Uppgift 1 - planering

Planering, fas 1: läsa igenom hela uppgiften och få en uppfattning om vilken typ av kod som måste skrivas. Uppskattad tid i minuter mellan parenteser.

Både 1a och 1b läser in en sträng från tangentbordet och letar sedan igenom strängen (10) för att returnera ett visst värde. Felhantering ska implementeras i fall användaren matar in fel (10). En form av loop som går igenom strängen och analyserar värden ska skrivas. (10)Allt detta har gjorts i kursen 1DV402, så det krävs knappt tid för att utforska tillvägagångssättet. Koden ska sedan testas (15).
Total tid 1a: 45 min. Total tid 1b: 45 min.

1c kräver att användaren matar in 10 heltal, och programmet ska sedan returnera det näst största talet. Det som gör uppgiften en aning mer komplicerad är att man inte får använda arrayer, så möjligheten att dela talen i en array, sortera och returnera det näst-sista talet kan inte användas. Först ska programmet läsa in heltalen och hantera eventuella fel, utifrån antalet heltal användaren väljer att denna vill mata in (15). Sedan ska talen sorteras utan användning av arrayer eller datastrukturer (15). Lite tid för informationssökning i referensboken och på Google läggs till för att tänka ut bästa sättet att sortera på (15). Koden ska skrivas och testas (15).
Total tid 1c: 60min.

Total tid för planering av uppgift 1 inklusive planering: 30 + 45 + 45 + 60 = 180 min = 3h0min

Planering, fas 2:

Arbete	Verklig tid (planerad tid)	Vilka fel	Orsaker till avvikelsen från planeringen
läsning av strängen	5 (10)	-	Endast några rader kod, tid över
felhantering	10 (10)	-	Lade till en while loop så att programmet inte skulle stängas ner vid fel.
analys med loop	10 (10)	-	-
testa och färdigställa	15 (15)	-	-

Tabell 1: dokumentation för uppgift 1a

Arbete	Verklig tid (planerad tid)	Vilka fel	Orsaker till avvikelsen från planeringen
läsning av strängen	5(10)	-	snabbare än förväntat
felhantering	5(10)	-	snabbare än förväntat
analys med loop	15(10)	räknade först nollor som jämna tal. Sedan räknades alla jämna tal som nollor.	ändrade från for till foreach
testa och färdigställa	10(15)		

Tabell 2: dokumentation för uppgift 1b

Arbete	Verklig tid (planerad tid)	Vilka fel	Orsaker till avvikelsen från planeringen
research	3(15)	-	tänkte för svårt, problemet var mycket enklare än vad jag trodde
inmatning och fel-hantering	15(15)	antal gissningar ökade även vid fel inmatning.	
implementera sortering	20(15)	Deklarerade input-variabeln till 0 från början, men missade att det blir fel med negativa tal. Ändrade till int.MinValue.	som ovan, problemet var enklare än jag tänkte från början
testa och färdigställa	20(15)		Rätta till felen ovan

Tabell 1: dokumentation för uppgift 1c

1.3: Reflektera över planering och genomförandet av uppgiften.

Eftersom uppgifterna var ganska små var det relativt enkelt att göra en planering. Jag läste först igenom uppgifterna och försökte analysera vad koden skulle göra och vilka delar jag behövde skriva.

Algoritmen för uppgifterna 1a och 1b kan sammanfattas såhär:
* Ta emot och kontrollera argument som användaren matar in.
* Ge användaren en ny chans i fall den matar in fel.
* Räkna förekomsten av karaktärer med vissa egenskaper (a eller A, jämna eller udda)
* Returnera hur ofta värdena förekommer.

Eftersom alla dessa delar har varit vanliga inslag under tidigare kurser bestämda jag mig för att sätta igång med programmeringen och först göra den enklast möjliga versionen av applikationen. När det funkade att ta emot, räkna och returnera började jag testa olika scenarion. Syftet var att testa felhanteringen, samt att se om det finns buggar i applikationen när man matar in något som inte är det förväntade. Jag testade även att använda programmen enligt specifikationen, för att se om de gör det de ska.

Utifrån dessa testernas resultat antecknade jag vad som funkade och inte funkade, och vad som jag behövde åtgärda. Jag försökte sedan lösa ett av dessa problem åt gången, och när jag hade implementerar lösningen testade jag om det funkade.

För uppgift 1c var min första tanke att använda en array och sortera den, men kraven var att lösa den utan arrayer eller datastrukturer. Jag funderade lite på olika lösningar och bestämde sedan att det enklaste sättet skulle vara att ha två variabler, en för största och en för näst största talet, och sedan tilldela variablerna värden som matas in av användaren i fall de uppfyller kraven. Bästa implementeringen för den typen av algoritm var enligt mig en if/else if/else sats.

Jag gjorde som för 1a och 1b och skrev en enkel variation av programmet som jag sedan började testa. Jag noterade vad jag behövde åtgärda. Det visade sig att inmatning som inte var en integer fortfarande räknades som ett tal i serien, så jag gjorde om räkningen av antalet inmatade tal. Jag provade mig sedan fram i att lösa olika delproblem och fortsatte om ett visst problem klarade några tester.

Att utveckla stegvis i olika iterationer fungerar bra för mig, eftersom man kan prova sig fram och lösa problem allt eftersom de uppstår. Nu var de här uppgifterna väldigt små, men i större program har jag upplevt det mycket svårare att försöka förutspå alla möjliga problem och planera hela programmet innan än att jobba med små steg i taget utifrån en första version.

Uppgift 2: Förändring och förbättring.

Ett alternativ är att inte bara bestämma vad första versionen av programmet ska kunna och sedan testa fram lösningar till olika deluppgifter, men att istället planera flera olika milstolpar eller releases. Till exempel version 1.0 för uppgift 1a ska kunna läsa in strängen och lagra den, 1.1 ska räkna antal ‘a’ och ‘A’, 1.2 ska returnera till användaren samt implementera felhantering. Fördelen som jag skulle se med det är att felmarginalen minskar i program som är större än de i uppgift 1, eftersom man skapar en tydligare struktur. Nackdelen är då att man upptäcker en del problem allteftersom man väl börjar koda och testa, som kan vara svåra att planera för innan. Därför kan det vara svårt att uppskatta hur mycket tid det kan kräva.

Ett annat alternativ är att använda vattenfall-modellen, där man planerar ut hela programmet innan man börjar programmera. Fördelen med detta hade varit att man vet exakt vad som ska göras innan man sätter igång. Nackdelen som jag upplever med detta sätt är samma som ovan, att man måste tänka på allt som kan gå fel innan man väl börjar skriva koden.

Har man en större uppgift kan man även bestämma sig för att leverera vissa delar av den vid ett visst tidpunkt, så kallad inkrementell ansats.

Att förutse vilka fel man kommer att göra är väldigt svårt, eftersom man troligtvis inte skulle göra fel om man visste att de skulle komma. Vissa allmänna åtgärdar eller rutiner kan man följa för att minska potentialen för fel. En första är att följa konventioner som finns i programspråk kring hur man namnger variabler, klasser och metoder/funktioner. En annan är att notera alla namn på ett papper och försöka skissa och dra streck mellan dem för att förstå hur de hänger ihop och fungerar.

Relaterad till detta är att man måste ha klara och tydliga krav innan, och en detaljerad specifikation om hur programmet ska se ut och fungera med en tidsgräns för när arbetet ska vara klart.

När man skriver program som ska interagera med värden som angetts av en användare är det dessutom inte bara fel i själva koden som kan ställa till problem, men fel eller buggar kan uppstå utifrån användarens input. Därför är det viktigt att tänka ut innan vad som kan gå fel, och implementera en felhantering som tar hand om detta, och informerar användaren om att den måste göra om för att programmet ska gå att köra. Ett exempel är när man ska ha ett program som tar födelsedatum som argument, att det kan matas in i olika format samt att det finns ogiltiga värden (t.ex. den 38 januari).

För att minska konsekvenserna af fel eller oförutsedda händelser är det viktigt att tilldela tillräckligt med tid och resurser till uppgiften alternativt till varje deluppgift. Till exempel planerade jag för 15 minuter “planera och färdigställa” i uppgifterna ovan, eftersom jag visste att även om själva testerna (att mata in några olika teststrängar) inte skulle ta mer än 2 minuter måste utrymme finnas att justera och göra om.

De förbättringsåtgärdar som jag kommer att implementera i min vidare planering är att planera för flera releases och att sammanställa en tydligare kravspecifikation innan.

Uppgift 3a: Palindrom.

Programmet ska kontrollera om en inmatad sträng är en palindrom. Följande behövs:
* Användaren ska mata in en sträng som ska lagras
* Viss felhantering ska finnas om användaren inte matar in en sträng (release 1, tid: 5 min)
* Strängen ska vändas baklänges
* Det ska kontrolleras om strängen är samma baklänges som originalsträngen (release 2, tid: 5 min)
* Svaret ska returneras till användaren (release 3, tid: 5 min)

Programmeringstiden uppgår till 15 minuter. Vi lägger till 5 minuter för oförutsedda händelser och 5 minuter för testning och kommer fram till planerad tid på 25 minuter. Detta bör vara rimlig tid eftersom jag har gjort liknande uppgifter innan och programmet är litet.

Arbete	Verklig tid (planerad tid)	Vilka fel	Orsaker till avvikelsen från planeringen
inmatning av sträng och felhantering	5(5)	-	-
vända baklänges och kontrollera	2(5)	-	väldigt enkel kod
returnera svaret	1(5)	-	väldigt enkel kod
testa och färdigställa	5(5)	-	-

Uppgift 3b: Bråktal.
Det ska skrivas en klass för hantering av bråktal. Följande behövs:
* Medlemsvariabler, konstruktor, get/set metoder
* Metoder enligt kravspecifikation: bestämma lämpliga returtyper och argument och skapa tomma metoder att utgå ifrån (release 1, 15 min)
* Skapa ett program som kan testa klassen
* Koda metoderna isEqual och isNegative, som gör enklare jämförelser. Testa dem. (release 2, 20 min)
* Läsa på och tänka ut hur metoderna add och multiply ska fungera samt hur den ska kodas.
* Koda metoderna add och multiply. Testa. (release 3, 30 min)
* Skapa metoden toString
* Testa igenom hela klassen (release 4, 20 min)

Programmeringstiden uppgår till 85 minuter. 15 minuter läggs till för oförutsedda händelser, totala tiden uppgår då till 100 minuter.

Arbete	Verklig tid (planerad tid)	Vilka fel	Orsaker till avvikelsen från planeringen
Release 1	10(15)		mindre kod än förväntat
Release 2	20(20)		
Release 3	20(30)		enklare än förväntat
Release 4	30(20)		Förbättrade gränssnitt till testprogrammet

Uppgift 4

Om man utgår från ett team på 5 personer med olika kompetenser är första steget att analysera vilka områden det finns inom uppdraget och till vilka personer det bör fördelas. Här antas det att infrastrukturen med servrar med mera tillhandahålls av tredje part.

Enligt visionsdokumentet ska en applikation skapas som tillåter användaren att ladda upp, skapa och redigera/hantera dokument i ett OOXML-format. Grunden till applikationen är databaslagret. Om tjänsten blir framgångsrik kommer stora mängder att data lagras, därför är det viktigt att en av medlemmarna har kunskap om hur man skapar effektiva och snabba databaser.

Sedan kan vi dela upp arbetet i (övrigt) backend och frontend-utveckling. Frontend måste applikationen designas och användargränssnittet kodas, samt att en WISYWIG-redigerare måste skapas. För personerna i dessa roller är det mycket viktigt att de har kunskap om och passion för att utveckla intuitiva och tilltalande gränssnitt.

På backenden måste hantering för användare och deras dokument implementeras, samt att man måste skapa ett slags versionshantering för att kunna se vem som har gjort senaste ändringarna och vad de ändringarna var. Personerna i dessa roller måste kunna skriva snabb och effektiv kod som gör att applikationen fungerar utan problem eller fördröjningar.

I uppgiften antas det att applikationen endast ska ha ett webb-gränssnitt, och att en desktop-klient inte ska utvecklas i nuläget.

I uppgiften antas det att den som skrivit visionsdokumentet är en av teamet, och att denna även tar på sig en ledarroll i utvecklingsarbetet. Teamet antas vara uppbyggd av följande roller (förkortning för planeringstabellen mellan [ ]):
* Frontend utvecklare med fokus på design av gränssnitt. [F1]
* Frontend utvecklare med fokus på applikationer; kommer ha stor roll i redigeraren [F2]
* Backend utvecklare med fokus på databasutveckling och lagring/hantering av stora mängder data [B1]
* Backend utvecklare med fokus på kopplingen mellan databaslagret och användargränssnittet [B2]
* Allround utvecklare som agerar som utvecklingsledare samt ansvarar för kontinuerliga tester [UL]

För planeringen använder vi oss av ett flertal releases med olika krav för varje release. Utifrån varje release påbörjas sedan en ny iteration i utvecklingen av applikationen, så att den gradvis förbättras tills den slutliga produkten är färdigställd.

Release 0.1:
Det ska finnas ett mycket enkelt gränssnitt för användaren att ladda upp en fil och se sina filer. Det ska även finnas en input-ruta där användaren kan skriva ej formaterad text, som blir lagrad som en fil i databasen. Möjlighet ska finnas för användaren att registrera sig och sedan logga in med sina uppgifter. Databasen ska innehålla en tabell med användarnas uppgifter, en eller flera tabeller som innehåller filerna som laddas upp, och en vy som kopplar filerna till användaren. Det ska visas vem som är loggat in när användaren är inne i applikationen. En första testapplikation ska utvecklas som testar felhantering vid inmatning, eventuella säkerhetsrisker, och om inloggningen fungerar.

Arbete	Planerad tid (dagar)	Person
Presentation av projektet, genomgång av rutiner och processer i arbetet så att alla jobbar enligt samma konventioner, diskussion och beslut kring vilka verktyg och programmeringsspråk ska användas.	3	alla
Planeringsmöten om hur databasen ska struktureras och hur det ska hänga ihop med front/backend	2	alla
Frontend: skapa enkelt uppladnings-gränssnitt att bygga på och utveckla applikationen ifrån.	2	F1,F2
Skapa klass som tar hand om uppladdade filer och lagrar dem i databasen, initiera databas	2	B1
Skapa klass som tar hand om inmatad text och lagrar den som fil i databasen	2	B2
Skapa formulär till att logga in eller registrera sig	1	F1,F2
Skapa klass och databas-koppling för att hantera inloggning och registrering	1	B1, B2
Utveckla testapplikation, testa applikationen, justera eventuella fel	3	alla
Total planerad tid:	11 arbets-dagar	

Release 0.2:
Designen på gränssnittet ska utvecklas och bli mer tilltalande för användaren. Input-rutan ska läggas under en egen flik och utvecklas till ett enklare redigeringsverktyg som tillåter olika textstil, färger, storlekar och typsnitt. Backend-funktionalitet som konverterar texten som matas in till OOXML ska utvecklas. Uppladdningsfunktionaliteten ska utvecklas så att den innehåller verifikation av filtyp, samt en viss komprimering för att minska belastningen på databasen. Testapplikationen ska utökas för att testa den nya funktionaliteten.

Arbete	Planerad tid (dagar)	Person
Utveckla design på användargränssnittet, inkl. ny utseende/plats för redigerare-vyn	5	F1
Utveckla redigeraren, lägga till enklare redigering	10, (5)	F2, (F1)
Konvertering av inmatad text till OOXML, inkl. säkerhets/felhantering	10	B1
Skriva tester för front-end samt redigeraren	5	F1,F2,UL
Hantering och komprimering av uppladdade filer	5	UL,B2
Optimering av databasen	5	B2
Testa och korrigera	5	alla
Total planerad tid:	25 arbets-dagar	

Release 0.3:
Designen för gränssnittet ska utvecklas vidare för att även fungera bra på mobila enheter, i olika webbläsare och operativsystem, samt granskas för tillgänglighet för användare med funktionshinder. Redigeraren ska utvecklas vidare med funktioner som brukar finnas med i textredigerare, såsom stöd för tabeller, bilder, olika stilmallar, med mera. Uppladdningen ska utökas med en funktion som kan konvertera filer som inte är skrivna i OOXML-formatet till OOXML.

Arbete	**Planerad tid (dagar)**	**Person**
Utveckla responsiv design	10	F1, F2
Utveckling av redigeraren - frontend	10	F1,F2
Utvecklingen av redigeraren och konvertering till DB - backend	10	B1
Konvertering av ej-OOXML filer till OOXML	10	B2
Utökning av testprogrammet	5	UL
Testa och korrigera	10	alla
Total planerad tid:	25 arbets-dagar	

Release 0.4:
Stöd för dokument som är tillgängliga för fler än en användare ska utvecklas. På frontenden måste det synliggöras om vilken relation man har till ett dokument, samt att gränssnittet måste uppdateras så att användaren kan skilja bland egna och delade dokument och utföra handlingar utifrån detta. Även ett formulär där man kan välja vilka man vill dela dokumentet med ska finnas.
På backenden ska denna funktionalitet implementeras genom att applikationen ska kolla med databasen vem som äger applikationen, vilka användare som har vilka rättigheter, och säkerställa att alla ägare till ett dokument är överens ifall ett dokument ska raderas.
Testapplikationen ska utvecklas för att testa nya funktionerna.

Arbete	Planerad tid (dagar)	Person
Gränssnitt för inställningar kring delade dokument	5	F1
Gränssnitt som skiljer mellan delade och egna, visuell representation av olika typer av dokument	5	F2
Databaskoppling för att skilja mellan egna och delade dokument för en viss användare	5	B1
Klass som tilldelar rättigheter till skapade och uppladdade dokument	5	B2
Funktionalitet kring radera av dokument	5	B1
Implementera versionshantering, revisionshistoria (backend)	5	B2
Visuell representation av revision/versionshantering i gränssnittet		
Utöka testerna	5	UL
Testa och korrigera, färdigställa	10	alla
Total planerad tid:	20 arbets-dagar	

Total arbetstid för projektet kommer till 71 arbetsdagar, eller 14 arbetsveckor. För att bygga in marginal för oväntade händelser lägger vi till 10% och rundar 78,1 av uppåt till 80 arbetsdagar, eller 16 arbetsveckor.

Man får räkna med att utvecklingen kommer att fortsätta efter detta, eftersom applikationen kommer att börja användas av riktiga kunder som efterfrågar nya funktioner eller förbättringar, samt att det är möjligt att de hittar buggar eller problem som man har missat under testningen.