NTNU
Det skapende universitet

Spillifisering for å stimulere utvikling av praktiske ferdigheter i Eclipse

Espen Selquist Stenmark

Master i informatikk
Innlevert: januar 2015
Hovedveileder: Hallvard Trætteberg, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Sammendrag

I denne oppgaven er det blitt sett på om spillifisering er en teknikk som kan brukes for å stimulere utviklingen av praktiske ferdigheter til studenter i et utviklingsverktøy. Spillifisering er et populært tema som går ut på å ta i bruk mekanikkene som finnes i spill, og anvende disse til et ikke-spill miljø for å motivere til atferd. Det er blitt tatt utgangspunkt i læringsmålene til et programmeringskurs ved universitet og implementert en spillifisert utvidelse til utviklingsverktøyet som brukes i kurset. Til slutt er det gjennomført brukertesting av prototypen hvor brukerevalueringen viste bevis for at spillifisering kan brukes til å stimulere for utvikling av praktiske ferdigheter.

Summary

This thesis has looked into the possibility of using techniques from gamification to stimulate the development of students' practical skills in using a software development tool. Gamification is a popular trend which involves taking game mechanics and applying them to a non-game environment to motivate a certain behavior. Based on the learning objectives from a university programming course, a gamification plugin for the development tool used in the course has been implemented. A usability test of the prototype was conducted, and the results indicate that gamification can be used to stimulate the development of practical skills in computer programming.

Forord

Denne oppgaven er en del av mitt masterstudie i informatikk innenfor retningen Data- og informasjonsforvaltning ved Norges teknisk-naturvitenskapelige universitet (NTNU). Oppgaven teller 50% av masterstudiet, og ble startet våren 2014.

Jeg vil først og fremst takke min veileder, Hallvard Trætteberg, for sine gode ideer, tilbakemeldinger og innnspill gjennom hele perioden. Dette er noe jeg har satt utrolig stor pris på.

Deretter vil jeg takke mine medstudenter for testing, tilbakemeldinger og korrekturlesing.

Til slutt ønsker jeg å takke både familie og venner for oppmuntring og støtte.

Espen Selquist Stenmark
12. januar 2015

Innhold

Tabeller

Figurer

Kapittel 1

Introduksjon

I programmeringsfag som undervises ved universiteter lærer studentene å skrive kildekode som datamaskinen kan tolke. Fagene som underviser programmering benytter seg av et øvingsopplegg der studentene skal tilegne seg de helt elementære ferdighetene gjennom å løse programmeringsoppgaver. Ved innføring i programmering blir det ofte benyttet et enklere utviklingsverktøy i form av en simpel tekstbehandler, der funksjonene som brukeren kan gjøre er begrenset. Motivasjonen for å bruke disse verktøeyene er at de krever tilnærmet ingen opplæring, da det bare er å opprette en programmeringsfil og begynne å skrive kildekode. Etterfølgende programmeringsfag som krever at studenten har kjennskap til programmering, benytter seg ofte av et mer avansert utviklingsverktøy som fungerer på kryss av operativsystemene og har mulighet for å implementere store systemer. Overgangen fra en simpel tekstbehandler og til et slikt integrert utviklingsmiljø krever dermed opplæring i programvaren for å kunne komme i gang med å skrive kildekode samt å kunne dra fordel av alle funksjonene som det tilbyr. Da slik opplæring koster betraktelig tid og ressurser stiller det store krav til at studentene selv tilegner seg de erfaringene og ferdighetene som kreves for å gjennomføre øvingsopplegget. Dermed blir studentenes motivasjon til å lære seg praktisk bruk av utviklingsverktøyet en viktig faktor for at de faktisk kommer til å gjøre dette. Å bruke mesteparten av tiden sin på å utforske et utviklingsverktøy i blinde, lese gjennom flere sider med kompliserte beskrivelser, eller tyde uoversiktlige hjelpevideoer på internettet, er ikke veldig motiverende for en travel student. Optimalt sett ville det vært ønskelig å kunne gi denne informasjonen direkte til studentene på en rask og enkel måte, samt finne en måte å motivere studentene til å fortsette å tilegne seg ferdigheter i utviklingsverktøyet. En stigende trend som prøver å løse slike problemer er spillifisering [Google, c]. Spillifisering handler om å utnytte egenskaper og mekanismer fra spillverdenen og bruke disse i en ikke-spill sammenheng for å motivere brukeren til å utføre en handling [Kapp, 2012]. Da det finnes bevis for at bruk av spillifisering kan øke motivasjonen og stimulere til atferd [O'Donovan et al., 2013], vil det være interessant å se på mulighetene til å bruke teknikker innenfor spillifisering som skal motivere studentene til å tilegne seg praktiske

erfaringer i et utviklingsverktøy.

1.1 Forskningsspørsmål

Hovedideen bak denne oppgaven er å utforske mulighetene til å ta i bruk noen av mekanismene ved spillifisering for å oppnå ferdigheter innenfor bruk av utviklingsverktøy til programmering. Dette har resultert i hovedspørsmålet:

RQ1: Hvordan kan spillifisering brukes til å stimulere ferdigheter som ikke nødvendigvis måles i dagens øvingsopplegg ved programmeringskurs?
Spillifisering er et vidt begrep bestående av mange spillmekanismer og spillelementer som kan benyttes til å motivere for en handling. Gjennom prosjektet vil det utforskes hva spillifisering er, hvordan dette er blitt gjort tidligere og hvordan dette kan bli gjennomført. Dette hovedspørsmålet vil bli besvart gjennom noen underspørsmål:

RQ1.1: Hvordan kan Eclipse utvides med hjelp av mekanikker innenfor spillifisering for å stimulere til bruk av relevante bruksoppgaver?
Relevante bruksoppgaver i Eclipse vil bli identifisert ved å studere læringsmålene til kurset TDT4100-Objektorientert programmering. Da vil det komme fram hvilke praktiske ferdigheter som det er ment at studentene skal tilegne seg i løpet av kurset. Videre vil det gjøres et litteraturstudie av hva spillifisering er og hvordan dette kan knyttets opp mot en løsning som skal motivere for at studentene ønsker å tilegne seg ferdigheter i Eclipse.

RQ1.2: Hvordan kan utvidelsen implementeres og gjøres tilgjengelig som et rammeverk for å støtte de relevante bruksoppgaver og potensielt utvikles videre?
Utviklingsverktøyet Eclipse er en stor programvare med komplisert funksjonalitet. Her vil det gjøres en empirisk undersøkelse om det faktisk er mulig å lage en utvidelse til et utviklingsverktøy som kan teste de praktiske ferdighetene som ønskes at brukeren tilegner seg. Deretter vil denne utvidelsen gjøres så enkel som mulig for utviklere ved å ha oversiktlig kildekode som lett lar seg modifiseres.

1.2 Resultater

Resultatene av dette prosjektet vil bli presentert avslutningsvis. Der vil det bli lagt fram hvordan den egne løsningen er oppbygd og hvordan den fungerer i praksis. Evaluering av egen løsning vil komme fram, der det blir lagt vekt på brukernes testing av systemet. Basert på dette vil det vurderes om forsøksspørsmålene er blitt godt besvart. Til slutt vil det bli presentert en konklusjon der det gjøres en oppsummering av prosjektet som helhet.

Kapittel 2

Metode

Dette kapittelet handler om metoder som er benyttet i prosjektet.

2.1 Utviklingsmetode

Forskningsmetoden som er valgt for dette prosjektet er basert på strategien "Design og Utvikling" [J., 2006], en forskningsstrategi som fokuserer på å utvikle nye produkter og artifakter innenfor informasjonsteknologi. For at prosjekter innenfor denne strategien skal kunne karakterisere seg som forskning, er det viktig at det ikke bare blir lagt fram tekniske ferdigheter og dyktighet. Istedet skal prosjektet kunne vise til akademisk kvalitet gjennom analysering, forklaring, argumentering, begrunnelse og kritisk evaluering. Denne strategien blir beskrevet av Oates [J., 2006] som en tilnærming til problemløsning gjennom en iterativ prosess bestående av fem steg:

- Bevissthet (eng.:awareness) - Gjenkjenne problemer gjennom litteraturstudie.
- Forslag (eng.:suggestion) - Komme opp med en tentativ plan for å løse problemet.
- Utvikling (eng.:development) - Implementasjon av det tentative designet.
- Evaluering (eng.:evaluation) - Vurdere utviklet produkt i forhold til forbedringer og avvik fra målsetting.
- Konklusjon (eng.:conclusion) - Oppsummering av designprosess, ny kunnskap blir identifisert sammen med løse tråder som kan bli forklart i videre forskning og utvikling.

Gjennom å følge disse stegene som en iterativ syklus, vil økt forståelse av et punkt føre til større innsikt i et annet punkt, som igjen fører til at forskeren stadig lærer gjennom utvikling.

2.2 Litteraturgjennomgang

Litteraturstudiet i løpet av denne oppgaven vil foregå ved å initielt ta i bruk dagens mest brukte søkemotor, Google [Google, b], for å finne resultater som er relevante [eBizMBA Inc.,]. Grunnen til at det er valgt å bruke en vanlig søkemotor, er at det er en rask og enkel måte å få tak i et stort omfang av resultater. Ulempen med å bruke en slik vanlig søkemotor til å finne søkeresultater er at den ikke klarer å skille vitenskapelige resultater fra ikke-vitenskapelige resultater, i tillegg til at den ikke har tilgang til alle vitenskapelige databaser som finnes. Dermed vil ikke denne metoden alene være en god måte for å skaffe vitenskapelige referanser.

Etter det initielle søket vil det bli gjort et nytt søk ved hjelp av en søkemotor som har tilgang til flere vitenskapelige databaser enn et helt vanlig "Google-søk" har. Det vil bli tatt i bruk et søkeverktøy som heter "Google Scholar" [Google, a], som fungerer slik som et vanlig "Google-søk" bortsett fra at det er utformet slik at den har tilgang til å søke gjennom akademisk litteratur. På den måten vil det forekomme mer relevante resultater enn ved et vanlig Google-søk, samt færre ikke-relevante resultater.

Til slutt vil det bli utført et mer vitenskapelig litteratursøk ved hjelp av ulike vitenskapelige databaser med akademisk litteratur. Den første som vil bli brukt heter Association for Computing Machinery (ACM) [ACM,] og krever en autorisasjon for å få tak i deres innhold, noe NTNU har skaffet seg. Ved å ta i bruk deres database til litteratursøket vil resultatene være høyst vitenskapelige og relevante. Andre ressurser som vil bli brukt for å lete etter vitenskapelige søkeresultater er IEEE Xplore [IEEE,] og BIBSYS Ask [Bibsys,].

2.3 Datainnsamling

Innsamlingen av data i dette prosjektet vil foregå ved hjelp av:

- Brukertesting der testpersoner skal prøve ut prototypen.
- Spørreskjema der testpersonene skal belyse hvor brukervennlig de synes at prototypen er.
- Intervju med testpersonene der prototypen vil bli diskutert.

2.3.1 Brukertesting

Det vil bli gjort en brukertesting av pluginen etter at det er blitt implementert en testbar prototype som inneholder de ønskede funksjonene. De utvalgte testpersonene vil

være reelle studenter der noen har erfaring med både programmering og programmeringsverktøyet Eclipse, mens andre testpersoner vil ha begrenset eller ingen erfaring. Hensikten med testingen vil være å få tilbakemeldinger fra brukerne om utformingen, hvor enkelt det var å bruke pluginen. Brukertestingen vil bli gjennomført ved å observere mens hver bruker utforsker pluginen og utfører oppgavene som finnes der. Observasjon er blitt valgt for å kunne følge med på hvordan brukeren faktisk reagerer gjennom interaksjon med pluginen.

2.3.2 Spørreundersøkelse

Testpersonene skal gjennom en spørreundersøkelse fylle ut et SUS-spørreskjema angående hvordan de opplevde brukervennligheten til prototypen. Deretter skal hver testperson fylle ut et spørreskjema hvor det skal belyses i hvilken grad de følte at prototypen var nyttig, samt hvor tilfreds de var med den. Formatet til spørreskjemaene vil bli besvart i form av "Likert-skala" [Wikipedia, b], som er en graderingsskala der svaralternativene graderer seg fra "1-Sterkt uenig" til "7-Sterkt enig".

2.3.3 Intervju

En lengre samtale vil ta plass etter at testpersonene har testet prototypen og fylt ut spørreskjemaene. Her vil det bli spurt om i hvilken grad prototypen hadde en motiverende faktor, hva som var bra med den, hva som ikke var så bra, hva som kunne vært gjort annerledes og hva som eventuelt manglet. Meningen her er at testpersonene ut ifra sine egne tanker og meninger skal vurdere prototypen som er laget.

Kapittel 3

Litteraturgjennomgang

Dette kapittelet viser gjennomgang av litteraturen som er relevant for å kunne sette seg inn i dette prosjektet. Det vil bli forklart hva som forventes av studentene i et programmeringskurs ved NTNU, hva Eclipse er for noe, samt hva spillifisering er for noe og hvordan dette kan kombineres og brukes.

3.1 Programmeringskurs

Faget TDT4100 - Objektorientert programmering ved NTNU tar for seg grunnleggende objektorientert programmering i språket java [NTNU,]. Studentene skal gjennom undervisning og øvinger oppnå en rekke læringsmål:

- Kunnskap om de viktigste begrepene og mekanismene i objektorienterte språk, og hvordan slike programmer kan struktureres og testes.
- Ferdigheter innenfor objektorientert programmering og bruk av relevante programmeringsmetoder (koding, testing, feilfinning) og moderne utviklingsverktøy.
- Kompetanse til å bruke objektorientert programmering for å løse praktiske problemer og utnytte mulighetene i moderne utviklingsverktøy.

3.2 Eclipse

Eclipse [Foundation, 2014] er et integrert utviklingsmiljø (IDE) som blir brukt til å lage applikasjoner. Programvaren har en lagdelt arkitektur med støtte for utvidelser i form av pluginer. Den har et arbeidsområde der ressursene deles opp i ulike prosjekter, pakker og filer.

3.2.1 SWT og JFace

SWT (Standard Widget Toolkit) er et bibliotek for å lage grafiske brukergrensesnitt-elementer. Dette biblioteket har tilgang til det grafiske brukergrensesnittet til det underligge operativsystemet. JFace ligger som et lag på toppen av SWT og inneholder et rammeverk som forenkler enkelte av oppgavene til SWT.

3.2.2 Brukergrensesnitt

Når programmet Eclipse åpnes presenteres en arbeidsbenk (*workbench*), som er miljøet hvor brukeren kan arbeide og navigere seg rundt i ressursene som finnes i arbeidsområdet (*workspace*). En arbeidsbenk kan ha mange forskjellige perspektiver, for eksempel et java-perspektiv når brukeren skal implementere java-kode, eller et debugger-perspektiv når brukeren skal debugge eksisterende java-kode. Slike perspektiver skal inneholde relevante elementer som vil være nyttige for brukerens gitte formål.

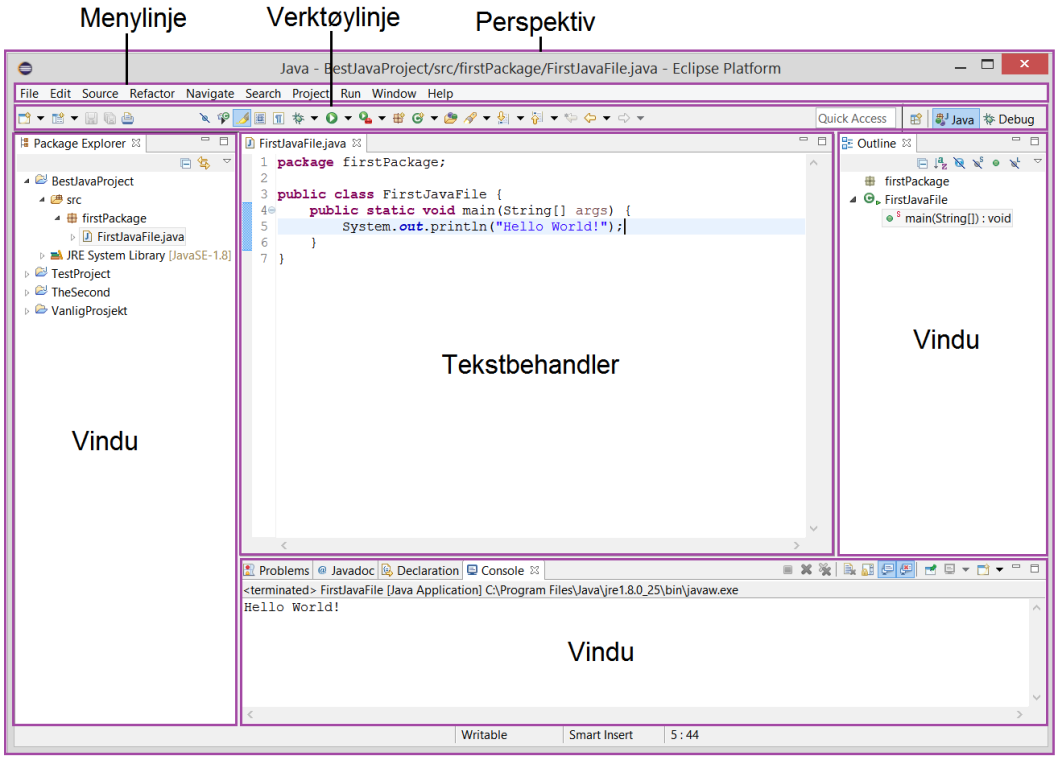
En *editor* er et element som kan inngå i et perspektiv. Dette vil typisk være et lite vindu som fungerer som en tekstbehandler som gjør det mulig for brukeren å redigere tekstfiler.

I tillegg til editorer kan et perspektiv bestå av ulike *views*, som er små vinduer der de presenterer informasjon om ressursene i arbeidsområdet. Dette kan være informasjon som viser alle prosjektene i arbeidsområdet samt hvilke filer som befinner seg i disse prosjektene; konsoll som viser output-data fra et kjørende program samt fungerer som input-data til et program; properties view: viser egenskaper til et valgt element; eller en rekke andre vinduer som er ønskelig for brukeren å legge til sitt perspektiv.

De øverste elementene i Eclipse sin arbeidsbenk heter *menylinje*. Dette er en samling med menyer som hører til selve programmet Eclipse og til innholdet i workbench. Typisk er dette vanlige menyer som finnes i de fleste programmer. “File, Edit, Navigate, Search, Project, Run, Window, Help”, er noen eksempler på slike menyer.

Nedenfor menylinja ligger det noe som heter *verktøylinje*. Dette er elementer som fungerer som verktøy til det nåværende perspektivet som brukeren raskt kan benytte seg av. I et java-perspektiv kan eksempler på slike elementer være “New, Save, Save All, Print, Run”.

Figur 3.1, viser en skjermdump av brukergrensesnittet til Eclipse med forklaringer til de ulike elementene.



Figur 3.1: Skjermdump av brukergrensesnittet til Eclipse med forklaringer.

3.3 Spillifisering

Et begrep som bare blir mer og mer populært for tiden er spillifisering [Google, c]. Spillifisering går ut på å utnytte egenskaper og mekanismer fra spillverdenen og bruke disse i en ikke-spill sammenheng for å motivere brukeren til å utføre en handling [Kapp, 2012]. Selv om dette er et begrep som kan virke litt fjernt, er det noe de aller fleste av oss har vært borti en gang i livet. Et eksempel på akkurat dette kan være at hvis du gjorde hjemmeleksen på barneskolen ble du belønnet med en gullstjerne. Denne stjerna kalles i spillverdenen en "badge", eller et merke, og er en form for belønning som fungerer som et visuelt bevis på at du har oppnådd noe. Uttrykkene "stjerne i margen" og "pluss i boka" har alle hørt. Dette er to uttrykk som ofte dukker opp i det norske språk, og fungerer som en belønning til en eller flere personer som har gjort noe bra og derfor fortjener en anerkjennelse. Disse merkene er enkle og effektive belønninger som kan vises fram både til seg selv og de rundt seg. Et sted hvor merker er flittig i bruk er i speideren, hvor medlemmene kan tjene ulike merker som fungerer som en belønning for at de har fullført en aktivitet.

En annen form for belønning er poeng. Dette er noe som er helt vanlig i de fleste spill og blir opptjent basert på prestasjoner, hurtiget og i noen tilfeller utforskning av omgivelsene. Et trivielt eksempel på hvor du kan ha vært borti en slik belønning er hvis du har spilt terningspillet yatzy, der meningen med spillet er å oppnå høyere sluttpoengsum enn motstanderne ved å kaste rett terningkombinasjon til rett tid. Dette er en belønningstype som er veldig effektiv for å kunne sette spillerprestasjoner opp mot hverandre.

Når det er mange spillere som etterhvert opparbeider seg poeng, er det nødvendig å opprette en liste med oversikt over spillerne og antall poeng de har tjent. Dette kalles i spillverdenen for en highscore-liste og er konstruert slik at den spilleren med de beste prestasjonene, altså den spilleren med flest poeng, er plassert øverst på lista og de andre spillerne blir plassert nedenfor i lista, sortert etter synkende poengsum. Dette kan for mange være en effektiv belønningsmekanisme, da denne lista viser helt tydelig og presist hvor gode prestasjoner de forskjellige spillerne har klart å oppnå, som kan gi økt motivasjon til å opparbeide seg flere poeng enn highscore-lederen.

Et svensk sosialt eksperiment kalt "Tomglasspelet" [Rolighetsteorin, 2010], er et godt eksempel på hva spillifisering kan få oss mennesker til å gjøre i hverdagen. Her har den tyske bilprodusenten Volkswagen forvandlet en helt vanlig avfallscontainer for glass om til et arkadespill [Wikipedia, a] med både lys, lyd og poengsum. Den fungerer på den måten at det er seks resirkuleringshull og en lampe over hvert hull som indikerer hvilket hull spilleren skal kaste glassflasken i, samt en skjerm som viser spillerens nåværende poengsum og den totale poengsummen som er oppnådd. Dette konseptet er en måte for å motivere folk til å resirkulere glassflasker ved at selve innleveringen blir en morsom og interessant opplevelse. Eksperimentet fikk en positiv respons blant folket og resulterte i at over 100 personer benyttet seg av spillet og resirkulerte glass på bare én kveld, sammenlignet med en nærliggende avfallscontainer for glass som kun hadde to personer som resirkulerte glass.

3.3.1 Spillmekanikker

Dette er verktøy, teknikker og elementer brukt som byggesteiner for å spillifisere en applikasjon [Wiki,].

Utmerkelser og prestasjoner (eng.:achievements)

En fysisk eller virtuell representasjon på at du har gjennomført noe. Dette kan for eksempel være en badge, et merke, som du har gjort deg fortjent til ved å fullføre en bestemt oppgave. Disse er med på å vise progresjon, samt gi et spill en viss karakter

og utfordring. Ofte kan slike utmerkelser bygge på hverandre slik at du må oppnå en bestemt utmerkelse for å kunne ha muligheten til å oppnå en annen utmerkelse.

Atferdsmomentum (eng.:behavioral momentum)

Går ut på å få spillerne til å fortsette å gjøre det de har gjort. Hvis spillet oppleves eller føles nyttefult vil det være større sannsynlighet for at brukeren dermed ønsker å spille videre.

Lykksalig produktivitet (eng.:blissful productivity)

Selv om spilleren bruker mye krefter og energi på å spille et spill, så vil det gjøre han mer lykkelig enn å slappe av og ikke gjøre noen ting. Dette er fordi mennesker tilfredsstilles av å utføre et anstrengende arbeid som både er meningsfullt og givende. Et eksempel på dette kan være at en person etter en hard arbeidsdag velger å spille dataspill fremfor å slappe av foran tven.

Bonuser (eng.:bonuses)

Dette er en form for belønning som brukeren får etter å ha fullført en rekke oppgaver eller kjernefunksjoner. I noen tilfeller kan slike bonuser være ekstra poeng som brukeren har opptjent ved å løse en oppgave på kort tid eller uten å bruke hjelpemidler.

Minst mulig ledetråder (eng.:cascading information theory)

Det skal kun tilgjengeliggøres små mengder med informasjon og ledetråder om gangen, slik at brukeren oppnår en ønsket forståelse underveis i spillet. Ved å ikke utsette spilleren for alt av informasjon på engang, vil det gi rom for at spilleren klarer å ta til seg det elementære og deretter mer utfyllende informasjon stegvis.

Oppgaver (eng.:challenges/quests)

Det kan være en eller flere oppgaver bestående av deloppgaver som brukeren kan eller må fullføre. Ofte er det slik at for å klare en oppgave så må brukeren fullføre tilhørende deloppgaver.

Nedtelling (eng.:countdown)

Brukeren får en viss mengde med tid til å gjøre noe. Dette kan være en nedtellingsklokke som viser hvor mange minutter og sekunder brukeren har igjen på å fullføre en oppgave. Ved å ta i bruk en slik mekanisme blir brukeren tvunget til å gjøre aktiviteter på kortere tid en han normalt sett ville gjort.

Oppdagelse (eng.:discovery)

Gi brukeren belønning for å motivere han til å utforske noe. Dette kan være bonuspoeng eller utmerkelser som brukeren oppnår ved å ha utforsket et område i et spill eller tatt i bruk en funksjon i en applikasjon.

Uendelig spillbarhet (eng.:infinite gameplay)

Det finnes ikke en fastsatt slutt på spillet.

Poengtavle (eng.:leaderboard)

En rangert liste av spillere basert på antall poeng de har oppnådd, der spilleren med flest poeng er plassert øverst på lista. Dette er ofte den mest motiverende spillmekanismen, da en rangert liste med de beste spillerne kan vekke lysten til å stadig strebe etter å være personen med flest poeng.

Nivå (eng.:levels)

Fungerer som en rampe som brukeren må overstige ved å utføre en rekke oppgaver og opptjene et visst antall poeng. Ofte blir oppgavene mer utfordrende for hvert nivå, og dermed vil poengene for å fullføre en oppgave øke. Ved å komme til et nytt nivå kan brukeren bli presentert for nye oppgaver, funksjoner og muligheter, noe som kan virke veldig motiverende for at brukeren ønsker å oppnå flest mulig nivåer.

Tapsaversering (eng.:loss aversion)

Påvirke spilleren til å ta handlinger for at han skal unngå å bli straffet. Dette kan være at spilleren må gjøre en bestemt handling for at han skal kunne beholde de opptjente poengene eller belønningene. Et eksempel på dette kan være at spilleren må logge seg inn og gjøre noe fornuftig i et nettbasert spill for å forhindre at han mister poeng eller andre goder.

Poeng (eng:points)

En numerisk verdi brukeren opptjener ved å utføre en eller flere handlinger. Noen handlinger kan gi flere poeng enn andre. Dette kan være med på å gi brukerne en idé om hvilke handlinger som er mest lønnsomme eller mer viktige enn andre, samt motivere brukeren til å utføre en viss handling.

Progresjon (eng.:progression)

En dynamikk som viser til hvordan brukeren sin progresjon i spillet er. Dette kan representeres ved en progresjonslinje.

Status

Viser spillerens nåværende rangering. Ved å opptjene en viss mengde poeng og oppnå et visst nivå, kan spillerens rangering øke.

Turbasert handling (eng.:taking turns)

Spillerne har kun mulighet til å utføre handlinger når det er deres egen tur til å gjøre noe. Dette er en spillmekanikk som ofte benyttes i bordspill eller brettspill. Denne måten spillerne utfører handlinger på gir rom for å observere hva de andre spillerne gjør samtidig som egne handlinger planlegges.

3.4 Relevant arbeid

Det er blitt valgt ut to eksisterende arbeider som skal være med på å utforme den endelige løsningen der begge disse inneholder funksjonalitet som er ønskelig å benytte seg av og bygge videre på.

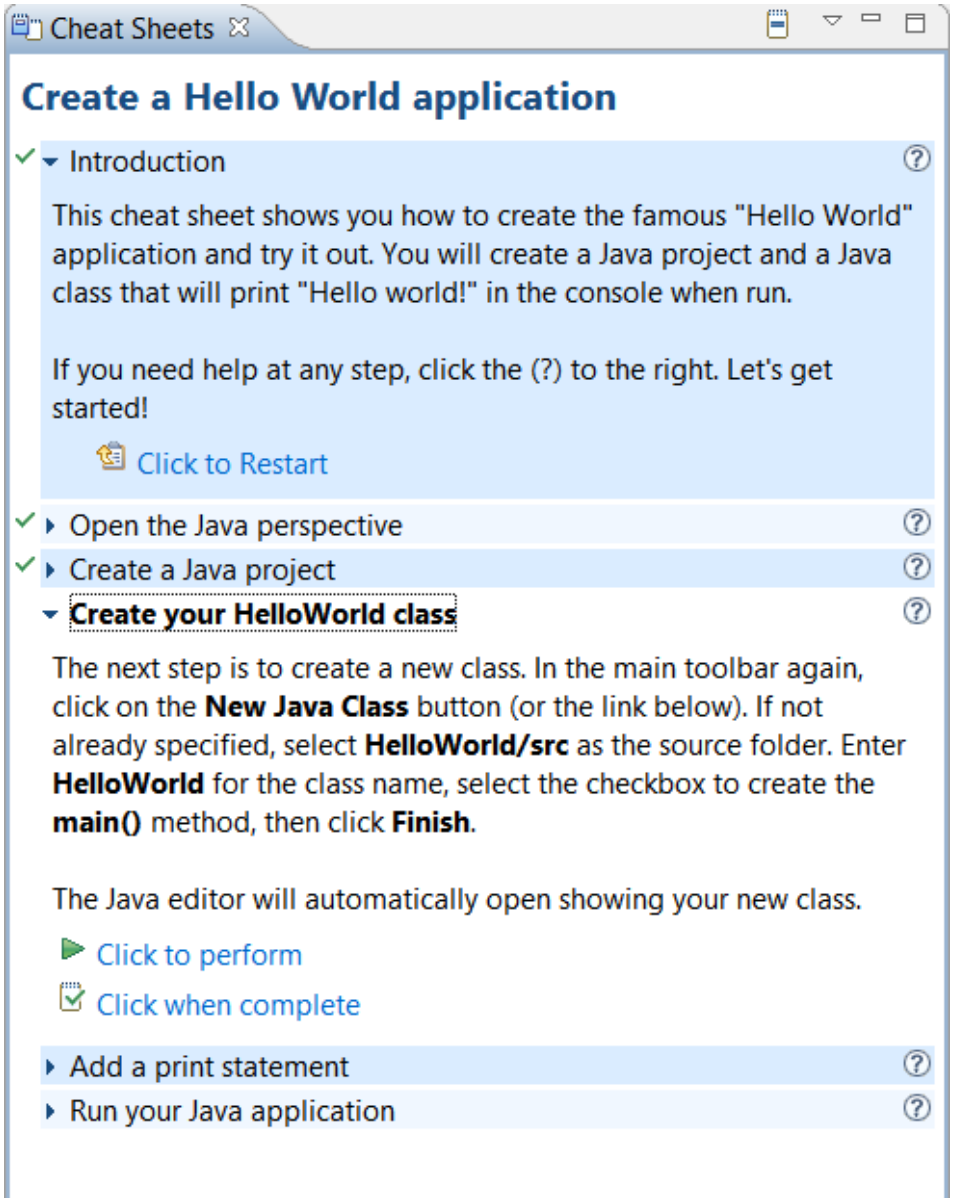
3.4.1 Eclipse Cheat Sheet

Et vindu i Eclipse som inneholder funksjonalitet til å guide brukeren skritt for skritt med å fullføre praktiske oppgaver. Dette er utformet slik:

- En hovedoppgave som er delt opp i mange deloppgaver.
- Et stort problem som blir delt opp i mange små delproblemer, der hvert delproblem er enkelt å løse.
- Deloppgavene må løses i rekkefølge for å fullføre hovedoppgaven.
- Det er en logisk rekkefølge på hvordan deloppgavene skal løses, det vil si at den neste deloppgaven er avhengig av den forrige.
- En fullført deloppgave markeres med en avkrysning.
- Det blir gitt tilbakemelding til brukeren som indikerer at en deloppgave er blitt løst.

- Hver deloppgave har en spørsmålstegn-knapp som gir ekstra hjelp til hvordan oppgaven kan løses for å unngå at brukeren setter seg fast.
- Hjelpe brukeren med å automatisk utføre oppgaver ved at systemet utfører deler eller hele deloppgaven for å opprettholde flyt i oppgaveløsningen.
- Når brukeren klikker på knappen som markerer deloppgaven som fullført, aktiveres automatisk neste ikke-fullførte deloppgave.

Figur 3.2 viser en skjermdump av Eclipse sitt innebygde "Cheat Sheet" der oppgaven går ut på at brukeren skal opprette et javaprosjekt med en javafil som skal printe ut teksten "Hello World!".



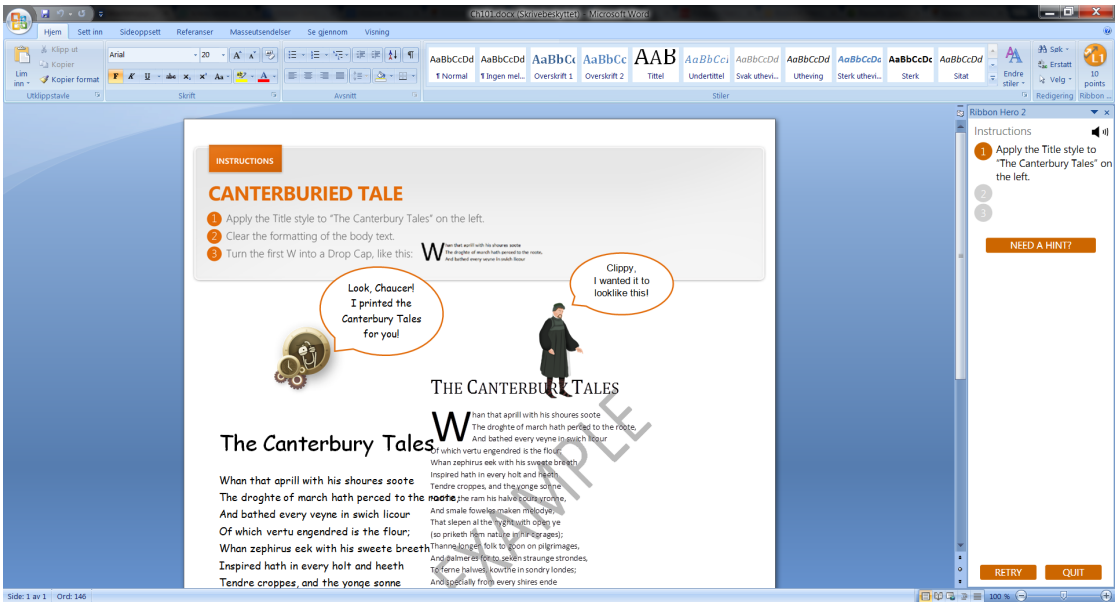
Figur 3.2: Skjermdump av Eclipse Cheat Sheet.

3.4.2 Ribbon Hero 2

Ribbon Hero 2 [Corporation, 2011], er et gratis spillifisert læringsverktøy utviklet av Office Labs som lar brukeren utforske og utføre en stor mengde av de grunnleggende og nye funksjonene til Microsoft Office 2007/2010. Verktøyet baserer seg på at brukeren skal få kunnskap og erfaring gjennom direkte interaksjon med noen av Microsoft Office sine programmer:

- Microsoft Word - Tekstbehandlingsprogram
- Microsoft Excel - Regnearkprogram
- Microsoft OneNote - Notatprogram
- Microsoft PowerPoint - Lysbildepresentasjonsprogram

Brukeren følger en spillhistorie der de hjelper Clippy, den tidligere office-assistenten, med å reise til ulike tidsepoker og løse de tilhørende oppgavene. Poengene som oppnås etter å ha gjennomført en oppgave, gjenspeiler ferdighetene brukeren har tilegnet seg. Figur 3.3, viser en skjermdump av Ribbon Hero 2 hvor oppgaven er at brukeren skal anvende en tittelstil på en overskrift.



Figur 3.3: Skjermdump av Ribbon Hero 2.

Mekanismer innenfor spillifisering

Spillmekanismer innenfor spillifisering som finnes i Ribbon Hero 2.

Oppgaver (eng.:challenges)
For hvert nivå er det oppgaver som hver består av et sett med deloppgaver. Oppgavene blir mer utfordrende for hvert nivå.

Progresjonslinje (eng.:progression bar)
Viser nåværende poengsum og hvor mange poeng som trengs for å nå et nytt nivå.

Nivå (eng.:levels)
De ulike tidsepokene blir låst opp en etter en for hvert nivå som oppnås.

Poeng (eng.:points)
Det blir gitt poeng for hver oppgave som blir fullført.

Bonus (eng.:bonuses)
Det er lagt opp til at brukeren kan oppnå bonuser i løpet av spillet. Hvis en oppgave blir fullført uten å ta i bruk hint vil brukeren få bonuspoeng.

Oppdagelse (eng.:discovery)
Det er gjemt bonuspoeng rundt i de forskjellige programmene til Microsoft Office som spilleren tjener ved å utforske funksjoner.

Status
Spillerens ferdighetsnivå indikerer hvilket nivå spilleren har oppnådd.

Atferdsmomentum (eng.:behavioral momentum)
Å løse oppgaver knyttet til Microsoft Office sine programmer skal føles nyttefult for en som ønsker å lære seg de grunnleggende funksjonene. Hvis det føles nyttefult vil brukeren dermed fortsette å spille videre.

Tapsaversjon (eng.:loss aversion)
Spillet styrer bort i fra å straffe spillere som ikke klarer eller sliter med en oppgave. Hvis en spiller bruker lang tid på en oppgave, vil hint-knappen blinke og spilleren kan få små hint til framgangsmåte. På den måten vil alle spillere alltid klare å fullføre oppgavene.

Uendelig spillbarhet (eng.:infinite gameplay)
Spillet følger en historie og har en slutt, men det er mulig å bla gjennom tidsepokene og å spille gjennom oppgavene på nytt i ettertid.

Kapittel 4

Design

Basert på litteraturgjennomgangen i i forrige kapittel, blir det i dette kapittelet gjort rede for de ulike valgene som er tatt angående spillifisering og funksjonalitet.

4.1 Generell beskrivelse av applikasjonen

Designet til den ferdige løsningen er inspirert av læringsmålene til programmeringsfaget TDT4100, Ribbon Hero 2, Eclipse sin innebygde cheat sheet, samt generell spillifisering og spillmekanikkene som er forklart i det forrige kapittelet. Målet er å motivere studenter til å tilegne seg praktiske ferdigheter i Eclipse ved å ta i bruk teknikkene til spillifisering.

4.2 Valgte bruksoppgaver

Basert på programmeringskurset TDT4100 sine læringsmål er det blitt laget et sett med oppgaver bestående av stegvise deloppgaver.

4.2.1 Opprette et Java-prosjekt

1. Klikke på knappen “New” i verktøylinja.
2. Velge “Java Project”.
3. Klikke på knappen “Next”.
4. Skrive inn prosjektnavn og klikke på knappen “Finish”.

4.2.2 Åpne Debuggingsperspektiv

1. Klikke på knappen “Window” i menylinja.
2. Klikke på “Open Perspective”.
3. Klikke på knappen “Debug”.

4.2.3 Åpne Konsollvinduet

1. Klikke på knappen “Window” i menylinja.
2. Klikke på “Show View”.
3. Klikke på knappen “Console”.

4.2.4 Åpne Java Build Path

1. Klikke på et vilkårlig javaprosjekt i “Package Explorer”.
2. Høyreklikke på det valgte javaprosjektet.
3. Klikke på knappen “Properties”.
4. Klikke på “Java Build Path”.

4.2.5 Åpne Innstillinger

1. Klikke på knappen “Window” i menylinja.
2. Klikke på “Preferences”.

4.2.6 Bruke innholdsassistenten

1. Åpne en vilkårlig javafil.
2. Skrive “sysout” inne i en metode.
3. Klikke på “Ctrl” + “SPACE” på tastaturet.

4.3 Valgte spillmekanikker

De spillmekanikkene som ble valgt ut og implementert i den ferdige løsningen, er de som kan implementeres med Eclipse sitt applikasjonsprogrammeringsgrensesnitt (API) og innenfor prosjektets omfang.

4.3.1 Oppgaver

Det er et sett med oppgaver, kalt "Scenarioer", hvor hvert gjennomført Scenario skal resultere i en ferdighet.

4.3.2 Deloppgaver

Hvert Scenario består av en rekke handlinger, kalt "Tasks", som brukeren skal utføre for å komme et steg nærmere å fullføre et Scenario.

4.3.3 Atferdsmomentum

Å løse oppgaver knyttet til programmet Eclipse skal oppfattes som nyttefult og lærerikt hos brukeren. Ved å ha interessante og meningsfylte oppgaver, vil dette øke sannsynligheten for at brukeren ønsker å fortsette å spille videre.

4.3.4 Tapsaversjon

Den ferdige løsningen er designet slik at det er meningen at alle skal klare å løse oppgavene som er der. Hvis en deloppgave oppleves som vanskelig og brukeren ikke vet hvordan denne skal løses, finnes det en hjelpeknapp som viser hvordan deloppgaven skal løses.

4.4 Rettningslinjer for design

Det finnes forskjellige retningslinjer for hvordan et brukervennlig system kan designes.

4.4.1 Prototype

I konteksten av en applikasjon, er en prototype en ufullstendig versjon av et tenkt system. Det skilles mellom to hovedtyper av prototyper:

1. Horisontal prototype: Gir en kort oversikt over systemet som helhet uten å fokusere på noen typer funksjonalitet.
2. Vertikal prototype: Gir en bred og ferdig modell av en spesifikk funksjon til det tenkte systemet.

4.4.2 Brukervennlighet

Blir beskrevet som hvor enkelt et system kan oppfattes av de som skal bruke det. Jakob Nielsen [Nielsen, 2012] nevner fem attributter som skal definere et brukervennlig system:

- Læringsdyktighet: Hvor raskt en bruker som ikke kjenner til systemet til å begynne å bruke det.
- Effektivitet: Ved å ha kjennskap til systemet skal brukeren kunne oppnå høy produktivitet.
- Memorisering: Brukere som sjeldent benytter seg av systemet skal ikke trenge å lære seg fra bunnen av hvordan systemet skal brukes.
- Feil: Det skal forhindres at brukerne gjør feil, og hvis de oppstår skal de være enkle å komme tilbake fra.
- Tilfredsstillelse: Tilpass systemet etter den som benytter systemet for å sørge for at brukeren liker det.

4.5 Utviklingsverktøy

Denne delen forklarer de ulike verktøyene som er blitt brukt for å implementere prototypen.

4.5.1 Eclipse

Eclipse er blitt brukt til å skrive kildekoden som utgjør den ferdige løsningen. I tillegg til å være et utviklingsverktøy for implementering, har Eclipse også vært brukt til kontinuerlig testing og debugging i løpet av utviklingen.

4.5.2 Visio 2013

Visio 2013 er et applikasjonsverktøy fra Microsoft som tillater brukeren å lage store komplekse diagrammer der disse har et oversiktlig utseende.

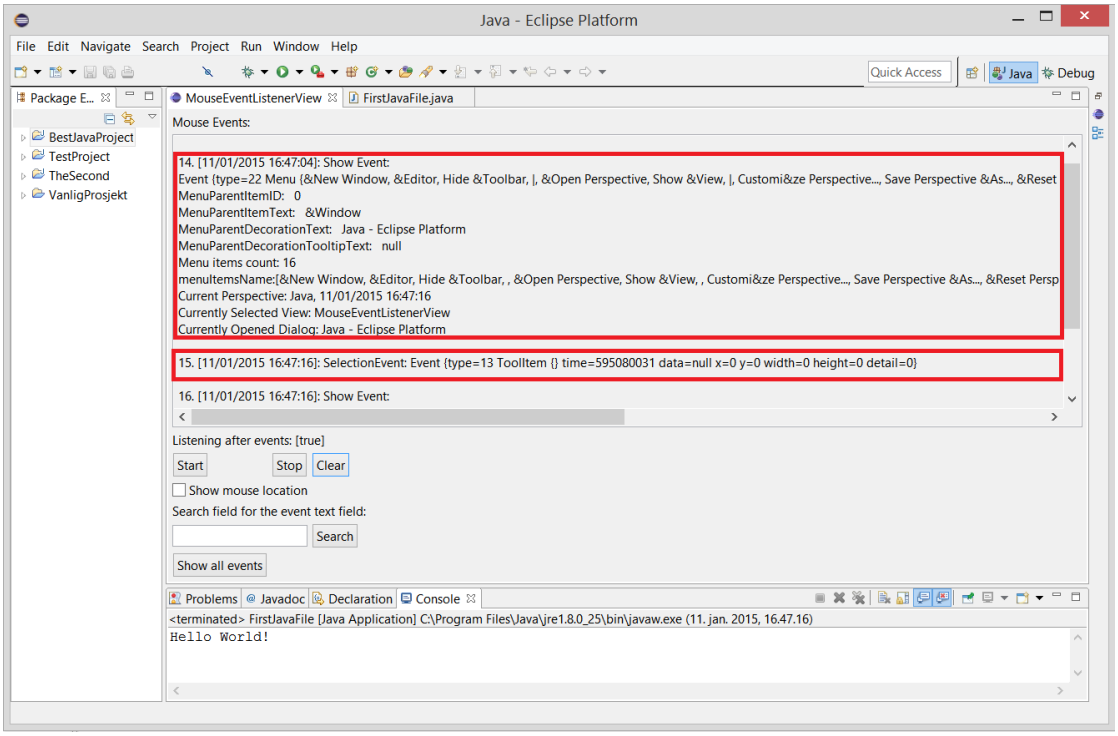
Kapittel 5

Implementasjon

Dette kapittelet beskriver hvordan prototypen ble implementert, arkitekturen og den ferdige løsningen.

5.1 Utforskning av Eclipse

I startfasen av implementeringen har det blitt gjort en undersøkelse av hvordan Eclipse kan utvides for å utnytte dets funksjoner og underliggende arkitektur. Her er det blitt lagd et utforskningsvindu kalt "MouseEventListenerView", som skriver ut nyttig informasjon basert på ulike måter en bruker interagerer med Eclipse. Formålet med dette vinduet var å utforske hvilke hendelser som i praksis trigget de forskjellige lytterne med tanke på bruksoppgavene som ble laget. Brukergrensesnittet til vinduet har et simpelt design bestående av en tekst-control, start-knapp, stop-knapp, clear-knapp, avkryssningsboks for å vise koordinater til musepekeren, søkefunksjon for å kunne søke gjennom tekst som har blitt skrevet ut, og en knapp for å vise alt som er blitt skrevet ut i tekstfeltet.



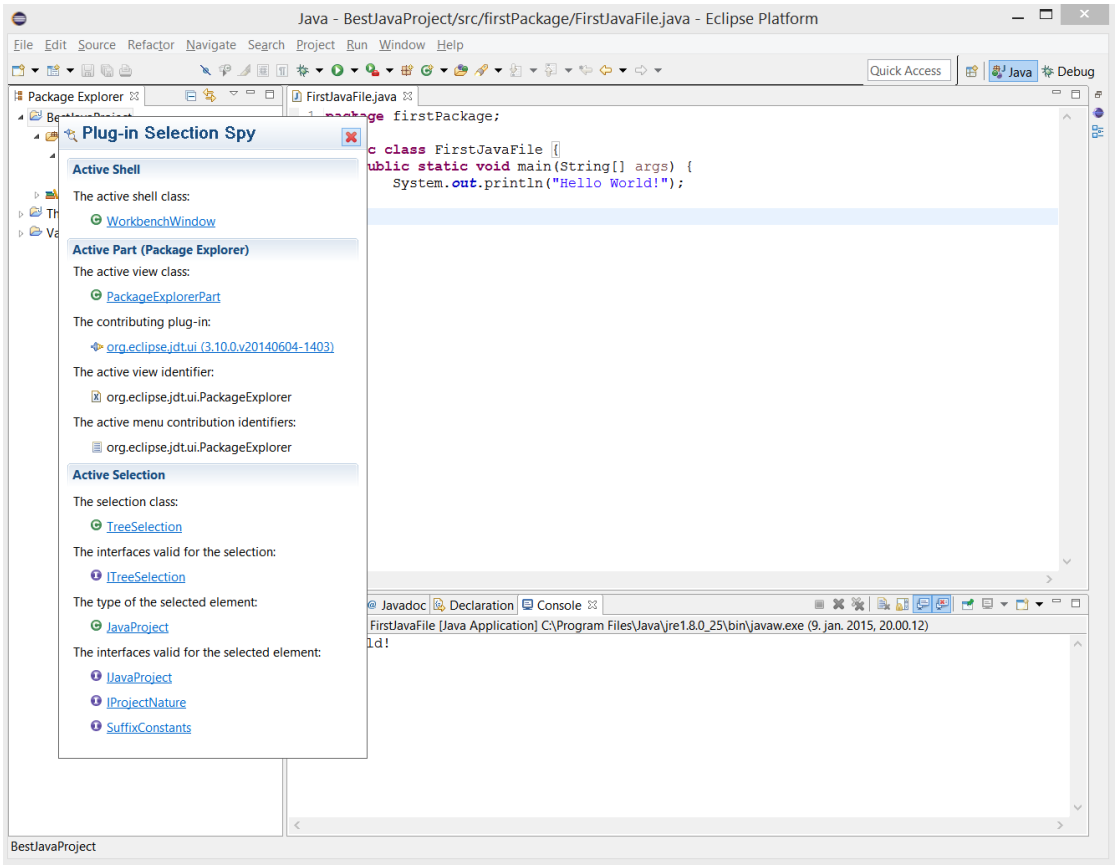
Figur 5.1: Skjermdump av MouseEventListenerView i Eclipse.

Øverst i brukergrensesnittet ligger det en tekst-control, et objekt som gjør det mulig å skrive inn eller vise tekst. I det brukeren utfører handlinger i Eclipse og det oppstår en hendelse, vil lytteren som ble trigget generere en tekststreng med informasjon om hendelsen og skrive ut tekststrengen i tekst-control. Knappene nederst i brukergrensesnittet utfører operasjoner på objektet tekst-control. Ved å klikke på start-knappen, vil hendelser som oppstår bli skrevet ut i tekst-control. Stop-knappen gjør at tekst-control ikke vil skrive ut hendelser som oppstår, og clear-knappen vil simpelthen fjerne teksten som befinner seg i tekst-control. Muligheten til å starte og stoppe printingen av hendelser er lagt til for å unngå støy i form av nye uønskede hendelser. Dette er relevant når det ønskes å utføre en nærmere undersøkelse av hendelsene som allerede befinner seg i tekst-controller. Søkefunksjonen er lagt til for å kunne filtrere vekk alle andre resultater når det er ønskelig å konsentrere seg om utskrift av en bestemt type. Knappen "Show all events", er lagt til for å fylle tekstfeltet med alle tekststrenger som er blitt skrevet ut i løpet av den gjeldende kjøringen. Det vil si at hvis søkefunksjonen blir brukt til å kun vise tekststrenger av en bestemt type, vil et trykk på "Show all events" endre innholdet i tekstfeltet til å vise utskriften som var der før søkefunksjonen ble benyttet.

Figur 5.1, viser at det er forskjellige lyttere som blir trigget av ulike handlinger. Ved å bruke musepekeren til å klikke på en av knappene som befinner seg i menylinja, vil

lytteren av type "SWT.Show" bli trigget, mens et klikk på en av knappene i verktøylinja vil føre til at lytteren av type "SWT.Selection" vil bli trigget. Dette er et eksempel på hvordan "MouseEventListenerView" har vært en viktig brikke i det å kunne oppdage og forstå hvordan Eclipse fungerer.

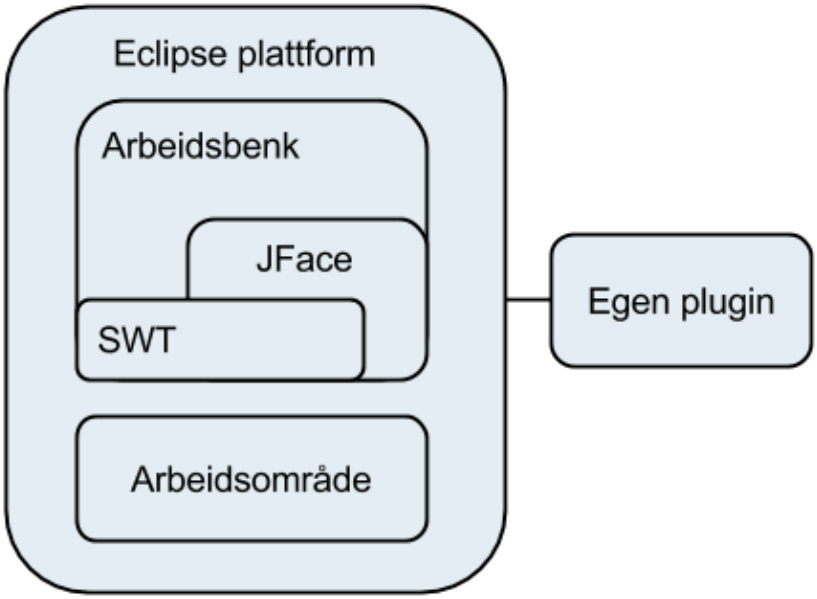
I tillegg til "MouseEventListenerView" er det blitt brukt et verktøy innebygd i Eclipse som gjør det mulig å hente ut informasjon som kan være relevant i utviklingen av nye pluginer. Dette blit kalt for "PDE Incubator Spy" [Foundation,], også kjent som "plugin-spy", der det ved å klikke "ALT+SHIFT+F1" på tastaturet vil dukke opp et lite vindu med informasjon om den valgte brukergrensesnitt-komponenten i Eclipse. Figur 5.2 viser en skjermdump av plugin-spy som er blitt brukt til å hente ut informasjon om et valgt javaprosjekt i vinduet "Package Explorer" i Eclipse.



Figur 5.2: Skjermdump av plugin-spy i Eclipse.

5.2 Arkitektur

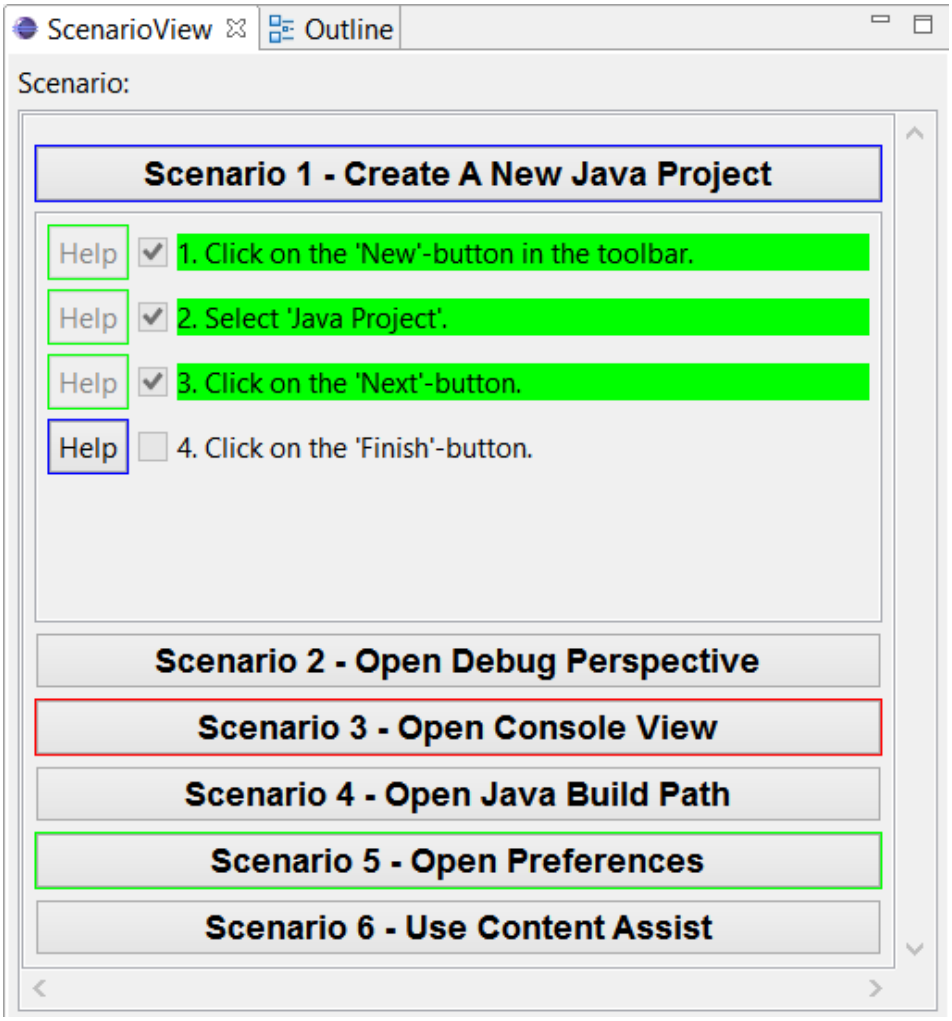
Den ferdige løsningen er i form av en plugin som installeres direkte inn i Eclipse. Figur 5.3 viser hvordan den implementerte pluginen er koblet opp mot Eclipse.



Figur 5.3: Integrasjon av plugin opp mot Eclipse

5.2.1 Brukergrensesnitt

Det grafiske brukergrensesnittet består av en ScrolledComposite(en beholder for grafiske elementer) med en rekke knapper der hver knapp inneholder navnet samt beskrivelsen til hvert Scenario. Ved å klikke på en av scenario-knappene, vil den underliggende modellen sette dette som det gjeldende Scenarioet, og en relevant Composite vil komme til syne i brukergrensesnittet under knappen. Det gjeldende scenarioet vil bli markert ved å ha en blå kantlinje rundt Scenario-knappen. Når et Scenario først er blitt aktivert, er det ikke mulig å aktivere et annet Scenario før det gjeldende er blitt deaktivert. Dette er blitt gjort for å gjøre det enklere å lytte etter hendelser som brukeren forårsaker samt unngå konflikter mellom lytterne, i tillegg til at det vil være mer oversiktlig for brukeren å forholde seg til et Scenario om gangen. Figuren 5.4, er en skjermdump av brukergrensesnittet til prototypen som er plassert som et vindu inne i programmet Eclipse.



Figur 5.4: Skjermdump av ScenarioView i Eclipse

Etter at alle deloppgavene i et Scenario er fullført, vil Scenarioet regnes som fullført og kantlinjen rundt scenario-knappen vil bli fargelagt grønn, i tillegg til at en dialogboks vil dukke opp hvor brukeren blir gratulert for å ha fullført Scenarioet. Hvis brukeren underveis ønsker å avbryte et Scenario, vil et trykk på tilhørende Scenario-knapp trigge en dialogboks der brukeren blir forespurt om å deaktivere Scenarioet. Dersom et ikke-fullført Scenario deaktiveres, vil den tilhørende TaskComposite bli skjult, og kantlinjen rundt Scenario-knappen vil bli fargelagt rødt for å markere at ikke alle deloppgavene i Scenarioet er fullførte.

Hvert Scenario har sin egen Composite, kalt TaskComposite, som kommer til syne i det et Scenario blir aktivert. Et Scenario består av et sett med Task-objekter, deloppgaver, som blir representert av en help-knapp, en avkryssningsboks og en label. Beskrivel-

sen av hver deloppgave til Scenarioet kommer fram i labelen. Avkrysningsboksen foran hver label, viser om en deloppgave er fullført eller ikke. Når en deloppgave er aktivert, blir dette markert ved å fargelegge kantlinjen rundt Help-knappen blå, og brukeren prøve å løse denne ved å lese beskrivelsen til oppgaven og foreta seg handlinger i Eclipse. Dersom disse handlingene førte til at deloppgaven ble fullført, vil brukergrensesnittet representere dette ved å deaktivere help-knappen og krysse av avkrysningsboksen, samt fargelegge kantlinjen til Help-knappen grønn og fargelegge labelen grønn. Help-knappen som ligger ved avkrysningsboksen, er et hjelpemiddel for automatisk utførelse av den gjeldende deloppgaven. Det er meningen at brukeren skal kunne klikke på denne knappen hvis det skulle trenges hjelp til å klare deloppgaven. Dersom denne blir klikket på, vil pluginen aktivere kode som fullfører deloppgaven ved og programmatisk simulere brukerens interaksjoner.

Den underliggende modellen fungerer slik at når et Scenario aktiveres, vil den første ikke-fullførte deloppgaven i scenariet automatisk aktiveres. Ved fullført deloppgave, vil pluginen sjekke om det finnes flere uløste deloppgaver. Hvis dette er tilfelle vil den neste deloppgaven aktiveres, ellers vil Scenarioet bli markert som fullført. Grunnen til at pluginen automatisk aktiverer en deloppgave, er for å unngå unødvendig kompleksitet ved at brukeren selv må klikke på en knapp for å aktivere den neste deloppgaven. En annen grunn er at dette vil på en enkel måte sikre at deloppgavene blir løst i riktig rekkefølge.

5.2.2 Rammeverk

Etter hvert som pluginen gjennom iterasjoner er blitt utvidet til å støtte mer funksjonalitet, er kildekoden blitt gjort om til et rammeverk for å opprette scenarier og de tilhørende Task-objektene. Dette er blitt gjort for at det enkelt skal kunne opprettes nye og spennende scenarier i framtiden, i tillegg til å utnytte fordelen med å opprettholde enkel og ryddig kode.

Objektorientert Løsning Et design-pattern som er brukt i den ferdige løsningen for og enkelt holde flere objekter konsistente med et annet objekt, kalles for observatør-observert-teknikken [Trætteberg, 2014]. I denne teknikken skilles det gjerne mellom to ulike roller som et objekt kan ha. Den første rollen kalles en observatør - et objekt som lytter etter endringer, mens den andre rollen er observert - et objekt som blir lyttet på.

Observatør - objektet som lytter
For at den observerte skal kunne si ifra om tilstandsendringer til alle sine lyttere, må observatørene ha en felles lytter-struktur slik at de kan bli sett på som lytter-objekter. Denne strukturen blir definert i et grensesnitt som kalles for et lytter-grensesnitt, som

er en gruppe med abstrakte metoder som observatørene må implementere i sine klasser. Disse metodene fungerer som en kobling slik at lytterne kan motta beskjed om at det har skjedd en endring i det objektet som de lytter på. Lytter-grensesnittet kan bestå av mange metoder, der en metode kan gi beskjed om at noe er aktivert/deaktivert, mens en annen metode kan gi beskjed om at noe er tilføyd/slettet.

Observert - objektet som blir lyttet på
Egenskaper som kjennetegner den observerte er at dette objektet skal kunne registrere observatører. For å holde kontroll på lytterne er det ønskelig at objektet som blir lyttet på inneholder en samling med alle observatører. Antallet lyttere som finnes i denne dynamiske samlingen endres ved å kalle den observerte sine metoder for å tilføye eller fjerne lyttere. Dette gjør det mulig for at objekter kan starte å lytte på et objekt når som helst, samt slutte å lytte etter at en gitt endringshendelse har inntruffet. En annen egenskap er at den observerte skal kunne endre tilstanden sin. Grunnen til at objekter ønsker å lytte på et annet objekt, er fordi at den observerte nettopp skal kunne endre sin tilstand. Den observerte skal kunne gi beskjed til alle lytterne sine om at en endringshendelse har inntruffet. Dette er hovedegenskapen i observatør-observert-teknikken og selve mekanismen for å holde tilstanden til de andre observatørene konsistente. Måten observatør-observert-teknikken fungerer på i praksis er som følger:

1. Den observerte legger til observatører i sin samling med lyttere.
2. Den observerte endrer tilstanden sin.
3. Den observerte itererer gjennom samlingen med observatører og kaller hver observatør sin metode som er relatert til den inntrufne tilstandsendringen.

Scenario.java Et Scenario er en oppgave som kan fullføres ved å løse et sett med deloppgaver (Task). Et eksempel på et Scenario kan være å opprette et Java-Prosjekt. Dette scenarioet vil da bestå av et sett med bestemte deloppgaver som vil resultere i at et Java-Prosjekt blir opprettet.

Scenario-klassen inneholder en samling av Task-instanser, samt logikk for å aktivere/deaktivere et Scenario og dets neste uløste Task. For at brukergrensesnittet skal være konsistent med tilstanden til de underliggende objektene, er observatør-observert-teknikken blitt benyttet. I forhold til denne teknikken har Scenario-klassen rollen som "observert", ved at den har mulighet til å registrere lyttere, også kalt "observatører", og gi beskjed til alle lytterne når det skjer tilstandsendringer. I denne sammenhengen vil lytterne få beskjed når en Task er aktivert, når en Task er fullført og når et Scenario er fullført.

Task.java En Task er en deloppgave som brukeren skal løse. Dette kan for eksempel være at brukeren skal trykke på en bestemt knapp.

Task-klassen er en abstrakt klasse som inneholder logikk for å aktivere, deaktivere og fullføre en Task. Det at klassen er av typen abstrakt vil si at dette kun er en mal for hvordan en Task skal se ut , noe som betyr at denne klassen ikke kan instansieres. Subklassene arver Task sin logikk, samt to abstrakte metoder addListeners() og removeListeners(). At disse to metodene er abstrakte vil si at de er ufullstendige og at subklassene må implementere de ferdige. Meningen med addListeners() er at i det en Task blir aktivert, så skal det legges til relevante lyttere som lytter etter hendelser som forekommer i Eclipse. Når ønsket hendelse inntreffer og det er verifisert at denne fullførte en Task, så blir en Task markert som fullført og lytterne skal fjernes ved å kalle removeListeners().

SwtTask.java En SwtTask er en abstrakt spesialisert subklasse av Task som oppretter en lytter av hendelsestypen SWT. Noen eksempler på en slik type hendelser kan være seleksjon, tastattur- eller muse-hendelser. Subklasser av SwtTask må implementere den abstrakte metoden isTaskEvent(), en boolsk metode som sjekker om den inntrufne hendelsen førte til at en deloppgave ble fullført.

MenuBarClick.java Denne klassen inneholder kode for en bestemt deloppgave, hvor denne blir fullført i det brukeren klikker på en av knappene på hovedmenybaren. Et eksempel på en slik oppgave kan være at brukeren skal klikke på knappen “File”.

Dette er en subklasse av SwtTask og lytter etter hendelser av typen SWT.Show, en hendelsestype som inntreffer når for eksempel et vindu eller en meny blir vist. Logikken for å avgjøre om en inntruffet hendelse fullførte deloppgaven ligger i isTaskEvent(). Når det opprettes en instans av MenuBarClick, vil det via en parameter bli spesifisert hvilken menybar-knapp som det skal lyttes etter.

ScenarioView.java Denne klassen er den grafiske representasjonen av pluginen, som viser en oversikt over alle Scenario som brukeren kan velge mellom, hvilke Tasks disse består av, hvilke Tasks som er fullført og hvilke Scenario som er fullførte/ikke fullførte. I forhold til observatør-observert- teknikken har denne klassen rollen som observatør. Dette blir gjort ved at ScenarioView registrerer seg som lytter på hvert Scenario, samt implementerer lytte-grensesnittet ScenarioProgressListener sine metoder.

Et Scenario blir representert ved en Scenario-knapp som brukeren kan klikke på for å aktivere eller deaktivere det gjeldende scenarioet.

Hvert Scenario inneholder Tasks, hvor den enkelte Task blir representert av en "Help"-knapp, en avkryssningsboks og en oppgavebeskrivelse. Ved usikkerhet rundt hvordan en deloppgave skal løses, kan brukeren trykke på "Help"-knappen for å få programmet til og automatisk utføre den og alle tidligere deloppgaver i det gjeldende Scenarioet. For hver deloppgave som fullføres, vil avkrysningsboksen automatisk bli krysset av.

Til slutt når alle deloppgavene i et Scenario er fullførte, regnes Scenarioet som fullført og kantlinjen rundt Scenario-knappen blir fargelagt grønn.

Hvis brukeren allerede jobber med et Scenario og heller ønsker å jobbe med et annet Scenario, må først det gjeldende Scenarioet lukkes ved å klikke på den respektive Scenario-knappen. Da vil Scenarioet regnes som ikke-fullført før det deaktiveres, og kantlinjen rundt Scenario-knappen vil bli fargelagt rødt.

ScenarioProgressListener.java Dette er selve lytter-grensesnittet som observatørene må implementere for å kunne lytte på tilstandsendringer hos den observerte. Grensesnittet består av metodene:

- taskActivated(Task task) - kalles når en Task er blitt aktivert.
- taskCompleted(Task task) - kalles når en Task er blitt fullført.
- scenarioCompleted(Scenario scenario) - kalles når et Scenario er blitt fullført.

Når en Task er blitt fullført, har det dermed oppstått en tilstandsendring hvor Scenario.java (den observerte) gir beskjed om endringen til ScenarioView.java (observatøren). Denne beskjeden blir gitt ved at den observerte kaller metoden taskCompleted(Task task) som ligger i observatøren. I dette tilfellet vil Scenario kalle ScenarioView sin taskCompleted-metode, en metode som sørger for at brukergrensesnittet blir oppdatert ved å deaktivere "Help"-knappen og krysse av avkrysningsboksen til den fullførte deloppgaven.

I det et Scenario er blitt fullført, vil Scenario.java gi beskjed om endringen ved å kalle ScenarioView sin metode kalt scenarioCompleted(Scenario scenario). Metoden vil oppdatere brukergrensesnittet til ScenarioView ved å vise et popup-vindu som gratulerer brukeren med å fullføre et Scenario, samt fargelegge kantlinjen til Scenario-knappen grønn.

AutoExecution.java Enhver Task kan inneholde et objekt av typen AutoExecution, som er en abstrakt klasse som inneholder kode for å automatisk utføre en deloppgave uten interaksjon fra brukeren. Hvis deloppgaven er av typen MenuBarClick hvor

knappen som skal trykkes på i hovedmenybaren er “File”, vil objektet av typen AutoExecution inneholde koden som flytter musepekeren til “File”-knappen og simulerer et museklikk.

Klassen er av typen abstrakt og inneholder kun en tittel og en abstrakt metode kalt execute(). Tittelen er en strengvariabel og blir lagt til som parameter ved subklasseinstansiering av denne abstrakte klassen. Den tomme metoden execute() som er av typen abstrakt, må bli implementert i subklassene da den vil inneholde koden som programmatisk utfører deloppgaven.

MenuBarClickExecution.java Denne klassen inneholder kode for å utføre et programmatisk klikk på en av knappene som befinner seg i hovedmenybaren til Eclipse. Et eksempel på dette kan være å utføre et klikk på “File”-knappen i hovedmenybaren til Eclipse helt uten interaksjon fra brukeren.

Dette er en subklasse av AutoExecution og legger ved en tittel som parameter ved instansiering. Basert på tittelen gitt ved parameter, vil metoden execute() utføre kode som plasserer musepekeren til den tilhørende knappen på hovedmenylinja, og simulerer et museklikk.

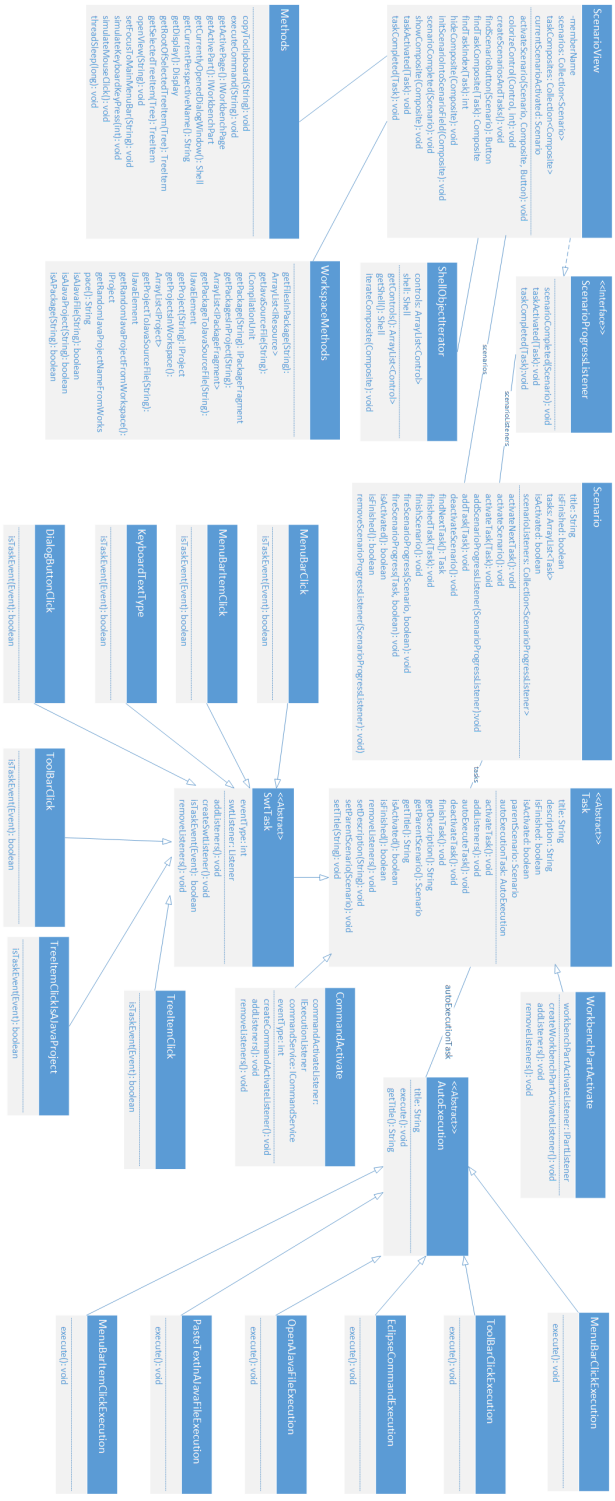
Methods.java En omfattende klasse som inneholder generelle statiske hjelpemetoder som kan kalles fra de andre klassene. Formålet med denne klassen er at metodene skal gjøre grovarbeidet slik at de andre klassene kan oppnå en enklere og ryddigere kode. Eksempler på metoder som finnes i denne klassen er simulateMouseClick og simulateKeyboardPress, hvor begge er metoder som simulerer interaksjon fra brukeren.

WorkspaceMethods.java En omfattende klasse som inneholder statiske hjelpemetoder som utfører arbeid på brukeren sitt workspace, som er selve arbeidsområdet til Eclipse der alle prosjektene og tilhørende filer befinner seg. Her finnes det metoder som for eksempel itererer gjennom workspace og henter ut alle prosjektene som finnes der, samt en metode som sjekker om en fil er en javaklasse.

ShellObjectIterator.java En klasse som tar inn et objekt av typen Shell eller Composite, itererer rekursivt gjennom denne og returnerer en liste med alle control-objektene som den finner. Denne listen kan brukes til å sjekke om et Shell eller en Composite inneholder et control-objekt av en bestemt type.

5.2.3 Klassediagram

Klassediagram i figur 5.5 viser oversikten over alle klassene som er blitt opprettet og hvordan de henger sammen.



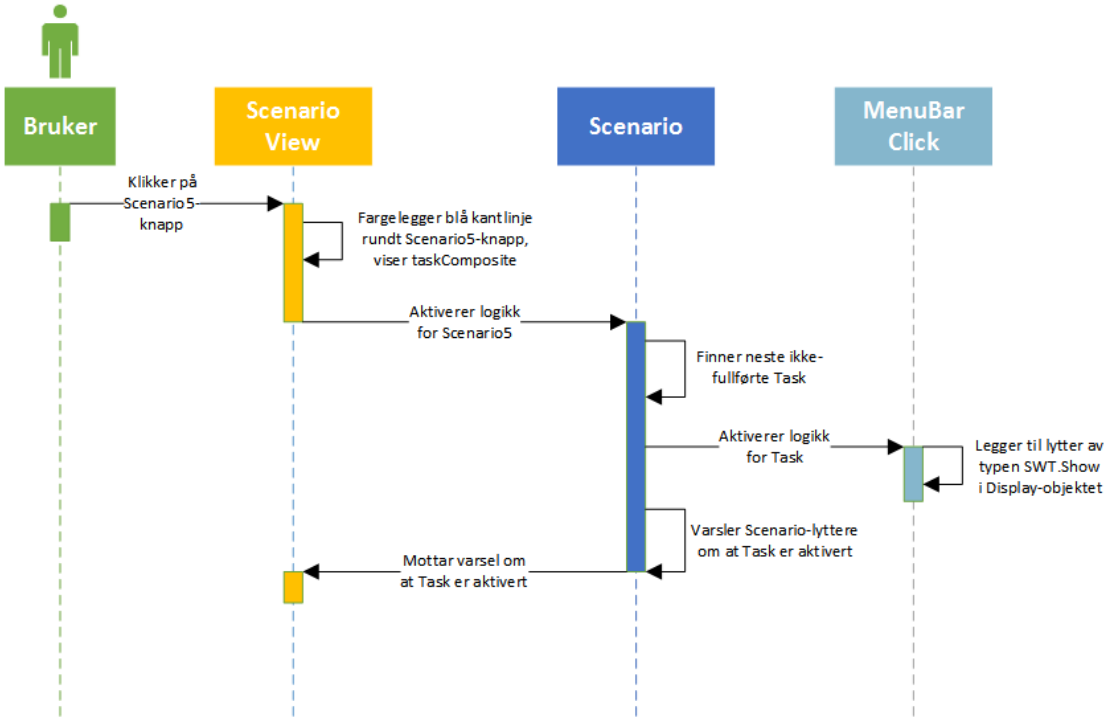
Figur 5.5: Klassediagram over pluginen.

5.2.4 Sekvensdiagram

Et sekvensdiagram viser hvordan systemet fungerer i det en spesifikk tilstand oppstår som følger av at brukeren utfører handlinger.

Aktivere et Scenario

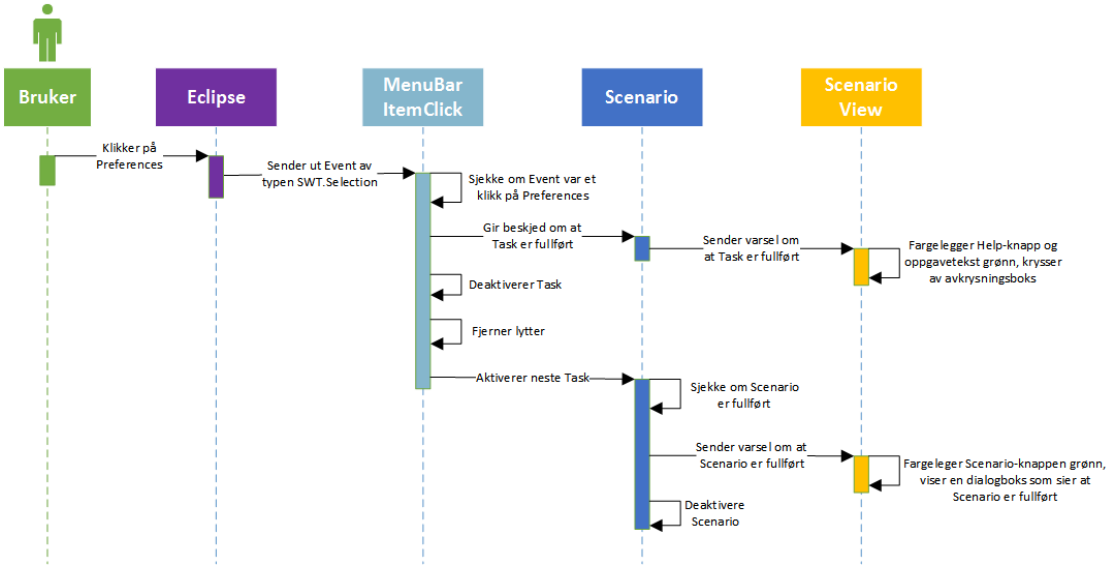
Figuren 5.6 illustrerer hvordan systemet reagerer når ingen Scenario er aktivert, og at brukeren klikker på Scenario 5 - knappen.



Figur 5.6: Bruker klikker på Scenario-knapp 5.

5.2.5 Fullføre et Scenario

Figuren 5.7 illustrerer hvordan systemet reagerer når brukeren fullfører Scenarioet ved å klikke på Preferences-knappen.



Figur 5.7: Bruker har fullført et Scenario.

5.2.6 Utvidelse av kode

Pluginen består av et rammeverk hvor det er en løs kobling [Pressman, 2010] mellom klassene, noe som gjør det enkelt å legge til nye Scenarios og Tasks i ettertid uten å risikere at pluginen slutter å fungere. Nedenfor følger et eksempel på hvordan det kan opprettes et nytt Scenario med en Task hvor brukeren skal bruke tastaturet til å utføre globale hotkeys i Eclipse.

Først må det opprettes en ny klasse som utvider SwtTask:

```
1  public class KeyboardHotkeyPush extends SwtTask{
2      public KeyboardHotkeyPush(String hotkeyCombination, String description,
       AutoExecution autoExecuteTask){
3          super(hotkeyCombination, description, SWT.KeyUp, autoExecuteTask);
4      }
5      protected boolean isTaskEvent(Event event){
6          if( eventCompletesHotkeyCombination ){
7              return true;
8          }else{
9              return false;
10         }
11     }//isTaskEvent.
12 }//KeyboardHotkeyPush.
```

Deretter må det opprettes en ny klasse som programmatisk skal utføre et hotkey-trykk:

```
public class KeyboardHotkeyPushExecution extends AutoExecution{
    public KeyboardHotkeyPushExecution(String hotkeyCombination){
        super(hotkeyCombination);
    }
    public void execute(){
    //Kode som programmatisk utfører et keyboard hotkey push.
    }//execute.
}//KeyboardHotkeyPushExecution.
```

Til slutt må det opprettes et nytt Scenario bestående av Task-instanser i ScenarioView:

Opprette et nytt Scenario

```
Scenario scenario7 = new Scenario("Scenario 7 - Kommentere ut en linje");
```

Legge til Task til Scenario

```
scenario7.addTask(new KeyboardHotkeyPush ("Ctrl+/", "1.Hold down 'Ctrl' and press
    '/' on the keyboard.", new KeyboardHotkeyPushExecution ("Ctrl+/")));
```

Legge til Scenario i samlingen med Scenario-objekter i klassen "ScenarioView"

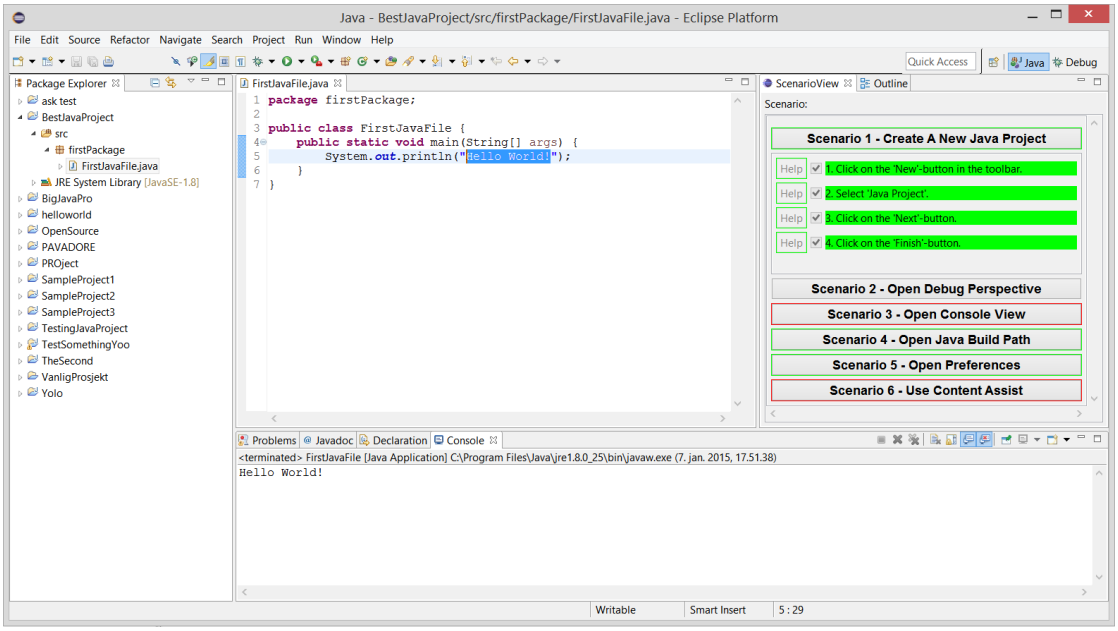
```
scenarios.add(scenario7);
```

5.3 Skjermbilder av prototype

Det blir vist skjermbilder av hvordan prototypen ser ut etterhvert som brukeren utfører ulike interaksjoner med den gjennom Eclipse.

5.3.1 Oversiktsbilde

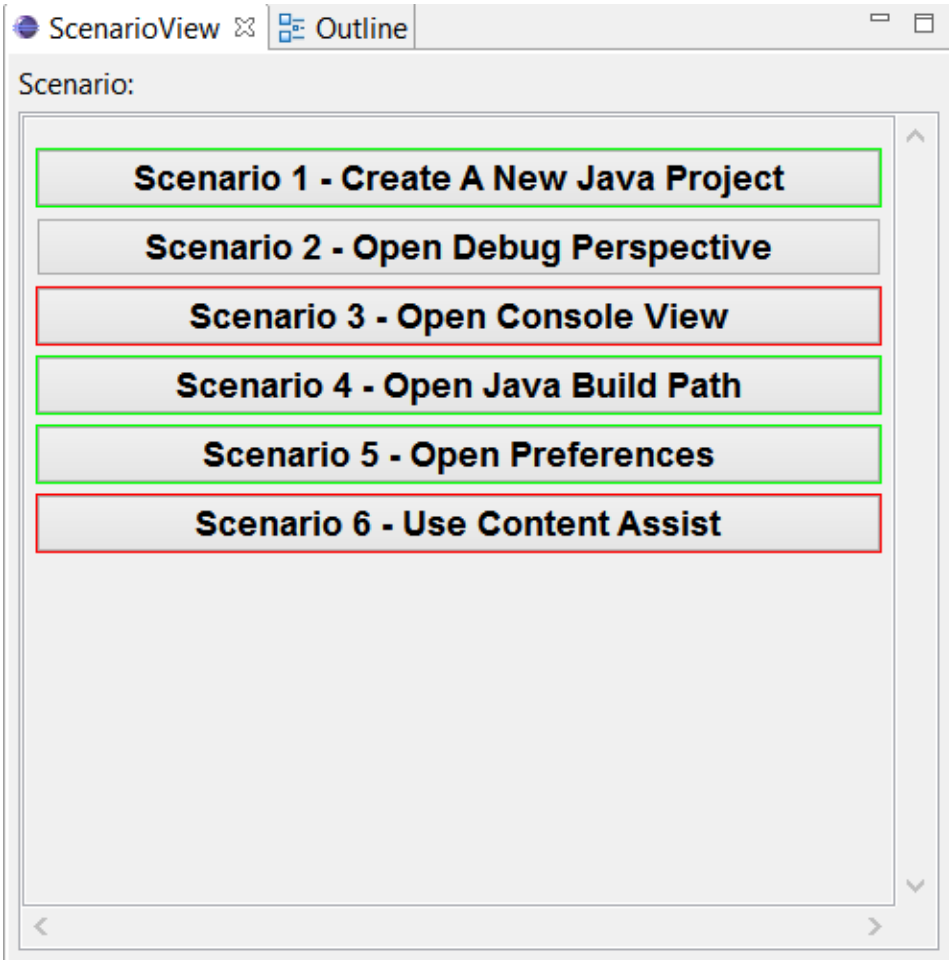
Pluginen er i form av et vindu som brukeren kan plassere etter eget ønske i brukergrensesnittet til Eclipse. Figuren 5.8 viser en skjermdump av Eclipse hvor ScenarioView er plassert som et vindu helt til høyre i brukergrensesnittet til Eclipse.



Figur 5.8: Skjermdump av Eclipse hvor ScenarioView er plassert til høyre.

5.3.2 Ikke Aktivert Scenario

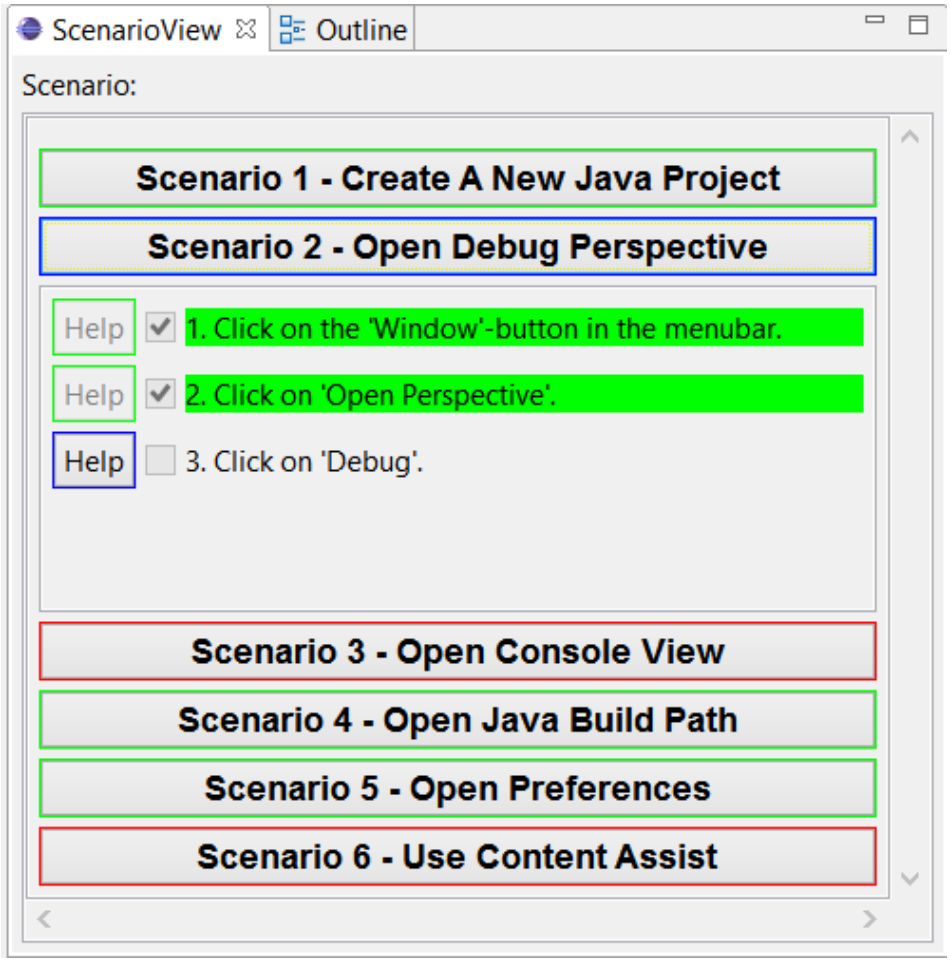
Hvis det ikke er aktivert et Scenario, vil brukergrensesnittet til ScenarioView kun vise en oversikt over de scenariene som pluginen består av. Dette vil være i form av knapper som inneholder tittel og beskrivelse til scenariet, samt en grønn eller rød ramme rundt knappen til de scenariene som henholdsvis er fullførte og avbrutte. Figur 5.9 viser en skjermdump av prototypen der ingen scenarier er aktivert, men der det blir illustrert hvilke scenarier som er fullført, avbrutt og ennå ikke påbegynt.



Figur 5.9: Skjermdump av ScenarioView når det ikke er aktivert et Scenario.

5.3.3 Aktivert Scenario

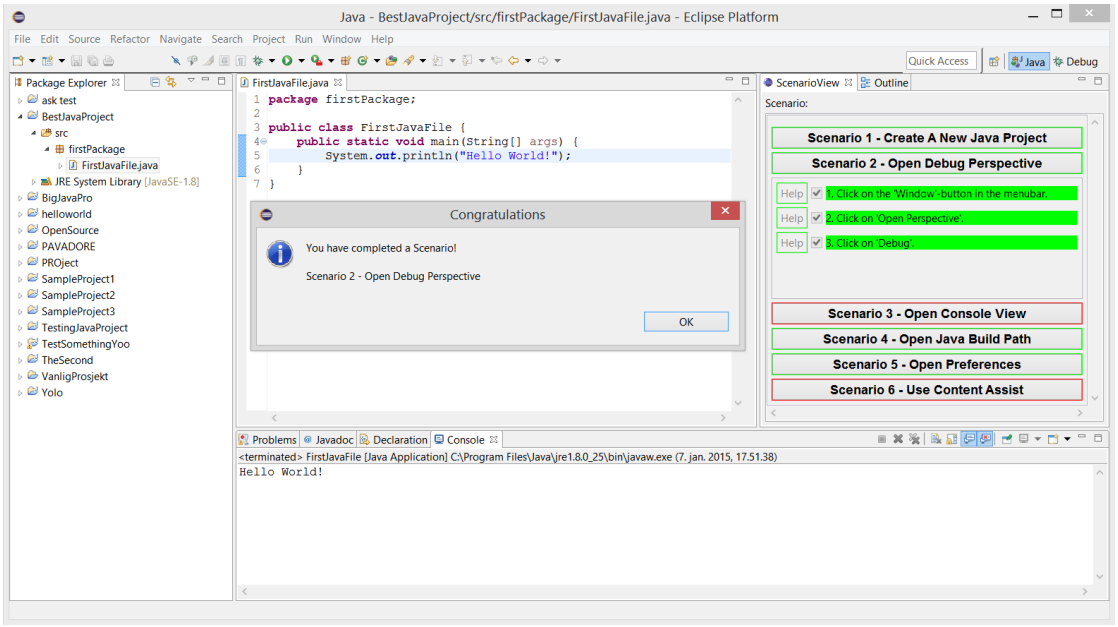
Når det er blitt aktivert et Scenario vil brukergrensesnittet til ScenarioView vise de tilhørende deloppgavene, hvor det blir krysset av i avkrysningsboksen samt markert med grønt hvilke som er fullførte. Dette blir gjort ved å fargelegge kantlinjen til help-knappen og bakgrunnen til oppgavebeskrivelsen. Dersom det finnes en Task som ikke er fullført, vil pluginen aktivere denne og brukeren kan starte å løse denne.



Figur 5.10: Skjermdump av ScenarioView når Scenario 2 er aktivert.

5.3.4 Fullført Scenario

I det et Scenario er blitt fullført av brukeren blir Scenario-knappen og den siste del-oppgaven markert grønn farge, samt et pop-up-vindu vil dukke opp hvor brukeren blir gratulert for å ha gjennomført et Scenario.



Figur 5.11: Skjermdump av Eclipse når Scenario 2 er blitt fullført.

Kapittel 6

Brukerevaluering

Brukerevalueringen ble utført i slutten av prosjektet da det var implementert en kjørende prototype som inneholdt de ønskede funksjonene. Formålet med denne typen testing er å sjekke hvor brukervennlig systemet er for brukerne og avdekke eventuelle svakheter med løsningen. Dette kapittelet presenterer personene som ble valgt ut til brukerevalueringen, forberedelsene og gjennomføringen, og til slutt resultatene av denne.

6.1 Testpersoner

Nesten alle personene som ble valgt til å teste prototypen hadde gjennomført emnet "Informasjonsteknologi grunnkurs" [ITGK, 2014], og var derfor innenfor målgruppen. For å få et bredt erfaringsmessig omfang av testsubjekter, ble det valgt ut en person med bred erfaring innen programmering og bruk av Eclipse som utviklingsverktøy. I tillegg ble det også valgt ut en testperson som ikke hadde noen tidligere erfaring med hverken programmering eller bruk av Eclipse. Dette valget ble gjort for å ha et testpanel som representerer hele skalaen av erfaringer. Antallet testpersoner ble satt til fem personer, etter Jakob Nielsens brukertesting [Nielsen, 2000], der det kommer fram at jo flere personer som blir testet jo mindre nytt vil den neste testpersonen avdekke. Tabell 6.1, viser en oversikt over testpersonene som ble utvalgt til brukertestingen og hvilke kvalifikasjoner den enkelte har.

Person	Kjønn	Alder	Yrke	Ferdigheter
P1	Mann	19	Student: Energi og Miljø	Grunnleggende ferdigheter i prosedyreorientert programmering. Aldri brukt Eclipse før.
P2	Mann	24	Mobil- og webutvikler	Bred erfaring i programmering og bruk av Eclipse.
P3	Kvinne	20	Student: Informatikk	Grunnleggende ferdigheter i programmering og bruk av Eclipse.
P4	Mann	24	Student: Fysikk og Matematikk	Grunnleggende ferdigheter i programmering. Aldri brukt Eclipse.
P5	Kvinne	19	Student: Kjemi	Ingen ferdigheter i programmering eller bruk av Eclipse.

Tabell 6.1: Informasjon om testpersonene

6.2 Forberedelser og gjennomføring

I forkant av testingen ble hver personene kontaktet og det ble avtalt tid og sted til hver av disse. En av fordelene med at prototypen kunne testes på en datamaskin, er at personene kunne velge et testlokale der de følte seg komfortabel. Da det ikke var garantert at hver person hadde det utstyret som krevdes for å teste prototypen, ble det valgt ut én bærbar datamaskin som ble brukt av alle testpersonene. Før testingen startet ble hver enkelt person forsikret om at det var prototypen som skulle testes, og ikke personen selv. Det ble ikke gitt ut noen scenarier som personene skulle gjennomføre, da prototypen i seg selv inneholder oppgaver som beskriver hva brukeren skal gjøre. Etter at hver person hadde fått prøve ut prototypen ble det gjennomført to spørreundersøkelser. Den første gikk ut på hva de mente om den generelle brukervennligheten, mens den andre gikk ut på hva de mente om utformingen og nyttigheten. Til slutt ble det gjort et intervju i form av en åpen samtale med hver person der de fikk muligheten til å komme med egne meninger.

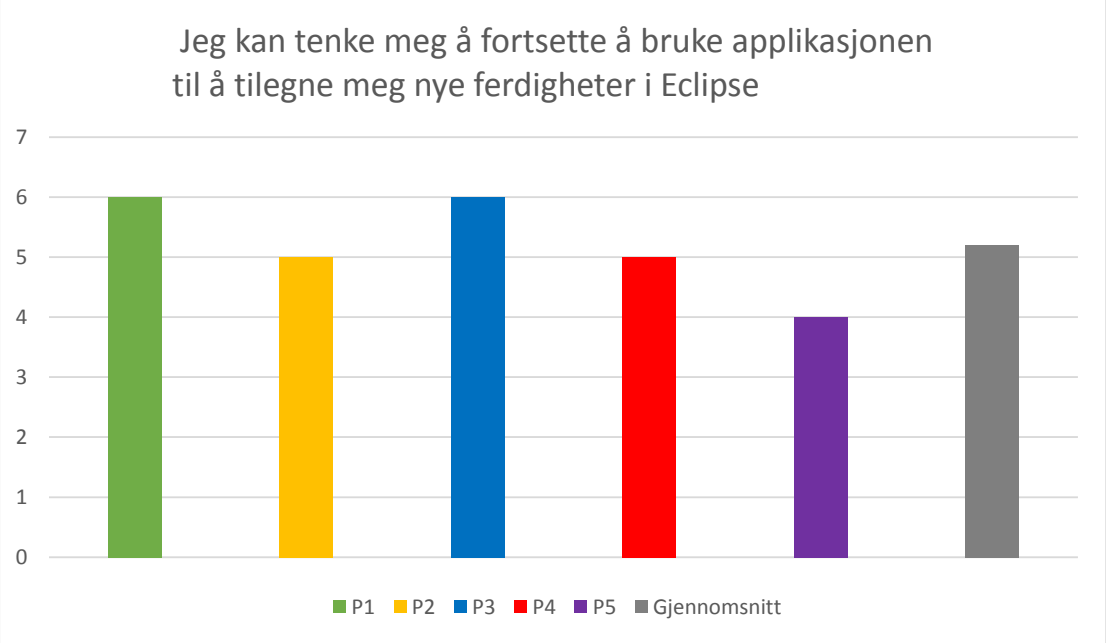
6.3 Resultater

Her blir det presentert de resultatene som ble avdekket gjennom brukerevalueringen.

6.3.1 Innvirkning på motivasjon

En av hovedhensiktene med løsningen som er implementert er at den skal motivere brukerne til å løse oppgaver. Figur 6.1 viser hva brukerne svarte på et av spørsmålene i

spørreundersøkelsen i appendiks C.1.



Figur 6.1: Innvirkning på motivasjon

Resultatene fra spørsmålet i figur 6.1 viser at brukerne i gjennomsnitt var over middels enige i at de kunne tenke seg å bruke prototypen til å tilegne seg nye ferdigheter i Eclipse. Dette var noe som ble bekreftet gjennom intervjuene. I tillegg kom det fram forslag til forbedringer:

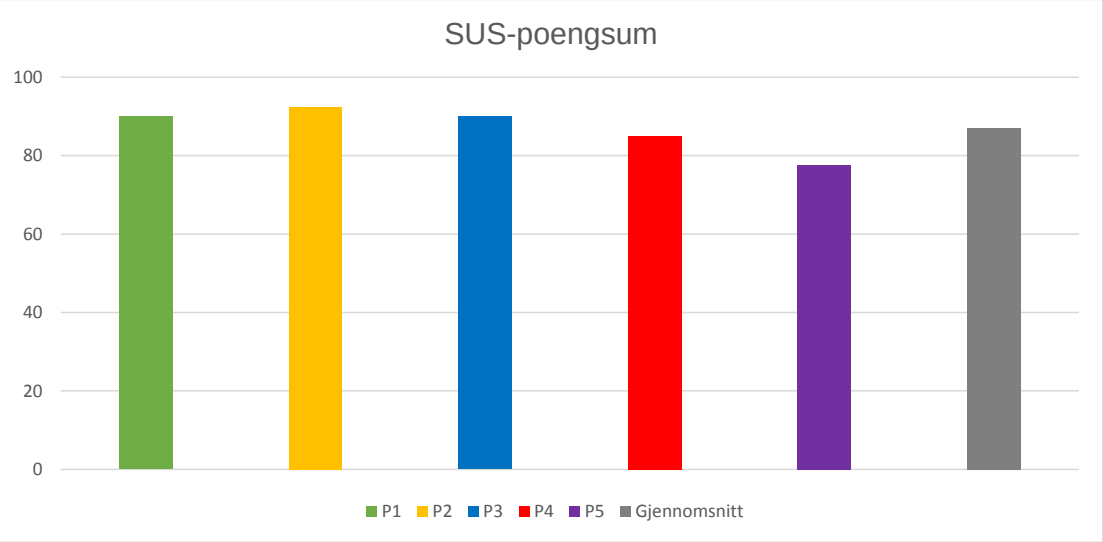
"Å løse oppgavene var i og for seg greit, men jeg kunne tenkt meg noen form for poeng etter at en oppgave ble løst. Jeg tror også at det ville vært interessant å kunne gå opp et nivå etter man har løst mange nok oppgaver".

Dette er et sitat fra P3 og ble sagt i løpet av intervjuet. Her kommer det fram at hun ville bli mer motivert hvis det var mulighet for å tjene poeng for å løse oppgaver. I tillegg mente hun at det ville være motiverende å innføre mange nivåer som brukerne kunne oppnå ved å løse mange nok oppgaver.

"Det kunne vært mer likt XBOX, der det dukker opp en liten boks og en lyd, og man får en ny achievement".
Dette ble sagt av P2 i løpet av intervjuet. Han mener at det ville være motiverende å innføre ulike merker som brukerne kan oppnå ved å løse oppgaver.

6.3.2 Generell brukervennlighet

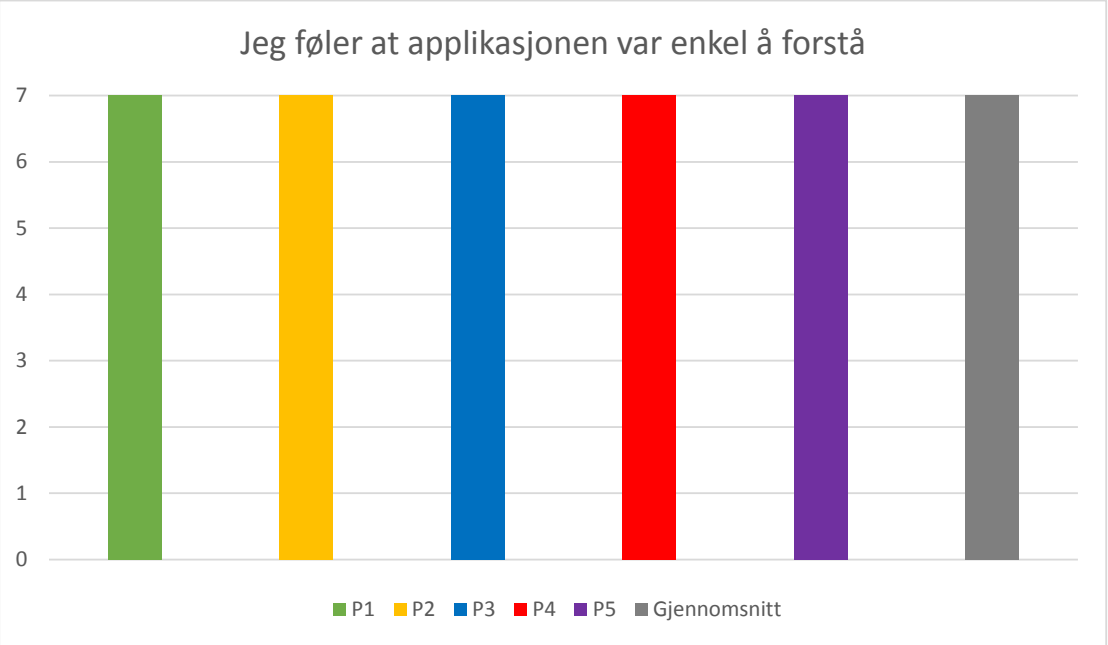
For å måle den generelle brukervennligheten til den ferdige løsningen, fylte alle testpersonene ut et eget SUS(System usability Scale)-spørreskjema (appendiks C.1) etter at de hadde fått prøve ut prototypen. Dette er en generell og rask måte for å måle den generelle brukervennligheten til et system [Brooke, 1996]. Figur 6.2, viser et diagram over SUS-poengsum til hver av testpersonene.



Figur 6.2: Oversikt over SUS-poengsum til testpersonene.

Som figur 6.2 viser, er den gjennomsnittlige SUS-poengsummen på 87 og vil si at brukerne mente at prototypen var veldig brukervennlig. Hvis denne poengsummen skulle blitt presentert i form av en bokstavkarakter ville den fått A [Sauro, 2011]. Det er verdt å nevne at resultatene fra hver person var ganske jevne, da den høyeste poengsummen var 92,5 (P2) og den laveste 77,5 (P5). Den gode brukervennligheten ble bekreftet gjennom spørsmålene til spørreundersøkelsen C.1.

Figur 6.3 viser hva hver testperson mente om applikasjonen som helhet. Her viser resultatene at alle testpersonene var sterkt eninge i at prototypen var enkel å forstå.



Figur 6.3: Hvor enkelt det var å forstå prototypen

6.3.3 Generelle tilbakemeldinger

Dette er generelle tilbakemeldinger fra brukerne:

- Enkelt og oversiktlig utseende.
- Likte at det tok kort tid å starte et Scenario.
- Det kunne vært flere scenarier og oppgaver.
- Likte at det var korte og presise oppgavetekster.
- Likte at det kom umiddelbar tilbakemelding i det en oppgave ble løst.
- Hjelpefunksjonen var interessant.
- Fargene satte preg på prototypen.
- Likte at det var mulig å løse scenarier i hvilken som helst rekkefølge.
- Ryddig å ha løsningen som et hjelpeprogram integrert i Eclipse.
- Mer interessant enn å lese en lang guide.
- Bedre enn å se gjennom utydelige hjelpevideoer på nett.

Kapittel 7

Diskusjon og konklusjon

I dette kapittelet vil forskningsspørsmålene samt resultatene fra brukerevalueringen bli diskutert.

7.1 Forskningsspørsmål

RQ1: "Hvordan kan spillifisering brukes til å stimulere ferdigheter som ikke nødvendigvis måles i dagens øvingsopplegg ved programmeringskurs?"

Dette forskningsspørsmålet er i løpet av oppgaven prøvd å bli besvart gjennom to underspørsmål:

7.1.1 RQ1.1

"Hvordan kan Eclipse utvides med hjelp av mekanikker innenfor spillifisering for å stimulere til bruk av relevante bruksoppgaver?"
Her er det blitt gjort en undersøkelse av læremålene til et programmeringsfag ved Norges Tekniske Naturvitenskapelige Universitet(NTNU), og satt sammen et sett med oppgaver bestående av stegvise deloppgaver som skal resultere i ferdighet. Oppgavene som er laget er varierte både i innhold og vanskelighetsgrad.

Når det kommer til de utvalgte spillmekanikkene i den ferdige løsningen er dette mekanikker som allerede finnes i et lignende spillifisert verktøy 3.4.2. Da det er blitt sett at det lignende spillifiserte verktøyet er noe som fungerer, er det dermed naturlig å tro at dette vil fungere for den egne løsningen. Brukerevalueringen som ble foretatt støtter opp mot denne tanken.

7.1.2 RQ1.2

"Hvordan kan utvidelsen implementeres og gjøres tilgjengelig som et rammeverk for å støtte de relevante bruksoppgaver og potensielt utvikles videre?"

Det er blitt undersøkt hvilke muligheter som finnes ved utvidelse av Eclipse. Dette har vært en meget ressurskrevende prosess hvor det har blitt satt av betraktelig med tid og energi på å lese seg opp på arkitekturen til Eclipse, samt empirisk eksperimentering for å bekrefte i praksis hva som er mulig og hvordan det kan gjøres. Forarbeidet har resultert i et rammeverk basert på objektorienterte løsninger. Arkitekturen til rammeverket er konstruert slik at det enkelt skal kunne bygges videre på.

7.2 Begrensninger

7.2.1 Metode

I løpet av prosjektet har det vært brukt mye tid og energi på å sette seg inn i hvordan Eclipse fungerer og hvordan det kan utvikles en utvidelse til Eclipse. Dermed har dette ført til at det har tatt noe lengre tid å produsere en kjørende prototype som inneholder ønskede funksjonaliteter som brukeren kan teste. Dette har igjen ført til at det ikke er blitt gjort så mange og hyppige brukerevalueringer underveis. Her kunne det i de tidligere fasene av prosjektet blitt tatt i bruk testing i form av papirprototyping, noe som ville gjort det mulig å raskt teste ut tenkt utseende og funksjonalitet på en kosteffektiv måte [Snyder, 2003].

7.2.2 Testpersonene

Personene som ble valgt ut til å teste den ferdige prototypen var alle nåværende eller tidligere studenter fra forskjellige linjer ved NTNU. Dataferdighetene og programmeringsferdighetene til testpersonene dekket hele skalaen, fra å ikke ha noen ferdigheter til å ha bred erfaring. Et problem her var at det var få testpersoner. Dersom det hadde vært flere kunne det blitt avdekket flere problemer og mulige forbedringer av prototypen.

7.2.3 Prototypen

Prototypen som ble implementert i dette prosjektet var laget for å motivere studenter til å tilegne seg praktiske ferdigheter i Eclipse ved å ta i bruk teknikker fra spillifisering. Utseende til prototypen er en svært enkel løsning uten forstyrrende elementer, men hvor funksjonaliteten er tilstrekkelig. Dette er noe som gjør at det er enkelt og intuitivt for brukeren å kunne benytte seg av systemet.

Arkitekturen til prototypen er et rammeverk som bærer preg av gode objektorienterte løsninger, noe som har gjort det enkelt å opprette Scenarioer underveis i prosjektet. Dette er også en egenskap som åpner opp muligheten for enkel utvidelse av pluginen for andre utviklere i framtiden.

Prototypen ligner en del på Eclipse Cheat Sheet, men skiller seg ut ved å kunne:

- Utføre deloppgaver programmatisk: Cheat Sheet kan kun utføre en enkelt kommando per deloppgave. Dersom en deloppgave krever flere kommandoer rett etter hverandre, vil ikke Cheat Sheet kunne utføre disse programmatisk, noe som prototypen klarer.
- Registrere at en deloppgave er fullført: Det er kun noen få deloppgaver som Cheat Sheet klarer å registrere at er blitt fullført. De som ikke blir automatisk registrert må bli registrert manuelt av brukeren. Dette stiller dermed krav til at en bruker selv må vite at oppgaven er blitt fullført, i stedet for å få en bekreftelse fra systemet om at oppgaven er fullført. Prototypen klarer automatisk å registrere alle deloppgavene som er fullført. Dersom en fullført deloppgave fører til at et Scenario er fullført, vil prototypen kunne registrere dette.
- Starte neste deloppgave automatisk: Dersom ikke Cheat Sheet automatisk klarer å registrere at en deloppgave er fullført, må brukeren klikke på en knapp som sier at deloppgaven er løst for å starte neste deloppgave. Når prototypen registrerer at en deloppgave er fullført, vil den automatisk starte neste deloppgave.

7.3 Videre forskning

Resultatene fra brukerevalueringen viser at brukerne kunne ønske seg noen form for belønning etter at en oppgave er blitt fullført, enten i form av poeng, merker, lyder eller en progresjonsbar. I framtidige versjoner av prototypen er dette noe som absolutt lar seg integrere, da den ferdige løsningen er et rammeverk som enkelt kan bygges videre på samt inneholder arkitekturen som muliggjør dette. Etter at den ferdige løsningen var implementert, ble det vurdert om det var hensiktsmessig med tanke på gjenstående tid å sette i gang en ny iterasjon der det skulle integreres spillmekanikker som brukerevalueringen avslørte. Basert på tidsaspektet samt muligheten til å støte på uventede problemer, ble det gjort et valg om å sette en strek over videre implementering og heller sette fullt fokus på å gjennomføre empirisk testing samt rapportskriving.

I brukerevalueringen er det blitt brukt få personer til å teste prototypen. Derfor burde det i videre forskning bli sett på muligheten til å prøve ut prototypen i større skala med

relevante testpersoner og vurdere dette opp i mot læringsmålene til programmeringskurset.

I framtidige versjoner vil det være mulig å bygge videre på pluginen ved å opprette mer varierte oppgaver bestående av mer komplekse deloppgaver. En stor utfordring med å lage komplekse deloppgaver, vil være å implementere kode som gjør at datamaskinen automatisk utfører handlinger som kreves for å fullføre en deloppgave. En slik automatisert testing av brukergrensesnittet er noe av det mest utfordrende å gjennomføre i en applikasjon, da det må bli tatt høyde for utallige hendelser og tilstander som kan inntreffe i tillegg til mulige avhengigheter mellom disse [Belli, 2001]. Gjennom arbeidet med prototypen har nettopp dette slukt mest tid og krefter.

Å kunne dele sine prestasjoner med andre ville vært en måte å innføre et konkurranseaspekt til bruken av systemet. Disse kunne blitt delt via sosiale medier, hvor prototypen hadde konstruert et praktisk Eclipse-sertifikat som viser hvilke oppgaver brukeren har gjennomført, hvilket nivå brukeren har oppnådd, og hvor det også kommer fram opptjente poengsummer og merker.

Når det dukker opp dialogvinduer i Eclipse er det ikke mulig å klikke på elementer som befinner seg utenfor eller bak dette vinduet. Derfor burde det bli sett på muligheten til å implementere funksjonalitet som gjør at "ScenarioView" kan koble seg til slike dialogvinduer.

7.4 Konklusjon

I løpet av dette forskningsarbeidet er det blitt opprettet et sett med bruksoppgaver som baserer seg på læringsmålene til programmeringskurset TDT4100 ved NTNU. Deretter har det vært gjennomført et litteraturstudie om spillifisering hvor hensikten var å velge spillmekanikker som kunne passe til et spillifisert læringsverktøy. Det ble valgt ut to arbeider som inneholdt elementer som ble sett på som relevante i forhold til utforming av en framtidig løsning. Videre ble det gjort en grundig undersøkelse av mulighetene til å utvide Eclipse. Her har det vært implementert et vindu som skulle hente ut informasjon etter hvert som det ble foretatt handlinger i Eclipse. Gjennom å kombinere alle disse stegene har det blitt laget en spillifisert utvidelse til Eclipse som skal stimulere til utvikling av praktiske ferdigheter. Etter brukerevalueringen er det grunn til å tro at denne prototypen vil fungere for brukerne da resultatene viser at:

- den gjennomsnittlige poengsummen til brukervennligheten var på 87.
- samtlige testpersoner mente at applikasjonen var enkel å forstå.

- de fleste var over middels enige i at de kunne tenke seg å fortsette å bruke applikasjonen til å tilegne seg nye ferdigheter i Eclipse.

Bibliografi

[ACM,] ACM. Acm digital library. http://dl.acm.org/. Lest: 12.01.2015.

[Belli, 2001] Belli, F. (2001). Finite state testing and analysis of graphical user interfaces. In *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*, pages 34–43.

[Bibsys,] Bibsys. Bibsys - vi gjør kunnskap tilgjengelig. http://ask.bibsys.no/. Lest: 12.01.2015.

[Brooke, 1996] Brooke, J. (1996). Sus: A quick and dirty usability scale. http://www.itu.dk/courses/U/E2005/litteratur/sus.pdf. Lest: 12.01.2015.

[Corporation, 2011] Corporation, M. (2011). Ribbon hero 2 - clippy's second chance. http://www.ribbonhero.com/. Lest: 12.01.2015.

[eBizMBA Inc.,] eBizMBA Inc. Top 15 most popular search engines | january 2015. http://www.ebizmba.com/articles/search-engines. Lest: 12.01.2015.

[Foundation,] Foundation, T. E. Pde incubator spy. http://www.eclipse.org/pde/incubator/spy/. Lest: 12.01.2015.

[Foundation, 2014] Foundation, T. E. (2014). About the eclipse foundation. https://eclipse.org/org/. Lest: 12.01.2015.

[Google, a] Google. Google scholar. https://scholar.google.com/. Lest: 12.01.2015.

[Google, b] Google. Google web search. https://www.google.com/. Lest: 12.01.2015.

[Google, c] Google, T. Google trends - gamification. http://www.google.com/trends/explore?hl=en#q=gamification. Lest: 12.01.2015.

[IEEE,] IEEE. Ieee xplore digital library. http://ieeexplore.ieee.org/Xplore/home.jsp. Lest: 12.01.2015.

[ITGK, 2014] ITGK (2014). Undervisningsinformasjon om it grunnkurs. https://itgk.idi.ntnu.no/faginfo/undervisning.php. Lest: 12.01.2015.

[J., 2006] J., O. (2006). *Researching Information Systems and Computing*. SAGE Publications Ltd.

[Kapp, 2012] Kapp, K. (2012). *The gamification of learning and instruction : game-based methods and strategies for training and education*. Pfeiffer, San Francisco, CA.

[Nielsen, 2000] Nielsen, J. (2000). Why you only need to test with 5 users. http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/. Lest: 12.01.2015.

[Nielsen, 2012] Nielsen, J. (2012). Usability 101: Introduction to usability. http://www.nngroup.com/articles/usability-101-introduction-to-usability/. Lest: 12.01.2015.

[NTNU,] NTNU, T. Tdt4100 - objektorientert programmering. http://www.ntnu.no/studier/emner/TDT4100#tab=omEmnet. Lest: 12.01.2015.

[O'Donovan et al., 2013] O'Donovan, S., Gain, J., and Marais, P. (2013). A case study in the gamification of a university-level games development course. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, SAICSIT '13, pages 242–251, New York, NY, USA. ACM.

[Pressman, 2010] Pressman, R. (2010). *Software engineering : a practitioner's approach*. McGraw-Hill Higher Education, New York.

[Rolighetsteorin, 2010] Rolighetsteorin (2010). Tomglasspelet. http://www.rolighetsteorin.se/tomglasspelet-0. Lest: 12.01.2015.

[Sauro, 2011] Sauro, J. (2011). Measuring usability with the system usability scale (sus). http://www.measuringu.com/sus.php. Lest: 12.01.2015.

[Snyder, 2003] Snyder, C. (2003). *Paper prototyping the fast and easy way to design and refine user interfaces*. Morgan Kaufmann, Elsevier Science, San Francisco, Calif.

[Svanæs, 2006] Svanæs, D. (2006). Sus norsk versjon. http://www.brukskvalitet.no/wp-content/uploads/2010/01/SUS-norsk.pdf. Lest: 12.01.2015.

[Trætteberg, 2014] Trætteberg, H. (2014). Observatør-observert-teknikken. https://www.ntnu.no/wiki/pages/viewpage.action?pageId=66879660. Lest: 12.01.2015.

[Wiki,] Wiki, G. Game mechanics. http://gamification.org/wiki/Game_Mechanics. Lest: 12.01.2015.

[Wikipedia, a] Wikipedia. Arkadespill. http://en.wikipedia.org/wiki/Arcade_game. Lest: 12.01.2015.

[Wikipedia, b] Wikipedia. Likert scale. http://en.wikipedia.org/wiki/Likert_scale. Lest: 12.01.2015.

Tillegg A

Installasjon og oppsett

A.1 Installasjonsguide

For å installere pluginen kreves det en datamaskin med Eclipse installert, helst med versjonen Luna 4.4 eller nyere.

1. Kopier plugin-fila.
2. Naviger til installasjonsmappen der “Eclipse.exe” ligger.
3. Dobbeltklikk på mappen “dropins” eller “plugins“.
4. Lim inn plugin-fila.
5. Åpne Eclipse.
6. Window -> Show View -> Other -> Eclipse Listeners.
7. Velg “ScenarioView”.
8. Klikk “OK”.

A.2 Importere Prosjekt

1. Åpne Eclipse.
2. File -> Import -> Plug-ins and Fragments.
3. Import from directory - velg lokasjonen til plugin-fila.
4. Klikk “Next“.

5. Marker pluginen og klikk “Add“.
6. Klikk “Finish“.

Tillegg B

Bruksoppgaver i Eclipse

B.1 Case Oppgaver

På grunnlag av hva læringsmålene i et programmeringsfag ved NTNU tilsier at studenter skal ha ferdigheter i, er det blitt satt sammen ett sett med oppgaver bestående stegvise deloppgaver som skal resultere i ferdighet.

B.1.1 Debugging

1. Åpne et Debug-perspektiv i Eclipse.
2. Debugging av en for-løkke som summerer tallene fra 0 til og med 10.
3. Gå fra Debug-perspektiv og tilbake til Java-perspektiv i Eclipse.

B.1.2 Bruk av java-editor for kode-templates

1. Bruke Eclipse sin innebygde Content Assist til å fullføre en kommando:
   (a) Lage en main-metode
   (b) Lage en if-setning
   (c) Lage en for-løkke
2. Bruke Eclipse sin innebygde Content Assist til å vise forslag til metoder og variabler som er tilgjengelige
   (a) Lage en default konstruktør
   (b) Vise variabler i en main-metode
3. Generere Javadoc til en vilkårlig metode

B.1.3 Oppsett av et Java-prosjekt

1. Opprette et java-prosjekt i Eclipse
2. Opprette en klasse i et java-prosjekt i Eclipse
3. Generere Javadoc til en vilkårlig metode
4. Legge til en JAR-fil i Build Path til et Java-prosjekt i Eclipse
5. Opprette et java-prosjekt og samtidig legge inn ønsket JAR-fil i Build Path
6. Åpne Java Build Path

B.1.4 Hurtigtaster

1. Kommentere eller fjerne kommentarlinjer kildekode
2. Skrive “System.out.println()”
3. Slette en rad i kildekoden
4. Kjøre en applikasjon

B.1.5 Diverse

1. Automatisk opprette Getters og Setters til instansvariabler
2. Legge til ulike Show View i Eclipse
3. Kommentere eller fjerne kommentarlinjer til markert kode

Tillegg C

Spørreskjemaer og intervjumal

C.1 Spørreskjema fra brukerevaluering

Spørsmål som ble stilt under spørreundersøkelse i sammenheng med brukertestingen anående hvor tilfreds de var med applikasjonen, der 1 er sterkt uenig og 7 er sterkt enig:

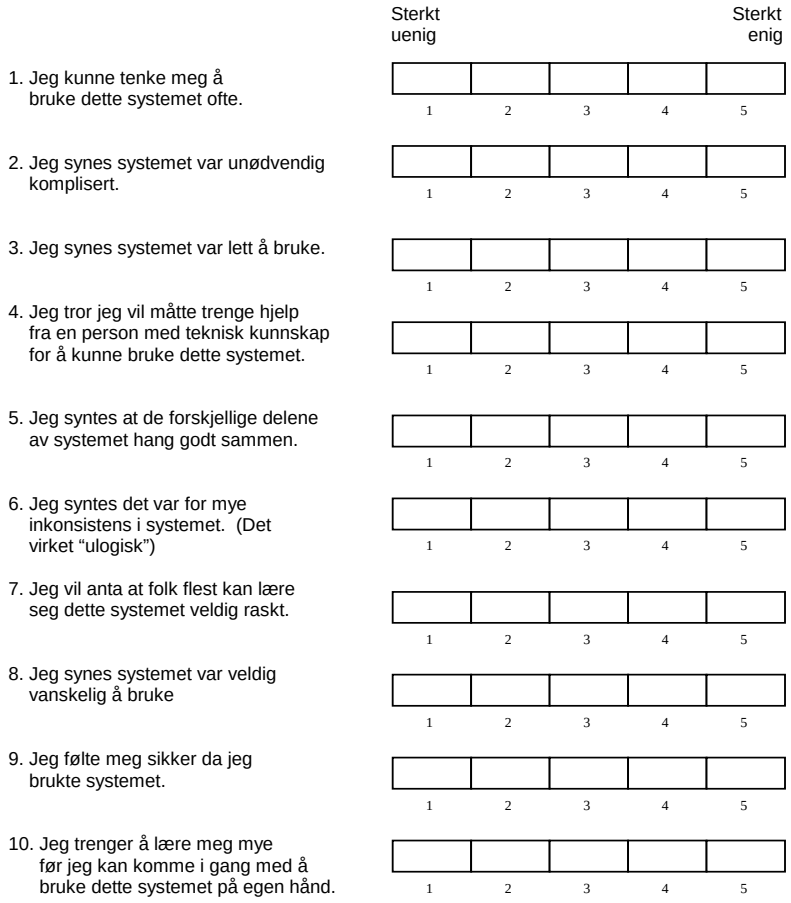
1. Jeg kan tenke meg å fortsette å bruke applikasjonen til å tilegne meg nye ferdigheter i Eclipse.
2. Jeg lærte noe nytt ved å bruke applikasjonen.
3. Jeg føler at oppgavene var interessante.
4. Jeg føler at oppgavene ikke var lærerike.
5. Jeg føler at det var for mange deloppgaver i hvert Scenario.
6. Jeg føler at oppgavene var for enkle.
7. Jeg føler at hjelp-funksjonen var nyttig.
8. Jeg føler at jeg dro nytte av applikasjonen.
9. Jeg føler at applikasjonen inneholdt forstyrrende elementer.
10. Jeg føler at applikasjonen var enkel å forstå.

C.2 SUS - spørreskjema

Den norske oversatte versjonen av SUS-spørreskjemaet som ble brukt i prosjektet [Svanæs, 2006].

Noen spørsmål om systemet du har brukt.

Vennligst sett kryss i kun en rute pr. spørsmål.



|

SUS
Norsk versjon ved Dag Svanæs
NTNU 2006

Figur C.1: Norsk oversatt versjon av SUS-spørreskjema.

C.3 Intervju fra brukerevaluering

Spørsmål som ble stilt under intervjuene under brukerevalueringen:

1. Hva er ditt førsteinntrykk av applikasjonen?
2. Hva tenker du om utseende og utformingen av applikasjonen?
3. Hva tenker du om hjelp-funksjonen?
4. I hvilken grad synes du at dette var en effektiv måte å tilegne seg nye ferdigheter i Eclipse?
5. I hvilken grad føler du at applikasjonen motiverte deg til å gjøre flere oppgaver?
6. I hvilken grad førte applikasjonen til at du lærte noe nytt?
7. Hva føler du at applikasjonen manglet?
8. Hva kunne blitt gjort anderledes?