Karl Peter Skjelvik

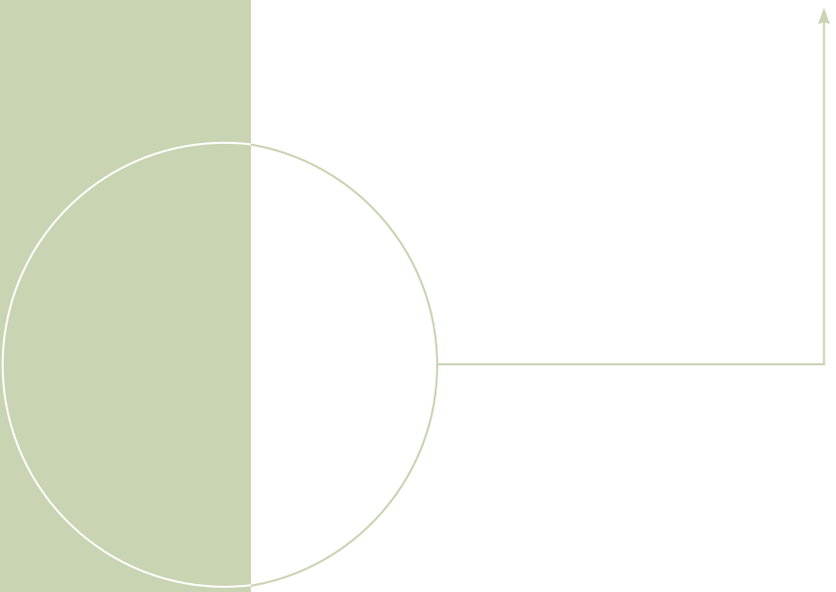
Innfyllingsoppgaver integrert i Visual Studio Code for bruk i programmeringsutdanning

Juni 2021

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Masteroppgave

Masteroppgave Juni, 2021 Karl Peter Skjelvik



NTNU
Kunnskap for en bedre verden

NTNU
Kunnskap for en bedre verden

NTNU
Kunnskap for en bedre verden

Innfyllingsoppgaver integrert i Visual Studio Code for bruk i programmeringsutdanning

Karl Peter Skjelvik

Datateknologi
Innlevert: Juni 2021
Hovedveileder: Hallvard Trætteberg

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknologi og informatikk

Forord

Denne masteroppgaven markerer slutten på 2 år med datateknologistudier, og 5 år på NTNU for meg. Dette prosjektet har bestått av en fordypningsoppgave som ble gjennomført høsten 2020 og en masteroppgave som ble gjennomført våren 2021

Jeg vil rette en stor takk til min veileder Hallvard Trætteberg, som har bidratt med verdifull kompetanse, veiledning og tilbakemeldinger som har gjort dette til et veldig spennende prosjekt.

Jeg vil også takke studentene som deltok på brukertest av det jeg har jobbet med og ga meg mye nyttige tilbakemeldinger.

Trondheim, Juni 2021
Karl Peter Skjelvik

Sammendrag

Det å lære seg å programmere er en utfordring for mange studenter. Når man programmerer er det mange ting man må fokusere på samtidig, fra logikk og syntaks til å skrive vedlikeholdbar og lesbar kode. Uansett om øvingen du jobber med egentlig bare skal lære deg om ett tema i programmering, så kommer du ikke unna å bruke mental kapasitet på alle de andre aspektene ved programmering

Målet med denne masteroppgaven var å utvikle et tillegg til Visual Studio Code som gjør det mulig å lage og utføre innfyllingsoppgaver integrert i Visual Studio Code, for bruk i programmeringsutdanning. Disse oppgavene lar studentene fokusere på å lære om enkelttema uten å måtte bruke like mye mental kapasitet på andre ting. For at tillegget skal kunne brukes til dette formålet må det være lett å komme i gang med å løse oppgaver som bruker tillegget, og det må fungere bra å bruke det, og det ble derfor gjennomført brukertester med studenter som tok emnet TDT4100 - Objektorientert programmering våren 2021.

Basert på brukertestene som ble foretatt av tillegget så er tillegget lett å komme i gang med, og det fungerer bra å bruke det. 75% av studentene som deltok på brukertest tror at de hadde hatt nytte av å ha denne typen oppgave tilgjengelig når de skal lære om nye tema i programmering.

Abstract

Learning to program is a challenge for a lot of students. When you're programming there are many things to pay attention to at once, from logic and syntax to writing maintanable and readable code. Even if the exercise you are working on is meant to teach you about a specific topic in programming, you can't escape spending mental capacity on all the other aspects of programming.

The goal of this thesis was to develop an extension for Visual Studio Code that makes it possible to create and solve fill-in-the-blank exercises integrated in Visual Studio Code, for use in programming education. These exercises let the students focus on learning about specific topics without spending as much mental capacity on other things. In order for the extension to be able to fulfill this purpose it has be easy to get started doing exercises with this extension, and using it has to work well, and therefore the extension went through user testing with students taking the course TDT4100 - Object-Oriented Programming the spring semester of 2021

Based on the user tests that were performed, the extension is easy to get started with, and using it works well. 75% of the students that participated in the tests think that having the option to do exercises like this when learning about new topics would be useful.

Innholdsfortegnelse

Figurer

Ordliste

API	Application Programming Interface, programmeringsgrensesnitt, grensesnitt som tillater at deler av et program kan aktiveres fra andre programmer.
Arbeidsflate	Område som vises på en dataskjerm, brukergrensesnitt (eng: workbench)
Arbeidsområde	En instans av et kodelager åpnet i en IDE (eng: workspace)
Emne	Et kurs som holdes i et tema, for eksempel TDT4100 - Objektorientert programmering
IDE	Integrated development environment, program for å assistere med utvikling og koding
Kodelager	Område der kode oppbevares, for eksempel på Github (eng: repository)
Mønster	Gjenbrukbar løsning på et kjent problem (eng: pattern)
Omgjøre	Å gjøre om igjen en handling som tidligere er blitt angret (eng: redo)
Sak	Et notat av et problem, forslag til funksjonalitet eller tilsvarende i et kodelager (eng: issue)
Tillegg	Komponent som legger til funksjonalitet til eksisterende programvare (eng: plugin)
Utvidelse	Komponent som utvider funksjonaliteten til et spesifikt program (eng: extension)
VS Code	Visual Studio Code, en IDE utviklet av Microsoft

Kapittel 1

Introduksjon

1.1 Motivasjon

Det å lære seg å programmere er en utfordring for mange studenter, og blant årsakene til dette er at oppgavene som gis oppfattes som repetitive og kjedelige.[1] Programmering er komplisert, og det er mange ting man må sette seg inn i som ny student, fra logikk og syntaks til brukbarhet og lesbarhet i koden, i tillegg til verktøyene man bruker, som en IDE. Ved hjelp av moderne teknologi som Gitpod og VS Code kan det være mulig å utvikle nye typer programmeringsoppgaver som gjør det lettere å lære å programmere ved å ha la studentene fokusere mer på spesifikke læringsmål i en oppgave, uten å måtte fokusere like mye på de aspektene av programmering som ikke er hovedformålet med oppgaven.

1.2 Tidligere arbeid

Denne masteroppgaven er en fortsettelse av et fordypningsprosjekt som undersøkte hvordan verktøyet Gitpod kan brukes i undervisning i faget TDT4100 - Objektorientert programmering. I dette prosjektet ble det funnet at Gitpod støtter tillegg laget for VS Code, og at dette kan utnyttes for å tilby nye typer programmeringsoppgaver. Fordypningsprosjektet så også på hvordan programmeringsoppgaver kan tilby et spissere fokus og færre distraksjoner enn tradisjonelle programmeringsoppgaver, og fant ut at man kan bruke prinsipper fra Parsons-problemer. Parsons-problemer er basert på at man får ferdige kodesnutter, og så er oppgaven å legge disse i riktig rekkefølge, i stedet for at man må skrive all koden selv.[2]

Konklusjonen i fordypningsprosjektet ble at man kan utvikle tillegg som støttes av Gitpod som lar deg lage og gjennomføre innfyllingsoppgaver inspirert av Parsons-problemer.

1.3 Forskningsspørsmål

Forskningsspørsmålene som har styrt denne masteroppgaven er:

Spørsmål 1: Hvordan burde den konseptuelle modellen for innfyllingsoppgaver i programmeringsutdanning se ut?

Spørsmål 2: Hvordan kan et tillegg for innfyllingsoppgaver best integreres i VS Code?

Spørsmål 3: Hvordan kan man best designe og utvikle et tillegg til VS Code som støtter å lage og gjennomføre innfyllingsoppgaver?

1.4 Mål og krav

Målet med masteroppgaven er å fortsette arbeidet slik det ble lagt frem i fordypningsoppgaven som beskrevet i seksjon 1.2, og å svare på forskningsspørsmålene definert i seksjon 1.3. Dette arbeidet vil bestå i å utvikle et tillegg til VS Code, som også vil støttes i Gitpod, basert på designarbeidet gjort i fordypningsprosjektet, som beskrivet i subseksjon 2.1.3. Spørsmålet om hvordan den innfyllingsoppgavene skal fungere ble også jobbet noe med i fordypningsprosjektet, og de resultatene finnes også i subseksjon 2.1.3. Når tillegget utvikles og det samles mer informasjon om hvilke muligheter som tilbys i API-et for utvikling av tillegg til VS Code, vil også den konseptuelle modellen for innfyllingsoppgavene endres og utvides for å tilby mer funksjonalitet.

Målet med fordypningsprosjektet som ledet inn til denne masteroppgaven var å forbedre programmeringslæringen i emnet TDT4100 - Objektorientert programmering. Et viktig bruksområde for programmeringsoppgavene som bruker tillegget vil være for å introdusere studenter til nye tema i programmering, ved å la studentene fokusere mer på det nye temaet uten å måtte tenke på alle andre aspekter ved programmering. For at det skal være verdt det for studentene å bruke dette tillegget er det veldig viktig at det både er intuitivt å komme i gang med, og lett å bruke.

For at det skal være nyttig må også funksjonaliteten for å lage oppgaver bygges ut, slik at det kan bli laget oppgaver til studentene. Funksjonaliteten for å lage oppgaver er ikke like avhengig av å være lett å komme i gang med, da den skal primært brukes av forelesere eller andre som har sterk faglig kunnskap og klarer å sette seg inn hvordan å bruke systemet fra en brukermanual. Prioriteten her er heller at systemet er effektivt å bruke for å lage oppgaver når man har satt seg inn i hvordan det fungerer, slik at man kan lage nye oppgaver uten at det er mer arbeid enn nødvendig.

Kravliste

Disse overordnede kravene stilles til tillegget og utviklingsprosessen fra inngangen til prosjektet.

Funksjonelle krav

- Det som utvikles skal være et tillegg til VS Code
- Det skal være mulig å løse innfyllingsoppgaver
- Det skal være mulig å lage innfyllingsoppgaver

Ikke-funksjonelle krav

- Utviklingen skal følge prinsipper fra smidig utvikling
- Utviklingen skal følge prinsipper fra lean utviking
- Utviklingen skal følge prinsipper fra brukersentrert utvikling
- Tillegget skal være brukervennlig og intuitivt å bruke for å løse oppgaver
- Tillegget skal være effektivt å bruke for å lage oppgaver når man har erfaring med det
- Tillegget skal være godt integrert i VS Code-arbeidsflaten.
- Tillegget skal programmeres på en måte som gjør det lett å utvide det med ny funksjonalitet
- Tillegget og informasjonen samlet inn under utvikling av tillegget skal lagres på en måte som gjør det lett å bygge videre på av andre utviklere

1.5 Bidrag

Bidraget til denne masteroppgaven kommer i form av et tillegg til VS Code som finnes på `https://github.com/kape142/vscode-parsons`. Dette tillegget lar deg lage og gjennomføre en ny type programmeringsoppgave inspirert av Parsons-problemer direkte i VS Code. Denne masteroppgaven dokumenterer hvordan denne typen oppgaver og dette tillegget er blitt utviklet og testet, og viser at det har potensiale for bruk i programmeringsutdanning.

Kapittel 2

Bakgrunn

2.1 Fordypningsprosjekt

Denne seksjonen vil inneholde en oppsummering av den viktigste informasjon som ble funnet i fordypningsprosjektet, og de viktigste resultatene fra fordypningsprosjektet.

2.1.1 Parsons-problemer

Parsons-problemer er en type programmeringsoppgave der man blir gitt ferdige kodesnutter og skal legge dem i riktig rekkefølge, i stedet for å skrive koden manuelt.[2] I fordypningsprosjektet ble det funnet at Parsons-problemer kan være en god måte å lage programmeringsoppgaver som fokuserer på mindre tema, men at det også har en del ulemper som har ført til at det ikke har sett så mye bruk i høyere utdanning enda. Disse ulempene handler hovedsaklig om at det krever mye jobb å lage oppgavene, koden må skrives på samme måte som andre oppgaver, men den må i tillegg deles opp i seksjoner som skal sorteres. Fordi studentene kun skal sortere kodesnutter og ikke skrive kode selv går også oppgavene mye fortere å løse, så for å ha en tilsvarende mengde arbeid må det lages store mengder oppgaver.

2.1.2 Gitpod

Gitpod er en nettbasert IDE for bruk med GitHub og andre git-vertstjenester.[3] Fordypningsprosjektet så på flere måter bruk av Gitpod i undervisning kan forbedre programmeringslæring, for eksempel ved å forenkle prosessen med å komme i gang med bruk av IDE, fordi man slipper å installere ting lokalt på sin PC. Det viktigste som ble avdekket i fordypningsprosjektet om bruk av Gitpod er muligheten for å installere tillegg som er laget til VS Code i Gitpod.[4] Dette gjør det mulig å utvikle et tillegg til VS Code som så kan brukes i Gitpod for å tilby programmeringsoppgaver basert på Parsons-problemer.

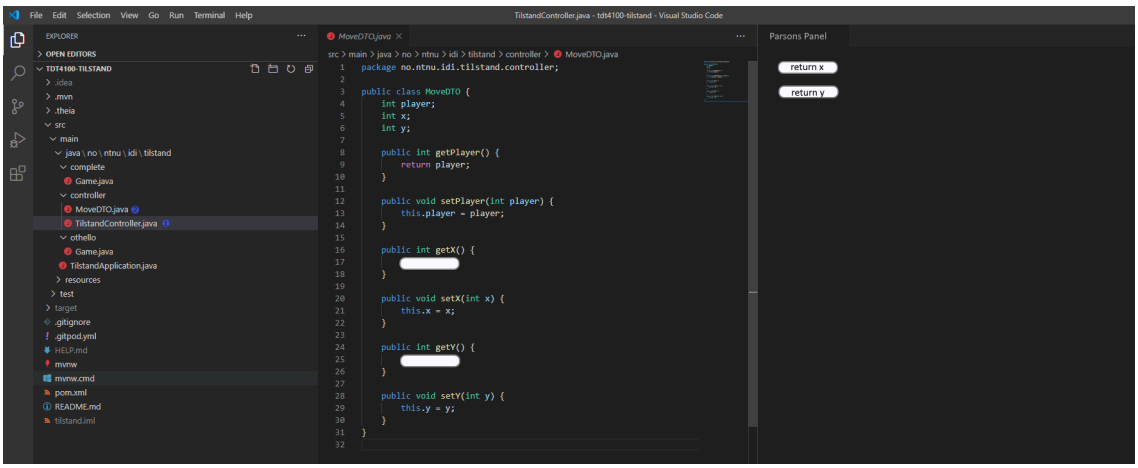
2.1.3 Resultater

I tillegg til den teoretiske bakgrunnen var resultatet av fordypningsprosjektet en tekstlig beskrivelse av tillegg for Parsons-problemer i Gitpod, skisser av hvordan det kan se ut, og modeller av hvordan oppgaveinformasjon kan lagres.

Beskrivelse og skisser

I fordypningsprosjektet ble det planlagt at verktøyet vil bestå av to tillegg til VS Code, ett tillegg for å lage oppgaver, og ett for å løse oppgaver. Mye forskjellig potensiell funksjonalitet ble beskrevet, og noe av funksjonaliteten ble også laget skisser av. Den grunnleggende funksjonaliteten er at en javafil har noe kode fjernet og erstattet med et tomt felt, og at man kan dra kodesnutter fra et panel til de tomme feltene. Dette er illustrert i figur 2.1. I tillegg beskrives det flere forskjellige utvidelser av denne funksjonaliteten:

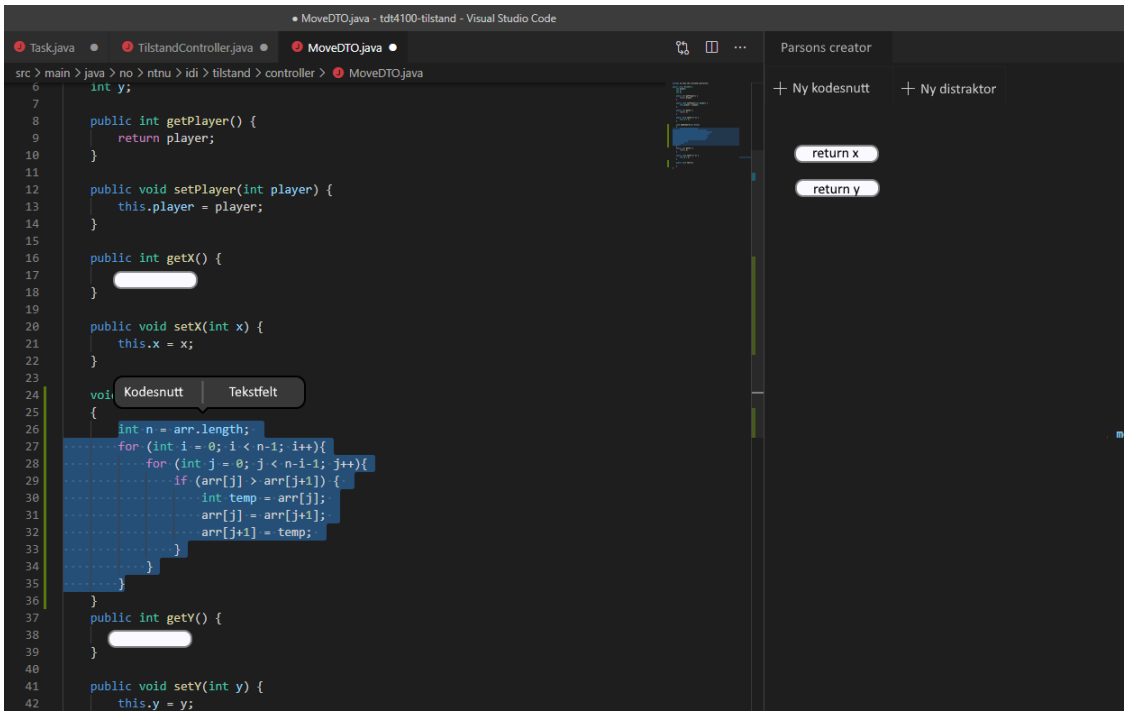
- Tomme felter som fylles ved å skrive kode manuelt i stedet for å bruke en kodesnutt
- Bruke flere kodefiler sammen, med samme panel for kodesnutter, slik at man må finne ut hvilke kodesnutter som hører til i hvilken kodefil
- "Distraktorer", kodesnutter som ikke hører til noe sted men som må vurderes opp mot det riktige svaret
- Kompilering og kjøring av koden med de svarene som er fylt inn
- Støtte for flere programmeringsspråk
- Valg av indentering for kodesnutter når de plasseres



Figur 2.1: Skisse som viser en kodefil med to mangler, panel til høyre med to kodesnutter, og fil-tre til venstre der to av filene har ikoner som symboliserer at de har mangler

Også tillegget for å lage oppgaver er beskrevet, og her er tanken at man har et tilsvarende panel med kodesnutter på høyresiden, men med knapper for å legge til

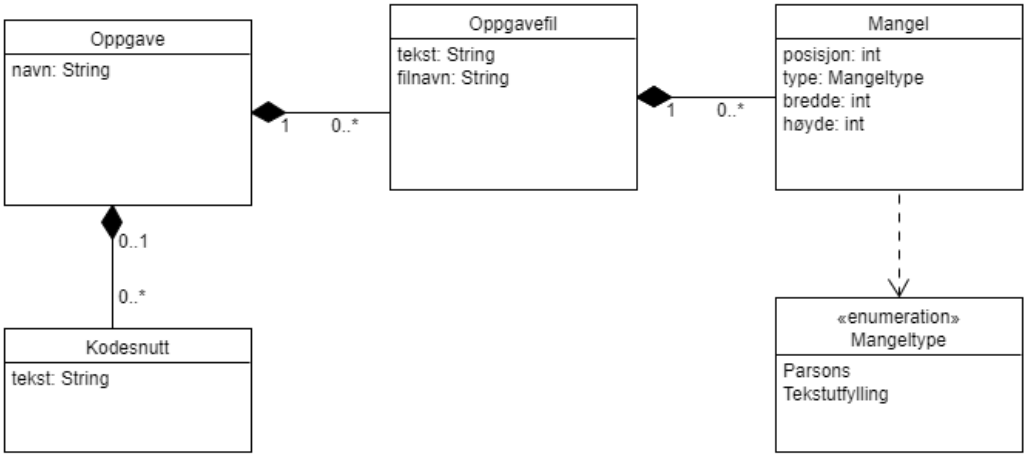
nye kodesnutter. For å fjerne kode fra filen og lage et tomt felt markerer man teksten som skal fjernes og trykker på et av valgene som dukker opp. Dette er illustrert i figur 2.2



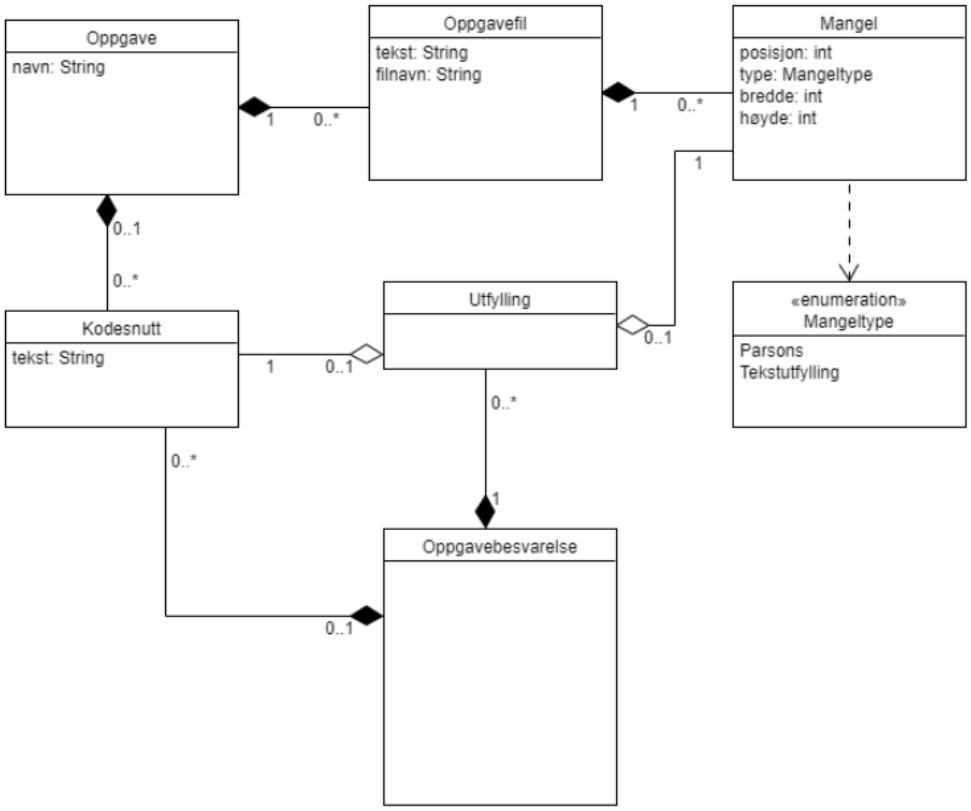
Figur 2.2: Skisse som viser panel til høyre med mulighet for å legge til nye kodesnutter, og hvordan det kan se ut når man markerer tekst for å lage en ny oppgave

Modeller

Fordypningsprosjektet definerte også en informasjonsmodell for lagring av en oppgave, og for lagring av en oppgaveutfylling. En oppgave vil inneholde all informasjonen for å definere en oppgave med kodefiler som har mangler, og kodesnutter som kan brukes for å fylle dem inn. Dette er modellert i figur 2.3. En oppgaveutfylling lagrer også informasjon om en besvarelse, hvilke kodesnutter som er brukt til hvilke mangler, i tillegg til en liste av kodesnutter som er skrevet for hånd til mangler som bruker tekstutfylling i stedet for ferdige kodesnutter. Dette er modellert i figur 2.4



Figur 2.3: Modell av en oppgave



Figur 2.4: Modell av en oppgaveutfylling

2.2 Visual Studio Code

Visual Studio Code, eller VS Code, er en IDE utviklet av Microsoft for Windows, macOS og Linux.[5] VS Code har åpen kildekode[6], som vil si at kildekoden er

offentlig tilgjengelig og kan inspiseres, modifiseres og forbedres.[7] VS Code har støtte for tillegg som legger til ny funksjonalitet[6], og det er denne muligheten, og muligheten for å bruke disse tilleggene også i Gitpod, som er interessant for dette prosjektet.

2.2.1 Tillegg

Et tillegg til VS Code defineres i en manifest-fil som skal kalles `package.json`. Denne filen definerer meta-informasjon om tillegget som navn og kategori, hvilken funksjonalitet tillegget bidrar med, hvilke hendelser som skal aktivere tillegget, inngangspunktet for utvidelsen, hvilke avhengigheter tillegget har, og skript for kompilering, bygging, testing etc.

Tillegg til VS Code skrives i utgangspunktet i JavaScript, men det er også mulig å bruke programmeringsspråk som kan kompileres til JavaScript, som for eksempel TypeScript, men disse må kompileres til JavaScript som en del av byggeprosessen til tillegget.

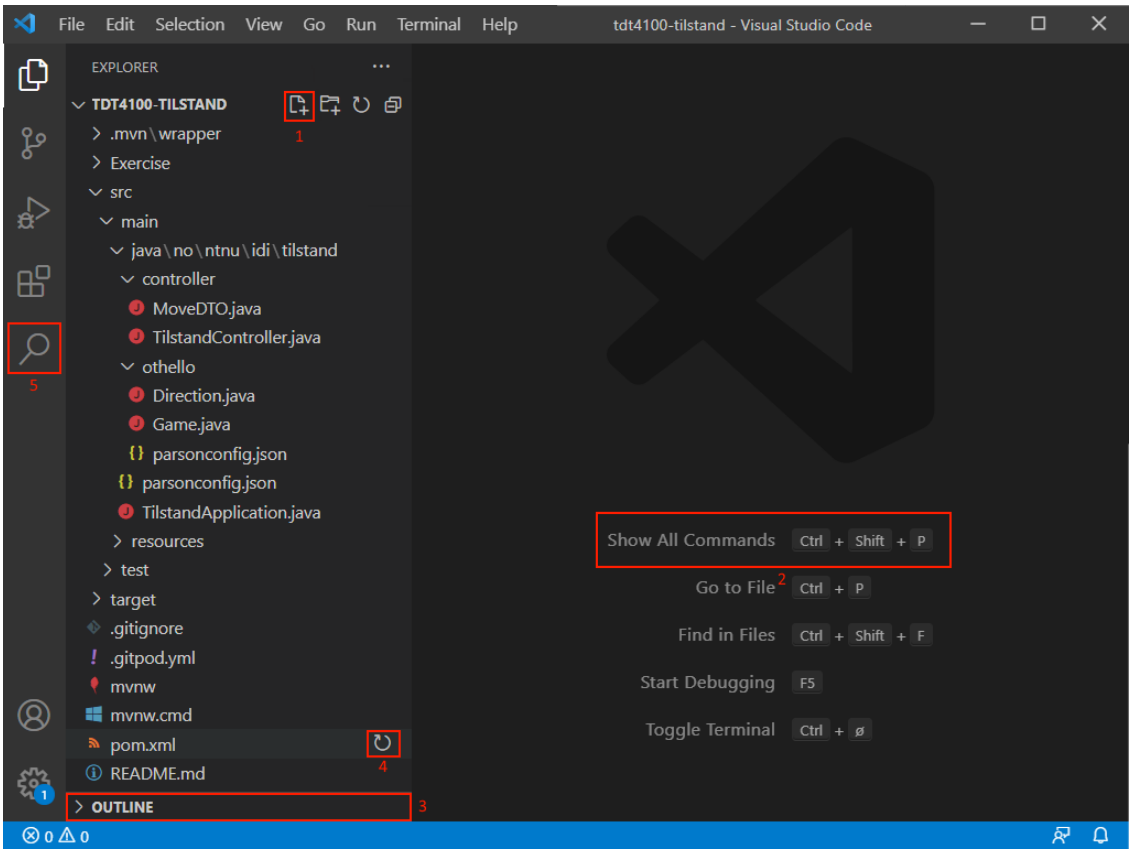
API

VS Code har et omfattende API for utvikling av tillegg som gjør det mulig å utvide IDE-en på mange forskjellige måter.[8] Noen av mulighetene er:

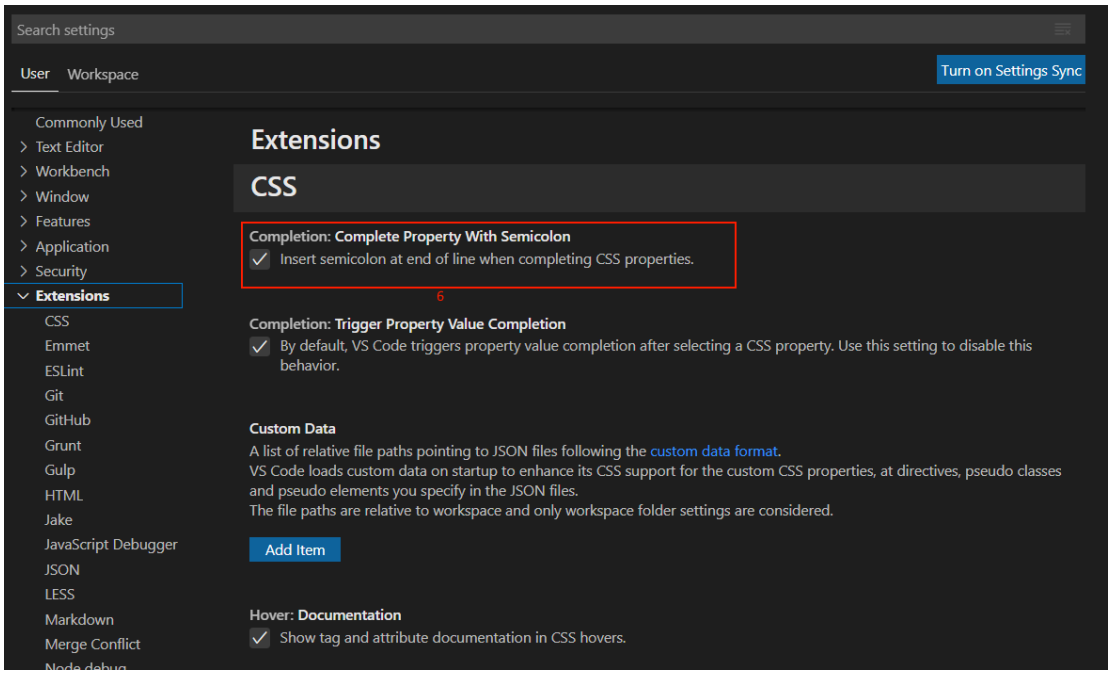
- Lage tema for å endre farger og utseendet til VS Code
- Utvide brukergrensesnittet med egne komponenter og visninger
- Lage egne webvisninger med HTML/CSS/JS integrert i VS Code
- Legge til støtte for nye programmeringsspråk
- Utvide feilsøkingsfunksjonaliteten

API-et lar deg utvide brukergrensesnittet med egne komponenter og visninger, og dette kan brukes for å lage tillegg som er godt integrerte i VS Code og som føles som en naturlig utvidelse av systemet og ikke en separat applikasjon. I figur 2.5 og figur 2.6 vises seks av utvidelsespunktene der det er mulig å legge til egen funksjonalitet.

1. Legge til funksjoner som påvirker data som vises i en tre-visning, her filsystemet
2. Legge til kommandoer i kommandopaletten
3. Legge til en ny visning av data som en trestruktur
4. Legge til funksjoner som påvirker et konkret element i en tre-visning
5. Legge til et nytt panel for visninger
6. Legge til innstillinger som kan endres i innstillingsmenyen.



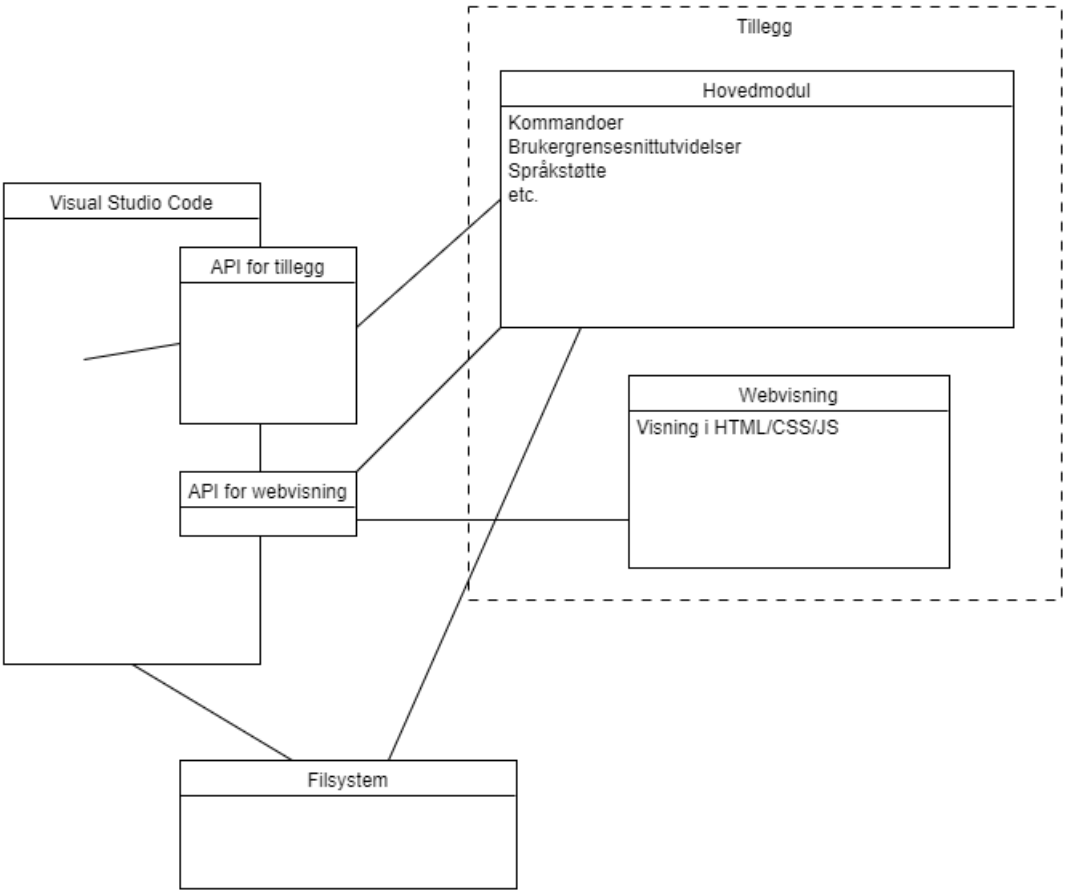
Figur 2.5: Figur som viser ulike måter grensesnittet i VS Code kan utvides



Figur 2.6: Figur som viser en innstilling i VS Code som er lagt til av et tillegg

Arkitektur

Et tillegg til VS Code kjøres isolert fra resten av VS Code, og tillegget har kun tilgang til de delene som eksponeres gjennom API-et. Tillegget har tilgang til det lokale filsystemet og kan gjøre operasjoner direkte med det uten å gå gjennom VS Code. Dersom tillegget inneholder egne webvisninger med HTML/CSS/JS er disse ytterligere isolert, og de har verken direkte tilgang til filsystemet eller det vanlige API-et for tillegg. I stedet må det bruke et eget API som gjør det mulig å sende meldinger mellom webvisningen og de andre delene av tillegget. Denne arkitekturen er modellert i figur 2.7



Figur 2.7: Modell av arkitekturen til et tillegg til VS Code, og hvilke deler som har tilgang til hva

2.3 Vedlikeholdbarhet og utvidbarhet

Et viktig krav til det som utvikles i dette prosjektet, som definert i seksjon 1.4, er at det skal være mulig å videreføre arbeidet. Det vil si at både det som programmeres må følge gode utviklingsmønstre som sikrer at koden kan vedlikeholdes og utvides, og det må legges til rette for at informasjonen som er samlet inn under utviklingen kan brukes til å styre prosjektet videre.

2.3.1 Vedlikeholdbar hode

Det å skrive kode som er lett å vedlikeholde er en grunnleggende tanke i programmering, og en måte å oppnå dette på er å bruke etablerte mønstre for design av kode, som kan finnes for eksempel i [9].

To viktige prinsipper som også hjelper med dette i objektorientert programmering er løse koblinger og høy kohesjon. Høy kohesjon handler om hvor spesialisert fokuset til et objekt eller en modul er, mens løse koblinger handler om at forskjellige objekter eller moduler ikke skal være mer avhengig av hverandre enn nødvendig. Ved å følge disse prinsippene gjør man det lettere å utvide systemet med ny funksjonalitet og nye moduler.

2.3.2 Utvidbar kode

Som en del av det å utvikle kode som skal kunne utvides er det også lurt å definere punkter der koden er planlagt å utvides eller der det er viktigst at det er mulig å kunne utvide koden. Dette kan gjøres for eksempel ved hjelp av språkmekanismer som arv, som gjør at nye deler av systemet kan ta nytte av det som allerede er utviklet, eller delegering, som gjør at det er lettere å erstatte eller utvide deler av systemet uten å endre resten. En annen måte det kan gjøres på er å definere rammeverk som gjør det mulig å utvide systemet uten å direkte endre det som allerede er skrevet av kode, slik som for eksempel API-et til VS Code som beskrevet i subseksjon 2.2.1.

2.3.3 Videreførbart prosjekt

For at det skal være mulig å forvalte og videreføre prosjektet holder det ikke at koden er utvidbar, de andre delene av prosjektet må også kunne arbeides videre med. Dette gjelder kodelageret der koden oppbevares, planlagte endringer og funksjonalitet som ikke enda er implementert, og informasjon som er samlet inn som en del av prosjektet, som for eksempel resultater av brukertesting.

For at all denne informasjonen skal være tilgjengelig for utviklere som vil videreføre prosjektet, kan innsamlet informasjon som resultater fra brukertesting legges i kodelageret sammen med koden. Kodelager-tjenester som Github har støtte for å legge inn saker, som kan brukes for å holde styr på planlagt funksjonalitet og problemer som er oppdaget men ikke løst enda.

Kapittel 3

Metode

Dette kapittelet beskriver metodikken som brukes i dette prosjektet. Metodikken tar elementer fra smidig utvikling, lean utvikling og brukersentrert utvikling, og bruker de elementene som er best egnet for dette prosjektet og for å oppnå de målene som dette prosjektet har.

3.1 Smidig utvikling

Smidig utvikling er en utviklingsmetodikk som er basert på å ikke bare følge en forhåndsbestemt plan, men å justere planen basert på informasjon samlet underveis i utviklingen, å samarbeide direkte med kunden som skal bruke produktet underveis i prosessen og å ha et større fokus på fungerende programvare over omfattende dokumentasjon. Prinsippene for smidig utvikling ble definert i "Manifestet for smidig programvareutvikling"[10] i 2001 og smidig utvikling er i dag svært utbredt. Så mye som 85.9% av profesjonelle utviklere oppgir i 2018 at de bruker smidig metodikk i sitt arbeid.[11]

Smidig utvikling er populært, og det er også knyttet til større suksessrate sammenlignet med tradisjonell utviklingsmetodikk.[12] Ved å bruke elementer fra smidig utvikling i utviklingen i dette prosjektet økes også sannsynligheten for at dette prosjektet blir en suksess og kommer nærmere å oppnå målene som er satt for prosjektet.

3.1.1 Smidig utvikling i dette prosjektet

For å kunne være trygg på at det som utvikles i dette prosjektet er av god kvalitet skal det følges en del prinsipper fra smidig utvikling. Mange av fordelene med smidig utvikling kommer mest til uttrykk i prosjekter der man er flere som jobber sammen, men det er også mange gode prinsipper man kan bruke i prosjekter med kun én utvikler.

Et av de viktigste prinsippene i smidig utvikling er iterasjoner, og dette prosjektet vil bestå av mange iterasjoner, og etter hver iterasjon skal det reflekteres over resultatet av iterasjonen og det vil gjøres vurderinger om noe burde gjøres om igjen eller endres i planen som er lagt videre.

I mange prosjekter som bruker metodikk fra smidig utvikling har man iterasjoner som består av faste tidsintervaller, som to eller tre uker, med konkrete mål for hver

iterasjon og oppfølging av de målene etter hver iterasjon. På grunn av mye usikkerhet rundt hvor lang tid det vil ta å fullføre mål så er det vanskelig å planlegge riktig arbeidsmengde for en iterasjon som har en fast tidsramme i dette prosjektet, og iterasjonene er derfor heller fokusert rundt hvilken funksjonalitet som skal fullføres, uavhengig av hvor lang tid det tar.

Hviken funksjonalitet som skal være i hver iterasjon er planlagt i neste subseksjon, men dette er ikke en plan som ikke kan endres, og når man følger smidig utvikling er det naturlig at informasjon som samles underveis i utviklingen vil påvirke videre veivalg. Dette vil påvirke både hva som arbeides med videre, men også hvordan neste iterasjon skal defineres.

Refleksjon over hvilken ny informasjon som har dukket opp i løpet av forrige iterasjon vil gjøres i samarbeid med veileder for prosjektet, som i tillegg til å være veileder med kunnskap om utvikling av programmeringsoppgaver også er en relevant bruker av foreleserversjonen.

Hvordan iterasjonene ender opp med å bli gjennomført dokumenteres i tillegg A.

3.1.2 Hovedplan

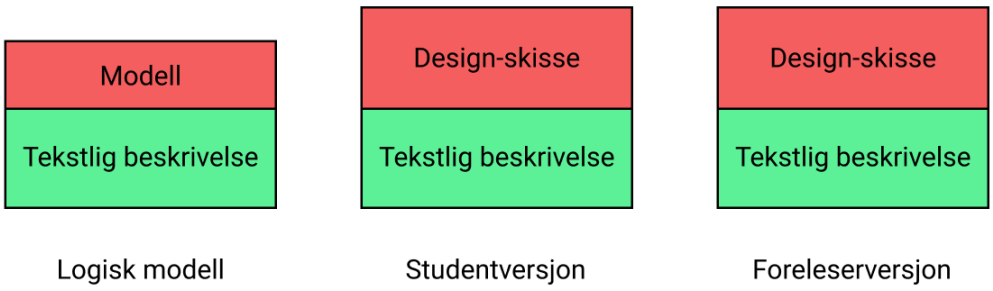
Utgangspunkt

I fordypningsprosjektet ble det utforsket forskjellige måter oppgaven kan løses på og resultatet ble et utkast til en løsning med tekstlige beskrivelser, noen modeller, og noen design-skisser.

I denne planen vil prosjektet deles opp i tre spor som kan prioriteres ulikt i de forskjellige iterasjonene. De tre sporene er:

- Den logiske modellen
- Tillegget for å løse oppgaver (Studentversjonen)
- Tillegget for å lage oppgaver (Foreleserversjonen)

Den logiske modellen forteller hvordan systemet skal lagre informasjon, hvordan en oppgave blir lagret og hvordan informasjon som fylles inn i en oppgave blir lagret. Studentversjonen vil brukes hovedsaklig av studentene som skal løse oppgaver, mens foreleserversjonen brukes hovedsaklig av forelesere som skal lage oppgaver for å løse i studentversjonen.



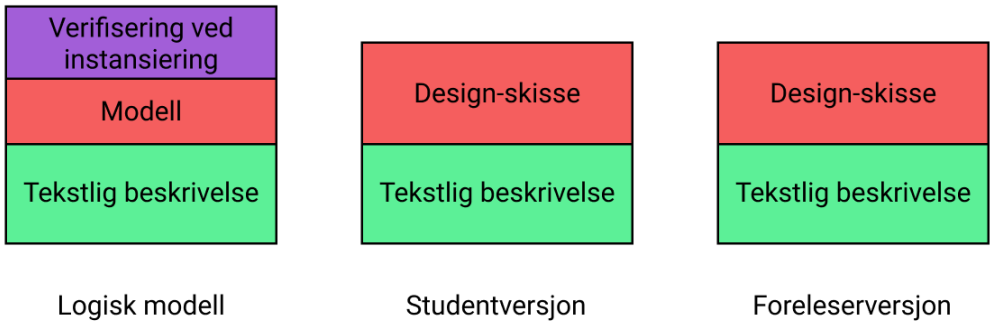
Figur 3.1: Situasjonen for de 3 sporene ved oppstart av masteroppgaven

Ved oppstart av masteroppgaven har alle tre sporene en tekstlig beskrivelse i tillegg til visuelle komponenter som ble laget i fordypningsprosjektet, og denne fremgangen er visualisert i figur 3.1. Den logiske modellen har to UML-diagrammer men disse er ikke arbeidet like mye med som design-skissene for studentversjonen og foreleserversjonen, så boksen for modellen er litt mindre.

Boksene er fargekodet med tre forskjellige farger, grønn for tekstlige elementer, rød for visuelle elementer, og lilla for kode-baserte elementer.

Iterasjon 1

Første iterasjon vil bestå av å teste ut modellen som ble designet i fordypningsprosjektet, ved å lage testdata som bruker modellen. Ved å lage mange forskjellige instanser kan man finne ut om modellen støtter alle de egenskapene det er ønsket at den skal støtte på en god måte, eller om den burde endres.



Figur 3.2: Iterasjon 1

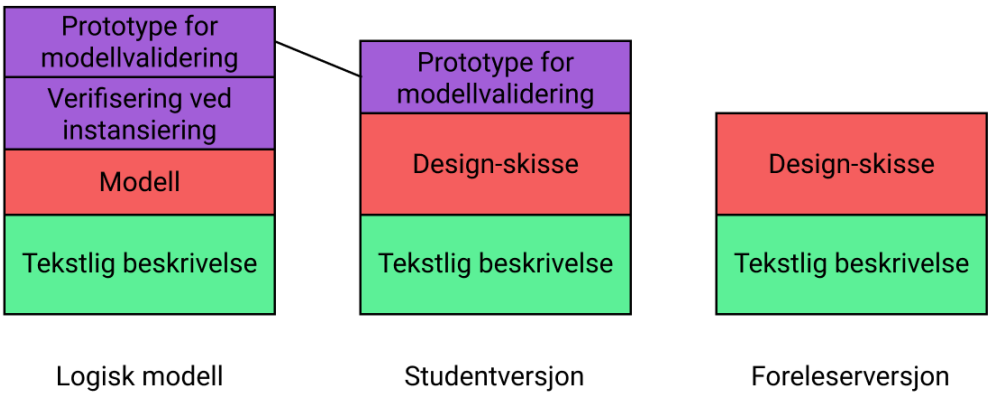
I figur 3.2 er blokken "Verifisering ved instansiering" lagt til i sporet "Logisk modell".

For å være så trygg som mulig på at mulige svakheter blir avdekkes, burde valg av eksempeldata skje systematisk, og dekke så mange muligheter for forskjellige oppsett som mulig.

Denne verifiseringen gjøres tidlig i prosessen for å unngå at man på et senere tidspunkt må gå tilbake og endre modellen etter at mye arbeid er blitt gjort basert på den første versjonen av modellen.

Iterasjon 2

Andre iterasjon vil bruke eksempeldata som ble laget i iterasjon 1 i en prototype på studentversjonen som kun viser koden fra oppgaven uten noe interaktivitet. Dette gjør at man får testet modellen også som en datastruktur å hente data ut fra for bruk i en situasjon som ligner på den som modellen er ment å brukes i.

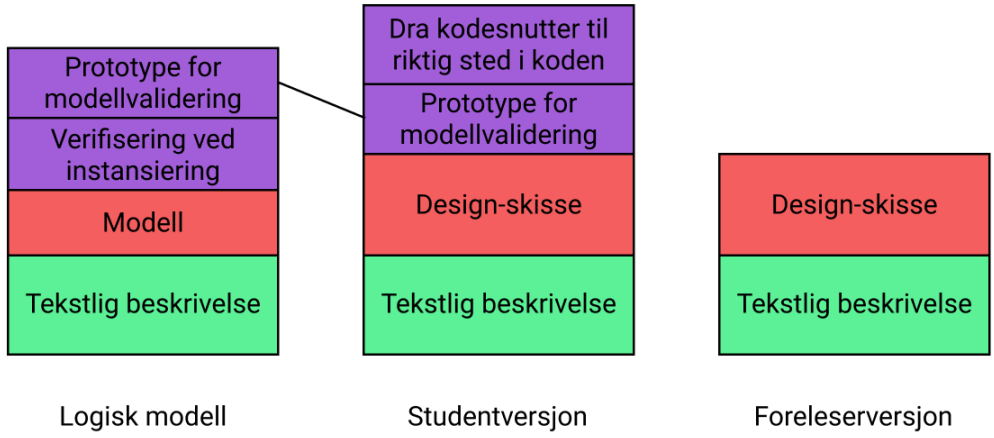


Figur 3.3: Iterasjon 2

I figur 3.3 er blokken "Prototype for modellvalidering" lagt til i sporene "Logisk modell" og "Studentversjon", med en kobling mellom dem. Formålet med denne iterasjonen er fortsatt å bekrefte at den logiske modellen passer for oppgaven, men i tillegg vil resultatet av iterasjonen fungere som en prototype på studentversjonen og vil jobbes videre med som en del av studentversjon-sporet, så blokken finnes i begge sporene.

Iterasjon 3

Tredje iterasjon bruker prototypen fra forrige iterasjon som grunnlag for studentversjonen og legger til de første interaktive elementene.



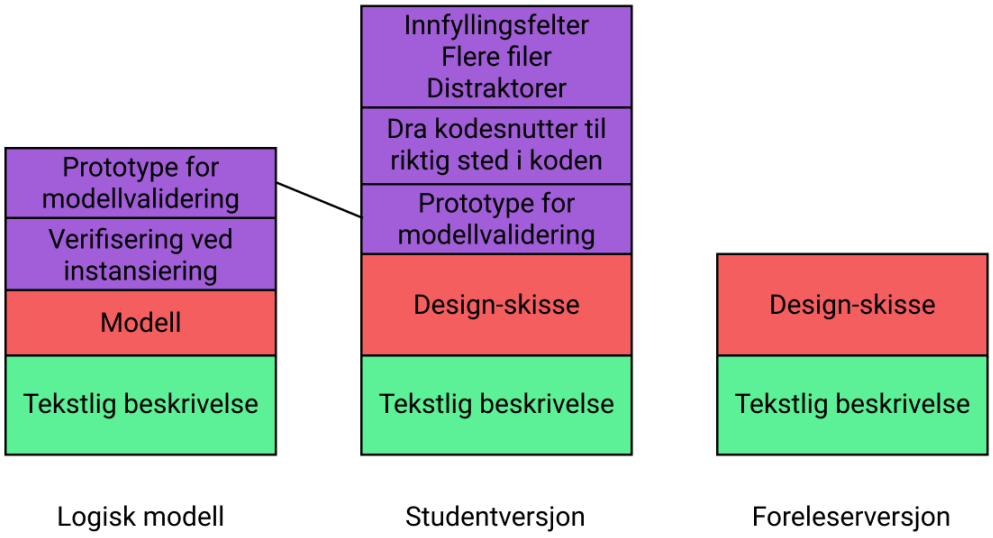
Figur 3.4: Iterasjon 3

I figur 3.4 er blokken "Dra kodesnutter til riktig sted i koden" lagt til i sporet "Studentversjon".

I studentversjonen som lages i denne iterasjonen vil man fortsatt være avhengig av eksempeldata fra iterasjon 1 for å kunne teste tillegget. Dersom man vil teste noe som ikke er blitt dekket av den eksempeldataen er man nødt til å manuelt lage ny eksempeldata. Foreleserversjonen som også skal lages vil kunne lage denne typen data som kunne blitt brukt til testing, men den er ikke laget enda. Det hadde vært mulig å prioritere foreleserversjonen over videreutvikling av studentversjonen etter at man er ferdig med å verifisere og validere den logiske modellen, og det alternativet er utforsket mer i subseksjon 3.1.3 "Alternativ plan".

Iterasjon 4

Fjerde iterasjon bygger videre på studentversjonen og legger til mer avansert funksjonalitet. Etter denne iterasjonen skal studentversjonen støtte oppgaver der man skal skrive inn kode manuelt i stedet for å dra kodesnutter til riktig sted, det skal være støtte for oppgaver som bruker flere kodefiler, og det skal være støtte for distraktorer, kodesnutter som ikke er en del av løsningen men som er lagt til for å gjøre oppgaven mer utfordrende.

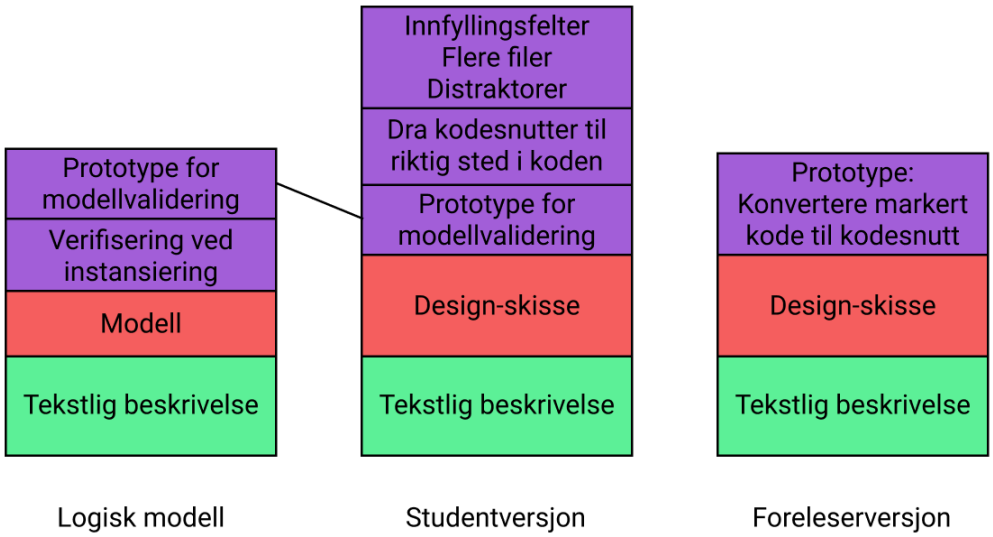


Figur 3.5: Iterasjon 4

I figur 3.5 er blokken "Innfyllingsfelter - Flere filer - Distraktorer" lagt til i sporet "Studentversjon"

Iterasjon 5

Femte iterasjon består av å lage en prototype av foreleserversjonen for å kunne lage oppgaver som kan løses i studentversjonen. Prototypen vil støtte å fjerne kodesnutter fra kodefiler som så skal plasseres på riktig sted igjen når oppgaven løses i studentversjonen

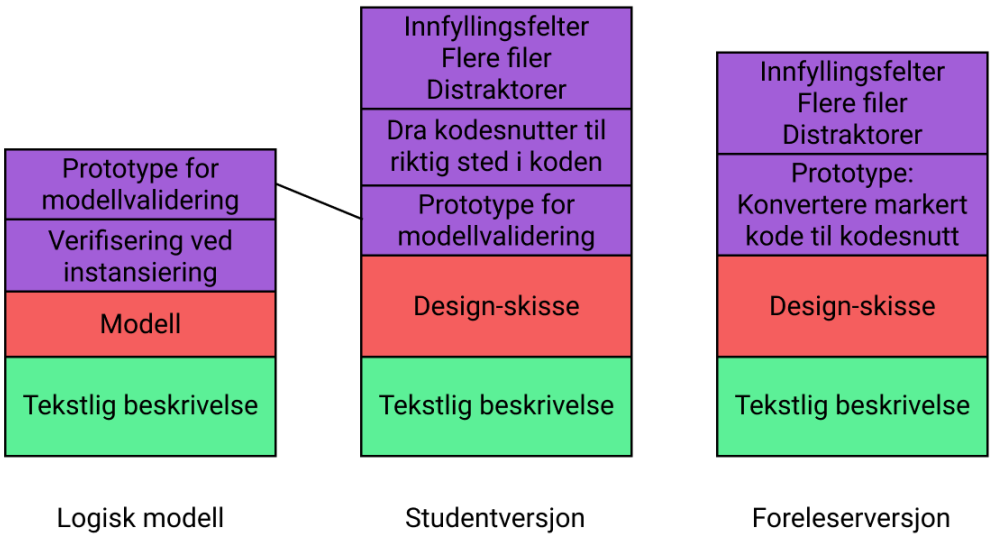


Figur 3.6: Iterasjon 5

I figur 3.6 er blokken "Prototype: Konvertere markert kode til kodesnutt" lagt til i sporet "Foreleserversjon"

Iterasjon 6

Sjette iterasjon bygger videre på foreleserversjon-prototypen, og legger til tilsvarende funksjonalitet som ble lagt til i studentversjonen i iterasjon 4.

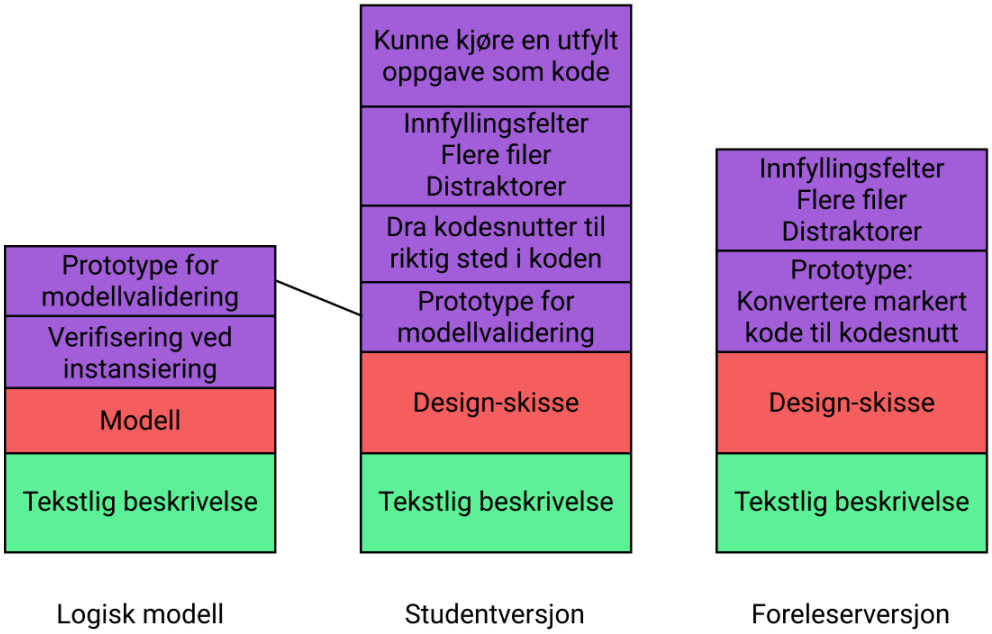


Figur 3.7: Iterasjon 6

I figur 3.7 er blokken "Innfyllingsfelter - Flere filer - Distraktorer" lagt til i sporet "Foreleserversjon"

Iterasjon 7

Syvende iterasjon skal legge til funksjonalitet i studentversjonen for å tillate at utfylt oppgave skal kunne kompileres og kjøres. Dette gjøres først etter at foreleserversjonen er laget fordi å teste denne funksjonaliteten grundig vil kreve mange forskjellige typer oppgaver, inkludert oppgaver som er basert på større og mer kompliserte kodeprosjekter, som vil være mye jobb å konvertere til oppgaver manuelt.



Figur 3.8: Iterasjon 7

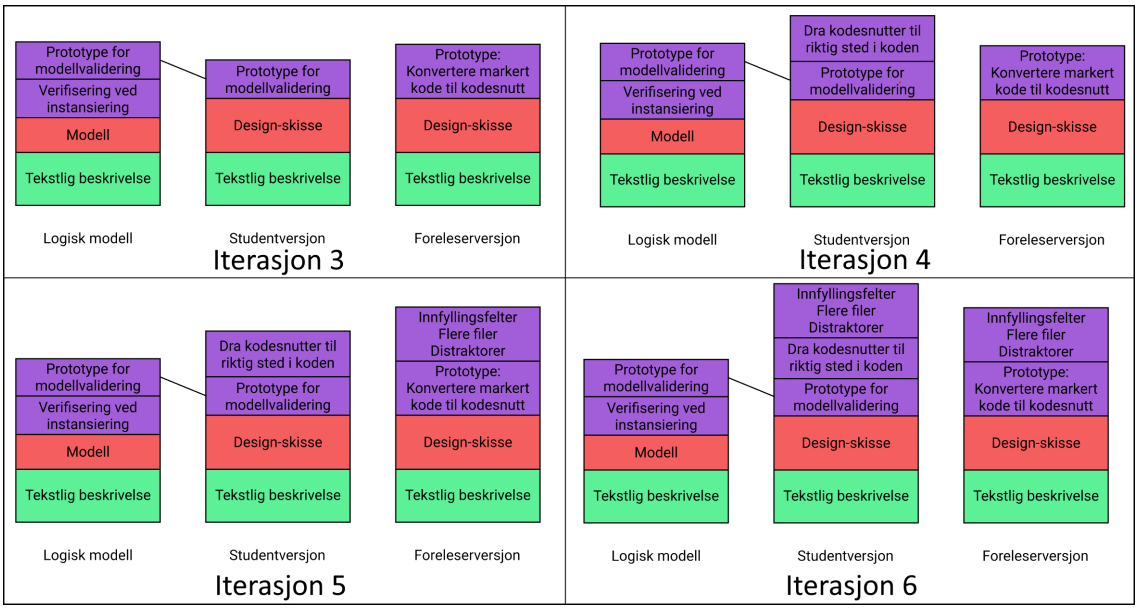
I figur 3.8 er blokken "Kunne kjøre en utfylt oppgave som kode" lagt til i sporet "Studentversjon".

3.1.3 Alternativ plan

I planen for iterasjoner i dette prosjektet er det lagt opp til at studentversjonen skal prioriteres først og jobbes mye med før man går over til foreleserversjonen. Dette medfører at man må bruke manuelt konstruerte oppgaver for testing, som vil bety noe ekstra arbeid med å manuelt lage oppgaver sammenlignet med om man hadde laget foreleserversjonen tidligere i prosessen og brukt den for å lage oppgaver til testing.

Grunnen til at studentversjonen likevel blir prioritert over foreleserversjonen i dette prosjektet er at det medfører minst risiko for å måtte gjøre mye arbeid om igjen, ved å gjøre mye fremgang på studentversjonen og potensielt avdekke problemer som oppstår når man introduserer mer komplisert funksjonalitet, sammenlignet med om man implementerer den enkle funksjonaliteten både i foreleserversjonen og studentversjonen for så å senere oppdage at den må endres begge steder.

Dersom man skulle gått for en plan som prioriterte foreleserversjonen over studentversjonen kunne man erstattet iterasjonene fra tre til seks med denne planen:



Figur 3.9: Alternativer til iterasjon 3-6

I figur 3.9 er det lagt opp en plan som bygger ut funksjonalitet i studentversjonen og foreleserversjonen side om side i stedet for å bygge så mye som mulig av studentversjonen ferdig først.

I denne planen utvikles ny funksjonalitet i foreleserversjonen først, slik at denne kan brukes for å generere oppgaver til studentversjonen som gjør det mulig å teste studentversjonen grundig uten å måtte lage mange nye oppgaver manuelt.

Etter disse fire iterasjonen vil både studentversjonen og foreleserversjonen være på samme sted som etter iterasjon 6 i hovedplanen. Iterasjon 7 fra hovedplanen handler hovedsaklig om studentversjonen og vil sannsynligvis ikke trenge så mange endringer i foreleserversjonen, så den vil komme som iterasjon 7 også i denne planen.

Elementer fra denne planen kan brukes dersom det viser seg at det å lage oppgaver i forbindelse med testing av funksjonalitet eller brukertesting av systemet blir en flaskehals for fremgangen i prosjektet.

3.2 Lean utvikling

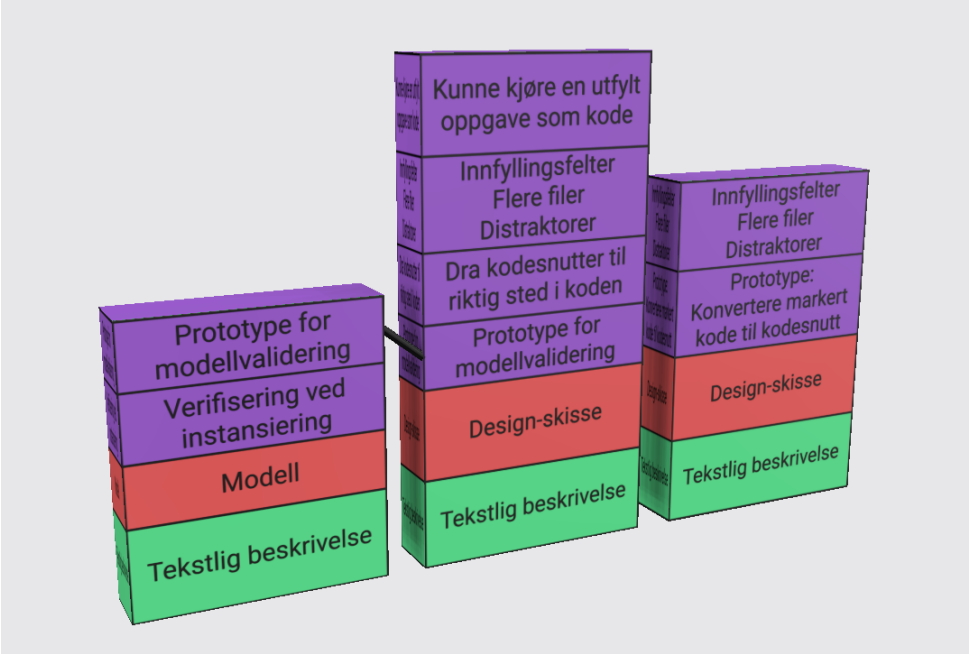
Lean utvikling, eller "slank" utvikling direkte oversatt til norsk, er et tankesett for systemutvikling som er basert på et produksjonssystem utviklet av Toyota på 1940-tallet. Denne tankegangen ble først adaptert for systemutvikling i boka "Lean Software Development: An Agile Toolkit" av Mary Poppendiek og Tom Poppendiek fra 2003.[13]

Lean utvikling handler hovedsaklig om å eliminere alle unødvendige elementer (gjerne kalt "waste"), og å levere så tidlig som mulig. Et viktig konsept i lean utvikling er "MVP", "Minimum Viable Product", eller "minste brukbare produkt" på norsk. Dette kan defineres som den versjonen av produktet som lar deg samle inn maksimalt med info med minimal innsats. Ved å utvikle en MVP får man mulighet til å teste produktet og få tilbakemeldinger fra kunder eller brukere så tidlig som mulig i prosessen, og man kan bruke disse tilbakemeldingene for å styre den videre

utviklingen.[14]

3.2.1 Lean hovedplan

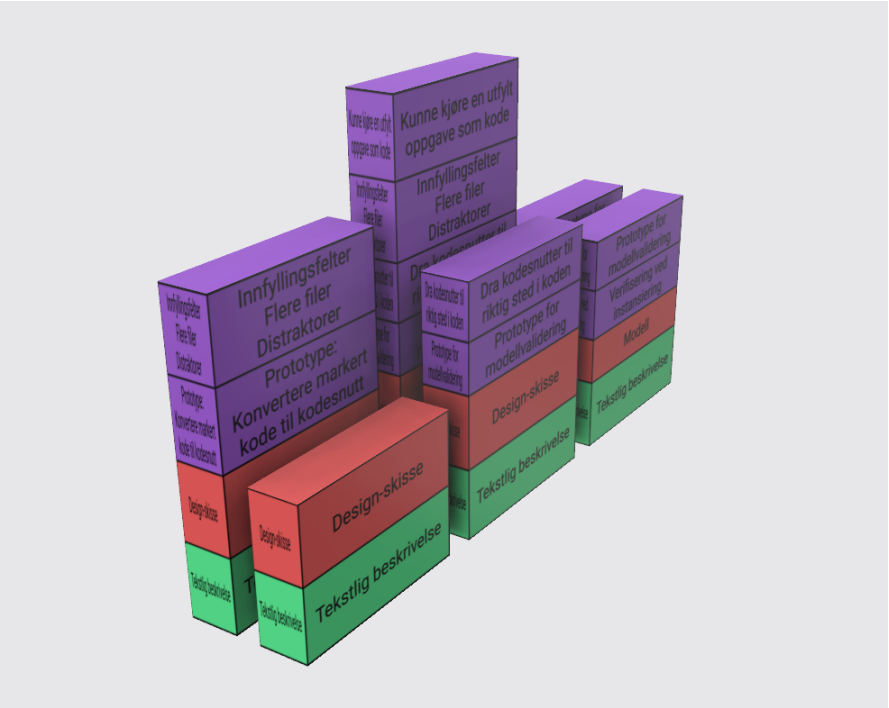
I tillegg til at funksjonaliteten i systemet utvikles trinnvis, visualisert med klosser som er lagt oppå hverandre i figurer som figur 3.8, kan man også tenke på utviklingen i en annen dimensjon, og gi klossene en dybde.



Figur 3.10: Visualisering av utviklingsprosess med dybde

For å gjøre rask fremgang med utviklingen og eliminere mest mulig risiko tidlig i prosjektet, kan også denne dybden kuttes ned på, og man kan bruke prinsipper fra lean utvikling og lage en slankere versjon. I den logiske modellen som er beskrevet i figur 2.4 er det med mange detaljer som ikke er strengt tatt nødvendige for å kunne lage en versjon som er mulig å teste, blant annet støtte for forskjellige oppgavetyper og beskrivelse av hvordan en utfylt beskrivelse kan lagres, som kan være mulig å utsette implementasjon av til senere i prosjektet, for å kunne få et testbart produkt (MVP) så tidlig som mulig, og så ofte som mulig når det legges til nye elementer. Ved å lage testbare versjoner tidlig og ofte er det mulig å få evalueringer fra utenfor prosjektet som kan gi ny innsikt.

Når man har utviklet denne tynne versjonen av de forskjellige klossene og fått testet de aspektene man ønsker å teste kan man jobbe videre med prosjektet ved å lage nye rader med klosser som utvider funksjonaliteten fra å være så tynn som mulig og fører prosjektet nærmere en endelig versjon.



Figur 3.11: Visualisering av utviklingsprosess med 2 rader med klosser som har blitt utviklet i ulik grad (sett fra baksiden)

I figur 3.11 vises det en oppstilling av blokkene slik som i figur 3.10 men med en ny rad med klosser stilt opp bak. Den første raden representerer en MVP som kun implementerer det som er absolutt nødvendig, mens raden bak utvider funksjonaliteten videre fra bunnen og opp.

I praksis vil ikke utvidelsen og oppbygning av nye rader skje så jevnt som tenkt her fordi noen blokker vil være avhengige av mer dybde i andre blokker, for eksempel vil den enkleste mulige versjonen av støtte for innfyllingsfelter kreve at den logiske modellen støtter innfyllingsfelter, men innfyllingsfelter er ikke strengt tatt nødvendig for å lage en testbar versjon av den logiske modellen, så modellen må utvides litt mer før det er mulig å legge på alle blokkene.

Det er et mål for prosjektet å både bruke det blokkbaserte oppsettet fra hovedplanen i subseksjon 3.1.2 sammen med tanken om å redusere dybden der det er mulig. Disse to tankene sammen vil kunne gjøre at prosjektet har en testbar versjon så tidlig som mulig samtidig som prosessen med å jobbe videre fra den tidlige versjonen skal bli så smertefri som mulig.

3.2.2 MVP

Det er et mål i dette prosjektet å utvikle en MVP (Minimum Viable Product) som kan brukes til brukertesting så tidlig som mulig. Samtidig er det viktig at det som testes er en god representasjon av hvordan det endelige produktet kommer til å bli, så det er viktig å finne en balanse mellom å få hatt brukertesten tidlig og å ha muligheten til å teste en versjon som er nærmere det endelige produktet og få tilbakemelding på flere typer funksjonalitet.

Versjonen som brukes til brukertesting vil ikke vil være den aller første versjonen der

det er mulig å løse noe som minner om et Parsons-problem. Når så mye er utviklet er det sannsynlig at det å utvide systemet med litt mer funksjonalitet som man kan få tilbakemeldinger på vil være mer nyttig enn å få tilbakemeldingene på det helt grunnleggende systemet litt tidligere. Spesielt for å kunne si noe om hvor godt egnet systemet er for å ta i bruk i utdanning, burde det minst være funksjonsrikt nok til at det kan brukes til utprøving i introduksjonsemner i programmering, som for eksempel TDT4110 - Informasjonsteknologi, grunnkurs.

Akkurat hvilken funksjonalitet som er hensiktsmessig å ta med i MVP-en som brukes til brukertesting er vanskelig å bestemme på forhånd. Ved inngangen til dette prosjektet er det fortsatt mye usikkerhet rundt utviklingen av tillegget og bruk av VS Code API-et som gjør at det er vanskelig å beregne hvor mye tid ulike deler av utviklingen vil ta.

Det som er mulig å bestemme på forhånd er at det er viktig at testingen skjer tidlig, slik at tilbakemeldinger om hva som fungerer bra, hva som er nyttig, og hva som mangler, kan brukes til videre utvikling. Etter at det er utviklet en versjon som er mulig å bruke på et helt grunnleggende nivå, burde det kontinuerlig vurderes hvorvidt det er mer nyttig å utsette brukertestingen for å utvikle mer funksjonalitet som kan testes, eller om det er mer nyttig å samle inn informasjon for å styre den videre utviklingen i riktig retning.

3.3 Brukersentert utvikling

Brukersentrert utvikling er en tilnærming til systemutvikling som fokuserer spesifikt på å gjøre systemer brukbare.[15] I [16] beskrives tre prinsipper for brukersentrert utvikling:

- Tidlig fokus på brukere og oppgave
  - Strukturert og systematisk informasjonsinnsamling
  - Designere opplært av domeneeksperter før datainnsamling
- Empirisk måling og testing av produktbruk
  - Fokus på at systemet er lett å lære og lett å bruke
  - Teste prototyper på faktiske brukere
- Iterativt design
  - Produktet designes, modifiseres og testes gjentatte ganger
  - Tillat total overhaling og revurdering av design ved å teste tidlig, og ved å teste konseptuelle modeller og design-idéer

Deler av disse prinsippene brukes i dette prosjektet for å sikre at det som utvikles er brukervennlig og intuitivt. Prinsippet om iterativt design passer godt sammen med iterasjoner fra smidig utvikling, og tidlig fokus på brukere og oppgaver passer godt med tanken om en MVP fra lean utvikling. Det brukersentrert utvikling tilfører i denne konteksten er å sikre at vurderingene som gjøres i prosjektet, for eksempel

etter en iterasjon i smidig utvikling, gjøres med tilgang til informasjon hentet fra relevante brukere av systemet.

Det viktigste virkemiddelet som brukes fra denne metodikken er testing av systemet med faktiske brukere. Utvikling av en MVP for testing gjør det mulig å teste relativt tidlig i prosessen, og denne testen skal gjøres med relevante brukere.

I tillegg til at det skal gjøres brukertester med relevante brukere, så gjøres også den iterative prosessen med jevnlig kommunikasjon med veileder for prosjektet, som også er en relevant bruker av systemet, men av foreleserversjonen. Dette gjør at prosjektet fremover har en tett kobling mot relevante brukere gjennom hele prosessen.

3.3.1 Brukertesting

Når en MVP som kan brukes til brukertesting er ferdigutviklet, skal den testes med relevante brukere, som for dette prosjektet vil si studenter som har emnet TDT4100 - Objektorientert programmering det inneværende semesteret.

Brukertester kan kategoriseres som formative eller summative, der formative tester fokuserer på å hente inn informasjon som skal forme den videre utviklingen, mens summative tester samler informasjon mer for å dokumentere om systemet fungerer slik det er ment. På grunn av den begrensede tidsrammen for dette prosjektet vil det kun gjennomføres én omgang med brukertester, og den vil derfor bruke både elementer fra formative tester og summative tester. Testene skal svare på hvordan studentene opplever tillegget, om mange av studentene opplever problemer som forstyrrer opplevelsen deres, og om studentene tror tillegget vil være nyttig. Disse spørsmålene svares best på av summativ testing, og dette vil være hovedfokuset i testen. De formative elementene i testen vil ikke fokusere på å forme den helt grunnleggende funksjonaliteten i systemet men heller forme hvilke utvidelser av funksjonaliteten som burde prioriteres. Den grunnleggende funksjonaliteten i systemet ble definert som en del av fordypningsprosjektet og gikk gjennom vurderingsprosseser da, som beskrevet i subseksjon 2.1.3, som gjør at det vurderes til lite sannsynlig at brukertestene vil indikere at hele konseptet burde forkastes eller overhales fra bunnen av.

Brukertesten vil gjennomføres ved å gi studentene oppgaver som bruker tillegget. Oppgavene vil være av varierende kompleksitet og vanskelighetsgrad, så de får muligheten til å prøve ut systemet uten å bli overveldet, og så bli introdusert for mer kompleks funksjonalitet gradvis. Det vil dokumenteres hvilke problemer studentene møter på og i hvor stor grad disse problemene forstyrrer bruken deres av systemet eller om det er problemer de klarer å løse selv. Denne informasjonen vil brukes for å vurdere brukbarheten til systemet, og dersom det er større problemer kan det å fikse disse problemene prioriteres i den videre utviklingen.

Studentene vil bli bedt om å tenke høyt mens de utfører testen, en etablert metode i formativ brukertesting for å hente ut så mye informasjon som mulig om hva som får brukeren til å ta de valgene de tar.[17] Dette er tatt med fra formativ brukertesting for å kunne se hvordan studentene intuitivt tror at systemet kommer til å fungere, og hvor fort de plukker opp den funksjonaliteten som ikke stemmer overens med deres forutinntagelser om systemet. Dette er spesielt viktig for dette systemet, fordi systemet er avhengig av å være lett å komme i gang med for å kunne oppfylle den rollen det er tiltenkt som beskrevet i seksjon 1.4

En detaljert rapport over hva som gjøres i brukertestene vil dokumenteres i tillegg B

Kapittel 4

Resultat

I dette kapittelet vil resultatene av prosjektet presenteres. Resultatet av prosjektet er et tillegg til VS Code, og det vil presenteres hvordan det har blitt utviklet, hvordan det er bygget opp, hvilken funksjonalitet det tilbyr, og hvilken tilstand tillegget er i ved slutten av prosjektet. Det har også blitt gjennomført brukertester av prosjektet, og resultatene fra disse vil også presenteres her.

4.1 Utvikling

Blant kravene til dette prosjektet definert i seksjon 1.4 er krav om at prosjektet skal følge prinsipper fra smidig utvikling og prinsipper fra lean utvikling. Denne seksjonen vil ta for seg hvordan utviklingen i dette prosjektet har foregått, og valg som har blitt tatt underveis.

4.1.1 Iterasjoner

I subseksjon 3.1.2 ble det definert en plan for hvordan iterasjonene i dette prosjektet kunne se ut. Å oppdatere planer underveis i utviklingen når ny informasjon samles inn er en viktig del av smidig utvikling, og som forventet skjedde det en del endringer med de planene.

Dette prosjektet endte opp med å bestå av fem iterasjoner. Resultatene fra de fem iterasjonene kan kort oppsummeres på denne måten:

- Iterasjon 1: Prototype for validering av modell
- Iterasjon 2: Prototype av grunnleggende funksjonalitet
- Iterasjon 3: MVP for bruk i brukertesting
- Iterasjon 4: Utbygging av funksjonalitet basert på resultater fra brukertest
- Iterasjon 5: Refaktorering for utvidbarhet og kodekvalitet

En detaljert gjennomgang av hva som ble gjort i hver iterasjon og hvilke vurderinger som ble gjort etter iterasjonene finnes i tillegg A: Iterasjoner.

Utviklingen har fulgt prinsipper fra smidig utvikling, og oppfylt kravet om det.

4.1.2 MVP

I subseksjon 3.2.2 ble det beskrevet hvordan en MVP skal utvikles i dette prosjektet. En MVP skal være den versjonen av produktet som lar deg samle inn maksimalt med info med minimal innsats, og for dette prosjektet vil det si at muligheten for å gjennomføre testen tidlig, må veies opp mot om det å utsette testen og utvikle tillegget videre vil øke verdien av den innsamlede informasjonen mer enn verdien av å få samlet informasjon så tidlig som mulig.

Etter iterasjon to, beskrevet i seksjon A.2, var tillegget i en tilstand som gjorde et det var mulig å gjøre oppgaver med det, og det hadde derfor gått an å holde brukertester med denne versjonen. I subseksjon 3.2.2 ble det bestemt at når det er utviklet en versjon som er mulig å gjennomføre brukertester med, skal det kontinuerlig vurderes om det er mer nyttig å utvikle videre eller holde brukertesten. Etter iterasjon 2 ble mulighetene overveid, og det ble besluttet at det skal utvikles litt mer før brukertestene holdes, fordi det er sannsynlig at det er mer verdifullt for prosjektet å få holdt brukertester av et tillegg med flere funksjoner selv om det betyr prosjektet må gå lenger uten ekstern validering fra ekte brukere.

I iterasjon 3, beskrevet i seksjon A.3, ble det lagt til ny funksjonalitet til oppgavene, og det ble laget en egen visning som viser alle oppgavefilene i arbeidsområdet. Denne visningen gjør det mer brukervennlig å åpne oppgaver, og gjør det mulig å støtte flere kodefiler i en oppgave. Det å kunne teste denne visningen og dokumentere om den er brukervennlig er veldig nyttig for prosjektet, for å kunne dokumentere at kravet om at tillegget er brukervennlig or oppnådd.

Utviklingen har fulgt prinsipper fra lean utvikling, og oppfylt kravet om det.

4.1.3 Teknologivalg

Tillegget har blitt utviklet for VS Code og ved hjelp av VS Code sitt API for tilleggsutvikling. Dette var et krav for prosjektet, og det legger mange føringer for teknologivalg i oppgaven, men det er likevel noen valg som kan gjøres. Tillegg til VS Code er i utgangspunktet skrevet i JavaScript, men det er også mulig å bruke programmeringsspråk som kan kompileres til JavaScript.

For utviklingen av dette tillegget ble det besluttet å bruke TypeScript, et programmeringspråk utviklet av Microsoft som kan kompileres til JavaScript. Hovedfordelen med å bruke TypeScript er det er typesikkert, som tilrettelegger for god programmeringsskikk og senker risikoen for kjøretidsfeil. I dette prosjektet er det et krav at tillegget programmeres på en måte som gjør det lett å utvide det med ny funksjonalitet, og det å bruke et typesikkert språk gjør det lettere for utviklere som vil utvide tillegget med ny funksjonalitet å sette seg inn i koden.

For utvikling av webvisninger brukes egentlig HTML, CSS og JavaScript, men her ble også JavaScript erstattet av TypeScript, og i tillegg ble CSS ersattet av Less, som kompileres til CSS. Grunnen til at Less ble valgt over vanlig CSS er at det tilbyr flere snarveier for å skrive CSS som både tar mindre plass og er mer lesbar. For å kunne skrive modulær kode med funksjonalitet delt opp i flere filer også i webvisningen, ble det nødvendig å bruke modulpakkeren webpack. En webvisning består av statiske filer som lastes inn i koden, og dersom funksjonaliteten er delt opp i mange JavaScript-filer og CSS-filer, må alle disse lastes inn manuelt. Ved

hjelp av webpack så ble heller systemet konfigurert slik at alle TypeScript-filene som brukes i webvisningen kompileres og pakkes til én Javascript-fil, og alle Less-filene i webvisningen pakkes til én CSS-fil. Dette webpack-oppsettet ble også utvidet til å omfatte resten av systemet, så TypeScript-filene for resten av utvidelsen også kompileres til én JavaScript-fil.

4.2 Tilstand

Denne seksjonen skal se på tilstanden til tillegget og prosjektet, og hvorvidt den svarer kravene som ble stilt i seksjon 1.4.

4.2.1 Funksjonalitet

De funksjonelle kravene som ble stilt til prosjektet var veldig grunnleggende og er ment som overordnede krav, men de er oppnådd. Det er utviklet et tillegg til VS Code, og det tilbyr muligheter for å løse og lage innfyllingsoppgaver.

En detaljert gjennomgang av all funksjonalitet som er blitt utviklet finnes i tillegg A, sammen med detaljer om hvordan det har blitt utviklet, og hvorfor det har blitt gjort på den måten. I tillegg presenteres her en kort oppsummering av den tilgjengelige funksjonaliteten.

Funksjonalitet for å løse oppgaver

- En tilpasset editor for .parson filer som viser en kodefil med mangler og et panel med kodesnutter som kan brukes for å fylle inn manglene
- En visning av alle oppgaver i arbeidsområdet, kalt "Parson Explorer"
- Støtte for flere måter å fylle inn svar
  - Drag and drop
  - Skriving
  - Dropdown
- Støtte for flere kodespråk
  - Java
  - Python
- Støtte for oppgaver som bruker flere kodefiler i samme oppgave
  - Filene vises i Parson Explorer og man kan bytte mellom dem der
  - Parson Explorer holder styr på hvor mange ubesvarte mangler det er i hver fil og viser det
- Støtte for å kompilere og kjøre kode med svarene som har blitt gitt.

Funksjonalitet for å lage oppgaver

- Kommandoer for å lage oppgaver som kan aksesseres fra kommandopaletten i VS Code

  - 'Create Parsons Problem from file" konverterer den åpne filen til en mappe som inneholder filen og genererer alle konfigurasjonsfilene for en oppgave

  - "Create Parsons Problem from folder" konverterer mappen som inneholder den åpne filen til en mappe for en oppgave ved å generere alle manglende konfigurasjonsfiler

  - 'Export Parsons Problem to file" kompilerer mappen som inneholder den åpne filen til oppgavefiler, en `.parson`-fil og en `.parsondef`-fil, og lagrer dem der det er spesifisert at de skal lagres i `parsonconfig.json`

  - "Make a gap from the currently selected text" bruker den marketerte teksten for å lage en ny mangel og lagrer den som en kommentar under linja med markert tekst.

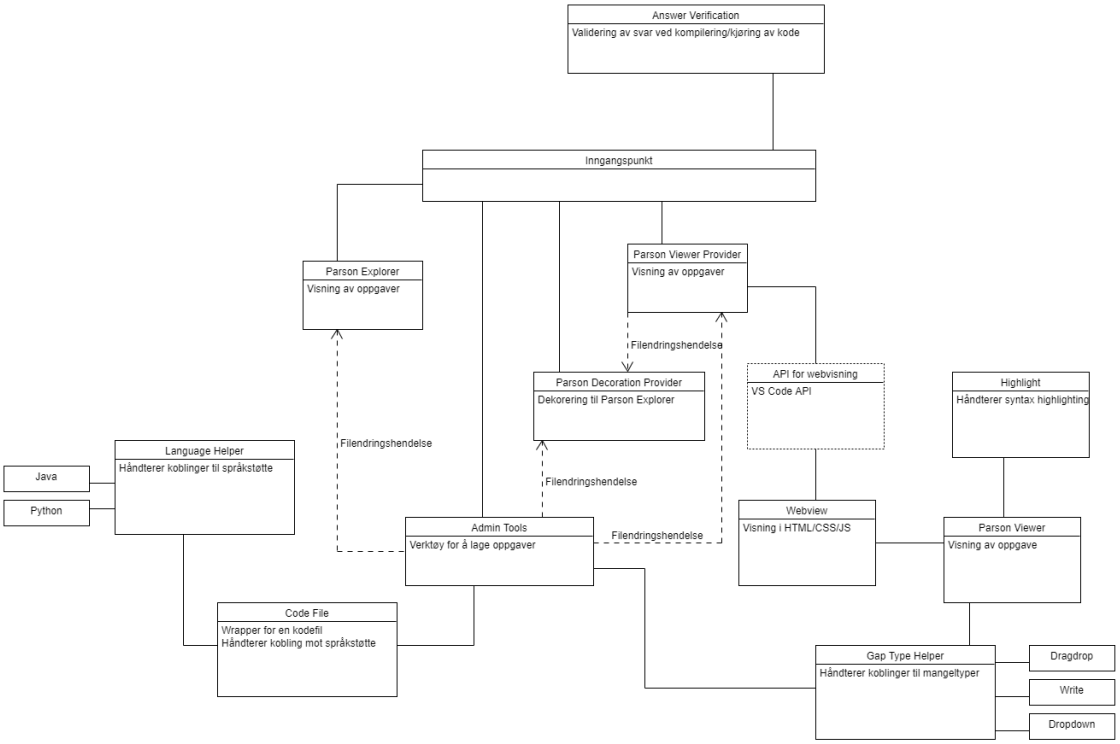
- Innstillinger som kan gjøre utvikling av oppgaver mer effektivt

  - Vis knapper for å oppdatere listen over oppgaver og innholdet i oppgaver

  - Automatisk eksporter endringer til oppgavefiler når en fil lagres i en mappe som inneholder en `.parsonconfig.json`-fil

4.2.2 Arkitektur

Denne modellen viser hvordan de forskjellige komponentene i systemet jobber sammen. Modellen fokuserer på koblingene mellom de forskjellige klassene som er utviklet for bruk i dette tillegget, så koblinger til andre komponenter som VS Code-API-et og filsystemet er ikke tatt med her, med unntakk av API-et for webvisning som måtte inkluderes for å vise hvordan webvisningen er koblet til resten av systemet.



Figur 4.1: Modell over hvordan de ulike komponentene i tillegget interagerer med hverandre. Koblinger til VS Code-API-et og filsystemet er ikke tatt med

Det mest interessante her er nok "Language Helper" og "Gap Type Helper" som ble laget for å gjøre det lett å utvide tillegget med nye programmeringsspråk og oppgavetyper. Dersom man legger til nye språk eller oppgavetyper og kobler dem på sine respektive hjelperklasser, så vil hjelperklassene håndtere koblingen til resten av systemet.

4.2.3 Integrasjon med VS Code

Et av forskningspørsmålene for dette prosjektet var hvordan et tillegg for innfyllingsoppgaver best kan integreres i VS Code. Det ble også stilt et krav om at tillegget skal være godt integrert i VS Code-arbeidsflaten.

Tillegget som har blitt utviklet i dette prosjektet har brukt API-et til VS Code for å tilby funksjonalitet på en måte som gjør at tillegget føles som en integrert del av VS Code. Den funksjonaliteten som ikke kan integreres direkte i VS Code-arbeidsflaten, det vil si den tilpassede editoren for oppgavefiler, er designet slik at den også ser ut som om den hører hjemme i VS Code. I figur A.10 vises en python-fil og en oppgave side om side, og de ser begge ut som om de hører hjemme i VS Code.

I tillegg A er det en gjennomgang av hele utviklingsprosessen, inkludert hvordan forskjellig funksjonalitet er integrert med VS Code-arbeidsflaten.

4.2.4 Utvidbarhet

Et av kravene for tillegget var at det skal programmeres på en måte som gjør det lett å utvide det med ny funksjonalitet. Å bruke etablerte kodemønstre og følge god programmeringsskikk bidrar til å gjøre det lettere å utvide med ny funksjonalitet senere, men for å oppfylle dette kravet ble det i iterasjon fem, beskrevet i seksjon A.5, gjort noen mer konkrete tiltak.

For å gjøre tillegget lett å utvide med ny funksjonalitet ble i iterasjon 5 store deler av tillegget refaktorert, slik at definisjonene av oppgavtyper og støtte for programmeringspråk kunne bli skilt ut i egne komponenter. Disse to elemente ble vurdert til å være de viktigste å ha gode muligheter for å utvide, flere oppgavetyper vil gjøre det mulig å lage mer varierte oppgaver, mens støtte for flere programmeringsspråk vil gjøre at tillegget kan brukes i flere emner og for å lære bort flere språk.

For å gjøre det lett å legge til nye oppgavetyper og programmeringsspråk ble det laget grensesnitt som definerer hva som må tilbys av en modul for å kunne integreres i systemet, og så ble det satt opp klasser som håndterer koblingen mellom resten av systemet og modulene. For å legge til et programmeringsspråk for eksempel, trengs bare å lage et objekt som implementerer grensesnittet for programmeringsspråk, og så må det objektet legges inn i hjelpeklassen for språkstøtte, og så vil automatisk alle filer med den spesifiserte filendelsen bruke den nye programmeringsspråkmodulen.

For å støtte et programmeringsspråk så må man definere både hvordan informasjon kan lagres i kommentarer og hvordan den kan hentes ut, og dette kan være litt komplisert, men mange programmeringsspråk har ganske lik syntaks for kommentarer, og derfor ble det også laget en generell løsning som hjelper med å lage støtte for alle språk som støtter kommentarer som kan gå over flere linjer med en syntaks for å starte kommentaren og en for å slutte, uten at det kreves noe annen syntaks i mellom dem.

4.2.5 Videreførbarhet

I tillegg til det som har blitt ferdig utviklet i dette prosjektet er det også blitt samlet inn og produsert informasjon som kan brukes for å veilede prosjektet videre, som resultater fra brukertester og beskrivelser og skisser av mulig funksjonalitet som ikke er blitt implementert enda. Det ble stilt et krav om at både tillegget og denne informasjonen skal være lett tilgjengelig for bruk i videreutvikling av tillegget. Deler av kravet er oppnådd, men det er også en del som mangler.

Tillegget og all kildekoden ligger tilgjengelig på Github, og kan hentes derfra av alle som er interesserte i å bygge videre på det.

Informasjonen som bør brukes for å styre den videre utviklingen er ikke like tilgjengelig. Mye av den er samlet og organisert i denne rapporten, men den er ellers ikke tilgjengelig. Det var planlagt at resultatene fra brukertestingen skulle være en del av kodelageret på Github, og at forslag til endringer og ny funksjonalitet skulle legges organiseres saker på Github, men det er ikke blitt gjort enda ved slutten av masteroppgaven.

Det var også planlagt å publisere tillegget i et register for VS Code-tillegg, som Open VSX [18], men dette ble heller ikke gjort innen slutten av masteroppgaven. Funksjonsmessig er tillegget egentlig klart for å publiseres, men det gjenstår jobb

med å se om det er noen spesielle krav som stilles til tillegg som skal publiseres og eventuelt oppfylle dem, og selve prosessen med å publisere tillegget.

4.3 Resultater fra brukertest

Som en del av prosjektet har det blitt holdt brukertester med studenter som er relevante potensielle brukere av systemet. Målet med testene var primært å dokumentere at tillegget er intuitivt og brukervennlig å bruke, men det var også et mål å samle informasjon om hva som burde prioriteres fremover i prosjektet.

Detaljert gjennomgang av brukertesten og resultatene fra den finnes i tillegg B. I tillegg presenteres her en kort oppsummering av resultatene.

4.3.1 Summative resultater

- Alle studentene åpner oppgaven uten problemer
- Ingen av studentene plasser kodesnutter uten problemer, alle forventer drag and drop-funksjonalitet. Tar likevel kort tid å komme finne riktig fremgangsmåte og svare på oppgaven.
- Alle studentene forstår distraktorer intuitivt
- Fire av åtte studenter har noe problemer med å bruke flere kodefiler i samme oppgave, men seks av åtte sier at de synes det er lett å bruke når de blir spurt
- Alle studentene forventer at lagring fungerer slik det gjør
- Alle studentene forventer at angring fungerer slik det gjør, men syv av åtte forventer at omgjøring skal fungere slik det gjør
- Alle studentene sier at de synes det fungerer bra å bruke tillegget
- Seks av åtte studenter sier at de tror de hadde hatt nytte av å kunne løse denne typen oppgaver når de skal introduseres til nye tema i programmering.

4.3.2 Formative resultater

- Alle studentene forventer at kodesnuttene har drag and drop-funksjonalitet, og fem av åtte tar opp dette som et forbedringspotensiale på slutten av testen
- Fem av fem spurte studenter er positive til dropdown-meny som en alternativ måte å svare på en oppgave
- Tre av tre spurte studenter er positive til å kunne validere at en oppgave er besvart riktig. To studenter etterspurte også mulighet for å validere svar uten å ha blitt spurt om det først
- Tre av tre spurte studenter er positive til å kunne kompilere og kjøre kode med de avgitte svarere

- To studenter ønsker muligheter for å kunne svare på oppgaver ved å skrive svaret selv.

Basert på disse testresultatene er det mulig å si at systemet er lett å komme i gang med og at det fungerer bra å bruke, og dersom drag and drop-funksjonalitet implementeres vil det være enda mer intuitivt å bruke.

Kapittel 5

Diskusjon

I dette kapittelet skal det diskuteres om prosjektet har oppnådd målene og fullført kravene som ble stilt i kapittel 1, og om forskningsspørsmålene er blitt besvart

5.1 Utvikling

I dette prosjektet ble det stilt krav om at utviklingen skal følge prinsipper fra smidig utvikling, lean utvikling, og brukersentrert utvikling.

5.1.1 Smidig utvikling

Dette prosjektet har brukt noen elementer fra smidig utvikling, herunder først og fremst oppdeling i iterasjoner. Iterasjoner i smidig utvikling har vanligvis en fastsatt lengde, noe de ikke har hatt i dette prosjektet. I stedet så ble iterasjonen definert ved å sette et mål for neste iterasjon, og så varer iterasjonen til dette målet er nådd. Dette oppnår mange av de samme fordelene, det har vært en vurderingsprosess av resultater for å bestemme veien videre etter hver iterasjon, og disse vurderingene har gjort at prosjektet ikke fulgte den planen som ble lagt opp på forhånd, men heller tilpasset planen etter ny informasjon som dukker opp under utviklingen.

Selv om iterasjonene egentlig skulle defineres basert på mål som skal gjennomføres, så ble likevel begge de to siste iterasjonene begrenset av tid i stedet, fordi det var begrenset med tid igjen som kunne brukes på utvikling, og disse iterasjonen bestod heller av å utvikle så mye som mulig på den begrensede tiden.

Sett under ett så har definisjonen av en iterasjon blitt brukt ganske løst her, men det betyr ikke nødvendigvis at arbeidet ikke har brukt prinsipper fra smidig utvikling. Det har blitt gjort flere vurderinger av resultatene så langt for å styre den videre utviklingen, og spesielt etter den iterasjonen som ble avsluttet med en brukertest ble resultatene veldig direkte brukt for å styre utviklingen i neste iterasjon.

5.1.2 Lean utvikling

Fra lean utvikling har dette prosjektet brukt konseptet om en MVP for å beslutte når det er tid for å holde brukertesten. I tillegg ble tanken om å ikke lage funksjonalitet

før du trenger brukt i de første to iterasjonene for å lage den veldig slanke prototypen som fungerte som en validering av at den konseptuelle modellen.

Prosjektet har ikke fokusert veldig mye på lean utvikling, men de prinsippene som har blitt brukt har kommet veldig godt med. Den første slanke prototypen ble utviklet raskt og gjorde det mulig å se at det er mulig med denne funksjonalitet i et VS Code-tillegg, og prinsipper for MVP gjorde at brukertesten kunne bli gjennomført på et tidspunkt som gjorde at mye kunne bli testet, men det var også tid til å bruke resultatene for å styre den videre utviklingen.

5.1.3 Brukersentrert utvikling

Dette prosjektet skulle bruke prinsipper fra brukersentrert utvikling, og det har derfor blitt gjennomført en brukertest som ble brukt for både å dokumentere kvalitetene til systemet, og for å styre den videre utviklingen. Utover brukertesten så har prosjektet hatt mange samtaler direkte med én relevant bruker, veilederen for prosjektet. Til sammen betyr det at både brukere som vil primært løse oppgaver og brukere som vil primært lage oppgaver har kommet med tilbakemeldinger i løpet av utviklingen som har blitt brukt for å styre utviklingen videre

Designarbeidet som ble gjort med den konseptuelle modellen for innfyllingsoppgavene ble derimot ikke utviklet med fokus på prinsipper fra brukersentrert utvikling, selv om den fasen av utvikling egentlig er sentral i brukersentret utvikling. Dette er potensielt negativt for prosjektet, men designet ble basert på innfyllingsoppgaver og Parsons-problemer som allerede har blitt designet og testet med brukere av andre, som gjorde det mer sannsynlig at den konseptuelle modellen likevel kan brukes for å lage et brukervennlig system.

5.1.4 Utvidbarhet

Et krav til tillegget var også at det skal være lett å utvide med ny funksjonalitet, og dette ble spesifikt addressert i iterasjon 5, beskrevet i seksjon A.5. Før denne iterasjonen ble det identifisert to punkter det vil være viktig å kunne utvide, støtte for programmeringsspråk, og måter å svare på oppgaver. Etter denne iterasjonen er disse delene av systemet blitt rimelig lette å utvide. Kravet slik det ble formulert sier at tillegget skal programmeres på en måte som gjør det lett å utvide med ny funksjonalitet, og to av de viktigste punktene er blitt lette å utvide, men det er store deler av tillegget som vil være langt fra like lett å utvide.

Underveis i utviklingen har det kanskje vært litt for lite fokus på å gjøre tillegget mulig å utvide. Det ble også stilt krav om at utviklingen skulle følge prinsipper fra lean utvikling, og fokuset på å få ferdig en MVP som kan brukes til brukertesting har fått prioritet over å passe på at tillegget er lett å utvikle. Dette er ikke nødvendigvis en dum ting, refaktoreringsprosessen i iterasjon 5 var ikke for krevende, og fordi det skjedde så sent i utviklingen var det samlet inn mye teknisk kunnskap som kom godt med i prosessen, og det er ikke sikkert at disse to delene av tillegget, som sannsynligvis er de to som er viktigst å kunne utvide, hadde vært like lette å utvide om det hadde vært et mye større fokus på å skrive kode som kan utvides underveis i stedet for å fokusere på funksjonalitet til MVP-en.

Sånn resultatet er nå så hadde nok tillegget vært tjent med enten et litt større fokus på å skrive kode som kan utvides senere under utviklingen, eller mere tid til refaktorering på slutten av prosjektet.

5.1.5 Testing

En vanlig del av utvikling av kode er å skrive tester. Det gjør at man kan verifisere at ting fungerer som det skal, og ved å kjøre testene når man gjør endringer kan man se om ting fortsatt fungerer som forventet etter endringene.

I dette prosjektet har det ikke blitt skrevet noen tester. Det er flere grunner til dette, i den første delen av prosjektet var alt fokuset på å få ferdig MVP slik at brukertester kunne gjennomføres, og da ble det ikke prioritert å skrive tester. Etter brukertestene var ferdig gikk fokuset over på å bruke informasjonen fra testene for å videreutvikle tillegget. Etter brukertestene var ferdig var det ikke så veldig mye tid igjen til utviklingen av tillegget burde stoppe, og det ble derfor ikke prioritert.

Usikkerheten rundt hvordan det vil fungere å utvikle tillegg til VS Code er noe som preget en del av valgene som ble tatt før utviklingen startet, og usikkerhet rundt hvordan det fungere å skrive tester for et VS Code tillegg er også noe av grunnen til at det ikke ble brukt mer tid på tester. Det er nok mye funksjonalitet som kunne ha blitt testet uten å måtte ta så mye hensyn til at koden brukes i et VS Code-tillegg, men det ble altså heller ikke gjort.

5.2 VS Code-tillegget

5.2.1 Funksjonalitet for å løse oppgaver

Det ble utviklet relativt mye funksjonalitet for å løse oppgaver, og basert på tilbakemeldingene fra studentene som deltok på brukertesting så kan dette være et nyttig verktøy i undervisning, men uansett hvor bra funksjonaliteten for å løse oppgaver er, så er den helt avhengig av at funksjonaliteten for å lage oppgaver også er bra, hvis ikke vil det aldri bli tatt i bruk.

5.2.2 Funksjonalitet for å lage oppgaver

Funksjonaliteten for å lage oppgaver bærer veldig preg av å være en midlertidig løsning. Den har nesten ikke noe brukergrensesnitt og bruker heller kommandoer som kan hentes frem i VS Code sitt kommandopalett eller bindes til tastatursnarveier.

Likevel fungerer det greit, og alle som er forventet å være interessert i å bruke dette tillegget for å utvikle oppgaver vil også sannsynligvis ha høyt nivå av teknisk kunnskap og domenekunnskap som gjør at det er mulig å lage oppgaver ganske effektivt likevel. Spesielt når man binder alle kommandoene til tastatursnarveier er all funksjonaliteten veldig tilgjengelig hele tiden selv om det ikke er et brukergrensesnitt for den.

5.3 Brukertest

Etter iterasjon 2 så var det egentlig nok funksjonalitet for å kunne holde en brukertest, men det ble besluttet å utsette testen og legge til en del mer funksjonalitet først. Dette var nok et riktig valg. Dersom brukertesten hadde blitt holdt så tidlig så hadde hovedresultatene sannsynligvis vært at kodesnuttene burde ha drag and drop-funksjonalitet, og at systemet funker.

Noe av grunnen til at det ble besluttet å utsette testen var også fordi det ble vurdert som usannsynlig at testen kom til å fortelle at hele konseptet burde endres, fordi det var basert på andre typer oppgaver som Parsons-problemer, men integrert i VS Code. Å ha testen tidligere kunne vært veldig nyttig dersom det var knyttet mer usikkerhet rundt hvorvidt selve konseptet var en god idé

Kapittel 6

Konklusjon og videre arbeid

6.1 Konklusjon

Programmeringslæring er noe mange synes er vanskelig, og programmering er komplisert, men ved hjelp av moderne verktøy er det kanskje mulig å lage oppgaver som kan hjelpe studenter med å lære seg å programmere. Denne oppgaven har sett på utvikling av et tillegg for VS Code som muliggjør forskjellig typer innfyllingsoppgaver som kan brukes for å la studenter fokusere på mindre tema innenfor programmering uten å måtte fokusere på alle andre aspekter ved programmering.

Mye fremgang har blitt gjort på tillegget i løpet av dette prosjektet, og basert på resultatene fra brukertestene er tillegget intuitivt, brukervennlig, og det ser ut som om det kan være nyttig for bruk i programmeringsutdanning. Det har blitt brukt flere prinsipper fra utviklingsmetodikk som smidig utvikling og lean utvikling, som også gjør det mer sannsynlig at tillegget er godt laget. Tillegget er ikke helt klart for å bli tatt i bruk, men det er veldig nærme, og mangler hovedsaklig å bli publisert i et register så det kan installeres.

6.2 Videre arbeid

Mye arbeid har blitt gjort med utvikling av tillegget, men ikke alle planer for prosjektet er gjennomført ved avslutningen av masteroppgaven.

6.2.1 Publisering

Det viktigste arbeidet som gjenstår ved slutten av prosjektet er publisering av tillegget. Tillegget er i stor grad klart for å bli publisert, men tidsbegrensninger gjorde at det ikke var mulig å sette av tid til å faktisk publisere det. Å publisere tillegget i et offentlig tilgjengelig register vil gjøre tillegget mye mer tilgjengelig for bruk både for studenter og forelesere, og det er en nødvendighet om man skal ha mulighet for å gjennomføre større tester av tillegget.

6.2.2 Gitpod

Utgangspunktet for fordypningsprosjektet som ledet til denne masteroppgaven var å se på mulighetene for å bruke Gitpod i programmeringsundervisning, og konklusjonen var at det kan utvikles et tillegg til VS Code, som vil støttes i Gitpod, for bruk i undervisning. I denne masteroppgaven har fokuset kun vært på tillegget i VS Code, og det er ikke gjort noe testing av hvordan det funker i Gitpod. Gitpod skal brukes i programmeringsutdanningen fremover, så det vil være nødvendig å sikre at tillegget fungerer som det skal også i Gitpod om tillegget skal kunne benyttes i undervisning.

6.2.3 Kodelager

Kildekoden for tillegget ligger på Github i et kodelager der kun github-bruker kape142 har tilgang til å gjøre endringer. Avhengig av planene for videreføring av tillegget burde det gjøres vurderinger på hvordan kildekoden best burde oppbevares. Blant mulighetene her er å flytte kildekoden over på Gitlab-instansen til NTNU, gi tilgang til Github-kodelageret til flere, eller å lage et nytt kodelager som er satt opp for bruk av flere bidragsytere fra starten av.

6.2.4 Utvikling av oppgaver

I kodelageret sammen med kildekoden for tillegget ligger det noen eksempeloppgaver, men om tillegget skal kunne tas i bruk i utdanning må det brukes en del tid på å utvikle gode oppgaver som benytter funksjonaliteten i tillegget på en god måte.

6.2.5 Større tester

Tillegget har blitt testet på noen studenter, og har fått gode tilbakemeldinger, men det hadde vært veldig interessant å teste det ut på en større skala, ved å for eksempel tilby oppgaver som bruker tillegget til studentene som tar TDT4110 - Informasjonsteknologi, grunnkurs neste semester. Da vil man kunne se på hvor mange som er interessert i denne typen tilbud, få mer robust data på hvordan systemet oppleves å bruke, og undersøke om studenter synes det faktisk er nyttig å bruke når de skal lære seg nye ting, i stedet for bare hva studenter tror om hvorvidt det kunne vært nyttig.

I tillegg til at tillegget må publiseres og testes ut på Gitpod før dette kan gjøres, vil også denne typen test kreve ganske mye arbeid med utvikling av oppgaver.

6.2.6 Utvikle ny funksjonalitet

Det finnes mye funksjonalitet som kan legges i tillegget, det er flere ting som ble beskrevet i fordypningsprosjektet som ikke er implementert enda, flere forslag fra brukertestene som ikke er implementert, og sikkert mange muligheter som ikke enda er påtenkt. Med utvidelsespunktene for oppgavetyper og programmeringsspråk er det i hvert fall mange muligheter der for utvikling av ny funksjonalitet.

Ett av de mer underutviklede aspektene ved tillegget er funksjonaliteten for å utvikle oppgaver, som for det meste er helt uten brukergrensesnitt. Her er det store rom for forbedringer, som vil være viktig om verktøyet skal være attraktivt for å utvikle oppgaver, noe det må være for å faktisk bli tatt i bruk.

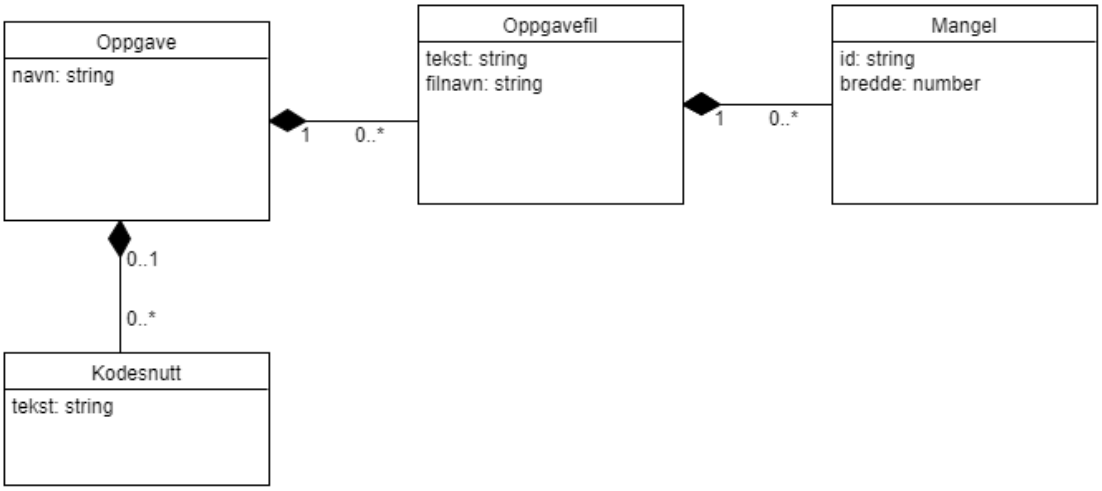
Tillegg A

Iterasjoner

I subseksjon 3.1.2 ble det beskrevet en plan for hvordan utviklingen av tilleggene kan deles opp i iterasjoner. Denne planen ble brukt som en veileder, men som forventet så ble det avdekket ting under utviklingen som gjorde at denne planen ikke ble fulgt helt nøyaktig. I dette tillegget vil iterasjonene slik de i praksis ble gjennomført dokumenteres.

A.1 Iterasjon 1

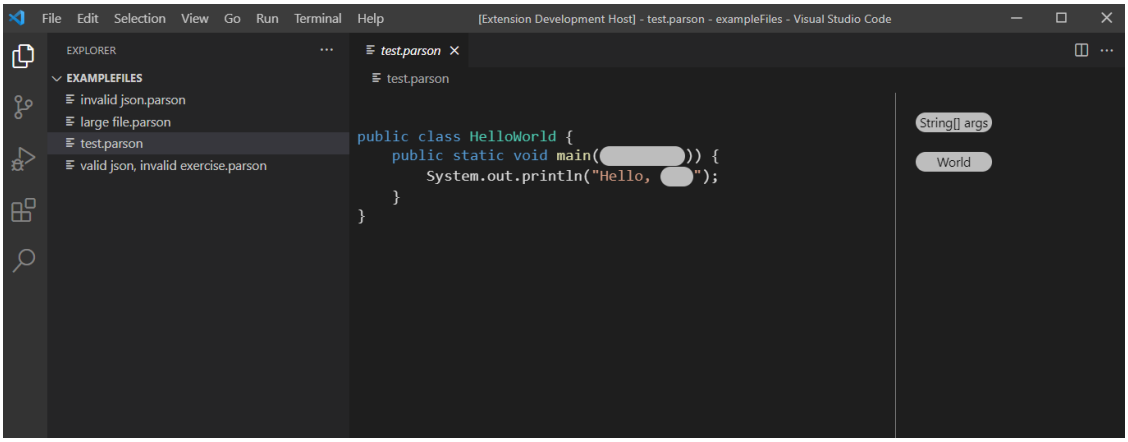
Iterasjon 1 ble i praksis en kombinasjon av iterasjon 1 og 2 fra hovedplanen, men en veldig slank versjon for kjapp testing. I stedet for å bruke hele den logiske modellen fra fordypningsprosjektet ble det laget en versjon av den logiske modellen som kun inneholdt de delene som var helt nødvendig for å kunne lage en tidlig prototype av studentversjonen uten noen interaktive elementer.



Figur A.1: Første og slankeste modell

I figur A.1 vises den logiske modellen slik som den ble brukt i iterasjon 1. Sammenlignet med figur 2.3 er modellen mye mindre omfattende, og det er i tillegg gjort noen endringer i hvilke egenskaper en mangel har, spesifikt har den fått en egenskap "id" som ble nødvendig for å kunne skille mellom manglene i en oppgave, og som

også brukes for å definere posisjonen til en mangel i en tekst i stedet for egenskapen "posisjon". På grunn av hvordan dette ble implementert i koden så er støtte for distraktorer allerede lagt inn i systemet, selv om det ikke skjer før i iterasjon 4 i hovedplanen. Andre endringer og kutt som er gjort i forhold til utgangspunktet fra fordypningsprosjektet er antatt at skal implementeres senere, men var ikke nødvendig for å få til den grunnleggende funksjonaliteten i prototypen.



Figur A.2: Skjermbilde av hvordan tillegget ser ut etter iterasjon 1

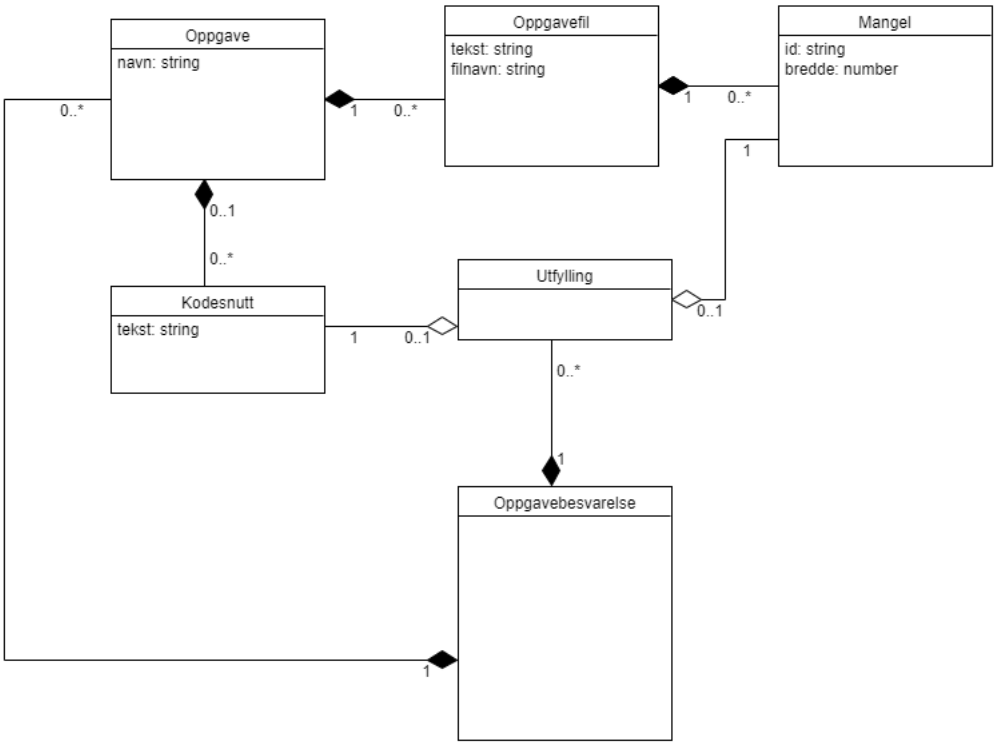
A.1.1 Resultater og plan for neste iterasjon

Resultatene i denne iterasjonen viser at det grunnleggende designet og modellene som ble laget i fordypningsprosjektet kan med mindre justeringer brukes for å lage et fungerende tillegg. Basert på dette kan neste iterasjon gå videre med å legge til mer funksjonalitet fra hovedplanen.

A.2 Iterasjon 2

Iterasjon 2 ble i praksis ganske lik som iterasjon 3 fra hovedplanen, og la til interaktivitet til prototypen, men en litt enklere versjon enn det som var planlagt i hovedplanen. I stedet for at man kan dra kodesnutter til riktig sted i koden velger man en kodesnutt ved å klikke på den, og klikker på den mangelen som skal fylles av kodesnutten. Denne endringen ble gjort for å kjappere komme videre med prototypen, men planen er at ordentlig "drag and drop" funksjonalitet skal implementeres på et senere tidspunkt i prosjektet.

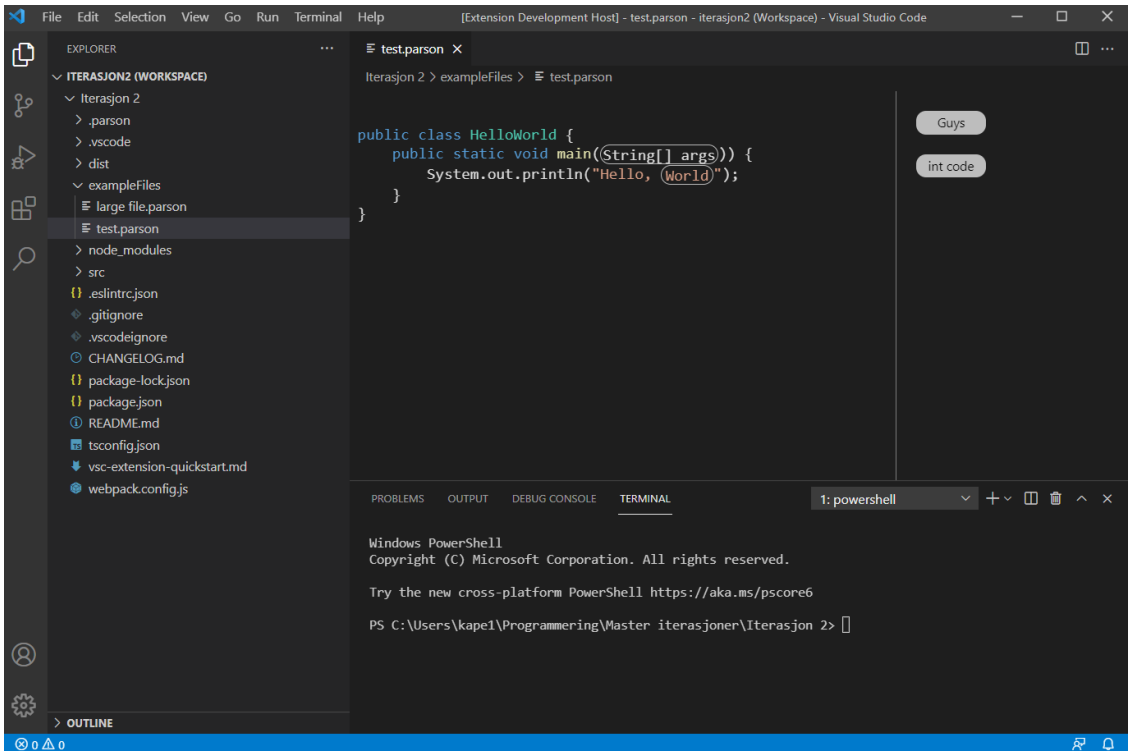
I iterasjon 1 så var modellen laget for å inneholde informasjon om en oppgave som skal løses, slik som i figur 2.3, men for at modellen skal kunne inneholde informasjon om hvilke kodesnutter som er blitt plassert i hvilke mangler så må modellen utvides og ta inn elementer fra figur 2.4. Den slankeste mulige modellen som støtter funksjonaliteten som var ønsket i prototypen etter iterasjon 2 er modellert i figur A.3



Figur A.3: Andre modell, med støtte for lagring av utførelser

I forhold til modellen fra fordypningsprosjektet er det også her gjort noen kutt og noen endringer, det som er fjernet er planlagt at skal implementeres senere, men en endring som er gjort er at en Oppgavebesvarelse nå inneholder en referanse til en oppgave, som gjør det enklere å lagre en all nødvendig info i en JSON-fil.

Som en del av iterasjonen ble det også implementert funksjonalitet for å lagre en oppgavebesvarelser til fil og laste dem inn igjen, så fremgang på en oppgave kan lagres. En oppgavebesvarelse-fil har en referanse til en oppgave-fil og inneholder ikke oppgaven selv, slik at når en oppgavebesvarelse-fil lagres vil ikke oppgave-filen påvirkes, og flere oppgavebesvarelses-filer kan bruke den samme oppgave-filen uten at fremgang på én oppgavebesvarelse påvirker den andre.



Figur A.4: Skjermbilde av hvordan tillegget ser ut etter iterasjon 2

A.2.1 Resultater og plan for neste iterasjon

Resultatene etter denne iterasjonen er et tillegg der det er mulig å løse oppgaver basert på Parsons-problemer direkte i VS Code. Det hadde vært mulig å bruke tillegget sånn det fungerer nå til en brukertest, men tillegget er ikke veldig brukervennlig enda, og en test av brukbarheten til denne versjonen vil ikke være så nyttig for å vurdere brukbarheten til den endelige versjonen. Neste iterasjon burde derfor fokusere på å gjøre tillegget mer brukervennlig. I tillegg er det ikke gitt at det er mer nyttig å utføre brukertesten så fort som mulig enn å utvikle noe mer funksjonalitet først slik at den også kan brukbarhetstestes. Spesielt funksjonalitet som utvider brukergrensesnittet er nyttig å brukbarhetsteste, og for eksempel støtte for å bruke flere kodefiler i en oppgave er en god kandidat for å utforske de tekniske mulighetene for å implementere.

A.3 Iterasjon 3

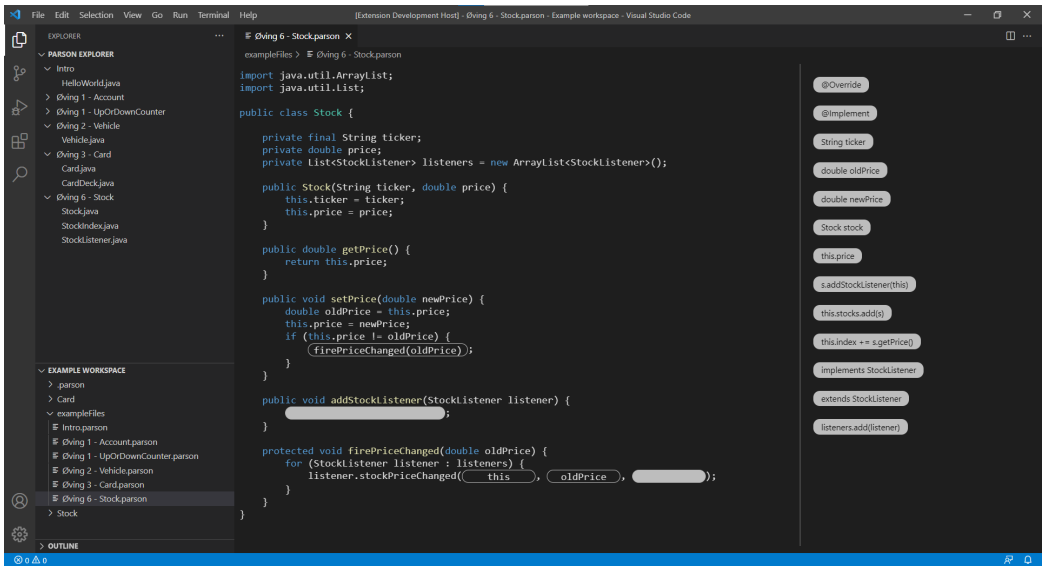
Fokuset for denne iterasjonen var å gjøre tillegget klart for brukertest, men det ble også lagt til en del funksjonalitet.

A.3.1 Støtte for flere filer

Den nye funksjonaliteten i studentversjonen denne iterasjonen er støtte for at en oppgave kan bruke flere filer, som er lagt til iterasjon 4 i hovedplanen. Dette ble prioritert før brukertestingen fordi det å bytte mellom filer krever et brukergrense-

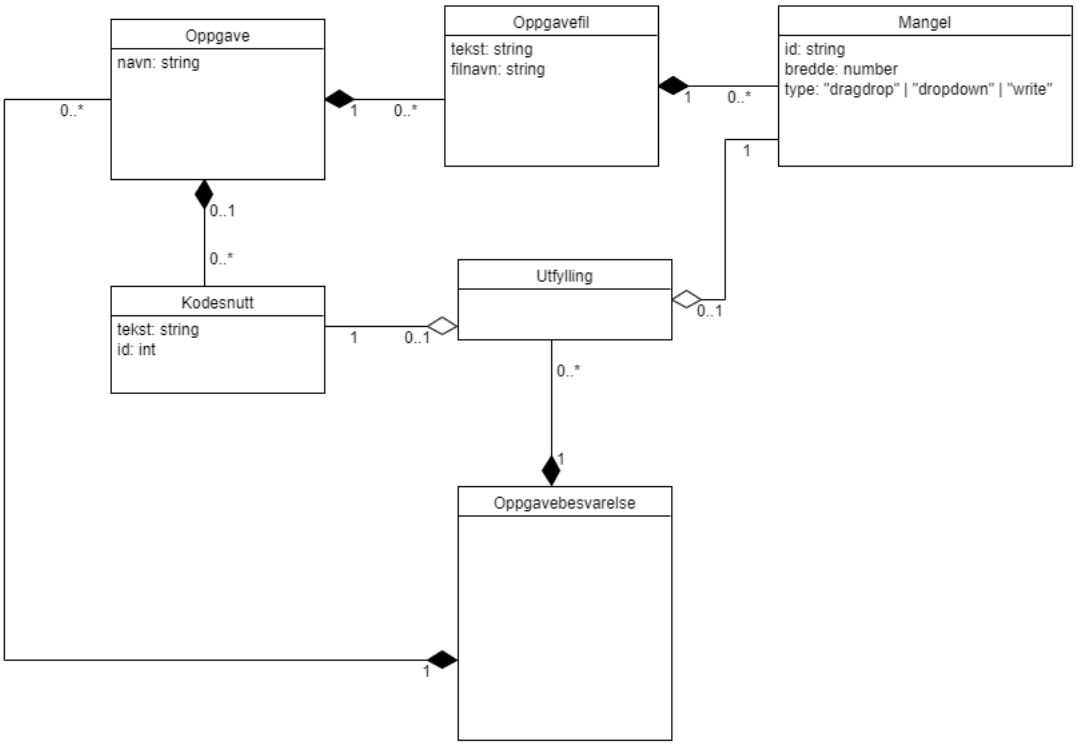
snitt, og da er det ønskelig å brukbarhetsteste det slik at man kan enten dokumentere at det er brukervennlig, eller jobbe for å gjøre det mer brukervennlig etter brukertestingen. Å teste funksjonaliteten for å støtte flere filer er også viktig for å kunne vurdere om dette tillegget vil være nyttig for større problemer og mer kompliserte tema i emner som TDT4100 - Objektorientert programmering, som ikke er mulig å dekke om man er begrenset til én kodefil per oppgave.

Som en del av utvikling av støtte for at en oppgave kan bruke flere kodefiler ble det utviklet en egen visning i brukergrensesnittet som viser alle oppgavene som finnes i arbeidsområdet som er åpent. Hvordan dette ser ut kan sees til venstre i figur A.7.



Figur A.5: Skjermbilde av hvordan tillegget ser ut når man løser oppgaver etter iterasjon 3

Det ble også gjort noen få endringer på modellen i denne iterasjonen. Det ble lagt til egenskapen ”id” på Kodesnutt, slik at også disse kan unikt identifiseres. Dette ble oppdaget at var nødvendig når mer kompliserte oppgaver ble laget og det at det ikke var mulig å unikt identifisere de forskjellige kodesnuttene skapte problemer med lagring av hvilken kodesnutt som hørte til hvilken mangel. I tillegg ble Mangel utvidet egenskapen ”type” som forteller hvilken type oppgave mangelen er. Denne egenskapen er ikke tatt i bruk enda i denne iterasjonen. Den nye modellen kan sees i figur A.6



Figur A.6: Modellen etter iterasjon 3

A.3.2 Støtte for utvikling av oppgaver

I denne iterasjonen gikk det også en del tid til utvikling av oppgavene som skulle gis i brukertestene, og det ble laget noen oppgaver manuelt først, men for de mer kompliserte oppgavene med flere kodefiler ble det veldig krevende arbeid, og som foreslått mot slutten av subseksjon 3.1.3 ble det tatt noen elementer fra den alternative planen for å unngå at for mye tid ble brukt på utvikling av oppgaver på en ineffektiv måte. Planen var originalt at dette skulle være et separat tillegg, men funksjonaliteten ble i stedet implementert i det samme tillegget som studentversjonen. Hovedgrunnen til at det ble gjort på denne måten i denne iterasjonen var for å ikke trenge å bruke tid på å sette opp et nytt prosjekt for det andre tillegget, og bygge det opp fra bunnen av.

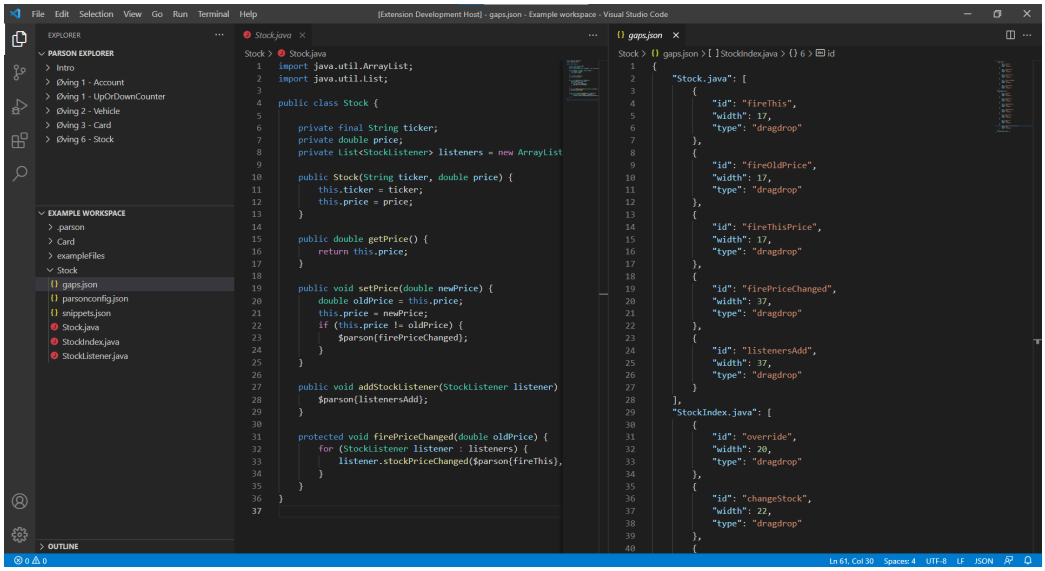
Det som ble utviklet av funksjonalitet for å lage oppgaver ble også en del annerledes enn slik det var planlagt i hovedplanen. I iterasjon 5, som er den første iterasjonen som jobber med foreleserversjonen, så er det planlagt at det skal lages en prototype som kan konvertere markert kode til kodesnutt, men den funksjonaliteten ble ikke med i denne iterasjonen. I stedet ble fokuset på å ta kodefiler som inneholder koden som skal brukes i oppgaver og kompilere dem til oppgavefiler som kan åpnes og løses i tillegget.

Denne funksjonaliteten ble designet og planlagt i samarbeid med veileder som i denne sammenhengen også fungerer som en ekte bruker av systemet.

Det som konkret ble lagt til av funksjonalitet var tre kommandoer som kan kjøres i VS Code. En kommando for å forberede en enkelt kodefil til å kunne konverteres til en oppgave, en kommando for å forberede en mappe med kodefiler til å kunne

konverteres til en oppgave, og en kommando for å kompilere en mappe med én eller flere kodefiler til en oppgave. Å forberede kodefiler til å kunne konverteres til en oppgave vil si at det genereres støttefiler som inneholder innstillinger, info om mangler og info om kodesnutter som legges i samme mappe som kodefilene. Dersom kommandoen brukes på en enkeltfil, så vil en ny mappe genereres som inneholder enkeltfilen, og støttefilene. Kommandoen for å kompilere en mappe til en oppgave samler informasjonen fra kodefilene og støttefilene og lager en `.parson` og en `.parsondef` fil som sammen kan åpnes og løses i tillegget.

I figur A.7 kan en mappe som er klar for å kompileres sees i filtreet til venstre, mens en kodefil der deler av koden er erstattet av idenfikatorer på formatet $parson{identifikator} sees i panelet til venstre, mens filen som definerer manglene, som bruker samme identifikatorer som finnes i koden, kan sees i panelet til høyre.



Figur A.7: Skjermbilde av hvordan tillegget ser ut når man lager oppgaver etter iterasjon 3

A.3.3 Resultater og plan for neste iterasjon

Resultatet av denne iterasjonen er en versjon av tillegget som er helt klart for å brukes til brukertester.

Resultatene fra brukertestene er dokumentert i tillegg B: Brukertestrapport. Disse resultatene dokumenterer at tillegget er brukervennlig, og at det tilbyr funksjonalitet som kan være nyttig. I tillegg inneholder resultatene informasjon om hvilke endringer og hvilken ny funksjonalitet som er mest ønsket av studentene som har testet tillegget. Basert på resultatene fra brukertesten er det aller viktigste å endre hvordan man svarer på oppgaver til å bruke drag and drop. Ellers burde også annen funksjonalitet utvikles basert på ønsker fra brukertestene og planene fra fordypningsprosjektet, men hva mer som skal utvikles og hvor mye mer som skal utvikles avhenger av teknisk utforskning av hvor krevende det er å implementere som vil måtte gjøres som en del av neste iterasjon.

I tillegg til resultatene fra brukertesting, så har også nye samtaler med veileder ført til noen forslag og ønsker til foreleserdelen av tillegget. For å gjøre det mer effektivt

å utvikle nye oppgaver er det ønskelig at systemet kan hente inn nye oppgaver automatisk når kodefilene som definerer en oppgave kompileres, slik at man kan raskere teste endringene man har gjort. Det er også et ønske at kodefilene som definerer oppgavene fortsatt kan kjøres som vanlig selv om de inneholder informasjon om mangler, det vil si at man ikke kan fjerne teksten og erstatte den med en identifikator, all informasjon må i stedet legges i kommentarer.

På grunn av at det er begrenset med tid igjen av prosjektet, og at tillegget allerede er funksjonelt, så er det ikke så veldig hensiktsmessig å definere nøyaktig hvilken funksjonalitet som skal fullføres i neste iterasjon, men å heller begrense den på tid. Det er ønskelig at så mye som mulig av funksjonalitet som ble foreslått i brukertestene og fordypningsprosjektet blir implementert, men fordi det begynner å nærme seg prosjektslutt og tillegget også burde gå gjennom refaktorering, og det er mye dokumentering som skal gjøres, så vil målet for den neste iterasjonen å implementere så mye funksjonalitet det er tid til før utvikling av ny funksjonalitet bør fryses for å ha tid til å fullføre resten av prosjektet.

A.4 Iterasjon 4

Målet med iterasjon 4 var å utvikle mer funksjonalitet til tillegget, nå som det er blitt testet og det er dokumentert at det fungerer godt for å løse programmeringsoppgaver og det er blitt samlet informasjon som kan brukes for å prioritere hvilken funksjonalitet som skal utvikles videre.

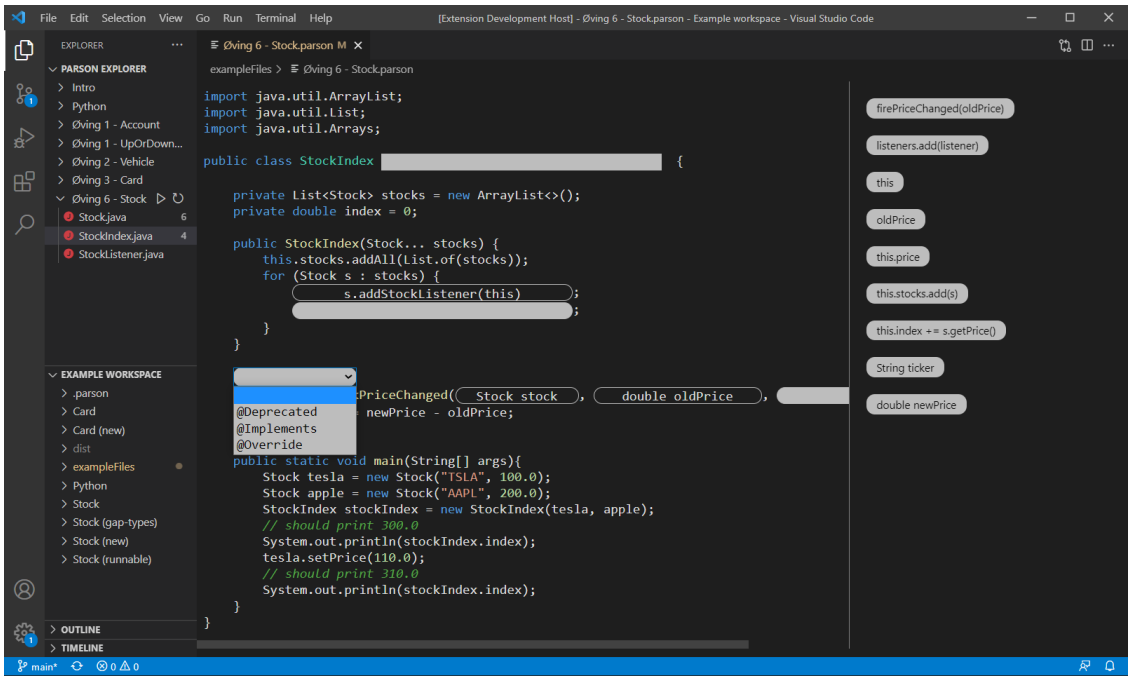
A.4.1 Funksjonalitet for løsing av oppgaver

Basert på resultatene fra brukertestene ble den høyest prioriterte funksjonaliteten å implementere drag and drop for kodesnuttene. Dette ble gjort ved hjelp av drag and drop API-et i HTML, så det oppfører seg slik man er vant til fra andre applikasjoner og nettsider.

Etter det ble det lagt inn støtte for to nye måter å fylle inn mangler på, ved å skrive inn svar manuelt, og dropdown-menyer. Begge disse måtene å svare på ble foreslått av flere studenter, og det vil gjøre det mulig å lage mer varierte oppgaver og kan gjøre noen oppgavetyper mer praktisk å gjennomføre.

Videre ble Parson Explorer, tre-visningen av oppgavene i arbeidsområdet, utvidet med informasjon om hvor mange mangler som gjenstår i hver fil. Dette er funksjonalitet som ble skissert i fordypningsprosjektet, se figur 2.1, og det ble prioritert nå fordi det er nyttig funksjonalitet og det at det er direkte integrert med arbeidsflaten til VS Code gjør også at tillegget som helhet oppleves som mer integrert i VS Code

I brukertestene var det flere studenter som var ønsket muligheter for å validere svarene sine, enten ved å direkte få vite om de hadde svart riktig eller ved å forsøke å kompilere og kjøre koden. I denne iterasjonen ble det implementert funksjonalitet for å kompilere og kjøre javafiler, men den er ganske begrenset. For å jobbe mot målet om at tillegget skal være godt integrert med VS Code-arbeidsflaten ble også denne funksjonaliten koblet sammen med Parson Explorer, ved at en oppgave som spesifiserer i konfigurasjonsfilene sine at den støtter kompilering og kjøring får en knapp med "play"-ikon, som er et etablert ikon for å symbolisere at noe kan kjøres.



Figur A.8: Skjermbilde av hvordan tillegget ser ut når man løser oppgaver etter iterasjon 4

I figur A.8 kan mye av den nye funksjonaliteten sees, det er et felt for manuell skriving øverst i kodefilen og en åpen dropdown-meny lenger ned i kodefilen, I Parson Explorer på venstresiden kan man se at hver kodefil viser hvor mange ubesvarte mangler som gjenstår, og ved navnet på oppgaven er det en knapp for å kjøre koden.

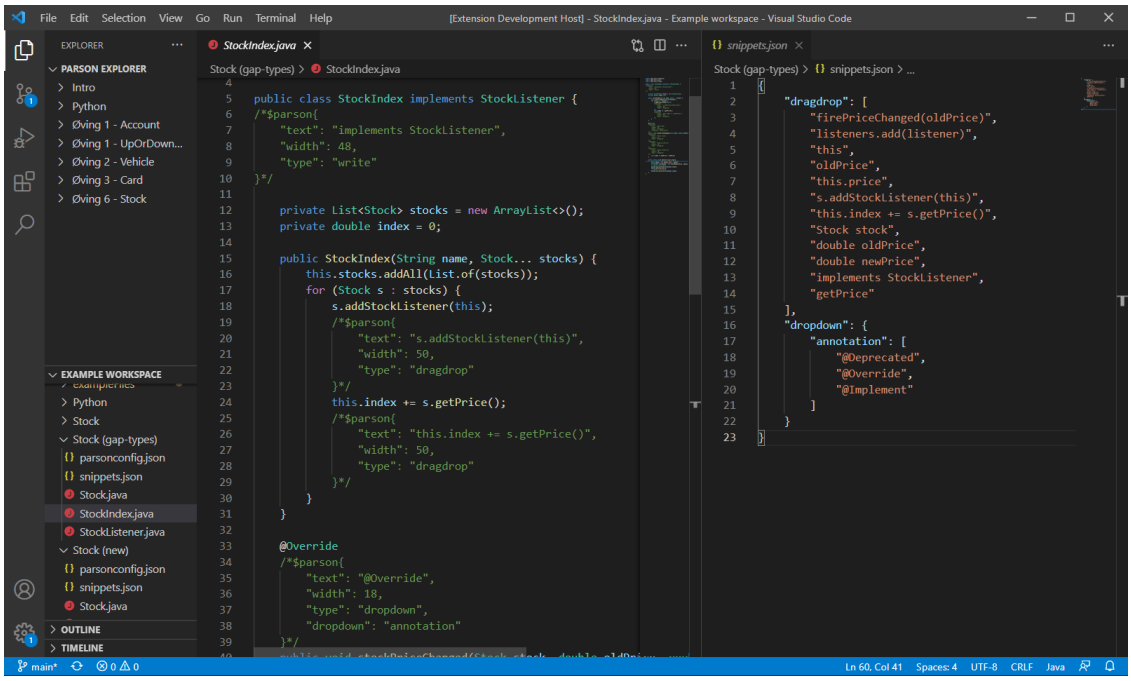
A.4.2 Funksjonalitet for å lage oppgaver

Det er et mål at det skal være effektivt å utvikle oppgaver til tillegget, og denne iterasjonen legger til noe funksjonalitet som bidrar til dette målet.

Et konkret ønske som kan fra veileder, som er en relevant bruker av systemet, var at informasjon om hvilke mangler som skal være i en kodefil ikke stopper kodefilen fra å kunne kompileres og kjøres. I løsningen som ble utviklet i iterasjon 3 må koden som skal gjøres om til en mangel erstattes av en identifikator, som både fjerner funksjonaliteten som kommer fra koden som blir gjort om til en mangel, og i de fleste tilfeller gjør at koden ikke lenger er gyldig kode i sitt språk.

I denne iterasjonen ble lagring av info om mangler flyttet til å bruke kommentarer i koden, slik at koden fortsatt kan kjøres, og dermed også arbeides videre med samtidig som man arbeider med å utvikle oppgaven som benytter kodefilen.

For å gjøre denne funksjonaliteten lettere å bruke, ble det lagt til en ny kommando som tar den teksten du har markert og konverterer den til en mangel som lagres som en kommentar under den linja du har markert tekst på. Funksjonalitet som dette ble planlagt i fordypningsprosjektet og skissert i figur 2.2, men implementasjonen her har ikke det samme brukergrensesnittet og bruker heller bare en kommando fra kommandopaletten. Dette fordi det ikke er en høy prioritet med brukervennlighet for utvikling av oppgaver på dette punktet.



Figur A.9: Skjermbilde av hvordan tillegget ser ut når man lager oppgaver etter iterasjon 4

For å gjøre utviklingen mer effektiv ble det også lagt til funksjonalitet for å oppdatere listen over oppgaver, og å laste inn en oppgave på nytt, slik at man mer effektivt kan se resultatene når man jobber med å lage oppgaver. Spesielt listen med oppgaver var veldig tungvint frem til denne oppdateringen, fordi den bare ble lastet inn når tillegget laster inn, som betydde at man måtte starte hele VS Code på nytt for å laste inn en ny oppgave.

Denne funksjonaliteten ble også koblet inn i Parson Explorer slik at den er en del av VS Code-arbeidsflaten. Alle oppgaver som vises i Parson Explorer, i tillegg til selve Parson Explorer-panelet, vil ha en knapp med en sirkulær pil, som er et etablert symbol for å oppdatere.

Funksjonaliteten for å oppdatere listen og innholdet i en fil blir også kalt når man bruker kommandoen for å kompilere til en oppgave blir kalt, så man vil ofte ikke trenge å trykke knappen selv, men muligheten er der hvis man vil gjøre endringer direkte i de kompilerte oppgavefilene.

Fordi denne funksjonaliteten ikke er relevant for studenter og brukere som ikke skal utvikle egne oppgaver ble det lagt til en innstilling for å skru visning av knappene av og på, som har av som standardverdi. Innstillingen er integrert i innstillingsmenyen til VS Code.

A.4.3 Resultater og plan for neste iterasjon

Etter denne iterasjonen ble det vurdert at det kunne gjøres én til tidsbegrenset iterasjon før utviklingen må avsluttes. Denne iterasjonen må fokusere på å refaktorere og gjøre det lettere å utvide tillegget med ny funksjonalitet, for å oppfylle kravet om det som ble stilt i seksjon 1.4.

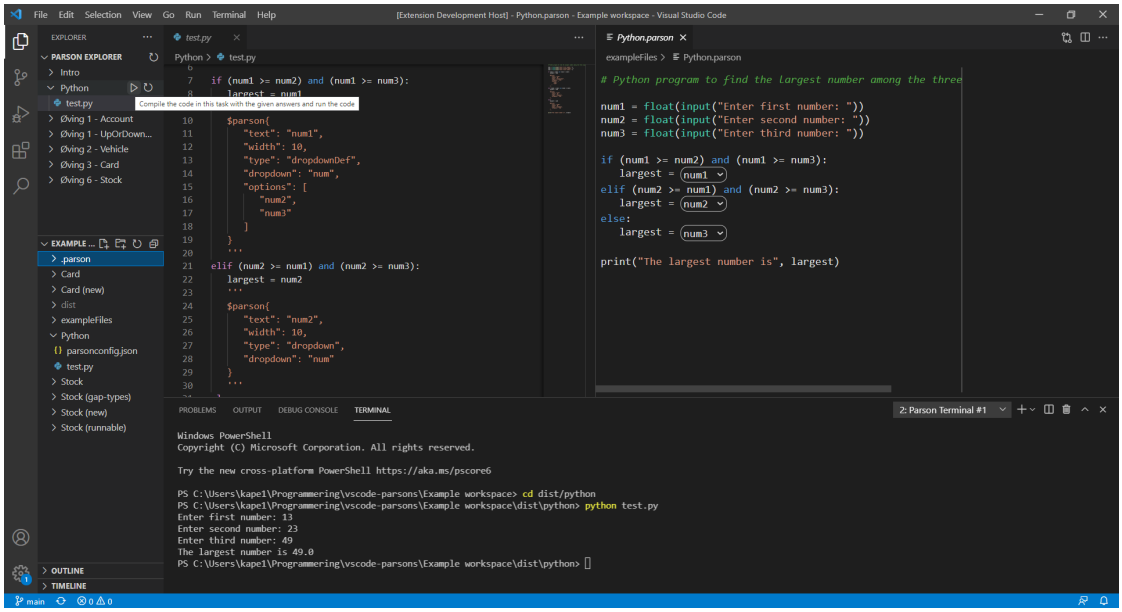
Det ble identifisert to deler av tillegget som vil være viktig å kunne utvide, støtte for programmeringsspråk, og oppgavetyper.

I tillegg er det et ønske fra veileder og potensiell bruker av systemet at tillegget kan lytte etter endringer i filene som definerer en oppgave og kompilere dem til oppgavefiler automatisk, for å gjøre prosesssen med å lage oppgaver enda mer effektiv.

A.5 Iterasjon 5

I denne iterasjonen gikk det mye tid på å refaktorere tillegget slik at det var mulig å skille ut funksjonaliteten for språkstøtte og oppgavetyper til egne moduler, slik at de er lette å utvide.

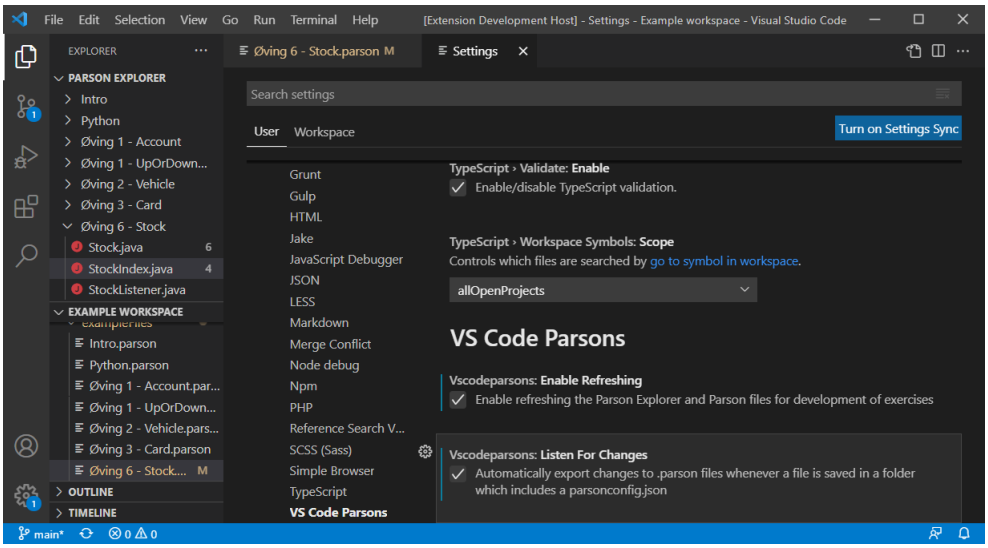
Etter at refaktoreringen var gjort ble det brukt tid på å legge inn støtte for automatisk kompilering av oppgavefiler når en kodefil som ligger i en mappe som inneholder konfigurasjonsfiler fra tillegget lagres.



Figur A.10: Skjermbilde av hvordan tillegget ser ut etter iterasjon 5

Skjermbildet i figur A.10 viser et oppsett for å effektivt utvikle oppgaver der man kan gjøre endringer i python-filen til venstre og oppgavefilen til høyre vil oppdateres for å reflektere disse endringene når man lagrer filen, eller automatisk nesten umiddelbart om man har skrudd på autolagring i VS Code

På samme måte som oppdateringsknappene som ble introdusert i forrige iterasjon er denne funksjonaliteten ikke relevant for brukere som ikke skal lage oppgaver, og den er derfor også lagt inn som en innstilling i innstillinger-menyen i VS Code, som vist i figur A.11



Figur A.11: Skjermbilde av innstillingene som tilbys av tillegget

Etter at språkstøttefunksjonaliteten ble refaktorert så ble støtten før å kunne kompilere og kjøre kun java-filer utdatert, så det ble også brukt noe tid på å gjøre denne generell, men den er gjort generell ved å legge alt ansvaret for kompilering og kjøring over på den som lager oppgaven. Hvilke kommandoer som skal kjøres defineres i konfigurasjonsfilen til oppgaven.

Tillegg B

Brukertestrapport

Som en del av smidig utvikling er det viktig å få tilbakemeldinger etter iterasjoner for å velge riktig vei videre, og en god måte å gjøre dette på er brukertesting. I dette prosjektet ble det gjort brukertester med åtte studenter som har emnet TDT4100 - Objektorientert programmering dette semesteret.

B.1 Hva som testes

Testene ble utført i uke 19, fra 10. til 12. mai, med den på det punktet mest oppdaterte kodebasen, lagret permanent på github som en utgivelse her:
`https://github.com/kape142/vscode-parsons/releases/tag/v0.1-alpha`
Denne versjonen støtter mye av den planlagte funksjonaliteten for studentversjonen i forhold til hovedplanen i subseksjon 3.1.2 men mangler mer på forelesersiden, som er fornuftig fordi det er studentversjonen som skal testes. I forhold til iterasjon 6 i hovedplanen så mangler innfyllingsfelter i studentversjonen, mens i foreleserversjonen mangler i tillegg funksjonaliteten for å konvertere markert kode til en kodesnutt. Mer detaljer om hva som ble med i denne iterasjonen og hvorfor finnes i tillegg A: Iterasjoner, der iterasjon 3 er siste iterasjon før brukertesten.

B.2 Testdeltakere

Testdeltakerene i denne brukertesten var frivillige studenter som er i sitt første år med studier og har emnet TDT4100 - Objektorientert programmering dette semesteret. For å rekruttere studenter ble det sendt ut en melding til alle studenter som har emnet med kort info om hva prosjektet går ut på, og forespørsel om at de som er interesserte i å delta på en brukertest kan ta kontakt på epost. Studentene ble bedt om å gi en kort vurdering av sine egne faglige egenskaper i eposten, slik at det ville være mulig å velge ut et spekter av studenter med varierende faglige egenskaper.

Det ble satt en grense på maks 10 deltakere, det vil være mange nok deltagere til at man både kan se etter resultater som er felles for mange deltakere og være trygg på at det er en trend, man øker sjansen for at man finner mindre åpenbare problemer som ikke alle deltakere vil møte på, og om man tester mange flere enn dette får man etter hvert lite nye informasjon i hver test.

Totalt 17 studenter meldte interesse, 10 av disse ble valgt ut og invitert til å være med på brukertest, og 8 av de endte opp med å delta på testen. Av de 8 som gjennomførte testen så var det fire som vurderte seg selv som faglig sterke, tre vurderte seg selv midt på treet, mens én vurderte seg selv som faglig under gjennomsnittet. Deltakerene hadde en jevn kjønnsfordeling, og de kom fra flere ulike studieprogrammer, herunder "informatikk", "datateknologi", "industriell økonomi og teknologiledelse (retning datateknologi)", og "ingeniørvitenskap og IKT".

B.3 Gjennomføring av testen

Gjennomføring av brukertestene ble påvirket av koronaviruspandemien som har preget store deler av 2020 og 2021, og for å ikke bidra til risiko for smittespredning ble alle brukertestene foretatt digitalt ved hjelp av videosamtaleverktøyet Zoom. Studentene ble med i en videosamtale med en PC som hadde en instans av VS Code kjørende med tillegget installert, og fikk overta kontroll av mus og tastatur på maskinen for å gjennomføre oppgavene. I tillegg til denne maskinen brukte jeg en annen PC for å ta notater og lese opp oppgaver og spørsmål uten at det forstyrrer studenten.

B.3.1 Konsekvenser av digital gjennomføring

Å gjennomføre testen over Zoom skapte noen problemer som ikke hadde oppstått dersom brukertesten kunne ha blitt gjennomført fysisk ved at de satt ved min pc.

At skjermbildet og mus- og tastatur-interaksjoner måtte sendes over internett gjorde at studentene opplevde noe forsinkelse i brukergrensesnittet, men ikke så mye at det hadde merkbar effekt på resultatene.

PC-en som kjørte VS Code brukte Windows som operativsystem, noe som skapte noen mindre problemer for de studentene som kontrollerte den fra en Mac.

For å bla horisontalt på en side kan man på en pekeplate dra fingrene horisontalt, men dette gjøres motsatt vei på Mac i forhold til på Windows, som gjorde at studentene som gjorde testen fra en Mac hadde noe problemer med å bla horisontalt når koden ble for bred for skjermen. Dette skapt mest problemer for den første studenten som opplevde det, da jeg ikke visste om at det var et problem som kunne oppstå, men i alle tilfeller hadde dette ikke noen andre konsekvenser for testen enn et lite opphold for Mac-brukerne for å venne seg til å bla motsatt retning horisontalt.

På Mac er snarveien for å angre ⌘+Z mens på Windows er det Ctrl+Z, og fordi Zoom bare sender direkte hvilke taster som blir trykket, var Mac-brukerene nødt til å bruke andre taster enn de er vant til for å teste angre-funksjonaliteten. Dette skapte få problemer, da denne funksjonaliteten ble testet ved å stille direkte spørsmål om at studentene skulle trykke disse tastene, og instrukser om å trykke ctrl+Z, selv om det ikke er den vanlige tastekombinasjonen, kunne gis til Mac-brukerene.

Zoom sender i tillegg til et skjermbilde også webkamera-bilde, og det skapte problemer for noen av studentene som opplevde at webkamera-bildene dekket over panelet med kodesnutter i den første oppgaven. Dette var nok det som i størst grad påvirket resultatene i størst grad, da de studentene som opplevde dette ikke skjønte hvordan systemet fungerte når de ikke kunne se en liste over kodesnutter og heller prøvde

å skrive direkte i feltene og måtte få beskjed om å sjekke om webkamera-bildene dekker over noe. Dette gjaldt bare to av åtte studenter og det ble løst i løpet av den første oppgaven og påvirket ikke resten av testen, men for de to studentene forstyrret dette muligheten for å samle informasjon om hvor intuitivt systemet er å bruke første gangen man ser det.

Alt i alt har å gjennomføre brukertestene digitalt skapt få problemer, og ingen av problemene har vært veldig store, men til sammen blir det en del forstyrrende elementer som gjør at det nok fortsatt hadde vært foretrukket om det var mulig å utføre brukertestene fysisk.

B.3.2 Oppgaver og spørsmål

Før oppgavene ble gitt ble studentene informert om at det er en test av tillegget og ikke av dem, verken av deres faglige kunnskaper, eller hvor flinke de er til å ta i bruk tillegget. De ble bedt om å tenke høyt når de interagerer med systemet, og spesielt dersom brukergrensesnittet ikke oppfører seg slik de forventer er det ønskelig at de forteller hva de hadde forventet at kom til å skje.

Da oppgavene ble gitt fikk studentene bare navnet på oppgaven de skulle løse, og beskjed om å fullføre oppgaven, og de fikk beskjed om å selv si fra når de føler at de er ferdig med oppgaven. Etter hver oppgave fikk de noen spørsmål om hvordan de følte det gikk å løse oppgaven, om de er kjent med oppgaven fra øvingsopplegget til emnet, og mulighet til å legge til ytterligere kommentarer. I tillegg ble det etter noen av oppgavene stilt spørsmål om spesifikke deler av den oppgaven.

Oppgavene er, med unntak av introduksjonsoppgaven, tatt fra øvingsopplegget til TDT4100 - Objektorientert programmering.

Intro

Første oppgave var en enkel "Hello World" oppgave, som er ment som en introduksjon til å bruke tillegget. Etter denne oppgaven ble studentene spurt om de synes det gikk greit å åpne oppgaven, om de opplevde at brukergrensesnittet oppførte seg som forventet, og om det er noe funksjonalitet de savner umiddelbart.

Øving 1 – Account

Andre oppgave er basert på en oppgave fra Øving 1 kalt "Account", og er den første oppgaven med distraktorer. Etter denne oppgaven ble studentene spurt om de de syntes det var intuitivt at de måtte velge ut de riktige alternativene fra lista og at ikke alle kodesnuttene skulle brukes til noe

Øving 1 – UpAndDownCounter

Tredje oppgave er basert på en oppgave fra Øving 1 kalt "UpAndDownCounter", og introduserer ikke noen nye elementer, men er en faglig vanskeligere oppgave.

Øving 2 – Vehicle

Fjerde oppgave er basert på en oppgave fra Øving 2 kalt "Vehicle", og fokuserer på synlighetsmodifikatorer. I oppgaven er alle synlighetsmodifikatorer fra hele klassen fjernet, og studentene må sette på plass én og én. Etter denne oppgaven ble studentene spurt om de følte dette var en praktisk måte å svare på oppgaven på, og om de har noen forslag til andre måter man kunne bedt om svar på i denne typen oppgave. Etter dette ble de spurt spesifikt om deres mening om å velge riktig svar fra en dropdown-liste for hver mangel i stedet for å måtte velge kodesnutter og sette dem på riktig plass.

Etter fem brukertester var gjennomført ble disse spørsmålene fjernet til fordel for nye spørsmål etter sjette oppgave.

Øving 3 – Card

Femte oppgave er basert på en oppgave fra Øving 3 kalt "Card", og består av to kodefiler, "Card.java" og "CardDeck.java", som deler en liste med kodesnutter. Det er den første oppgaven som har flere kodefiler som en del av samme oppgave. Etter denne oppgaven ble studentene spurt om de syntes det var intuitivt å bruke flere kodefiler i samme oppgave.

Øving 6 – Stock

Sjette oppgave er basert på en oppgave fra Øving 6 kalt "Stock", og består av tre kodefiler, "Stock.java", "StockIndex.java", og "StockListener.java", som deler en liste med kodesnutter. Denne er den mest omfattende oppgaven og den mest faglig utfordrende. Etter denne testen ble studentene stilt spørsmål om de tror det hadde vært nyttig å kunne validere om svarene deres er riktige, og om de tror det hadde vært nyttig å kunne kompilere og kjøre koden med svarene de har gitt.

Disse spørsmålene ble kun stilt i de siste tre brukertestene, etter en vurdering om at spørsmålene fra fjerde oppgave var blitt svart på nok ganger og at det ville være mer nyttig å legge til noen spørsmål etter denne oppgaven i stedet.

Andre spørsmål

Etter at alle oppgavene var gjennomført ble de stilt noen spørsmål om funksjonalitet som ikke var blitt testet direkte i oppgavene. Først ble de spurt om lagring, og hva de forventer at vil bli lagret dersom de lagrer oppgavefilen etter å ha svart på oppgaven. Etter det ble de spurt om angring og omgjøring og hva de forventer at vil bli angret dersom de trykker Ctrl+Z og hva tilsvarende hva som blir omgjort om de trykker Ctrl+Y

Til slutt ble de stilt noen avsluttende spørsmål om inntrykket deres av tillegget. De ble spurt om hvordan de synes det var å gjøre oppgaver med dette tillegget, om de tror det kunne hjulpet dem å løse oppgaver som bruker dette tillegget når de skal introduseres for nye tema i faget. Til slutt fikk de muligheten til å komme med forslag eller ønsker til nye funksjoner for tillegget eller andre måter det kan brukes på.

B.4 Mål med brukertesten

Det viktigste målet for brukertesten var å dokumentere at tillegget er intuitivt og brukervennlig å bruke for å løse oppgaver. I tillegg var det et mål å samle informasjon om hva som burde prioriteres av funksjonalitet videre i utviklingen.

B.4.1 Summative mål

For å dokumentere at tillegget er intuitivt å bruke ble studentene ikke gitt instruksjoner om hvordan å bruke tillegget, de ble kun gitt navnet på en oppgave og beskjed om å utføre oppgaven. Dette gjorde det mulig å vurdere brukbarheten til mange forskjellige deler av systemet ved å se på hvordan studentene prøver seg frem når de blir introdusert for nye elementer og om de klarer å finne riktig fremgangsmåte uten assistanse.

I tillegg til dokumentering av hvordan studentene bruker systemet, så ble studentene også stilt konkrete spørsmål om hvordan de synes tillegget var å bruke, og spørsmål om hvor intuitivt det var å bruke nye deler av systemet etter alle oppgavene som introduserte nye ting. Dette kan brukes sammen med observasjonene for å få ett mer komplett bilde av hvordan det oppleves å bruke tillegget.

Funksjonaliteten for å lagre oppgaveutfyllinger og å angre og omgjøre i en fil ble ikke testet direkte i noen av oppgavene, og ble heller stilt spørsmål om hvordan studentene forventer at det vil fungere, slik at brukbarheten av denne funksjonaliteten også kan testes.

Funksjonaliteten som skal vurderes for brukbarhet er:

- Menyen for å velge oppgaver
- Plassering av kodesnutter
- Bruk av distraktorer
- Flere kodefiler i en oppgave
- Lagring av en fil
- Angring og omgjøring i en fil

B.4.2 Formative mål

Det var også et mål for brukertesten å samle informasjon som kan brukes for å styre den videre utviklingen. Det ble gjort ved å se om studentene møter på noen problemer eller uventet oppførsel som burde prioriteres å endres, ved å se på hvordan de bruker systemet og hvordan de går frem for å finne svar på oppgaver som de ikke umiddelbart kan svare på, og ved å stille dem spørsmål om hva de føler burde fungert annerledes, hvilke ønsker de har for ny funksjonalitet, og noen spørsmål om hva de synes om noen konkrete forslag til funksjonalitet som enda ikke er implementert.

I den versjonen av systemet som testene ble gjennomført på svarte man på oppgaver ved å klikke på en kodesnutt i panelet, og så klikke på det feltet man vil plassere

den i. Det var et mål med brukertestene å undersøke om studentene forventer å kunne dra kodesnuttene fra panelet til dit de skal være i stedet, og det ble derfor fulgt med på om de forsøkte å dra i kodesnuttene før de prøvde å trykke på dem. I tillegg fikk de et spørsmål etter først oppgave om brukergrensesnittet oppførte seg som forventet, som ga dem muligheten til å ta opp denne funksjonaliteten om det var noe de syntes burde fungert annerledes

Etter fjerde oppgave, en oppgave der alle synlighetsmodifikatorer er fjernet fra en fil og må legges inn én og én, fikk de første fem studentene spørsmål om de syntes den oppgaven var praktisk lagt opp for å svare på, etterfulgt av et spørsmål om de har forslag til andre måter å gjøre det, før de til slutt blir spurt om hva de synes om tanken om en konkret alternativ løsning. Her var målet både å se om noen av studentene kommer frem til den samme alternative løsningen som allerede har blitt vurdert i prosjektet, og å få forslag til andre alternative måter.

Etter sjette oppgave fikk de siste tre studentene spørsmål om validering av oppgavesvar og kompilering og kjøring av kode. Svarene på disse spørsmålene kan brukes for å prioritere om denne funksjonaliteten burde bygges ut, og hvordan validering og kompilering/kjøring burde prioriteres i forhold til hverandre.

De avsluttende spørsmålene som ber om forslag til ny funksjonalitet vil kunne brukes både for å ta i mot nye forslag som ikke har blitt vurdert enda, men det vil også være nyttig å se om det er noen forslag som blir foreslått flere ganger, og om funksjonalitet som allerede er tenkt på til prosjektet også blir foreslått av studentene uten at de vet om at det er under vurdering. Informasjon om hvilken funksjonalitet som blir foreslått flere ganger vil kunne brukes til å gjøre prioriteringer for den videre utviklingen.

B.5 Resultater

Resultatene kan også deles inn i to kategorier basert på hvilke mål de svarer på, om de er summative og dokumenterer brukbarheten til systemet eller er formative og gir informasjon som kan brukes til å veilede videre utvikling.

B.5.1 Summative resultater

De summative resultatene dokumenter hvordan studentene opplever å bruke systemet, og skal svare på om systemet er brukervennlig og intuitivt å bruke for å løse oppgaver.

Menyen for å velge oppgaver

Alle studentene bruker menyen for å åpne riktig oppgave helt uten problemer. Samtlige studenter sier også når de blir spurt at det gikk veldig fint å skjønne hvordan man skal åpne oppgaven

Plassering av kodesnutter

Ingen av studentene prøver å klikke på kodesnutter og så hvor de skal plasseres som første valg. Syv av åtte studenter prøver å dra kodesnuttene til riktig sted i stedet, mens én student prøver å skrive i feltet. Den studenten som prøvde å skrive i feltet oppdager så at det er et panel med kodesnutter som har blitt dekket til av webkamera-bilde fra Zoom, og prøver da og dra kodesnuttene som sitt andre valg. I tillegg til at alle studentene prøvde å dra kodesnuttene før de prøvde den funksjonaliteten som faktisk var implementert på den første oppgaven, var det også flere av studentene senere i testen prøvde å dra i kodesnuttene på nytt av refleks, selv om de allerede hadde funnet ut hvordan det egentlig skulle gjøres.

Alle åtte studentene nevnte at de forventa å kunne dra kodesnuttene til riktig sted på spørsmål om brukergrensesnittet oppførte seg som forventet, mens syv av åtte studenter tok opp denne funksjonaliteten på spørsmål om det er noe funksjonalitet de savner, mens én student ønsket seg en tekststreng med kort forklaring av hvordan å bruke systemet i stedet.

Fem av åtte studenter tar opp denne funksjonaliteten på spørsmål om forslag til endringer helt på slutten av testen

Bruk av distraktorer

Alle åtte studentene skjønte at når det var det fler alternativer enn steder å plassere dem så betyr det at noen av dem kun er ment som distraksjoner. På spørsmål om de synes dette var intuitivt svarer alle åtte at det er det.

Flere kodefiler i en oppgave

Fire av åtte studenter sier seg ferdig med oppgaven etter å ha svart på den første kodefilen uten å åpne den andre kodefilen og svare på den. Når de får beskjed om at oppgaven ikke er fullført enda går tre av studentene videre til den andre kodefilen uten mer instruksjoner, mens én student spør om de også skal løse CardDeck, navnet på den andre kodefilen, og får bekreftelse på dette før de går videre.

På spørsmål om de synes det er intuitivt å bruke flere kodefiler i en oppgave svarer seks av åtte ja, selv om kun fire av åtte åpnet begge kodefilene før de sa seg ferdig med oppgaven. En av de som ikke svarte ja, svarer at det synes det var lett å forstå funksjonaliteten etter å ha fått beskjed om at oppgaven ikke er ferdig etter å ha fylt ut en kodefil, mens den andre svarer at selv om de i denne testen sa seg ferdig uten å åpne begge filene, tror de at om de hadde løst denne oppgaven på egenhånd så hadde det naturlig neste steget vært å åpne den andre kodefilen, og da hadde de oppdaget hvordan det fungerer og løst oppgaven som ment.

To av studentene kommenterer at de hadde forventet at de forskjellige kodefilene kom til å åpne seg i nye faner.

Lagring av en fil

På spørsmål om hva de forventes at lagres dersom de lagrer den åpne oppgavefilen, så svarer alle åtte studenter at de forventer at svarene du har avgitt blir lagret.

Angring og omgjøring i en fil

På spørsmål om hva de forventes at angres dersom de angrer i den åpne oppgavefilen så svarer alle åtte studenter at de forventer at den forrige handlingen de gjorde blir angret. Syv av åtte studenter sier at de forventer at forrige angring vil bli omgjort dersom de trykker snarveien for å omgjøre, mens én av studentene sier at det ikke er kjent med hvordan omgjøring fungerer i tekstredigeringsprogrammer, så de vet ikke helt hva som vil skje.

Totalinntrykk

På slutten av testen blir studentene spurt om hva de synes om å bruke tillegget, og om de tror det kunne hjulpet dem å løse oppgaver som bruker dette tillegget når de introduseres for nye tema i faget. Alle åtte svarer at de synes det fungerer bra å bruke tillegget. Seks av åtte sier at de tror at oppgaver med dette tillegget kunne hjulpet dem hvis det var tilgjengelig som et alternativ, men flere av dem påpeker at det er mange aspekter av programmering som man ikke lærer av å gjøre denne typen oppgaver. To av studentene tror at å bruke tid på å løse denne typen oppgaver ikke nødvendigvis vil gi dem noe mer læringsutbytte enn om de hadde brukt den samme tiden på å løse øvinger på den vanlige måten.

B.5.2 Formative resultater

De formative resultatene inneholder informasjon som kan brukes for å designe ny funksjonalitet til systemet, og prioritere hvor viktig forskjellig funksjonalitet er å implementere.

Plassering av kodesnutter

Alle studentene forventer at plassering av kodesnutter skjer ved at man drar kodesnuttene til riktig sted i stedet for å klikke på en kodesnutt og så hvor den skal plasseres, bekreftet både av hvordan de prøver seg frem i systemet og svarene deres på spørsmål om hvorvidt brukergrensesnittet oppfører seg som forventet. Mulighet for å dra kodesnuttene er også det mest populære endringsforslag i de avsluttende spørsmålene, foreslått av fem av åtte studenter.

Synlighetsmodifikatoroppgave

Av fem studenter som fikk spørsmål om denne oppgaven svarte fire positivt på at det er en praktisk måte å løse oppgaven på, men en av dem påpeker at det ikke hadde tatt så mye lenger tid å bare skrive ordene ”private” og ”public” selv, mens en annen synes oppgaven ble litt monoton med så mye likt, men at det er god trening. Én student mener at dette ikke er en praktisk måte å løse oppgaven på, og at det er blir rotete med så mange like alternativer.

På spørsmål om forslag til andre måter å legge opp oppgaven på kom det flere forskjellige forslag, og noen av dem ble foreslått av flere personer:

- Svare for flere felter samtidig, så man svarer for eksempel for alle gettere og settere samtidig.
- Kodesnutter som kan brukes flere ganger og ikke blir borte fra panelet. Foreslått av tre personer.
- Skrive teksten direkte i boksen selv. Foreslått av to personer.
- Velge fra en dropdown-liste på hver mangel. Foreslått av to personer.
- Bruke en kortere kodefil for denne typen oppgave.

Når studentene ble spurt om hva de synes om forslaget om å velge fra en dropdown-liste for hver mangel, er alle fem positive, men én påpeker også at det fortsatt vil være mange klikk involvert for å fylle inn alle manglene.

Validering og kompilering/kjøring

Tre studenter fikk konkrete spørsmål om de tror at det hadde vært nyttig å kunne validere koden og få vite om man har svart riktig, og om de tror det hadde vært nyttig om man kunne kompilere og kjøre koden. Alle tre studentene var positive til begge deler. Én av studentene, når de får spørsmål om validering av svar først, foreslår mulighet for å kompilere koden som et alternativ. Én av studentene er positiv til validering først, og når forslaget om å kompilere og kjøre koden presenteres foreslår de at det kan også kan gjøres som en del av valideringen. Den tredje studenten er positiv til begge, men har noen forbehold om validering, og er redd for at det kan gjøre det for lett hvis man kan validere hvert svar når det gis.

Utenom de konkrete spørsmålene, blir det å kunne validere svar foreslått av én student på spørsmål om det er noe funksjonalitet som savnes umiddelbart etter første oppgave, mens én student som ikke fikk disse spørsmålene etterspurte validering av svar på det avsluttende spørsmålet om ønsket funksjonalitet.

Andre forslag

På slutten av testen ble studentene bedt om forslag til endringer eller ny funksjonalitet, og dette er det de svarte:

- Bedre syntaks-highlighting, også i utplasserte kodesnutter.
- Mulighet for å dra kodesnutter i stedet for å klikke på dem. Foreslått av fem personer.
- Mulighet for å skrive teksten direkte i boksen selv. Foreslått av to personer.
- Validering av svar når man er ferdig. Foreslått av to personer.
- Mer feedback når man plasserer og fjerner svar, for eksempel en animasjon av at svaret flyttes.
- Når det er mange svar i panelet kunne det vært flere kolonner.

Andre vedlegg

I tillegg til dette dokumentet legges det ved en zippet mappe kalt `Vedlegg.zip`.

I denne mappen ligger en zip-fil kalt `kode.zip` som inneholder den samme koden som ligger lagret på Github på `https://github.com/kape142/vscode-parsons/`.

Mappen inneholder også en mappe kalt `Brukertest` som inneholder dataen samlet inn under brukertestene. Denne dataen kommer i form av to planer for gjennomføring av brukertester, `Brukertestplan 1-5.docx` og `Brukertestplan 6-8.docx`, og en mappe `Rapporter` som inneholder 8 dokumenter, en for hver gjennomførte test, med alle notater som ble tatt under testene.

Referanser

[1] T. Jenkins. (2002). «On the Difficulty of Learning to Program,» adresse: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.596.9994&rep=rep1&type=pdf. Lastet ned: 08.06.2021.

[2] D. Parsons og P. Haden. (2006). «Parson's programming puzzles: A fun and effective learning tool for first programming courses,» adresse: https://www.researchgate.net/publication/262160581_Parson%5C%27s_programming_puzzles_A_fun_and_effective_learning_tool_for_first_programming_courses. Lastet ned: 09.06.2021.

[3] M. Eysholdt. (2018). «Gitpod: A One-Click Online IDE,» adresse: https://www.typefox.io/blog/gitpod-a-one-click-online-ide. Lastet ned: 09.06.2021.

[4] Gitpod. (2021). «VS Code Extensions,» adresse: https://www.gitpod.io/docs/vscode-extensions/. Lastet ned: 09.06.2021.

[5] F. Lardinois. (2015). «Microsoft Launches Visual Studio Code, A Free Cross-Platform Code Editor For OS X, Linux And Windows,» adresse: https://techcrunch.com/2015/04/29/microsoft-shocks-the-world-with-visual-studio-code-a-free-code-editor-for-os-x-linux-and-windows/. Lastet ned: 09.06.2021.

[6] Ars Technica. (2015). «Visual Studio now supports debugging Linux apps; Code editor now open source,» adresse: https://arstechnica.com/information-technology/2015/11/visual-studio-now-supports-debugging-linux-apps-code-editor-now-open-source/. Lastet ned: 09.06.2021.

[7] Opensource.com. (2021). «What is open source software?» Adresse: https://opensource.com/resources/what-open-source. Lastet ned: 09.06.2021.

[8] Microsoft. (2021). «Visual Studio Code Extension API,» adresse: https://code.visualstudio.com/api. Lastet ned: 09.06.2021.

[9] E. Gamma, R. Helm et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[10] K. Beck et al. (2001). «Manifestet for smidig programvareutvikling,» adresse: https://agilemanifesto.org/iso/no/manifesto.html. Lastet ned: 06.06.2021.

[11] Stack Overflow. (2018). «Stack Overflow Developer Survey 2018,» adresse: https://insights.stackoverflow.com/survey/2018#work-_-which-methodologies-do-developers-use. Lastet ned: 06.06.2021.

[12] S. W. Ambler. (2014). «2013 IT Project Success Rates Survey Results,» adresse: http://www.ambysoft.com/surveys/success2013.html. Lastet ned: 07.06.2021.

[13] M. Poppendiek og T. Poppendiek, *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional, 2003.

[14] Agile Alliance. (2021). «What is a Minimum Viable Product (MVP)?» Adresse: https://www.agilealliance.org/glossary/mvp/. Lastet ned: 07.06.2021.

[15] S. L. Henry og J. Thorp. (2008). «Notes on User Centered Design Process (UCD),» adresse: https://www.w3.org/WAI/redesign/ucd. Lastet ned: 07.06.2021.

[16] J. Rubin, *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. John Wiley og Sons, Inc., 1994.

[17] J. Nielsen. (2012). «Thinking Aloud: The #1 Usability Tool,» adresse: https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/. Lastet ned: 07.06.2021.

[18] Eclipse Foundation. (2021). «Open VSX Registry,» adresse: https://open-vsx.org/about. Lastet ned: 11.06.2021.