Refleksjonsrapport

IT1901 Informatikk Prosjektarbeid 1

Studentnummer: XXXXXX

Gruppenummer: 07

Temanummer: 02

Tema: Kodekvalitet

Ordtelling: 1499

Innledende betraktninger

Kodekvalitet er en essensiell prosess for å effektivisere og gjøre kildekoden mer forståelig. Martin Fowler sier det enkelt: "Any fool can write code that a computer understand. Good programmers write code that humans understand." (Fowler, 1999, s. 15). Dette er spesielt viktig i gruppearbeid, hvor flere utviklere må forstå og samarbeide om den samme kildekoden. Kodekvalitet er imidlertid ikke kun viktig for utviklere. I boken "Clean Code" deler R. C. Martin et eksempel på en startup som til tross for umiddelbar suksess gikk konkurs grunnet lav kodekvalitet (Martin, 2007, s. 3). Ineffektiv kode førte til lav ytelse og en dårlig brukeropplevelse. Til slutt ble utviklingsprosessen så krevende at vedlikehold av kildekoden ble umulig.

I denne rapporten skal jeg se på hvordan gruppen min jobbet med kodekvalitet, deriblant hvilke faktorer som bidro til høyere kodekvalitet, og faktorer som reduserte kodekvaliteten. Til slutt skal jeg se på hvilke fremtidige tiltak som kan tas for å sikre høy kodekvalitet.

Forståelse av kodekvalitet

Basert på tilnærmingen tatt i innledningen og for å avgrense oppgaven vil jeg primært se på kodekvalitet knyttet til omstrukturering, effektivitet, vedlikeholdbarhet, konsistens og lesbarhet som er faktorer diskutert i pensum-stoffet og på forelesing. Jeg vil basere rapporten på Michael Feathers sin definisjon av høy kodekvalitet “Clean code always looks like it was written by someone who cares. There is nothing obvious that you can do to make it better.” (Martin, 2007, s. 10), og ta i bruk resonnementene til R. C. Martin og Martin Fowler gjennom rapporten.

Faktorer som bidro til høyere kodekvalitet

Omstrukturering til SSOT

I etterkant av den andre gruppeinnleveringen ble det tydelig at vi måtte rette fokuset mot kodekvaliteten i prosjektet. Tidligere hadde vi i stor grad jobbet individuelt på våre egne oppgaver, noe som resulterte i løsninger preget av "få det til å fungere" mentalitet – en tilnærming som står i kontrast til prinsippet for høy kodekvalitet av Michael Feathers.

På dette tidspunktet fungerte JavaFX-applikasjonen ved å sende objekter mellom kontrollerne til de ulike visningene. Dette oppsettet viste seg imidlertid å skape utfordringer når det gjaldt implementering av ny funksjonalitet, og det ble tydelig at en omstrukturering var nødvendig. Som Martin Fowler uttrykker det: "When you find you have to add a feature to a program, and the program's code is not structured in a convenient way to add the feature, first refactor the program to make it easy to add the feature, then add the feature." (Fowler, 1999, s. 7).

På bakgrunn av dette foreslo jeg for gruppen at vi måtte omstrukturere applikasjonen for å kunne hente ut data fra en"single source of truth" (SSOT). Dette ble realisert gjennom introduksjonen av et nytt MainController-objekt som ga alle JavaFX-kontrollerne tilgang til det samme User-objektet. Denne endringen forenklet implementeringen av ny funksjonalitet, da vi lettere fikk tilgang til den nødvendige dataen. Senere, da vi introduserte et Rest-API, krevde det kun å bytte ut User-objektet til et RemoteUserAccess-objekt og implementering av relevante metoder for API-forespørsler. Dette strategiske grepet legemliggjorde ikke bare prinsippene for omstrukturering, men bidro også til økt fleksibilitet og vedlikeholdbarhet i kildekoden.

Fremme konsistens og lesbarhet med DRY-prinsippet

I utviklingen av de ulike JavaFX-kontrollerne, opplevde vi utfordringer knyttet til lesbarhet, spesielt på grunn av omfattende og komplekse if-setninger som ble brukt for inputvalidering og visning av informasjonsmeldinger. Denne praksisen resulterte i uoversiktlige og langstrakte kodelinjer, som gjorde det vanskelig å lese og vedlikeholde koden. For å forbedre kodekvaliteten tok jeg initiativ til å flytte denne logikken til egne hjelpeobjekter. Dette strategiske grepet bidro til å organisere koden bedre, reduserte kompleksiteten i kontrollerne og førte til en generell økning i kodekvaliteten.

Videre observerte jeg gjentakelse av metoder for generering av rutenett i både Workout- og Overview-kontrolleren, samt gjentatt bruk av varslingsbokser og bekreftelsesbokser i flere av kontrollerne. Inspirert av Fowlers ord: "I think one of the most valuable rules is to avoid duplication. 'Once and only once' is the Extreme Programming phrase," (Shore, 2007, s. 319) så foreslo jeg for gruppen å lage dedikerte objekter for disse felles kodedelene. Dette initiativet resulterte i implementeringen av GridBuilder og UiUtils. Disse endringene ikke bare reduserte kodegjentakelsen, men fremmet også konsistens og vedlikeholdbarhet i kildekoden med hensyn til DRY-prinsippet (Don't Repeat Yourself).

Faktorer som reduserte kodekvaliteten

Ineffektivt Rest-API uten bruk av DTOs

Et område som reduserte kodekvaliteten var en ineffektivt implementasjon av Rest-API. For å unngå direkte tilgang til Core-objekter i JavaFX-UIet brukte vi en betydelig mengde kall til Rest-APIet. Selvom dette fungerer fint når man kjører serveren lokalt, så vil det føre til unødvendig ventetid og en dårlig brukeropplevelse når serveren er ekstern. Dette resulterte i lite effektiv kode og lav kodekvalitet.

For å løse dette problemet, kunne vi brukt en strategi Fowler introduserte: "When you're working with a remote interface (...) each call to it is expensive (...) and that means that you need to transfer more data with each call. The solution is to create a Data Transfer Object that can hold all the data for the call." (Fowler, 1999, s. 306). Ved å introdusere et WorkoutDTO-objekt, kunne vi redusert antall kall i Overview-kontrolleren fra åtte ganger exerciseCount til kun én.

Utfordrende implementasjon av Checkstyles og JavaDoc

For å følge de samme kode konvensjonene implementerte vi Checkstyle som et linteverktøy. Dessverre opplevde flere i gruppen utfordringer med å integrere det i sine IDE-er, noe som førte til inkonsistens i kildekoden. Det var også flere konvensjoner vi var uenige i, men vi fikk aldri tid til å endre konfigurasjonen. For eksempel i MainController-objektet har vi flere VBoxer for hver av visningene. Checkstyle tvang oss til å definere dem på separate linjer med flerlinjede kommentarer, selv om vi foretrakk å definere dem mer kompakt på én linje, noe som reduserte lesbarheten.

I tillegg påla checkstyle at vi skulle legge til JavaDoc på alle metoder, inkludert selvforklarende metoder som getters og setters. Dette hadde en negativ innvirkning på vedlikeholdbarheten og lesbarheten til koden. Selvom Martin skriver at "It would be difficult (...) to write Java programs without them." (Martin, 2007, s. 59) så er jeg mer enig med Fowler, "When you feel the need to write a comment, first try to refactor the code so that any comment becomes superfluous." (Fowler, 1999, s. 88). Jeg mener det bør være en balanse når det gjelder bruk av JavaDoc, og en overdreven bruk, som det vi implementerte, kan faktisk ha en negativ innvirkning på kodekvaliteten.

Fremtidig tiltak for å sikre høy kodekvalitet

Klar designstruktur og konvensjoner fra starten

Den viktigste strategien for å sikre høy kodekvalitet er å bli enige om designstrukturen og implementere alle nødvendige verktøy tidlig i prosjektet, og det er avgjørende at disse elementene fungerer for alle medlemmene i gruppen. Det er også viktig at alle i gruppen er enige og har forstått de vedtatte konvensjonene og designstrukturen for kildekoden. I vår gruppe merket vi at vi investerte like mye tid i å modularisere applikasjonen og rette opp checkstyle-varsler som vi brukte på selve programmeringsarbeidet. En tidlig enighet om å strukturere koden rundt en Single Source of Truth (SSOT) ville spart oss tid ved implementeringen av MainController. Det ville også gjort implementasjonen av Rest-APIet lettere hvis appen var strukturert for det fra starten. Hele poenget med å sikre høy kodekvalitet er å spare tid og sikre en effektiv kodebase. Derfor er det avgjørende å etablere dette fundamentet fra starten av prosjektet.

Regelmessige retrospektiver

Selv med et klart rammeverk for design og konvensjoner i gruppen, er det mulig å miste fokus over tid. Derfor kan implementering av regelmessige retrospektiver være et effektivt tiltak for kontinuerlig forbedring og opprettholdelse av kodekvaliteten. Disse refleksjonsøktene kan gjennomføres som en del av gruppemøter eller ved bruk av parprogrammering ofte som en konsekvens av "code-reviews". Dette gir en plattform der koden kan gjennomgås og omstruktureres på en måte som sikrer at alle gruppemedlemmer har en felles forståelse.

Resyme

For å konkludere, denne rapporten har undersøkt flere faktorer som påvirket kodekvaliteten i utviklingen av en JavaFX-applikasjon med Rest-API i

kurset Informatikk Prosjektarbeid 1. Tiltak som bedret kodekvaliteten inkluderte omstrukturering til en "single source of truth" for å forbedre strukturen og implementering av DRY-prinsippet for å forbedre konsistens og lesbarhet. Mens tiltak som førte til lavere kodekvalitet inkluderer en ineffektivt implementasjon av Rest-APIet som gjorde koden mindre effektiv og utfordrende bruk av Checkstyles og JavaDoc som gjorde lesbarheten og velikeholdbarheten til koden dårligere. For å sikre høy kodekvalitet i fremtiden, er det avgjørende å bli tidlig enig om designstruktur og konvensjoner i starten av prosjektet og vedlikeholde det ved hjelp av regelmessige retrospektiver. Ved å ha gjort det kunne vi unngått de utfordringene vi hadde knyttet til kodekvalitet, og dermed etablert et solid fundament for en mer effektiv og vedlikeholdbar applikasjon som oppfyller høye standarder for kodekvalitet.

Bibliografi

Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.

Fowler, M. (1999). Refactoring: Improving the Design of Existing Code. Addison-Wesley.

Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

Shore, J., & Warden, S. (2007). The Art of Agile Development. O'Reilly Media.