LINKÖPINGS UNIVERSITET

# TDP003 Projekt: Egna datormiljön

# TDP003 - Reflektionsdokument

Författare

Anton Sköld antsk320@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2017
Version 1.0

2017-10-18

# Innehåll



# 1 Revisionshistorik

| Ver. | Revisionsbeskrivning | Datum |
|---|---|---|
| 1.0 | Påbörjade dokumentet. Reflektioner tillagda | 2017-10-18 |

# 2 Inledning

Jag och min projektpartner William Utbult har i kursen TDP003 utvecklat en portfolio-sida. Detta dokument innehåller reflektioner över arbetet som vi har gjort.

Jag och William jobbat i helhet bra ihop. Vi har liknande kunskap och erfarenhet sen tidigare, så vi ligger ungefärligt på samma nivå.

Vi har bidragit ungefär samma mängd till de flesta delmål i projektet. Jag har arbetat lite mer på själva koden. William har dock arbetat mer än mig på dokumenten vi lämnar in eftersom hans stilkrav är något högre än mina egna.

För det mesta så delade vi inte upp arbetet, utan arbetade när vi kände motivation för det. När vi båda arbetade samtidigt så blev det ibland parprogrammering då vi båda satt på ett problem/en funktion. Ibland blev det separat arbete om det fanns flera saker att göra, t.ex att en kodar och en skriver LaTeX-dokument.

Jag och William är dock lite osynkade med tiderna vi känner för att jobba. Jag föredrar att arbeta en hel del på morgonen + eftermiddagen, medans William helst sitter kvällar. Ibland måste man knuffa igång den andra för att få mer gemenskap i arbetet.

Ibland om vi parprogrammerar ett svårt problem, så kan små konflikter uppstå i hur vi går tillväga att lösa problemen. Då bryter vi ofta parprogrammeringen och testar en lösning var lokalt, och den som blir klar

först eller har bäst lösning kör vi oftast med. Jag tycker att parprogrammering är något irriterande med två personer på ett tangentbord, det känns oeffektivt. Denna lösning fungerar då väldigt bra för oss båda.

# 3 Reflektioner

## 3.1 Projektet i helhet

Projektet i helhet gick smidigt att utveckla. Vi var nästan hela tiden långt före alla deadlines.

Vi hade bra planering med tydliga översiktsbilder, så vi visste alltid vad som skulle göras även om vissa detaljer var suddiga.

Bra specifikationer förklaras i: McConnell, Steve. 2004. *Code Complete, Second Edition*. Microsoft Press. Sida 38-43.

Boken säger att bra specifikationer leder till att användaren istället för programmeraren driver funktionaliteten, och att det kan hjälpa undvika argument.

## 3.2 Lo-Fi prototypen

Projektet började med en Lo-Fi prototyp. William skapade den i Photoshop och den såg jättefin ut. Några veckor senare så byggde jag HTML-mallarna lite snabbt, och jag kopierade designen från Lo-Fi prototypen så gott det gick.

I slutändan så blev vår sida väldigt fin, enligt mig en av de snyggaste av portfoliona i klassen.

## 3.3 JSON-filen

JSON-filen är datakällan till hela projektet. Filformatet visade sig vara väldigt lättanvändligt, speciellt inom Python.

Tyvärr så var det lite oklart vad som skulle vara med i JSON-filen. Det var väldigt tvetydlig information mellan systemspecifikationen, sidan för datalagret, och JSON-filen som användes i gemensamma testfallen på GitLab.

På grund av denna tvetydlighet så kan det finnas vissa detaljer som saknas i JSON-filen, men den bör uppfylla 95% av kraven ändå.

Bra specifikationer förklaras i: McConnell, Steve. 2004. *Code Complete, Second Edition*. Microsoft Press. Sida 38-43.

Boken förklarar viktigheten av att ha konkreta specifikationer, och att tvetydligheter kan leda till konflikter mellan användare och utvecklare, och icke uppfyllda krav.

## 3.4 Datalagret

Datalagret var väldigt smidigt att skapa. Specifikationen var tydlig med solklar information om vilken indata, och returdata som funktionerna ska ha.

På grund av denna information vad det enkelt att skapa ett ”kodskelett”, som man sedan fyller på med logik för att få korrekt utdata.

Arbetsfördelningen var jämn. Jag tog hand om skelettet, och en del funktioner. William tog hand om en del funktioner, inkluderat den större search()-funktionen.

### 3.5 Installationsmanualen

Manualen gick smidigt att skriva. Det fanns tydliga krav på det som skulle vara med, och det var enkelt att hitta informationen som behövdes för installation.

Manualen är just nu väldigt grundläggande. Den tar bara upp steg-för-steg hur man installerar Virtualenv och Flask, men det finns inga alternativa kommandon som man kör om originalkommandona inte fungerar av någon anledning.

### 3.6 Projektplanen

Projektplanen gick inte lika smidigt att skapa. Främst var det på grund av att vi inte var medvetna om inlämningstiden förrän en dag innan.

Jag har nu ett bättre system att hantera mitt schema och mina deadlines, så förhoppningsvis minskar det överraskningsfaktorn av framtida dokument.

Det kändes även lite otydligt vad exakt som skulle stå i projektplanen. Vi fick komplettera en gång, men efter det blev det godkänt. Det kändes dock som att vi repeterade en hel del information genom dokumentet.

### 3.7 Presentationslagret

Presentationslagret var enkelt och smidigt att skapa. Vi hade en Lo-Fi prototyp och systemspecifikation sedan innan, så vi visste nästan direkt hur layouten skulle bli.

Flask och Jinja var även enkelt att greppa. Det räckte med föreläsningen och en kort Youtube-tutorial för att greppa grunderna.

Grundläggande förberedelse och påläsningen om nya tekniker är extremt användbara innan man dyker ner i ett projekt.

Det var lite klurigt att komma på hur presentationen logiskt skulle fungera. Vi ville autogenerera checkboxes, en för varje teknik i datalagret, som hade tillräcklig meta-information för att kunna användas som en sök-term till search()-funktionen i datalagret.

Klurigheterna löste sig dock rätt snabbt när vi hittade <form>-objektet tillsammans med en POST-request på sidan.

### 3.8 Systemdokumentationen

Systemdokumentationen var överraskande enkel att skapa.

Det fanns krav på ett sekvens-diagram, och krav på en översiktlig bild över systemet. Vi kombinerade båda de punkterna till ett översiktligt sekvensdiagram.

Vi använde oss av http://sequencediagram.org/ för skapandet av sekvensdiagram. Sidan var mycket lättanvändlig.

Utöver översikten så krävdes det förklaring över alla metoder inuti data- och presentationslagret. Detta krav gick väldigt snabbt att uppfylla eftersom vi båda vet vad alla funktioner och parametrar har för syfte.

Vi skrev namnen på funktionen, vilka parametrar den tar och vad de betyder, och vad funktionen returnerar för objekt.

Dokumentet slutar med en del om tester, felhantering, och loggning. Där beskriver vi hur man vidareutvecklar projektet, hur man testar att allting fungerar, hur man kan debugga sin kod, och var man hittar loggar.

### 3.9 OpenShift

OpenShift-uppladdningen gick smidigt för oss båda. Instruktionerna fungerade väl, och det var lättförståligt hur man uppdaterar sidan i framtiden. Det var ju bara att pusha upp kod till Git-repot man skapade på OpenShift.

### 3.10 Git & GitLab

Detta är första projektet (för min del) som vi använde oss av versionskontrollsystem. Jag har lärt mig att versionskontroll är helt fantastiskt, och att jag förmodligen kommer använde mig av det resten av livet.

Vårt Git-användande är just nu rätt så simpelt, nästan endast git commit och git push. Även fast våran användning av Git är väldigt grundläggande, så slår det absolut versionshantering genom ZIP-filer skickade fram och tillbaka.

Fördelar av versionskontroll förklaras i: McConnell, Steve. 2004. *Code Complete, Second Edition*. Microsoft Press. Sida 668.

Boken säger att redigeringskonflikter nästan aldrig sker om två personer arbetar på samma fil, man kan enkelt uppdatera sin lokala kopia av projektet till nyaste versionen, och man har även en automatisk backup eftersom versionkontroll-kopian är ett säkerhetsnät (Filerna på GitLab).

### 3.11 PyCharm

Vi använde främst PyCharm för kodskrivning och debugging, och det har fungerat utmärkt. PyCharm har fin Syntax-highlighting, autofyllning av ord, bra debugging med mera.

PyCharm har även ett plugin som heter IdeaVim, vilket ger PyCharm simulerade Vim-kommandon för rörelse och redigering. Jag håller just nu på att lära mig och bli mer bekväm i Vim, så detta är nog ett plugin som jag kommer använda mig av i framtiden.

Fördelar av utvecklingsmiljöer så som PyCharm förklaras i: McConnell, Steve. 2004. *Code Complete, Second Edition*. Microsoft Press. Sida 710-711.

Boken beskriver flera fördelar med IDEs så som: Igenkänning av kompileringsfel från editorn, integrerade debugging-verktyg, hoppning mellan definitioner etc.

## 4 De viktigaste lärdomarna

Struktur och förberedelse är extremt viktigt. Om man har en specifikation tillsammans med ett schema så blir det mycket enklare att ta reda på om man ligger i fas eller har hamnat efter. Det är också mycket enklare att veta vad man ska arbeta på dag-till-dag.

# 5 Projektet i helhet

Projektet gick väldigt smidigt att utveckla hela vägen igenom, främst för att vi hade bra planering och arbetade mycket i förtid.

Det var häftigt att lära sig massor nya tekniker, främst Flask, Jinja, JSON, och Git.