	Impact	P*I Matrix	
Mandefald grundet eksterne begivenheder.	1	1	non-action	1
Fejl i kodningen som ikke kan ændres inden deadline.	1	2	non-action	2
undervurdering af projektets omfang	5	1	monitor	5
Ændring i ledelse eller en ledelse der ombestemmer sig.	2	3	monitor	6
Uforventede store krav sent i processen.	2	4	action	8
Mistolkning af kundens ønsker	3	3	action	9

- Afgør et cut off - hvilke risici er I klar til at acceptere? Med hvilken begrundelse?
    - Vi har lavet et (ovenstående) sheet hvori vi ud fra P*I Matrixen har fået et form op for hvor høj risici vi kan vurdere hvert scenarie ud fra. Da næsten alle af vores scenarier/cases er i de grønne felter og da de hverken har høj probability eller impact kan vi acceptere disse.
      Det eneste der har en "høj" probability og impact, er ved scenariet: "Mistolkning af kundens ønsker. Derved lave et fejlfyldt produkt"
        - Dette er vi som gruppe klar til at acceptere, da det er at vi inden aflevering har fået en underskrift af "Kunden" hvor det tydeligt fremgår præcis hvad der skal laves og hvad "Kunden" kan forvente. Hvis ikke det lever fuldt op til forventning og som beskrevet er det vores fejl og vi vil derfor opdatere og "rette" de manglende funktioner/fejl i produktet.
- Hvilke risici bør/kan forebygges
    - Da alle vores risiko-scenarier ligger indenfor det "acceptable" felt under vores risk-scale i PI-matrixen burde de alle kunne forebygget.
      Dette kan vi også se hvis vi kigger på probability raten for vores risiko-scenarier, da probability er lav er det derfor også nemmere at forebygge. Havde probability været meget høj er det en indikator for at muligheden for dette risikoscenarie er højt og dette er jo så også en indikator for at dette ville kunne forebygges og man kan derfor tænke på flere løsninger til at ordne dette og derudover sætte mere monitoring på scenariet.

- Svar kort på: Hvornår er det relevant at
    a. Overføre en risiko.

Hvis man har nogle ressourcer der er gode til at håndtere disse typer af risiko, kan det være en god ide at outsource disse risiko.

    b. Acceptere en risiko.

Hvis risikoen er acceptabel og konsekvensen er lille.

    c. Afbøde en risiko.

Hvis konsekvensen kan reduceres til en acceptabel grad. Dette er relevant hvis den hverken kan elimineres eller accepteres. Desuden skal det være muligt at overvåge den.

    d. Eliminere en risiko.

Hvis det giver mere mening at eliminerer en risiko end at leve med den kan det være en god ide at eliminere den hvis muligt. Generelt er det altid en god ide at eliminere risiko det kan ofte bare ikke lade sig gøre.

Design

Use case:

For at finde ud af, hvor vi skal starte med dette program vil vi lave en use case, der beskriver programmets forløb. I dette program er der kun en use case nemlig: at spille terningespillet. Da der kun er en use case er dette selvfølgelig også main use case. Da der er tale om et repetitivt spil i ture vil use casen beskrive hvordan en tur foregår, hvordan spillet starter og hvordan der findes en vinder. Der er kun en aktør i denne use case.

Use case navn: Spil terningespillet

Scope: Terningspillet

Level: User goal

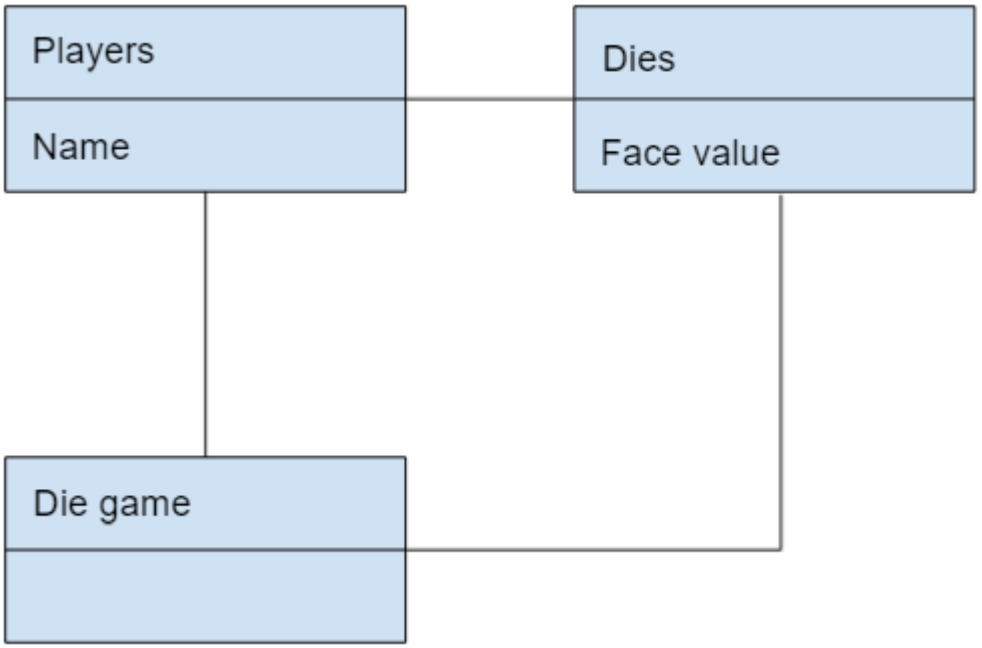
Primær aktør: Spilleren

Interessenten i denne use case er spilleren. Spilleren er interesseret i, at programmet kører uden forsinkelser. Spilleren er også interesseret i at han kan forstå spillet uden besvær.

1. Spillet starter.
2. Begge spillere har 0 point når spiller starter.
3. Spiller 1 slår med to terninger.
4. Spiller får point efter hvilken sum han slår.
5. Hvis spiller slår to ens får han lov til at slå igen og går til step 2.
6. Hvis spiller slår to ettere mister han alle sine point.
7. Hvis spiller slår to seksere to ture i træk vinder spilleren.
8. Efter spilleren har opnået 40 point skal spilleren slå to ens for at vinde spillet.
9. Spillet er nu slut

Domænemodel:

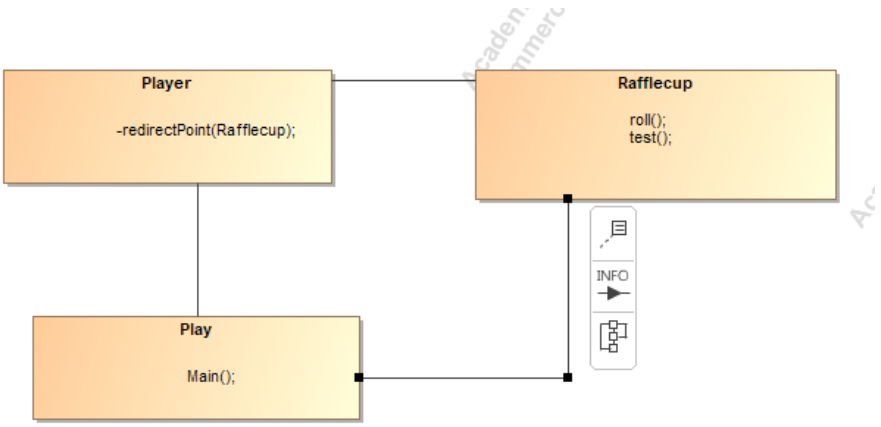
Vores domænemodel er blandt andet med til at give muligheden for, at give en ikke-IT-forstående kunde, et et overblik over hvad programmet indeholder.



Desuden er denne domænemodel et godt udgangspunkt, når man skal til at lave en klassemodel.

Klassemodel:

Vi har efter denne model lavet en mere kodenær model nemlig et klassediagram. Dette diagram ses nedenfor. I dette diagram ses tre klasser, Player, RaffleCup, play. hver klasse har egentlig en vigtig metode der får spillet til at fungere. Play klassen har en main metode der sætter spillet i gang. Rafflecup har en metode der slår med to terninger. Player har en metode der giver spilleren point. Desuden tænker vi det kunne være en god ide, at have en testmetode i klassen RaffleCup. Da det er et af de krav vi har fået. Når man kommer videre til implementering kommer der nok flere metoder, men disse metoder er ikke de centrale men blot metoder til at understøtte og hjælpe. At skrive alle disse små metoder fra start af ville gøre klassediagrammet rodet uden noge grund. Desuden ville det være dumt at fastsætte alle metoder, da det er nemmere at gøre når man sidder og koder. Herunder ses klassediagrammet.



Implementering

Til implementering af denne kode, tog vi inspiration fra vores domænemodel. Vi har altså tre forskellige klasser vi selv har skrevet til dette program. Vi har dog også brugt hjælp fra nogle af java's egne biblioteker. Dette ses for eksempel ved metoden Math.random(); som vi har brugt i vores kode til at få terninger der er tilfældige. Den vigtigste klasse i dette program er klassen RaffleCup. Denne klasse har fire klasse variable 3 simple og et objekt. 2 terninger summen af terningerne og objektet GUI. Som vi er blevet bedt om at bruge Denne klasse har metoden roll(); denne metode bruger Math.random(); til at få to tilfældige tal mellem et og seks. roll(); metoden regner også summen af de to terninger, og opdatere GUI'en med metoden setdice();. I denne klasse findes også metoden print(); denne metode printer terningerne individuelle værdi ud og summen. Denne metode bruges kun i forbindelse med roll(); derfor er denne metode private da den kun skal bruges i klassen selv. Dette undgår at man gentager sig selv. Desuden har denne klasse en anden vigtig metode der ikke ses når man spiller spillet.

Den anden klasse Player, bruges primært til at holde styr på og uddeler point til de forskellige spillere. Klassen to variable er spillerens navn og RaffleCup. Denne klasse kommer med en konstruktor, da en spiller skal have et navn og en spiller skal have et rafflebøger. Denne klasse har en public metode der hedder redirectPoint(); Denne metode vurderer om man har slået 2 ens eller ej. Derefter kalder den, den korrekte metode. Der er her tale om to private metoder. De to metoder er awardpoints(); og sameFaces(); disse to metoder gør hvad der er i de funktionelle krav. Desuden er der også metoden win(); der holder styr på om spilleren har vundet spillet og derved stopper det. Så er der metoden print point der bare printer et statement, dette er blevet lavet til en metode da det undgår gentagelse. Der er også en reroll(); metode. Denne metode bruges til at slå en ekstra gang med terningen når der er nødvendigt.

Det ses at vi har lavet en ændring i implementeringen i forhold til vores klassemodel da test ligger i en anden pakke. I vores anden pakke med test har vi metode test();Denne metode er metoden test(); denne metode brugte vi til at lave 1000 slag med roll(); metoden. Vores test er dokumenteret og bliver beskrevet senere i dokumentet.

Test

- Vi tester forekomsten af de mulige udfald i forhold til antallet øjne og specielle scenarier, for at sikre at det bliver så tilfældigt som muligt, for at illustrere et ægte terningspil. Vi har gennemført 1000 terningekast og opstillet et skema for forekomsten af hver kombination, sammenlignet med den statistiske sandsynlighed, for at vurdere præcisionen af programmet. Det statistiske udfald på 1000 terningekast med to terninger, ligner en binomialfordeling. Vi har også fået koden til at optælle alle de par, som bliver slået i vores test. Det viser nemlig, om der er nogle par, der forekommer oftere end andre. Det kan ses at vores kode er relativ tilfældig når det gælder antallet af par, hvilket understøtter, at vores kode den virker. Dokumentation for dette kan ses under bilag.

Konklusion

Vi kan på baggrund af de ovennævnte krav konkludere at vores terningespil fungerer som ønsket. Dette kan vi se ud fra de forskellige test-forsøg hvor terningspillet giver acceptable udfald i forhold til den statistiske sandsynlighed, hvilket giver siden vi testede 1000 terningekast. Jo flere terningekast jo mere præcis bliver testen. Hvis vi havde øget antallet af kast, ville vores testresultater højst sandsynligt have ændret sig en lille smule i retningen af den statistiske sandsynlighed. Vores antal af par blev også fordelt nogenlunde ligeligt ud over alle 6 muligheder, hvilket vi har fundet acceptabelt.

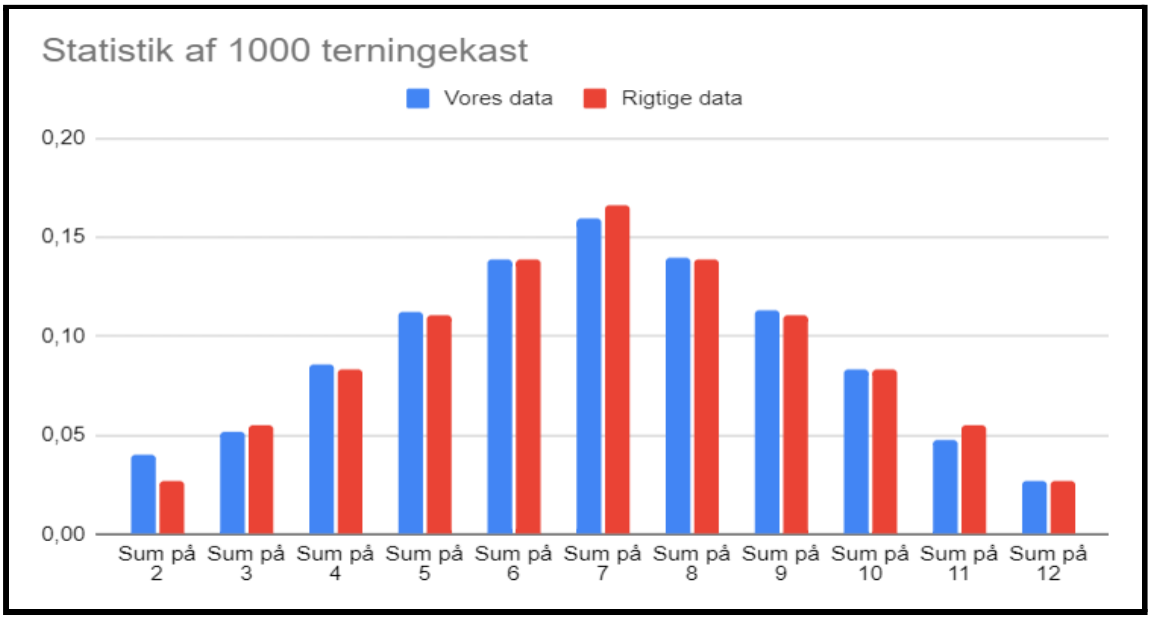
Derudover har vi også haft et dokumenteret eksempel på hvad det er at kunden har ønsket ud fra dette terningespil som vi så herefter har analyseret, designet og implementeret spillet efter. Vi har altså opfyldt de ønsker som kunden havde ved projektets start og derfor lavet et succesfuldt.

Til at opbakke denne konklusion har vi også opbygget vores kode efter diverse modeller, hvilket har været med til at give et overblik og formindske antallet af problematikker vi imødekom undervejs i projektet.

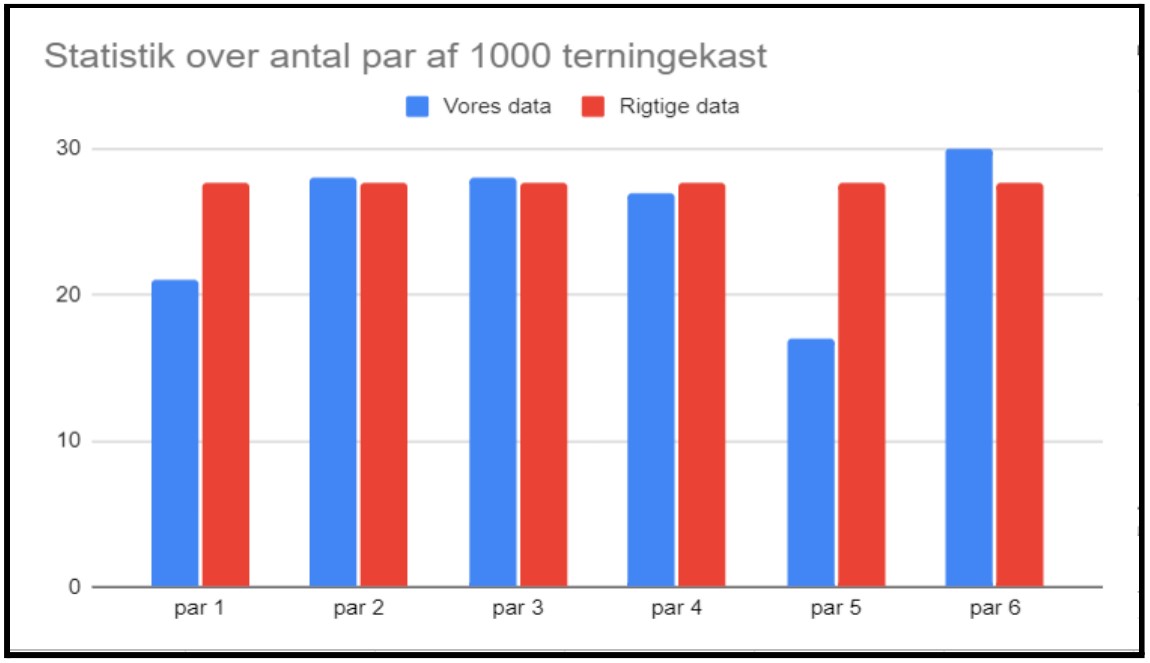
Litteraturliste

Larman Craig, Applying UML and Patterns, John Wait

Bilag:



Søjlediagram over 1000 terningekast



Søjlediagram over antal par over 1000 terningekast

Link til git repositroy: https://github.com/christianhyltoft/02_Terning