CHALMERS
UNIVERSITY OF TECHNOLOGY

Software Engineering Project

Prototyp

Lag Filip

Nicole Ascard
Filip Lindahl
Felix Lissåker
Peter Möller
Filip Nilsson
Henrik Tran

DAT255 Software Engineering Project
Chalmers Tekniska Högskola
2017-06-02

Innehållsförteckning

Code quality

Findbugs har använts för att detektera eventuella buggar i programmet. Resultatet ligger i sin helhet på GitHub för den intresserade. Ett initialt test resulterade i ett 20-tal buggar. Applikationen har uppdaterats och nu finner inte FindBugs längre buggar i programkoden.

Software testing

Unit Testing

Anledningen till att gruppen genomfört *Unit Testing* genom projektets utveckling är för att bedöma om mjukvaran fungerar enligt fördefinierade förväntningar. Vanligtvis genomförs testet på olika nivåer i kodningen för att isolera vilka funktionaliteter som funkar och vilka som inte gör det. Detta innebär att utvecklaren med valfri metod ska kunna mata in ett randomiserat värde och få ut ett väntat resultat. Alternativt att metoden konstaterar att indatan är felaktig och att användaren meddelas detta. Utvecklarna har kontinuerligt genomfört en stor andel manuella och semi-automatiska tester, dock inga automatiska.

Manuellt Unit Test

Denna typ av tester har genomförts för att testa funktionaliteten i en viss aspekt i programmet. Det som särskiljer de manuella testerna från resterande är att användaren interagerar med programmet och direkt ser att interaktionen ger resultat i programvaran. T.ex. kan användaren trycka på knappen “Updatera” i programvaran och direkt se att en lista med lotsoperationer lägg till i en lista. Detta visar att den underliggande metoden och dess tillhörande funktionalitet opererar på avsett vis.

Semi-Automatisk Unit Test

En del enhetstester sker semi-automatiskt på liknande sätt som de manuella testerna. Dessa urskiljer sig genom att användaren inte direkt behöver konfirmera att datasettet som erhålls av en interaktion är korrekt. Istället ges en konfirmation i konsollen som medger om metoden och dess funktioner genomförts på rätt sätt. Alltså finns det knappar både för att ta emot och skicka meddelanden som dyker upp i konsolen ifall något av anropen lyckats.

Automatiskt Unit Test

Det har inte gjorts några automatiserade Unit tests under detta projekt då fokus lagts på att få funktionaliteten att fungera med hjälp av manuella och semi-automatiska tester. Att implementera några automatiska tester hade dock underlättat utvecklingen eftersom att manuella och semi-automatiska tester kräver att applikationen körs och att utvecklaren kommer

ihåg vilka funktionaliteter som ska testas. Genom att implementera automatiska tester hade alltså projektets utveckling simplifieras.

Integration Testing

Denna del av testningen är mer kritisk för projektet än *Unit Testing. Integration Testing* undersöker nämligen kontakten med externa instanser, PortCDM. Eftersom att prototypen under hela projektet haft svårigheter att få kontakt med PortCDM, som följd av diverse serverbyten och problem p.g.a. olika versioner på servrar och APIer, har kontakten med de externa instanserna varit allt annat än stabil. Som följd har även *Integration Testningen* varit av bristande karaktär.

Den typen av integartionstesting som bedrivs har mestadels varit i kombination med semi-automatiska tester. Ett knapptryck i applikation konfirmerar att data skickats, semi-automatiskt. Sedan undersöks integrationen genom att se om datan som skickats in kan hämtas ut från applikationen igen. Trots att erfarna programmerare hos de externa servrarna har tittat på programvaran har dessa tester har i stor utsträckning misslyckats tills i slutet av projektets tidsfrist där skeppens ID hårdkodats in i applikationen.

System Testing

Slutligen genomfört testning av systemet som färdig produkt. Eftersom att produkten vid demot endast fungerade lokalt förväntades gruppen att inte kunna delta med de andra aktörerna. Genom att hårdkoda skeppets ID lyckades applikationen erhålla kontakt med sandbox-server och kunde därmed genomföra det slutliga systemtestet.

Detta test inkluderade möjligheten att hämta in skepp som efterfrågat lots och presentera dessa i applikationens lista. Utöver det kunde information om dessa skepp visualiseras i applikationen för att lots organisationen enkelt skulle kunna bedöma om en lots fanns tillgänglig för efterfrågan. Utifrån denna frågeställning kunde användaren acceptera eller avböja förfrågan och därmed eventuellt boka lotsen för uppdraget. Utöver det kunde applikationen skicka in en rad “*actual-tider”* och estimat baserat på lotsens skede i operationen samt boka förtöjningspersonal för olika anlöp.

Design rationale

API-levels

Valet av API-level var baserat på det som PortCDM själva har utvecklat. Detta gjordes för att det ansågs mest troligt att denna skulle vara mest kompatibel med PortCDM. Då PortCDM

fortfarande är under utveckling visade detta sig inte vara fallet. Bristande dokumentation och buggar i API försvårade arbetet med att jobba mot PortCDM.

De meddelanden som applikationen hämtar från PortCDM och sedan visar har filtrerats så att de alla är relaterade till lotsen. Denna filtrering gjordes för att lotsoperatören enbart är intresserad av dessa PortCallMessages och att visa fler meddelanden skulle kunna innebära att det blir svårare att hitta den information som är relevant för lotsoperatören.

External dependencies

Projektet har till stor del varit externt beroende. Visserligen har en produkt utvecklats som fungerar med en intern server men det har varit problematiskt att få den att fungera externt med PortCDM. Efter många timmars felsökande och även hjälp från utvecklarna på PortCDM lyckades applikationen tillslut få kontakt med servrar lagom till demot.

Database structure

Applikationen har inte haft någon egen databas, istället har den lokala databasen använts för att lagra information. Däremot har PortCDMs databas använts externt för att ladda upp information av olika slag såsom ETAer, olika requests, olika confirms och så vidare med hjälp av PortCallMessages. Detta möjliggjorde att gruppernas applikationer kunde kommunicera med varandra genom att en gemensam testserver sattes upp (Sandbox) så att ett scenario under slutredovisningen kunde spelas upp.

Overview

Behavioural

När applikationen startas möts man av en lista på skepp. Genom att trycka på ett av skeppsnamnen så får man fram information om just det skeppet och när det förväntar sig lotsning. Under informationen finns knappar man kan trycka på i olika syften.

När någon av knapparna i skeppsvyn trycks på i applikationen skapas ett *PortCallMessage.* Detta skickar då ut olika meddelanden beroende på knappens syfte. Skillnaden i dessa meddelanden är olika platser och tider vilka framgår genom vad som står på knappen. Applikationen har även ett inputfält som tar emot ett datum och tid som skickar iväg ett meddelande om när lotsoperationen beräknas starta.

Utöver skeppsvyn så får man även fram en “mobil enhet” som ska representera lotsen ombord fartyget. Den här “mobila enheten” har knappar som kan skicka iväg meddelanden om när lotsningen faktiskt sker.

Structural

Applikationen är en desktop applikation (innehållande en desktop applikation och en “teoretisk” handhållen applikation) som är utvecklad i Eclipse, i programspråket Java. För att ge en överskådlig förklaring av hur strukturen ser ut programmatiskt i applikationen hittas en beskrivning alla klasser och deras funktion nedan:

- `Main.java - Startar JavaFX applikationen(Genererad av Eclipse).`
- `Controller.java - Sätter ihop och skickar olika PortCallMessages. Tar emot och bearbetar viss data så att den kan presenteras i det grafiska gränssnittet.`
- `PortCallInfo.java - En hjälpklass för att underlätta sparandet och sedan hämtandet av relevant information tillhörande ett PortCallMessage.`
- `PortCallSummaryUtils.java - En hjälpklass för att enklare kunna hämta data ur en PortCallSummary.`
- `PortCDMApi.java - Hanterar` *`hur`* `portCallMessages sätts ihop och även` *`hur`* `de skickas och tas emot. Sätter upp APIer, med rätt användarnamn och lösenord och rätt server (lokal eller extern).`
- `application.fxml (JavaFX) - Hanterar utseendet på applikationen, var olika komponenter sitter, hur de ser ut, etc.`

Protocol

Initialt utvecklades programmet mot en lokal server för att få ordning på de visuella- samt de funktionella aspekterna. Allt eftersom skiftades utvecklingen av programmet mot externa servrar av olika slag. Under projektets gång har det varit oklart vilken server som är mest lämpad att använda som utvecklingsmiljö. Gruppens val att utveckla mot den lokala VirtualBoxen, baserat på kurslitteraturens rekommendation, visade sig vara ett av de sämre valen. Här skulle ett tidigt byte av utvecklingsmiljö underlätta kommunikationen med PortCDM. Nedan beskrivs de olika klienter och servrar som använts samt deras fördelar och nackdelar.

- Lokal klient - Fungerade tidigt att få igång och testa mot och fungerade även med versionen som vi hade på våra PortCallMessages. Det gick dock inte med denna att dela sina meddelanden med andra grupper vilket ledde till problem vid integrationstestet då den nya servern inte var kompatibel med de PortCallMessages som vi skickade.
- Sandbox - Gemensam server som slutpresentationen utfördes mot. De flesta grupper kunde skicka och ta emot meddelanden från denna. Denna servers PortCallMessages var av nyare typ än det vi först hade utvecklat. Detta innebar att applikationen inte kunde ta emot meddelanden från servern samt att servern inte korrekt hanterade applikationens meddelanden.
- Dev-server - Denna server hade en nyare typ av meddelanden än vad den lokala VirtualBoxen vi hade jobbat mot hade, vilket ledde till kompatibilitetsproblem mellan grupper som hade utvecklat mot olika servrar. Om applikationen till en början hade

utvecklats mot denna server hade eventuellt en del problem i utvecklingen i senare stadier av projektet kunnat undvikas.

User stories

Gruppen har varit konsekventa i användandet av *User Stories*. Inför varje sprint konstruerades historier i olika utsträckningar. Initialt var utförandet ganska trevande men fram mot slutet mer konkret och konsekvent. Ambitionen bakom sammanställningen av *User Stories* har också varit varierande beroende på stressnivå och arbetsbörda. Ambitionen har också ofta minskat eftersom att gruppmedlemmar varit införstådda i vad som ska åstadkommas under sprinten. Generellt har dock framställningen av *User Stories* blivit bättre under projektets gång men fortfarande med utrymme för förbättring, speciellt rörande “elefant carpaccio”- teorin. Nedan är en fullständig uppställning av *User Stories*:

https://github.com/filipni/Software-Engineering-Project/tree/master/documents/Sprints