Testning – Busify

Introduktion

Tester genomfördes innan varje pull request lämnades in via Github, under godkännande av pull requests samt under release-processer. Vi hade totalt 27 stycken pull requests och fem release-processer.

För att säkerställa att vi utförde vad vi ansåg lämplig mängd testning i förhållande till de resurser vi hade att utgå ifrån valde vi att inkludera code by contract (se reflektionsrapporten) på vissa Trello-kort. Det innebar då att alla var överens om vilka tester som skulle köras och att det var värdefull tid att lägga på den uppgiften. Vidare innebar detta att när velocity skulle sättas blev testningen medtagen i beräkningen. Det förhindrade att testningen skulle bli en uppgift utförd bristande i ”sista minuten”.

Validering genom acceptanstester

I reflektionsrapporten (D4) beskrivs utförligt hur användargränssnittet testades i nära, löpande interaktion med kunden Fredrik Persson på Göteborg Energi under projektets gång. Dessa intervjuer fungerade som acceptanstester och validering, vilket innebär att vi testade så att vi byggde rätt mjukvara. Kundintervjuerna fyllde även ett validerande syfte innan vi hade en mjukvara att testa, då det är viktigt att testa sin idé i form av ett koncept, innan själva produkten testas[1]. Detta gjordes med hjälp av Value Proposition Canvas, vilket beskrivs i reflektionsrapporten och dess appendix.

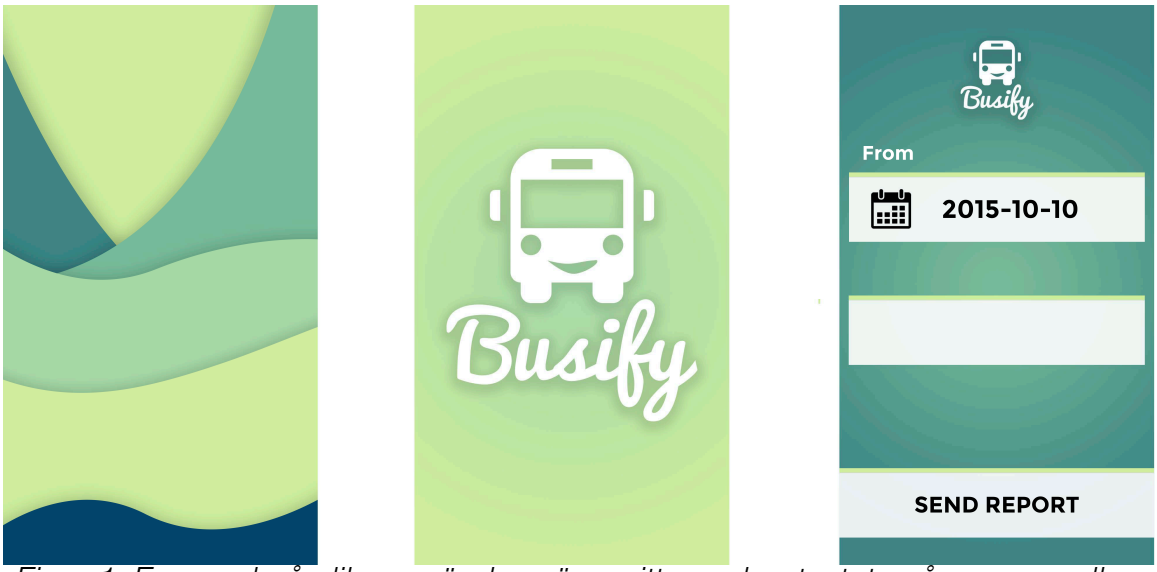
I viss mån fungerade även presentationsdagen den 25 maj 2016 på Lindholmen som acceptanstest då appens användargränssnitt testades mot en rad oberoende parter. Samtlig feedback dokumenterades och diskuterades i gruppen efter presentationsdagen. På så sätt kunde vi vara lyhörda för förändringar[2] och anpassa användargränssnittet till att bli mer intuitivt.

Vi har på sätt och vis även gjort interna acceptanstester, genom att en eller ett fåtal gruppmedlemmar initialt skapade appens användargränssnitt vilket sedan testades av övriga gruppmedlemmar. De kunde på så sätt ge input på vad som var svårt att förstå, ge förbättringsförslag, och beskriva vad som fungerade bra och intuitivt.

---

[1] Brunnegård, Viktor; Chalmers Ventures, Chalmers tekniska högskola. 2016. Entrepreneurship, föreläsning 2016-04-27.

[2] Brunnegård, Viktor; Chalmers Ventures, Chalmers tekniska högskola. 2016. Entrepreneurship, föreläsning 2016-04-27.



Figur 1. Exempel på olika användargränssnitt som har testats på gruppmedlemmar som inte skapade de grafiska delarna.

Verifiering

Vi verifierade mjukvaran, testade om den var byggd på rätt sätt, genom att göra unit tests, integration tests samt system tests. Då det är svårt att göra alla möjliga test, exempelvis för en loop med heltal, testades kod inför varje pull request med någonting som förväntades fungera och någonting som inte förväntades fungera. Exempelvis utfördes statiska tester för metoden stringDivision, vilken tar två Strings som inparametrar, i klassen CsvHandler. Tester utfördes med strängar som motsvarade siffror i form av heltal samt decimaltal, men även med strängar som inte representerade siffror. Då beräkning av strängarnas numeriska värde görs i metoden skapades metoden för att inte acceptera inparametrar som inte kunde omvandlas till decimaltal.

Vi arbetade både med att modellera samt att testa mjukvaran. Genom att initialt modellera backend i form av färdiga filer kunde vi arbeta mer iterativt och parallellt med databasimplementationen utveckla andra delar av produkten. Om vi hade arbetat efter vattenfallsmodellen och väntat tills databasen Firebase och Firebase Storage fungerade som vi planerat skulle vårt resultat med största sannolikhet inte motsvarat lika stort värde för våra intressenter.

Dynamiska tester utfördes framför allt vid pull requests, där vi försökte få koden att misslyckas såväl som att lyckas. Vi försökte utföra både white box och black box testning genom att den som programmerat koden, enligt gruppens överenskomna Design by Contract, testade någonting som fungerade och någonting som inte fungerade. Personen i fråga visste hur programmet, klassen eller metoden fungerade och utförde därmed white box testning. Black box testning försökte vi genomföra initialt under pull requestens genomgång. Då någon annan än den/de som kodade innehållet i pull requesten testade den visste personen i fråga inte om det exakta, kodmässiga innehållet. Genom att börja med att testköra programmet, exempelvis trycka på ”Share report”-knappen, försöka välja datum i DatePickern som ska fungera eller något som inte ska fungera, innan koden sågs över gjordes en form av black box testning. Testaren visste inte hur programmet såg ut inuti, men försökte få ut rätt effekt givet att personen använde appen på rätt sätt.

System- och integrationstester

Systemtester har gjorts för att testa hur Android appen fungerar tillsammans med databasen Firebase och Firebase Storage. Genom att exempelvis testa hur olika filvägar accepteras av metoder som uploadFile i klassen Storage Handler säkerställde vi att kopplingen mellan Android Studio och Firebase var korrekt. Nya filer genereras, från information i databasen, och lagras i Firebase Storage om Firebase Storage inte redan innehåller en fil för det valda datumet. Om så är fallet används existerande fil, istället för att appen ska anropa databasen, generera värden och skriva det till en rapport varje gång. På så vis ökas skalbarheten då arbetsbelastningen för appen blir lägre, särskilt om mängden data och antalet filer ökar. Detta testades genom att köra emulatorn på Android Studio och klicka på såväl rapportdatum som fanns i Firebase Storage och rapportdatum som inte fanns. Genom att samtidigt notera förändringar i Firebase Storage testades att kopplingen fungerade som önskat.

Systemtester mot databasen i Firebase gjordes genom att med hjälp av Log-funktioner göra utskrifter i logcat. I klassen DatabaseHandler lades exempelvis en ValueEventListener till för både databasreferensens rot och dess barn, för att se vilka utskrifter och vilken data som genererades från databasen. Vi försökte hämta information för dagar som inte hade data inskriven i databasen, vilket som förväntat inte fungerade men kastade ett undantag vilket vi istället valde att behandla med ett felmeddelande.

Integrationstester gjordes särskilt i release-processer och deras branches. Då testade den eller de som utförde releasen så att koden fungerade ihop och inte bara isolerat. Detta gjordes även under pull requests, då design- och programmeringsval diskuterades i Github och merge conflicts löstes. Exempelvis testades under release 7.0 att klicka hem rapporter för alla datum, rapporter utan datum, att köra FindBugs och agera på de buggar som upptäcktes samt att köra Androids inbyggda debug-program.

Enhetstester

Enhetstester dokumenterades för varje branch i gruppens gemensamma Google Drive. Exempelvis testades design med att köra appen på emulatorer med olika SDK Plattformar (exempelvis både Marshmallow och Lollipop) samt på smartphones, som en telefon från HTC. Vidare testades designen med olika storlekar för bakgrundsbilden, då vi upptäckte att bakgrunden inte syntes på telefonen trots att det fungerade i emulatorn.

För funktionen att skicka ett email utfördes tester i form av att skriva en mailadress och flera mailadresser med kommatecken, vilka båda förväntades fungera. Vidare testades det att skicka mail utan mottagare och med mellanrum ”som mottagare”, vilka som förväntat inte fungerade. Ett försök gjordes att implementera en metod som först kontrollerade så att epostaddressen fanns, men detta nedprioriterades eftersom vi ansåg att det tillhörde telefonens inbyggda mailapplikation och var utanför omfattningen av vårt projekt.

Exekveringen för funktionaliteten som anropar databasen och hämtar dess värden testades genom att ha knappen tryckbar under hela användningsprocessen. Vi upptäckte att vi behövde trycka två gånger på knappen för att hämta korrekt data, vilket vi märkte genom att utföra tester med utskrift i logcat genom Log-meddelanden. För att undvika dubbelklick testade vi att lägga in anropet till databasen i SplashScreen. Den bästa lösningen vi prioriterade att skapa var en tråd i DatabaseHandler, vilken ser till så att databasen anropas två gånger med 500 millisekunders mellanrum. Genom att inte finslipa lösningen ytterligare hade vi mer tid till att sammanställa en genomarbetad reflektionsrapport och refaktorering, vilket vi ansåg viktigt vid tillfället.

Funktionaliteten med att lägga till en femte kolumn byggdes upp stegvis. Initialt integrerades kod som hanterade databasanrop med MainActivity vilken tillsammans med DatePickern utsåg vilket datum som skulle visas i logcat om en fil för samma datum fanns i Firebase Storage. Genom att därefter först skapa en metod som endast tog värdena från databasen, och skapade en 11 x 4-matris, säkerställde vi att kopplingen till databasen fungerade som önskat. Därefter lades beräkningar av den femte kolumnen till innan den tillslut adderades till det tvådimensionella fältet.

Databasen byggdes, även den, upp steg för steg. Genom att endast ha information om alla tio bussar lagrad för en dag initialt kunde testning av utskrifter göras. Vi insåg att designen av databasen inte var som önskat och ändrade ordning så att datum fungerade som förälder till bussarnas identifierare, och inte tvärtom.

Under ungefär två veckors tid satt en del av gruppen fast på en uppgift som handlade om att bifoga en fil till telefonens inbyggda mailapplikation. Denna uppgift innehöll en hel del testande i utvecklingsprocessen, t.ex. att testa på både olika emulator och telefoner, att använda Gmails API, att placera filen i olika mappar i Android Studio (res, i en mapp i res, i en mapp i main och på nivån Busify), att skriva ut filvägen till filen med hjälp av en Toast, att skapa olika uses permissions, korrigera Manifest.xml för att skapa en så kallad provider för fildelning, att skicka bilder istället för filer med mera.

Rapporter har försökt genereras för datum som har data och för datum som inte har data, vilket då gav en Toast med felmeddelande. Test har också gjorts för att avbryta skickandet genom att få bakåt från mailapplikationen, vilket har gjort att användaren kunde välja ett nytt datum och att rapporten sparades som ett utkast.

När funktionaliteten som erhåller en URL från Firebase skapades upptäcktes problem med asynkrona anrop, vilket ledde till att Busify kraschade. Detta löstes med att emailskickandet först genomfördes efter framgångsrikt erhållande av URL, vilket upptäcktes genom Android Studios debug och Log-meddelanden i logcat. Datum där rapporter fanns samt datum där rapporter inte fanns testades och ett felmeddelande skapades istället för ett kastat undantag vid det senare fallet. Metoden fungerade inte på vissa emulatorer, varför uppdagades aldrig. Då funktionaliteten implementerades nära projektets deadline beslutade gruppen att inte gå vidare med problemet. Test gjordes av att upprepande klicka på ”Share report”-knappen, vilket öppnade en mängd fönster av mailapplikationen, och ledde till att vi gjorde knappen icke-klickbar efter första försöket.

Corner testing

Ett corner test utfördes även i samband med att beräkning av kolumnen vilken innehåller elektricitet per kilometer. Detta var för att säkerställa att division med noll inte förekom. Körsträckan sattes då till noll och resultatet av detta test blev #NaN# vilket var önskat resultat. Det indikerar då för användaren att något är fel eller alternativt att elektricitet per kilometer inte kunde beräknas då bussen inte kört den dagen. Det blir intuitivt klart för användaren då denna buss kommer ha noll för både elektricitet samt körsträcka.

Prioritering av tester

Precis som nämns i reflektionsrapporten var vårt fokus under projektet att skapa en produkt som överensstämmer med Fredrik Perssons behov, och varken gör mer eller mindre. Acceptans testerna har även hjälpt oss att identifiera problem som kan ha blivit förbisedda vid enhetstester eller integrationstester. Vidare har testerna agerat som en möjlighet för kunden att se hur långt vi har kommit i processen. Det har gjort att vi säkerställt att kunden förstår vad som är rimligt för oss att producera med de resurser vi har. Av dessa anledningar har acceptanstester varit högprioriterade. Vi har också prioriterat att kontinuerligt testa koden med enhetstester innan ändringar har pushats till git repot, så att fungerande kod finns på varje branch i så hög utsträckning som möjligt. Därefter har prioritet varit att

utföra integrationstester under releaseprocesserna och tidsmässigt har systemtester fått lägst prioritet, med tanke på att de utförts i slutet av projektet för att integrera en backend och öka applikationens tekniska komplexitet. På så sätt har vi testat, vad vi anser vara, det viktigaste och gjort det på bästa sätt.