1DV404 – Iterativ Mjukvaruutveckling

Laboration 1

Av: Mattias Wikström – mw222rs – grupp E2

Tidslogg:

Tid förbrukad	Dag	Vad?
1,5 h	2014-11-17	Skapade detta dokument och läste igenom uppgiften.
5 h	2014-11-22	Uppgift 1a – 1b
5 h	2014-11-23	Uppgift 1c
8h	2014-11-23	Uppgift 2, 3, 4

Uppgift 1a – RaknaA

1. Jag planerar att denna uppgift kommer att ta mindre än en timme för mig att genomföra, då jag i stort sett har gjort en likadan uppgift i JavaScript-kursen jag läser parallellt med denna. På grund av denna redan gjorda uppgift blir det heller inte särskilt många moment att gå igenom här, jag tänker helt enkelt gå tillbaka till den kod jag skrev i den tidigare uppgiften, kopiera de delar jag kan återanvända här och sedan lägga till de rader kod jag behöver lägga till för att räkna antal A och a i strängen.

   Detta har jag enkelt tänkt lösa genom att först omvandla strängen till en array med .split-metoden och stegar sedan igenom arrayen med .map och räknar då enkelt hur många A'n och a'n arrayen innehåller.

2. **Genomförande:**

   1. Skapade repositorie på github och kopplade mot cloud9 – 15 min.
   2. Skapade ett antal filer för denna uppgift och b- och c-uppgifterna – 10 min.
   3. Kopierade de rader kod jag hade användning för från tidigare uppgift och ändrade funktionen så att den istället för att manipulera en sträng (vilket var uppgiften i den tidigare uppgiften) till att helt enkelt räkna antal stora och små A i en sträng – 15 min.
   4. Provkörde programmet och allt verkar funka som det ska – 5min.

3. Jag vet inte riktigt hur mycket jag kan reflektera kring denna planering och mitt genomförande då jag egentligen visste exakt hur jag skulle lösa uppgiften direkt jag läste vad det var jag skulle göra – således behövdes alltså ingen planering. Själva genomförandet gick även det utan några större problem, jag skrev de rader kod jag behövde och det fungerade som väntat.

Uppgift 1b – RaknaSiffror

1. Jag planerar att även denna uppgift kommer att ta mindre än en timme att genomföra då jag i stort sett kan använda exakt samma kod som från a-uppgiften men i .map-metoden istället för att kolla om varje bit av en array är ett stort eller litet a kolla om det är en nolla, en jämn siffra eller en udda siffra.

   De jag moment jag måste genomföra är endast att byta ut några rader kod i .map-metoden, som i a-uppgiften med hjälp av ett par if-satser räknade antal a och A, för att istället få den att gå igenom arrayen (som jag skapar från den inmatade strängen med .split-metoden) och räkna siffror istället med hjälp av ett par lite annorlunda if-satser.

2. **Genomförande:**

   1. Gjorde en kopia på koden från a-uppgiften och ändrade i den funktion som där gick igenom arrayen och räknade a och A så att den istället gick igenom arrayen och med hjälp av tre if-satser räknade hur många nollor, jämna och udda siffror det inmatade heltalet innehåller. Hade några små problem med typomvandling som gjorde att jag missade nollorna, något som jag kom på efter lite testande. – 40 min.

   2. Provkörde och allt verkade funka. – 5min.

3. Likt a-uppgiften fanns det inte så mycket att planera för denna uppgift då jag i stort sett förstod hur jag skulle gå till väga för att lösa uppgiften redan då jag läste vad det var jag skulle skapa för program. Jag gjorde helt enkelt som jag planerat och ändrade i .map-metoden för att få det resultat jag eftersträvade.

Uppgift 1c – NastStorsta

1. Planeringen för denna uppgift är lite svårare, då jag inte riktigt vet hur jag skall lösa denna uppgift. Jag hade enkelt kunnat lösa den om det inte hade varit för notisen i uppgifts-förklaringen om att man ej får använda arrayer eller liknande. Hade jag fått använda arrayer hade jag ju bara använt .sort-metoden på arrayen och fått den fint sorterad efter storlek. Nu antar jag att jag måste sortera varje inmatat nummer. Jag planerar att det här kommer att ta ganska lång tid då jag inte riktigt vet vad jag skall göra och inte vet vad jag skall kolla upp innan, det är nog bara att sätta sig ner och försöka.

2. **Genomförande**

   1. Börjar med att lösa problemet med en array, sedan läser jag att man inte får använda arrayer i uppgiftsförklaringen. Börjar om från början. 30 min

   2. Gör om funktionen så att varje nummer som läses in sparas i en variabel som sedan jämförs med en biggestNumber-variabel, är det inmatade numret större så får biggestNumber värdet av det inmatade värdet. Hur man håller reda på det näst största numret är helt enkelt att om det inmatade numret är större än

biggestNumber så tar helt enkelt secondBiggestNumber värdet av biggestNumber och biggestNumber får det inmatade värdet istället. Det är dock inte så enkelt, för ett nummer kan ju vara mindre än det största men ändå större än det tidigare sparade näst största, så en ELSE IF-sats läggs till där vi undersöker om det imatade värdet är större än det tidigare sparade secondBiggestNumber. – 3,5 h

3. Problem jag kom i kontakt med handlar helt enkelt om att jag inte fick det att fungera som det skall om man bara matar in negativa siffror, något jag fick lösa med att instansiera biggestNumber- och secondBiggestNumber-variablerna med värdet av NEGATIVE_INFINITY, för om man instansierar utan värde blir det bara undefined av det hela, och om man – som jag gjorde i början – instansierade till 0 så kommer det inte att funka att testa storlek mot negativa tal. Lite halvtrassligt som JavaScript ofta verkar vara. – 1 h

3. Att det tog längre tid än väntat att lösa den här uppgiften beror inte så mycket på min bristande planering, även om jag hade kunnat spara lite tid genom att läsa igenom uppgiften bättre och inte lagt ner onödig tid på att lösa problemet med arrayer först, utan mera i att jag är väldigt ny med JavaScript och inte riktigt van vid den milda galenskap som fria typer är. Jag vet dock inte om det är något som jag hade kunnat lösa genom bättre planering eller förundersökning inför denna uppgift, det känns mera som något som jag lärde mig genom lösandet av uppgiften och de problem jag stötte på.

   Visserligen hade jag väl kunnat läsa på mera allmänt om JavaScript inför som en del av planeringsarbetet, men jag kände inte riktigt att det skulle varit möjligt att få det mera tidseffektivt i och med att jag inte visste vad jag skulle göra för fel innan jag så att säga gjorde de. Jag hade inte tänkt på att jag bara kunde börja med att instansiera variablerna till NEGATIVE_INFINITY för att vara säker på att de inmatade talen alltid blev större i de större än- mindre än-undersökningarna jag genomför (och på så sig försäkra mig om att även negativa tal hänger med i beräkningarna). Hade jag satt mig ner och bara bläddrat i JavaScript-bibeln hade det nog bara tagit ännu längre tid. Även om det kanske inte var den mest effektivt lösta uppgiften i historien så tror jag att det var mer eller mindre den snabbaste lösningen jag kunde genomföra.

Uppgift 2 – Förändring och förbättring

2a. Jag vet inte riktigt hur strategier spelade in i planeringen av dessa programmeringsuppgifter då de var så pass simpla att jag kunde lösa med en sorts råstyrka, slå sönder dörren istället för att leta reda på nyckeln-lösning – det vill säga att jag bara kunde sätta mig ner och försöka och försöka tills jag löste det hela istället och ändå göra det på samma tid som om jag först hade gjort en rad för rad-planering. Divide and conquer känns liksom inte så relevant som strategi när koden som löser problemet knappt är 10 rader lång.

2b. Det jag tror att man kan göra för att minska konsekvenserna av eventuella oplanerade fel eller andra saker är nog dels att dela upp problemet i mindre delproblem och lösa de eftersom, men även att planera in lite buffert-tid för att kunna klara av några smällar utan att planeringen helt fallerar. Jag vet inte om jag kan ge så många konkreta exempel baserat på uppgifterna hittills i denna uppgift då de som sagt varit lite för simpla för att kunna vinna

på att delas upp till mindre problem som sedan löses för sig innan man går på alltsammans, men visst hade jag nog vunnit på att inte vara så sent ute med själva laborationen i sig.

2c. Jag tror att en förbättring jag skulle kunna implementera i min planering är att i större grad planera vad jag skall göra i bestämda del-steg. Inte bara skriva ”jag skall lösa detta” och sedan göra det, utan planera i lite mera detalj som ”jag planerar att göra X och sedan Y för att gå vidare till…” och sedan ge en tidsram för detta. Kanske är detta svårt att använda i praktiken på de uppgifter som finns i denna laboration då det inte behövs så många steg för att lösa dem, men i kommande större uppgifter tror jag att det kan finnas fördelar med det. Det är dock farligt att fastna för länge i planeringen av ett större projekt istället för att börja med själva uppgiften.

En andra förbättring jag skulle kunna använda i min planering är nog att i största möjliga grad gå igenom uppgiftsförklaringen innan man börjar med genomförandet så att man är hundra procent säker på att man inte gör något, till exempel använder arrayer, när det i förklaringen står att det inte är tillåtet i uppgiften. Det är bara dumt tidsslöseri att sitta och jobba med något som man sedan bara måste slänga i papperskorgen på grund av att man läst slarvigt. Det man inte har i huvudet får man ha i benen, brukar man kanske säga? I detta fall får nog benen bytas ut mot fingrarna som springer över tangentbordet.

Uppgift 3a – Palindrom

1. Jag planerar att denna uppgift kommer att ta ungefär 45 minuter att genomföra. Detta tack vare att jag enkelt kan använda JavaScripts inbyggda array-metoder för att vända på och sedan göra om arrayen till en sträng igen för att sist men inte minst jämföra med den oförändrade inmatade strängen. Är de exakt likadana så är det en palindrom, är de olika är det inte en palindrom.
2. **Genomförande:**
    1. Tar den inlästa strängen och lägger till .split(””).reverse().join(””); för att väldigt effektivt på bara en rad först splitta strängen till en array, vända på ordningen i arrayen och sedan sedan sätta samman arrayen till en sträng igen. - 15 min
    2. På den andra raden i funktionen skriver jag helt enkelt `return (reversed === str) ? str + " är en palindrom" : str + " är inte en palindrom";` vilket då returnerar den inmatade strängen plus *”är en palindrom”* om den inmatade strängen är exakt likadan som den omvända varianten eller *”är inte en palindrom”* om den inte är det. 20 min.
3. Precis som uppgifterna 1a-c så finns det här inte särskilt mycket att reflektera över, då uppgiften löstes så relativt enkelt. Vissa kanske skulle tycka att min rätt så minimalistiska kodstil är svårläst - som att jag valt att splitta, vända på och sedan sätta ihop den imatade strängen på bara en rad istället för att dela upp det över tre - men jag tycker att när man jobbar med inbyggda metoder som har så förklarande namn som split, reverse och join så borde det inte vara några problem.

   Även JavaScripts conditional operator, som jag använder i min return-rad, kan kanske ses som svårläst men jag tror nog att det är en fråga om kodstil och smak. Jag tycker

personligen att den kompakta effektivitet man får av att lösa allt detta på en enda rad kod övervinner det eventuella problemet att det kanske är svårare att förstå vid första anblick jämfört med till exempel en if-sats hade varit.

Uppgift 3b – Fraction

1. Denna uppgift har jag lite problem med att planera då jag inte riktigt tror att den är skriven med JavaScript (som jag ju programmerat hela laborationen i) i åtanke, utan kanske skriven med mera klassiskt klass-baserade objektorienterade programmeringsspråk i åtanke (tex Java och C#).

   Jag tror dock att det skall gå att lösa på ett någorlunda acceptabelt sätt med JavaScript, även om det förmodligen kommer att medföra lite mera ”meckande” än det skulle ha gjort om jag skrivit detta i C# istället. Å andra sidan är väl just detta ”meckande” ganska så signifikativt för, och i min mening en stor del av charmen med, JavaScript. Det finns ju inte riktigt ett lika bestämt ”rätt sätt” att koda något i det väldigt fria JavaScript som i tillexempel det hårdtypade C#.

   Min plan är således att skapa en konstruktorfunktion i en separat fil som jag kallar Fraction.js som sedan skapar och initialiserar de efterfrågade medlemmarna och metoderna. Jag lägger metoderna på prototyp-objektet som tillhör konstruktorfunktionen.

2. **Genomförande:**

   1. Skapar filen Fraction.js i vilken jag skapar en konstruktor-funktion som skapar och initialiserar ett nytt bråktal som det skall, med täljaren och nämnaren som privata egenskaper och två priviligierade metoder, getNumerator och getDenominator, som jobbar som getters. Får fundera lite hur man skall skriva getters i JavaScript och säker den del på internet. 60 min

   2. Skapar en andra konstruktorfunktion som jag kallar fractionPrototype som egentligen bara är som en blueprint för prototyp-objektet. 20min

   3. Lägger till isNegative-metoden som kollar om det är negativt till min fractionPrototype-konstruktor. Inga egentliga problem kommer jag i kontakt med under genomförandet av detta. 10 min.

   4. Metoderna add och multiply skapas och läggs till i fractionPrototype-konstruktorn. Dessa plussar ihop, respektive multiplicerar det ursprungliga bråktalet med det bråktal man skickar med som argument. Får tänka lite hur man egentligen skall multiplicera bråktal, då det var många år sedan jag hade en mattelektion. Googlar mig fram till svar och skriver kod som jag tror skall lösa det någorlunda acceptabelt. 30 min

   5. Skapar och lägger till metoderna isEqualTo (som jämför huruvida två bråktal har samma värde, och skickar ett booleskt värde som svar) och toString (som överskuggar Object-metodens standard toString med en enkel utskrift av bråktalet med formen [Täljare]/[Nämnare]). Möter inga större motgångar. 40 min

   6. Skriver sedan raden: Fraction.prototype = new fractionPrototype(); och så vips så delar alla Fraction-objekt som skapas de metoder jag tilldelat prototypen. 5 min

3. De största problemen jag stötte på under programmeringen av denna uppgift var precis som jag trodde i min planering – att JavaScript inte är ett programmeringsspråk som har klasser som är lika klart specificerade som andra mera klassiska programmeringsspråk som C# och Java. Det finns liksom lite svängrum och plats för tolkningar.

   Till exempel getter-metoder kan skapas som en enkel metod på objektet som sedan returnerar den önskade egenskapen, men även läggas till som en egenskap genom lite mera invecklad kod genom att använda defineProperty. Skillnaden blir då, bland annat, att man kallar på tex getNumerator utan de efterföljande (), alltså som en egenskap och inte en metod på objektet. Detta är dock något nytt i ECMAScript 5, så IE 8 och tidigare hänger inte med. Jag valde därför i slutändan att inte använda defineProperty och bara skriva en enkel metod på Fraction-objektet som returnerar antingen T eller N.

   Jag vet inte riktigt hur jag borde ha planerat annorlunda för att kunna genomföra denna uppgift bättre, men det hade förmodligen varit enklare att genomföra om jag bara hade valt att göra det i C# istället. Att genomföra det i JavaScript var dock något jag ville göra för att få ut en liten extra bonus i det att det ger mig extra träning för den parallellt gående Webbteknik 1-kursen. Jag borde därför kanske lagt till en del i planeringen just om detta, att extra tid kommer att behövas men att det ses som väl värt det då vinsten i repetition av JavaScript ses som ett plus.

Uppgift 4 – Planering

4a.

a. De uppgifter vi måste genomföra är att först och främst bestämma vem som gör vad. En av de fem tar på sig ledar-mössan medan de fyra kvarvarande delas in i två grupper som tar han om serversidan (back end) och användargränssnittet (front end) respektive.

   Efter att ha bestämt vilka positioner och ansvar vi har inom gruppen så börjar vi vår inception-fas genom att sätta och ner och börja prata igenom olika användningsfall.

   Då CloudPortfolios plan är att skapa ett rätt så komplicerat system så finns det många användningsfall att gå igenom - det viktiga nu i början är dock att bara skapa enkla skisser av alla (vid denna tidpunkt) uttänkta användningsfall. Efter att dessa skisser skapats så väljer vi ut mellan 10-20% av de användningsfall som bedöms vara viktigast - viktigast gällande uppbyggnaden av systemets själva grundstomme, eller att de har ett stort värde för att kunna sälja produkten eller att de har högst risk kopplat till sig.

   Dessa 10-20% skrivs sedan ut i sin helhet och vi analyserar sedan som grupp dessa användningsfall intensivt för att få en någorlunda basal idé om vad det är vi måste börja med att skapa för att komma igång med projektet. Det skulle även kunna vara så att vi efter denna analys kommer fram till att projektet inte är realistiskt – det går inte att genomföra på ett godtagbart sätt

under våra förutsättningar. Det skulle ju då vara skönt att avsluta redan nu efter bara några dagar in i projektet och kunna börja med något annat, istället för att efter månaders slit märka att det omöjligt kommer att gå att genomföra arbetet. Som tur är tror vi oss ha kunskapen och motivationen med oss för att ro projektet i hamn så vi går vidare med CloudPortfolio.

b. Efter dessa inledande analyser hålls ett planeringsmöte där vi bestämmer att projektet skall genomföras på lite mera än ett halvår, uppdelat på 15 stycken två veckor långa iterationer. Under detta inledande möte delar ni upp er i de två front end- och back end-grupperna och börjar i par skissa på enkla UML-diagram för att få en liten idé om vad ni kan hinna med under den första iterationen.

   Planen är på inga sätt att ha en färdig prototyp efter den första iterationen. De första iterationerna ingår i elaboration-fasen och går ut på att under det första arbetet med själva kodandet börja få en mera verklighetsförankrad föraning om vad det är som kommer att krävas av er under detta projekt. I slutet av den första iterationen träffas vi och har ett andra planeringsmöte där vi med det vi upplevt under de gångna veckorna kan skapa en något mera verklighetsförankrad planering för den kommande iterationen, och även göra justeringar till tidsplanen för hela projektet. Kanske behövs det mera eller mindre tid för att hinna få en första funktionsduglig version av systemet klart?

c. Gruppen är som sagt uppdelat i tre delar – en person som har rollen att se över projektet och hålla koll på att alla håller sina deadlines och om de inte gör det, gå in och de-scopea målen för iterationen. Hen har också ansvaret att leta reda på testare och hålla kontakten med branschen. Kanske finns det delar av systemet som finns tillgängliga som Open Source-mjukvara som kan implementeras och kan på så sätt tid sparas på programmering? Det är upp till ledaren att hålla koll på. Denna ledarfigur har dock även möjligheten att sätta sig ner och koda om det skulle finnas tid och användning för det i någon av de två grupperna.

   De fyra kvarvarande medlemmarna är som sagt uppdelade i en front end- och en back-end del. Dessa är dock inte satta i sten, varken att det skall vara en klar uppdelning mellan grupperna eller vem som skall ingå i vilken. Kanske (förmodligen) kommer fokus att växla fram och tillbaka mellan front end och back end under utvecklingens gång och det kommer därför vara viktigt att kunna möta detta genom att kanske ha tre personer i en grupp under en iteration eller liknande lösningar. Kanske har en person tidigare erfarenheter av ett problem som uppstår i den andra gruppen – hen skall då enkelt kunna byta plats med någon i den andra gruppen för att underlätta utvecklingen av systemet. Ingen vi mot dom inom gruppen, alla skall sträva mot samma mål.

4b.

Svårigheterna att planera en uppgift som denna är att det aldrig går att veta vad som kommer att hända i ett senare skede av utvecklingen av mjukvara. Det är väldigt svårt, för att inte säga omöjligt, att exakt veta vad som kommer att behöva genomföras och sedan skriva en lång exakt lista på det innan man ens börjat med själva utvecklingen. Det blir därför

också omöjligt att ge en exakt tidsram och exakta roller till gruppen – allt detta är något som man med iterativ mjukvaruutveckling löser eftersom.

Genom att i början bara röra vid visionen och de viktigaste användningsfallen och sedan direkt börja utvecklingen med detta som bas så får man snabbt *verklig kunskap* som man sedan kan använda för att i slutet av varje iteration med större och större precision kunna planera vad man skall göra under nästkommande iteration, men även *hur* detta skall genomföras.

Detta i kontrast till att innan man börjar själva utvecklingen lägger en stor mängd tid på att skriva långa och utförliga planeringsdokument som i själva verket bara är baserade på vad man *tror* kommer att hända. Denna vattenfallsmodell är heller inte bara farlig i och med att man omöjligt kan veta vad som kommer att hända innan man börjar göra något – den är också dålig på att hantera förändringar. Med en agil utveckling utförs istället arbetet inkrementellt och iterativt och man är mycket mera beredd på eventuella förändringar eller störningar som kan uppkomma.